Improving semiempirical quantum chemistry with graph neural networks

A hybrid method trained on the QM9 dataset

Aleksandar Davidov

Thesis submitted for the degree of
Master in Computational Science: Chemistry
60 credits

Chemistry
Faculty of Mathematics and Natural Sciences

UNIVERSITY OF OSLO

Spring 2023

Improving semiempirical quantum chemistry with graph neural networks

A hybrid method trained on the QM9 dataset

Aleksandar Davidov

Improving semiempirical quantum chemistry with graph neural networks

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Acknowledgements

I would like to thank my supervisor, Simen Kvaal, for proposing this project to me and for all the help and support he has provided throughout the year. This has been a very interesting project, and I have thoroughly enjoyed every aspect of it. I would also like to extend my thanks to my co-supervisor, David Balcells, for assisting me with topics related to machine learning and for introducing me to Lucas Lang. I would like to express my gratitude to Lucas as well for all the assistance he has offered me throughout the year.

I would also like to thank Jørn and Mira, with whom I shared an office. I had many great discussions with both of you, and I have learned a lot from talking and discussing ideas with you. Lastly, I would like to thank my friend Roy, with whom I used to study. It was great getting to know all of you.

Abstract

Solving the time-independent Schrödinger equation for large systems is essential in many areas in quantum chemistry, such as molecular dynamics or electronic structure analysis. Semiempirical methods have traditionally been used for this due to their computational efficiency. However, these methods, while computationally efficient, often suffer from poor accuracy compared to **ab initio** methods, with their performance greatly dependent on the choice of parameters. Recently, machine learning models have been employed to accurately predict molecular properties, with Message-Passing Neural Networks (MPNNs) garnering particular interest for their ability to capture complex relationships within the data. Yet few have incorporated semiempirical methods as part of a broader machine learning architecture. In this study, we used an MPNN to serve molecule-dependent parameters to the semiempirical method PM3, aiming to leverage the strengths of both methodologies. The model, trained on the QM9 dataset, demonstrated substantial improvements over the bare PM3 method with a negligible increase in computational cost. The model was further validated on a distinct set of molecules, showing enhanced performance in comparison to the PM3 method, reinforcing the potential applicability and robustness of this approach.

Contents

List of Figures

List of Tables

Preface

Part I

The project

Chapter 1

Introduction

The aim of this thesis is to develop a Machine Learning (ML) model trained on the QM9 dataset [1] to predict the parameters for the PM3 [2–4] which is a well-established approach in computational chemistry, and this work aims to enhance its parameterization through iterative training. This means that after each iteration in the training procedure, the PM3 parameters will be updated by the influence of the weights in the ML model. These updated parameters are then used within the PM3 method to calculate internal energy at 0 K, and this calculated energy is compared with the target energy in the dataset. This process is then repeating in the training procedure until the model achieves convergence or meets other predefined stopping criteria. This research seeks to address the challenge of accurately predicting PM3 parameters, which are pivotal in the field of computational chemistry. By employing an iterative ML approach, it aims to bridge the gap between traditional quantum chemistry and modern computational techniques.

The ML model used in this study will be a Message-Passing Neural Network (MPNN), introduced by Gilmer et al. [5]. The choice of this network is motivated by several factors. Firstly, the graph structure of the MPNN is inherently well-suited to represent molecules, a capability that has been specifically demonstrated on the QM9 dataset by Gilmer et al. This suitability arises from the fact that MPNNs can naturally represent molecules as graphs, making them invariant to the ordering of atoms. This is a critical feature in molecular modeling, as predictions must not depend on how atoms are labeled or arranged within the molecule. Moreover, MPNNs offer flexibility in handling various types of input data, including those of variable length, such as the molecules found in the QM9 dataset. This scalability further underscores the appropriateness of MPNNs for this study.

After the design and implementation of the model, which is in a sense a hybrid model integrating the MPNN with the PM3 method, the next phase involves assessing the model's accuracy and robustness. This will be carried out by testing the model on unseen data consisting of 10000 molecules and its performance will be directly compared against the benchmark and the PM3 method.

Through the implementation, testing, and analysis of this research and future research, the hope is to provide insights into the seamless integration of two distinct computational paradigms: the MPNN and the PM3 method. By exploring how these models can interact

and merge into a more sophisticated methodology, this study aims to contribute new perspectives to the field of computational chemistry. While this task is ambitious, the primary aim is to investigate the feasibility and challenges of combining machine learning with traditional quantum chemistry techniques in this was and within the specific context of the PM3 method. The objective is to study how these methods can be combined, recognizing both the new opportunities and the challenges that this mix can create. The work aims to find a middle ground between what's possible and what's practical, striving to give a clear view of a complex task that brings together different scientific fields.

This thesis consists of the five part. The first part which is this introduction 1, the second part is about fundamental theory to this work II. It contains chapters on basic many-body quantum mechanics relevant for quantum chemistry 2, Hartree-Fock theory 3, as well as theory on semiempirical methods 4. It also contains a chapter on optimization 5 as well as a chapter on machine learning which includes Recurrent Neural Networks (RNNs) and Graph Neural Networks (GNNs) as they're essential in this study. The third part of this thesis III goes into the specific theory and implementation and design of the MPNN-PM3 Hybrid model. It contains a chapter about the QM9 dataset and the continuous kernel-based convolutional operator. The part also includes a chapter about the model 8, which goes into the theory and logic of the MPNN-PM3 Hybrid model. The fourth part contains the results of this work IV, and lastly the fifth part contains discussion about the project and future work V.

Part II

Theory

Chapter 2

Basic many-body quantum mechanics

2.1 The Hydrogen Atom

Usually, when we work with quantum mechanics, we want to solve the non-relativistic time-independent Schrodinger equation, which takes the form,

$$\hat{H}|\psi\rangle = E|\psi\rangle$$

where $\hat{H}$ is the Hamiltonian of the system of interest and E is its corresponding energy. $|\psi\rangle \in \mathcal{H}$ is the state vector[1] of the system we are solving for and $\mathcal{H}$ is the Hilbert space, the space of all states system's particles can be in. Usually, when dealing with particles, it is useful to represent the state in the position basis. This leads to the concept of wave function ψ which is the projection of the state vector into the position representation. The wave function is a probability amplitude which encapsulates all the information that can be known about the system and finding the wave functions squared magnitude $|\psi(\mathbf{r})|^2$ gives the probability density of finding a particle at position $\mathbf{r}$. For simplicity, we will assume that $\mathbf{r} \in \mathbb{R}^3$ where $\mathbb{R}^3$ is the set of all three-dimensional real numbers, specifying the position of a particle. For a hydrogen atom, the Schrödinger equation is

$$\left(-\frac{\hbar^2}{2m_e}\nabla^2_{m_e} - \frac{e^2}{4\pi\epsilon_0|\mathbf{r}|}\right)\psi(\mathbf{r}) = E\psi(\mathbf{r}),$$

where the first term is the kinetic energy of the electron and $\hbar$ is the reduced Planck's constant ($\frac{h}{2\pi}$), m_e is the mass of the electron, and ∇^2_e is the Laplacian operator which represents the second spatial derivatives with respect to the electron's coordinates. The second term of the equation represents the potential energy term of the electron due to the electrostatic attraction between the negatively charged electron and the positively charged nucleus. The quantity $-e$ is the charge on the electron and the factor e^2 takes into account both charges of the the nucleus ($+e$) and the electron ($-e$). Lastly, ϵ_0 is the

[1] Vectors on the form $|\psi\rangle$ are reffered to as kets while vectors on the form $\langle\psi|$, which are the adjoint, or complex conjugate of the ket vectors, are called bras. The inner-product $\langle\phi|\phi\rangle$ is called a braket and is anti-linear in the first coordinate.

vacuum permittivity which characterizes the amount of resistance that a vacuum offers to the formation of an electric field[2].

2.2 Molecular Hamiltonian

Solving the Schrödinger equation for a seemingly simple quantum system such as the Hydrogen atom can be a difficult task. The Hydrogen atom is also one of the simplest models we solve in quantum chemistry as it just consists of one nucleus and one electron, as well as their interaction. Since we in quantum chemistry are often interested in finding molecular properties, this usually involves the use of electronic structure theory which involves solving the Schrödinger equation for a molecular Hamiltonian. The molecular Hamiltonian describes the total energy of a molecular system and encapsulates the entire dynamics of the system according to quantum mechanics.

For a system with many interacting electrons and nuclei, the Hamiltonian operator

$$\hat{H}_{\text{mol}} = -\sum_{i=1}^{n} \frac{\hbar^2}{2m_e}\nabla_{e_i}^2 - \sum_{I=1}^{N} \frac{\hbar^2}{2m_I}\nabla_I^2 - \sum_{i=1}^{n}\sum_{I=1}^{N} \frac{Z_I e^2}{4\pi\epsilon_0 r_{iI}} + \sum_{I<J}^{N} \frac{Z_I Z_J e^2}{4\pi\epsilon_0 r_{IJ}} + \sum_{i<j}^{n} \frac{e^2}{4\pi\epsilon_0 r_{ij}} \tag{2.1}$$

where m_e is the mass of the electron and m_I is the mass of the Ith nuclei respectively. $r_{ij} = |\mathbf{r}_i - \mathbf{r}_j|$ is the absolute distance between the ith and jth electron, $r_{iI} = |\mathbf{r}_i - \mathbf{r}_I|$ is the distance between the ith electron and Ith nuclei, and $r_{ij} = |\mathbf{r}_I - \mathbf{r}_J|$ is teh distance between Ith and Jth nuclei. $\nabla_{e_i}^2$ and ∇_I^2 are the Laplacian operators with respect to electron i and nucleus I, while Z_I is the atomic number of nucleus I. The first term in the molecular Hamiltonian, $\hat{T}_e$, is the kinetic energy operator for the n electrons while the second term, $\hat{T}_N$, is the kinetic energy operator for the N nuclei. The third term, $\hat{V}_{eN}$, gives the Coulomb attraction between the electrons and the nuclei while the forth and fifth terms, $\hat{V}_{NN}$ and $\hat{V}_{ee}$, describe the repulsion between the nuclei and between the electrons respectively. We can hence use the labels of the operators to write the molecular Hamiltonian as such:

$$\hat{H}_{\text{mol}} = \hat{T}_e + \hat{T}_N + \hat{V}_{eN} + \hat{V}_{NN} + \hat{V}_{ee}.$$

The Electronic problem

Solving the Schrödinger equation for a molecular Hamiltonian can be a difficult task as we are dealing with coupled equations for the nuclei and electrons. For example, it requires us to solve for the motion of both electrons and nuclei simultaneously, as well as their correlation. Solving for such systems leads to a coupled differential equation which is very difficult to solve mathematically. Considering the molecular Hamiltonian from Equation 2.1, we can represent it using atomic units, meaning we set $\hbar = m_e = m_I = e = \pi = Z_I = 1$. The molecular Hamiltonian can then be written

$$\hat{H}_{\text{mol}} = -\sum_{i=1}^{n} \frac{1}{2}\nabla_{e_i}^2 - \sum_{I=1}^{N} \frac{1}{2}\nabla_I^2 - \sum_{i=1}^{n}\sum_{I=1}^{N} \frac{1}{r_{iI}} + \sum_{I<J}^{N} \frac{1}{r_{IJ}} + \sum_{i<j}^{n} \frac{1}{r_{ij}}. \tag{2.2}$$

[2]The factor $\frac{1}{4\pi}$ comes from the formulation of Coulomb's law in three-dimensional space. The factor takes into account the way the electric field spreads out in all directions. We can think about it as a sphere with surface $4\pi r^2$ where r is the distance from the charge.

However, since solving the Schrödinger equation for a molecular system is difficult, we will utilise the Born-Oppenheimer approximation (BOA) [6]. The BOA is based on the critical observation that the mass of a nucleus is much greater than the mass of an electron and consequently, nuclei move much slower relative to the electrons. It leverages this mass disparity by treating the nuclear positions as fixed parameters when considering the electronic motion. Essentially, it means that the electrons "adjust" instantaneously to any change in the nuclear position, meaning they move in a field of nuclei. The resulting equation can then be solved with the positions of the nuclei set as parameters which will then result in a potential energy surface (PES) forming the basis for solving the nuclear motion [7, Ch.3]. The total wave function $\psi(\mathbf{r}, \mathbf{R})$, which now depends on both the electronic and nuclear position as the equation is coupled, can then be factorized into a product of electronic and nuclear wave functions. It then becomes

$$\psi(\mathbf{r}, \mathbf{R}) = \psi(\mathbf{r}; \mathbf{R})\chi(\mathbf{R}).$$

Here, for easier distinguishability of the position of the electrons and the nuclei, we will use $\mathbf{r}$ to denote just the electronic coordinates and $\mathbf{R}$ to denote the nuclear coordinates. The electronic wave function $\psi(\mathbf{r}; \mathbf{R})$ depends parametrically on the nuclear positions, while the nuclear wave function $\chi(\mathbf{R})$ is a function of the nuclear coordinates alone.

By using the BOA, the second term in Equation (2.2) (kinetic energy for the nuclei) is then set to zero, and the last term (repulsion of the nuclei) is set as a constant. The last term can then be neglected because adding a constant to an operator does not effect the corresponding operator eigenvectors as it only adds to the eigenvalues. Doing this, we then get the electronic Hamiltonian

$$\hat{H}_{\text{elec}} = -\sum_{i=1}^{n} \frac{1}{2}\nabla_{e_i}^2 - \sum_{i=1}^{n}\sum_{I=1}^{N} \frac{1}{r_{iI}} + \sum_{i<j}^{n} \frac{1}{r_{ij}} + \hat{V}_{\text{NN}}$$

This electronic Hamiltonian represents the motion of n electrons in the field of N nuclei and solving it yields the corresponding electronic energy and wave-function. Hence the Schrödinger equation becomes

$$\hat{H}_{\text{elec}}\psi_{\text{elec}} = E_{\text{elec}}\psi_{\text{elec}}$$

where ψ_{elec} is the corresponding electronic wave-function and E_{elec} is the corresponding electronic energy for $\hat{H}_{\text{elec}}$.

2.3 Orbitals and Slater determinant

We know that elementary particles can be either *fermions*, such as electrons, or *bosons*, like photons. Both fermions and bosons are types of indistinguishable particles. This concept arises due to the fact that since we represent particles as mathematical objects, the wave function, which gives the probability of finding a particle in that particular location or with a particular set of properties. When there are two or more indistinguishable particles described by the same wave-function, it becomes impossible to distinguish one particle from another.

2.3.1 Spin orbitals

As we are in quantum chemistry primarily interested in many-body systems containing electrons, we will stick to the commonly accepted notation and define an orbital as the spatial part of the wave function of a single electron. For wave functions describing the electrons in a molecule, we will be using Molecular Orbitals (MOs). A spatial orbital is a wave function which depends on the position $\mathbf{r}$ of an electron and describes the spatial distribution of that particular electron. Spatial orbitals are assumed to be orthonormal, meaning:

$$\int \psi_i^*(\mathbf{r})\psi_i(\mathbf{r})\,\mathrm{d}\mathbf{r} = \delta_{ij}$$

where i and j are the indices labeling different wave functions in a quantum system. The wave function $\psi_i^*(\mathbf{r})$ indicates the complex conjugate of $\psi_i(\mathbf{r})$ while $\mathrm{d}\mathbf{r} = \mathrm{d}x\mathrm{d}y\mathrm{d}z$ represents an infinitesimal volume element in the space where the wave functions [3]. Lastly, δ_{ij} is the Kronecker delta defined as:

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{cases}$$

If $i = j$ then the integral of the product of a wave function with its complex conjugate over all space is 1, representing the normalization condition of the wave function. If $i \neq j$ then the integral is 0, indicating that the two different wave functions are orthogonal to each other.

But the position of an electron doesn't just depend on its spacial coordinates $\mathbf{r}$, but also its spin. Therefore, for a complete description of the wave function of an electron, it is necessary to include spin. Therefore, to include this, we will define the spin functions $\sigma_1 = \uparrow$ indication spin up, and $\sigma_2 = \downarrow$ indicating spin down. Again, if the spatial orbitals are orthonormal then the spin orbital is also orthonormal

$$\int \psi_i^*(\nu)\psi_i(\nu)\,\mathrm{d}\nu = \delta_{ij}$$

where ν indicates both spacial and spin coordinates.

For the remainder of this chapter, the wave function ψ_i will always be used to describe a spin orbital.

2.3.2 Pauli Exclusion Principle

When defining a multi-electron wave function, it is important to consider the Pauli Exclusion Principle [8]. It states that two or more fermions cannot occupy the same quantum state within a quantum system simultaneously. This is because if two electrons occupy the same position $\mathbf{r}$, their spin value has to be opposite. Electrons, and fermions in general are therefore said to be antisymmetric.

Suppose we have one electron in the state ψ_i and one electron in state ψ_j, then The combined wave function of of two electrons

$$\Psi(\nu_1, \nu_2) = \frac{1}{\sqrt{2}}\left[\psi_i(\nu_1)\psi_j(\nu_2) - \psi_j(\nu_2)\psi_i(\nu_1)\right]$$

where the factor of $\frac{1}{\sqrt{2}}$ is the normalization constant and Ψ is a multi-electron wave function, or two-electron in this case [9].

[3] In an integral, this would mean that we integrate over all all three spacial dimentions.

2.3.3 Slater Determinant

The concept of antisymmetry for two-electron systems can also be extended to larger multi-electron systems by introducint the Slater Determinant (SD) [10]. For an N electron quantum system, the SD is as follwos:

$$\Psi(\nu_1, \nu_2, \ldots, \nu_N) = \frac{1}{\sqrt{N!}} \begin{vmatrix} \psi_i(\nu_1) & \psi_i(\nu_2) & \ldots & \psi_i(\nu_N) \\ \psi_j(\nu_1) & \psi_j(\nu_2) & \ldots & \psi_j(\nu_N) \\ \vdots & \vdots & \ddots & \vdots \\ \psi_n(\nu_1) & \psi_n(\nu_2) & \ldots & \psi_n(\nu_N) \end{vmatrix} \tag{2.3}$$

where the factor $\frac{1}{\sqrt{N!}}$ is the normalization factor for a system of N electrons. The N electrons occupy N spin orbitals without any specification of which electron is in which orbital. The rows are labeled by the electrons while the columns are labeled by the spin orbitals. We can see that the SD meets the criterion for antisymmetry as interchanging the coordinates of electrons corresponds to interchanging two rows of the SD whih then changes the sign of the determinant. We also see that two electrons can occupy a spin orbital as it would make two columns of the determinant equal to zero which would as a consequence make the entire SD equal to zero.

Chapter 3

Hartree-Fock Theory

3.1 Basic description

The Hartree-Fock method represents an essential evolution in quantum chemistry, linking the foundational theories with modern computational practice. Through the use of self-consistent field equations, it allows for the systematic approximation of electronic interactions within a molecular system, despite certain simplifications such as the neglect of electron correlation. As a key method providing insights into electronic structure and behavior, it has laid the foundation for more advanced techniques that explore intricate electron-electron correlations, thus offering a bridge between historical theoretical advances and contemporary research applications.

In Hartree-Fock theory, the primary aim is to locate a single Slater Determinant that minimizes the expectation energy. This is realized through the application of variational principles, fostering an optimal depiction of the ground state. The energy of this Slater Determinant can be succinctly articulated as:

$$E = \langle \psi | \hat{H}_{\text{elec}} | \psi \rangle \tag{3.1}$$

where $\hat{H}_{\text{elec}}$ is the electronic Hamiltonian from Equation (2.2). This formulation accentuates Hartree-Fock's key objective and its role in bridging computational efficiency with a meaningful representation of electronic interactions, thereby forming a vital step in quantum chemical analysis.

To begin with, we will write the electronic Hamiltonian as the sum of a one and two-body operator

$$\hat{H}_{\text{elec}} = \sum_{i=1}^{n} h_i + \sum_{i<j}^{n} g_{ij} + \hat{V}_{\text{NN}}$$

where

$$h_i = -\frac{1}{2}\nabla_{e_i}^2 - \sum_{I=1}^{N} \frac{Z_I}{r_{Ii}}$$

and

$$g_{ij} = \frac{1}{r_{ij}}.$$

Here, we have introduced the one-electron operator, or core-Hamiltonian, h_i, which describes the motion of electron i in the field of all the nuclei, and g_{ij} is the two-electron operator giving the electron–electron repulsion. Further, we can now define

$$J_{ij} = \int \psi_i(1)^* \psi_j(2)^* \frac{1}{r_{12}} \psi_i(1)\psi_j(2)\,\mathrm{d}1\mathrm{d}2$$

and

$$K_{ij} = \int \psi_i(1)^* \psi_j(2)^* \frac{1}{r_{12}} \psi_i(2)\psi_j(1)\,\mathrm{d}1\mathrm{d}2$$

where we have used the short-hand definition and matrix element J_{ij} is the Coulomb integral and the matrix element is the K_{ij} is the exchange integral and $\mathrm{d}\nu_1\mathrm{d}\nu_2$ denotes the integration over spatial and spin coordinates of electrons 1 and 2. The matrix element J_{ij} represents the electrostatic repulsion between an electron in the spin-orbitals ψ_i and one in ψ_j while the exchange integral has no classical analogue. It originates mathematically from terms in the expansion of the Slater determinant that are distinct solely in the exchange of electrons.

By using the Coulomb and exchange terms, we can derive the energy expression (3.1) of the system in the following manner:

$$\begin{aligned} E &= \sum_{i=1}^{n} \langle \psi_i(1) | h_1 | \psi_i(1) \rangle \\ &\quad + \sum_{i<j}^{n} \left(\langle \psi_i(1)\psi_j(2) | g_{12} | \psi_i(1)\psi_j(2) \rangle - \langle \psi_i(1)\psi_j(2) | g_{12} | \psi_i(2)\psi_j(1) \rangle \right) + \hat{V}_{\mathrm{NN}} \\ &= \sum_{i=1}^{n} h_i + \sum_{i<j} (J_{ij} - K_{ij}) + \hat{V}_{\mathrm{NN}} \\ &= \sum_{i=1}^{n} h_i + \frac{1}{2} \sum_{ij} (J_{ij} - K_{ij}) + \hat{V}_{\mathrm{NN}} \end{aligned} \tag{3.2}$$

The factor of $\frac{1}{2}$ in in the third term accounts for the fact that the summation over $i < j$ includes only unique pairs, while the summation over all ij counts each pair twice. Thus, the expression is halved to avoid double-counting.

Further, we can rewrite the energy by introducing the Coulomb and exchange operators:

$$\hat{J}_i \psi_j(2) = \psi_j(2) \int \psi_i^*(2) \frac{1}{r_{12}} \psi_i(2)\,\mathrm{d}\nu_2$$

and

$$\hat{K}_i \psi_j(2) = \psi_i(2) \int \psi_i^*(2) \frac{1}{r_{12}} \psi_j(2)\,\mathrm{d}\nu_2$$

leading to the energy expression as:

$$E = \sum_{i=1}^{n} \langle \psi_i | h_i | \psi_i \rangle + \frac{1}{2} \sum_{ij} (\langle \psi_j | \hat{J}_i | \psi_j \rangle - \langle \psi_j | \hat{K}_i | \psi_j \rangle + \hat{V}_{\mathrm{NN}}.$$

We can now introduce the Fock operator which is central in HF theory:

$$\hat{F}_i = h_i + \sum_{j=1}^{n} (\hat{J}_j - \hat{K}_j)$$

which is often expressed as a sum of the core-Hamiltonian operator h_i and the effective potential v_{eff}, where $v_{\text{eff}} = \sum_{j=1}^{n}(\hat{J}_j - \hat{K}_j)$. This leads us to the Hartree-Fock equation:

$$\hat{F}_i\psi_i = \epsilon_i\psi_i. \tag{3.3}$$

which serves as the cornerstone of the of the HF method, where the spin-orbitals are the eigenstates, and their corresponding energies are the eigenvalues.

3.2 Basis-set

Solving the Hartree-Fock equations exactly is only possible for small highly symmetric systems. These systems include atoms which are spherical symmetric as well as molecules that exhibit significant symmetry in their geometric or electronic structure, often leading to simplifications in solving the equations. It is therefore normal to introduce basis functions for the expansion of the spatial part of the spin orbitals and solve a set of matrix equations. These basis functions are typically chosen from a predefined set that captures the essential features of the quantum system, allowing for a more tractable numerical solution. When the basis set approaches completeness, the spin-orbitals that one obtains will approach the exact Hartree-Fock spin orbitals. Even though the Hartree-Fock equation from (3.3) has the form of a linear equation, it is in fact a pseudo-eigenvalue equation. This characteristic arises because, unlike in a true eigenvalue equation where the operator is not dependent on the function on which it acts, here the Fock operator has functional dependence through the Coulomb and exchange operators. A pseudo-eigenvalue equation involves this type of functional dependency, adding complexity to the mathematical structure. This makes the Hartree-Fock equations nonlinear, which necessitates solving them iteratively [11, Ch.3]. A set of functions that is a solution to Equation (3.3) is called self-consistent field (SCF) orbitals. It is these orbitals we aim to find through the iterative process, which is known as the SCF loop.

However, expanding upon the choice of basis functions mentioned earlier, essentially all calculations use a basis set expansion to express the unknown MOs, or the SCF orbitals, in terms of a set of known functions. The selection of basis functions may encompass a diverse array, including but not limited to exponential, Gaussian, polynomial, cube functions, wavelets, and plane waves.

- They should have a behavior that agrees with the physics of the problem, since this ensures that the convergence as more basis functions are added is reasonably rapid. Specifically, the functions must exhibit a behavior that ensures a reasonably rapid convergence as additional basis functions are incorporated. In the context of bound atomic and molecular systems, this necessitates that the functions diminish to zero as the distance between the nucleus and the electron grows large.
- The chosen functions should make it easy to calculate all the required integrals [7, Ch.5].

When using basis sets, all molecular orbitals ψ are expanded in terms of basis functions ϕ using the linear combination of atomic orbitals (LCAO) method, which can be mathematically

expressed as

$$\psi_i = \sum_{\alpha=1}^{M} c_{\alpha i}\phi_\alpha,$$

where M is the number of basis functions, ϕ_α represents the individual basis functions, or atomic orbitals, and $c_{\alpha i}$ denotes the coefficients for the expansion of the molecular orbitals ψ_i in terms of the basis functions ϕ_α.

Further, we can write the Hartree-Fock equation (3.3) in matrix form, which yields the Roothaan-Hall eqation

$$\mathbf{FC} = \mathbf{S}\boldsymbol{\epsilon}, \tag{3.4}$$

where $\mathbf{F}$ is the Fock matrix with elements $F_{\alpha\beta} = \langle\phi_\alpha|\mathbf{F}|\phi_\alpha\rangle$, $\mathbf{C}$ is the matrix that contains the coefficients, and $\boldsymbol{\epsilon}$ is the diagonal matrix containing the orbital energies.

One critical aspect of this formulation is the overlap matrix $\mathbf{S}$, which contains the overlap elements

$$S_{\alpha\beta} = \langle\phi_\alpha(1)|\phi_\beta(1)\rangle = \int \phi_\alpha^*\phi_\beta \,\mathrm{d}\nu_1 \tag{3.5}$$

representing the extent to which two atomic orbitals "overlap" with each other. This measure takes a value between 0 and 1, with $S_{\alpha\beta} = 0$ indicating orthogonality and $S_{\alpha\beta} = 1$ signifying identical atomic orbitals.

In Hartree-Fock theory, it is preferred to write the Hartree-Fock elements as a sum of the density matrix and the two-electron integrals. To do that we first expand the Hartree-Fock elements:

$$\begin{aligned} F_{\alpha\beta} &= \langle\phi_\alpha|h|\phi_\beta\rangle + \sum_{j=1}^{n}\langle\phi_\alpha|\hat{J}_i - \hat{K}_i|\phi_\beta\rangle \\ &= \langle\phi_\alpha|h|\phi_\beta\rangle + \sum_{j=1}^{n}\left(\langle\phi_\alpha\psi_j|g|\phi_\beta\psi_j\rangle - \langle\phi_\alpha\psi_j|g|\psi_j\phi_\beta\rangle\right) \\ &= \langle\phi_\alpha|h|\phi_\alpha\rangle + \sum_{ab}^{M}\sum_{j=1}^{n} c_{aj}c_{bj}\left(\langle\phi_\alpha\phi_a|g|\phi_\beta\phi_b\rangle - \langle\phi_\alpha\phi_a|g|\phi_b\phi_\beta\rangle\right) \end{aligned} \tag{3.6}$$

Now we can introduce the density matrix $D_{ab} = \sum_j c_{aj}c_{bj}$ as well as the two electron part $G_{\alpha\beta ab} = \left(\langle\phi_\alpha\phi_a|g|\phi_\beta\psi_b\rangle - \langle\phi_\alpha\psi_a|g|\psi_b\phi_\beta\rangle\right)$. With these definitions, the Fock matrix elements can be succinctly written as:

$$F_{\alpha\beta} = h_{\alpha\beta} + \sum_{ab} G_{\alpha\beta ab}D_{ab}.$$

Consequently, this formulation leads to an expression for the energy (3.2) in terms of the density matrix, one- and two-electron integrals

$$E = \sum_{ab}^{M} D_{ab}h_{ab} + \frac{1}{2}\sum_{ab\alpha\beta}^{M}\left(D_{ab}D_{\alpha\beta} - D_{a\beta}D_{\alpha b}\right)\langle\phi_a\phi_\alpha|g|\phi_b\phi_\beta\rangle + \hat{V}_{\mathrm{NN}}$$

where the one-electron integral is

$$h_{ab} = \langle \phi_a | h | \phi_b \rangle = \int \phi_a(1) \left(-\frac{1}{2} \nabla_1^2 \right) \phi_b \, \mathrm{d}\nu_1 \tag{3.7}$$

$$+ \sum_{A=0}^{N} \int \phi_a(1) \left(\frac{Z_A}{r_{A1}} \right) \mathrm{d}1. \tag{3.8}$$

and the two-electron integral

$$\langle \phi_a \phi_\alpha | g | \phi_b \phi_\beta \rangle = \int \phi_a(1) \phi_\alpha(2) \frac{1}{r_{12}} \phi_b(1) \phi_\beta(2) \, \mathrm{d}1\mathrm{d}2. \tag{3.9}$$

This representation is particularly useful as it allows for a more concise expression of the HF equations, and it can be directly applied in numerical methods to solve for the electronic structure of molecular systems.

Alternatively, we cal also write the energy using the definition of the Fock operator from Equation (3.1) as

$$E = \sum_{i=1}^{n} \epsilon_i - \frac{1}{2} \sum_{ij}^{n} (j_{ij} - K_{ij}) + \hat{V}_{\mathrm{NN}}$$

with

$$\epsilon_i = \langle \psi_i | F_i | \psi_i \rangle = h_i + \sum_{j=1}^{n} (J_{ij} - K_{ij}).$$

As we can see, the Fock operator accounts for the electron-electron repulsion through the Coulomb and exchange operators. In calculating the energy, this repulsion is considered twice in the summation, necessitating the inclusion of a factor of $1/2$ to correct for this duplication. Looking at the equation, it is also clear that the total energy is not exact as it provides an average representation of an electron with all the other electrons by assuming their spatial distribution within a set of orbitals. Since the total energy only cointaines the averaged electorn-electron repultion and the calculation is based on a single SD which can not accurately describe most molecules, it cannot be exact, reflecting the method's mean-field approximation [7, p. 100].

Chapter 4

Semiempirical Methods

4.1 Introductory theory

When dealing with semi-empirical methods, we have to introduce some new terms. Specifically the notion of core and center. The core is the part of the atom that consists of the nucleus and the inner electrons. These inner electrons are often tightly bound to the nucleus and are not involved in chemical bonding or reactions. Therefore, they are sometimes treated separately from the outer (valence) electrons, which are responsible for most of the chemical behavior of the atom. The term "center" in the context of one-center and two-center integrals refers to an atomic nucleus or a group of nuclei. One-center integrals involve functions centered around a single nucleus, while two-center integrals involve functions centered around two different nuclei.

Semi-empirical methods use the Zero Differential Overlap (ZDO) approximation, a fundamental assumption that posits there is no overlap between atomic orbitals situated on different atoms. This means that if you have two atomic orbitals on different atoms, labeled as i_a and j_b, the product of these orbitals will be zero. Specifically, under the ZDO approximation, three- and four-center integrals are set to zero. This simplification is based on the extent to which differential overlap is disregarded. The concept of differential overlap, denoted as $\mathrm{d}S$ is described mathematically by the equation

$$\mathrm{d}S = \phi_i(1)\phi_j(1)\,\mathrm{d}\nu_1.$$

This approximation assumes that the change in overlap between two orbitals is negligible when an electron moves within the differential volume element $\mathrm{d}\nu_1$. Besides that, including this assumption leads to substantial computational savings, especially in complex systems since the three- and four- four center integrals take up most of the computing time. [12].

To elucidate the distinctions among semi-empirical methods, we employ a specific mathematical representation, recasting Equation (3.6) in semi-empirical notation. This transformation leads to a new expression for the Fock matrix element, denoted as F_{ij}, allowing us to understand and compare different semi-empirical techniques more effectively [7, Ch. 7]. The Fock matrix element in semi-empirical methods can be expressed as:

$$F_{ij} = h_{ij} + \sum_{r=1}^{m}\sum_{s=1}^{m} D_{rs}(\langle ij|rs\rangle - \langle ir|js\rangle)$$

where a two-electron integral is abbreviated as $\langle ij|rs\rangle$ involving the basis functions i, j, r, and s. We also have

$$h_{ij} = \langle i|h|j\rangle.$$

Here, h_{ij} as in the above equation, is the matrix element of the one-electron Hamiltonian between the basis functions i and j, and it encapsulates the kinetic energy of an electron and its potential energy due to interactions with the core electrons and the atomic nuclei. The summation term accounts for the cumulative contributions of electron pairs to the total electronic energy, factoring in their mutual interactions and those with atomic nuclei. Approximations are made for the one- and two-electron components. The semi-empirical notation simplifies the expression by abbreviating complex integrals and highlighting the underlying physical interactions. The recasting of the expression in semi-empirical notation allows for a systematic comparison of various semi-empirical methods by

- Isolating key components: By focusing on the distinct elements of the Fock matrix, we can identify the specific approximations, assumptions, and computational strategies that characterize different semi-empirical methods.
- Enhancing Interpretability: The simplified notation unveils the mathematical structure of the methods, making it easier to recognize the commonalities and differences.

4.1.1 Neglect of Diatomic Differential Overlap (NDDO)

In the NDDO approximation, which is relevant for the modern semiempirical methnods, there are no further approximations than those mentioned above. Using i and j to denote either an s- or p-type (p_x, p_y or p_z) orbital, the NDDO approximation is defined by the following equations, beginning with the overlap integral (3.5) which now takes the form:

$$S_{ij} = \langle i_A|j_B\rangle = \delta_{ij}\delta_{AB}$$

This simplification is based on the premise that overlap between atomic orbitals on different atoms can be neglected. The only non-zero overlap is between the same kind of orbital on the same atom. Assumptions are also made to the one-electron operator (3.2):

$$\begin{aligned} h &= -\frac{1}{2}\nabla^2 - \sum_A^N \frac{Z'_A}{|\mathbf{R}_A - \mathbf{r}|} \\ &= -\frac{1}{2}\nabla^2 - \sum_A^N V_A \end{aligned}$$

where Z'_A is the nuclear charge which has been reduced by the number of core electrons, and the term $\frac{Z'_A}{|\mathbf{R}_A-\mathbf{r}|} = V_A$ represents the potential energy of the electron, $\mathbf{r}$ due to its interaction with the Ath nucleus at position $\mathbf{R}_A$ determined by the charge Z'_A. Further, the one-electron integrals (3.7) now take the form:

$$\langle i_A|h|j_A\rangle = \left\langle i_A\right| -\frac{1}{2}\nabla^2 - V_A\left|j_A\right\rangle - \sum_{a\neq A}^N \langle i_A|V_a|j_A\rangle \tag{4.1}$$

with

$$\langle i_A|h|j_B\rangle = \left\langle i_A\right| -\frac{1}{2}\nabla^2 - V_A - V_B\left|j_B\right\rangle \tag{4.2}$$

and

$$\langle i_A | V_C | j_B \rangle = 0. \tag{4.3}$$

where Equation (**??**), represents the matrix element of the one-electron Hamiltonian between orbitals and on the same atom. The term V_b is the potential energy due to the nucleus A, while the second term on the right-hand side represents the potential energy due to all other nuclei. In Equation (4.2), potential energy operators V_A and V_B represent the electrostatic attraction between the negatively charged electron and the positively charged nucleus of atoms A and B. The equation is expressing the expectation value of the sum of kinetic and potential energy operators for an electron in a pair of orbitals, i_a and j_B. The Equation 4.3 states that the potential energy due to a third nucleus C is zero when calculated on different atoms A and B. This is a reflection of the NDDO approximation which assumes atomic orbitals on different atoms is negligible.

It is also important to note that the first one-center matrix element in Equation (4.1) equates to zero owing to the orthogonality of atomic orbitals, unless the two functions involved are the same, as stated by Jensen in 2017 [7].

$$\left\langle i_A \right| -\frac{1}{2}\nabla^2 - V_A \left| j_A \right\rangle = \delta_{ij} \left\langle i_A \right| -\frac{1}{2}\nabla^2 - V_A \left| i_A \right\rangle$$

In other words, the one-electron integral is zero unless the two orbitals are identical $(i = j)$. This is due to the orthogonality of the atomic orbitals, which means that the integral of the product of two different orbitals is zero.

Lastly, we have the two-electron integrals from Equation (3.9) is reduced to

$$\langle i_A j_B | r_C s_D \rangle = \delta_{AC}\delta_{BD} \langle i_A j_B | r_A s_B \rangle$$

which represent the interaction between electrons in orbitals i_A and j_B with electrons in orbital r_C and s_D. It is reflected through δ_{AC} and δ_{BD} that interactions between different atoms is negligible.

Generally, NDDO methods serve as the benchmark for all-purpose semiempirical methods, which is the focus of the remaining sections of this chapter [12, Ch.6].

4.2 Modified Neglect of Diatomic Overlap (MNDO)

Transitioning our discussion to MNDO, we begin an exploration of the first in a set of modern algorithms. These are distinct yet interconnected implementations of the NDDO model, each defined by a specific parameterization in terms of atomic variables. The intention behind this approach is to focus on the properties intrinsic to individual atoms, providing a deeper understanding of each atom's role and contributions within the model. These modern algoruthms, MNDO, AM1 and PM3, all stem from the foundational NDDO approximations but differ in their treatments of core-core repulsion and parameter assignments. These methods exclusively consider the valence s- and p-functions, which are modeled as Slater-type orbitals (STOs) with respective exponents ξ_s and ξ_p [7, Ch.7] [13].

STOs are mathematical functions used to describe the shape and orientation of atomic orbitals, which are regions in an atom where electrons are most likely to be found. They

decreases exponentially with increasing distance from the atomic nucleus, which makes them particularly effective in modeling the behavior of electrons in atoms. The exponents ξ_s and ξ_p control the rate of this decrease, which in effect governs the "spread" of the orbital. Specifically, an orbital with a larger exponent value (indicating a faster rate of decrease) is more localized close to the nucleus, while an orbital with a smaller exponent value is more diffuse, or spread out over a larger volume.

The MNDO method is widely recognized for providing fairly acceptable qualitative results when applied to a variety of organic systems. Nonetheless, there exist instances where MNDO fails to deliver accurate outcomes, either qualitatively or quantitatively. A notable flaw observed with this method is the tendency to underestimate computed electronic excitation energies. Specifically, this underestimation means that the computed transition between electronic states often predicts lower energy requirements than what is experimentally observed, leading to potential inaccuracies in understanding electronic behavior in these systems [14, Ch.4].

The evaluated one-center one-electron integrals encompass the energy of a single electron subject to experiencing the nuclear charge of its own atom, denoted U_{ss} or U_{pp} for s and p-orbitals respectively, and the cumulative potentials exerted by all other nuclei in the system Equation (4.1). This additional influence is parameterized in terms of the reduced nuclear charges Z' and a two-electron integral. Consequently, the one-electron core Hamiltonian matrix element between specific orbitals i_A or j_A, denoted as h_{ij}, includes both the kinetic and potential energy of an electron under the influence of its own nucleus (U_{ii}), as well as the potential effects from the rest of the system.

$$h_{ij} = \langle i_A|h|j_A\rangle = \delta_{ij}U_{ii} - \sum_{a\neq A} Z'_a\langle i_A s_a|h|j_A s_a\rangle$$

where

$$U_{ii} = \left\langle i_A\right| -\frac{1}{2}\nabla^2 - V_A\left|i_A\right\rangle .$$

We see that U_{ii} in effect, represents the expected value of the Hamiltonian operator for the electron in the given atomic orbital, meaning in the field of its own nucleus. Therefore U_{ii} corresponds to a one-center one-electron integral, of an electron in relation to its own nucleus. Importantly, when the atomic orbitals under consideration belong to different symmetry species, the core Hamiltonian matrix elements become zero due to the spherical symmetry of the kinetic plus potential energy operator [15, p. 325].

The two-center one-electron integrals given by Equation (7.6) are now reduced to an expression involving the overlap integral and atomic "resonance" parameters, β. This simplification directly multiplies the overlap integral with the average of the two atomic resonance parameters, indicative of the orbitals' inherent characteristics. This gives:.

$$\langle i_A|h|j_B\rangle = \frac{1}{2}S_{ij}(\beta_i + \beta_j)$$

where the overlap integral between the atomic orbitals $S_{\mu\nu}$ is calculated explicitly. The quantity is explicitly computed, thereby deviating from the traditional ZDO approximation. It quantifies the extent to which these orbitals share the same space and is used in the

computation of these matrix elements. Its magnitude provides a measure of the degree to which the electron wavefunctions overlap in space. This explicit computation of the overlap integral is the source of the "modified" descriptor in the method's nomenclature. The two atomic resonance parameters, averaged by to the factor of $1/2$, are typically determined empirically and reflect the average potential for electron exchange between the orbitals.

Moving to the one-center two-electron integrals taken into account in the NDDO approximation, specifically within the context of an s and p-basis set, these are represented by the following expressions:.

$$\begin{aligned}\langle ss|ss\rangle &= g_{ss}\\ \langle pp|pp\rangle &= g_{pp}\\ \langle sp|sp\rangle &= g_{sp}\\ \langle pp'|pp'\rangle &= g_{p2}\\ \langle ss|pp\rangle &= h_{sp}\end{aligned}$$

The g-type parameters are Coulomb terms, while the h parameter is an exchange integral. The g_{p2} integral involves two different types of p-functions (i.e. p_x, p_y or p_z). These integrals have distinct roles in accounting for electron-electron interactions. The parameters labeled with g are associated with Coulomb interactions. This involves interactions within the same types of orbitals, including $s-s$, $p-p$, and $s-p$; and interactions within different types of p-functions (e.g., p_x, p_y, or p_z), denoted by g_{p2}. The last integral type, represented by h_{sp}, is connected with the exchange integral, which takes into account the quantum mechanical exchange phenomenon arising from the Pauli exclusion principle. The distinction between g and h parameters provides a practical and efficient way to encapsulate key interactions involved in a system's electronic structure within the context of the NDDO approximation. For the MNDO method, these values have been obtained from experiments taken from the atomic spetra, while the others are fitted to molecular data [7]

The core-core repulsion in the context of the MNDO model refers to the repulsive interaction between the inner shell (or core) electrons of two different atoms. The core–core repulsion of the MNDO model [16], which refers to the repulsive interaction between the inner shell (core) electrons of two different atoms takes the form:

$$V_{CC}(A,B) = Z'_A Z'_B \langle s_A s_A | s_B s_B\rangle (e^{-\alpha_A R_{AB}} - e^{-\alpha_B R_{AB}} + 1),$$

where R_{AB} denotes the interatomic distance and the α variables is chosen as fitting parameters. In the case where the O—H and N—H bonds are involved, the model uses a different expression:

$$V_{CC}(A,B) = Z'_A Z'_B \langle s_A s_A | s_B s_B\rangle (R_{AB} e^{-\alpha_A R_{AB}} - e^{-\alpha_B R_{AB}} + 1). \tag{4.4}$$

In these cases, the core-core repulsion is additionally influenced by the bond length (R_{AB}) between the two atoms. Further, the MNDO method makes the approximation that the Slater-type orbital exponents for the s and p orbitals are equivalent ($\xi_s = \xi_p$) for some of the lighter elements. The core-core repulsion term replaces the nuclear-nuclear repulsion term in the energy expression. We therefore get

$$E = \sum_{i=1}^{n} \varepsilon_i + \frac{1}{2} \sum_{r=1}^{m} \sum_{s=1}^{m} D_{rs} h_{rs} + V_{\mathrm{CC}}. \tag{4.5}$$

Empirical data reveals that despite the theoretical nature of MNDO's parameterization approach, its overall performance remains largely unaffected. This method, which initially concentrated on ground-state characteristics, especially heats of formation and geometries, also included ionization potentials and dipole moments as auxiliary reference data. The choice to use heats of formation as reference data implied a generalized accounting for zero-point vibrational energies and thermal corrections between 0 and 298 K [17]. The parameterization process [2, 16] involved the selection of a training set of common, small molecules such as methane, benzene, nitrogen, water, and methanol, with a total of 34 molecules incorporated for the carbon, hydrogen, oxygen, and nitrogen set [12, Ch.6]. The overarching goal was to fine-tune six pivotal parameters to optimally represent four critical molecular characteristics—heat of formation, geometry, dipole moment, and ionization energy. Despite the intrinsic complexity and compromises in this process, the MNDO approach has proven to be a surprisingly effective tool for molecular property prediction. The reports of of the calculation on 138 compounds limited to the atoms as in the training set. The Mean Absolute Errors (MAEs) for the 138 compounds in heat of formation was 26kJ mol^{-1} while the value for these were in the range of -600 to 600kJ [16]. Given that the training set exclusively contains stable, ground-state molecules, MNDO struggles with systems where electron correlation significantly deviates from that of the ones in its training set.

4.3 Austin Model 1 (AM1)

Upon gaining insights through practical engagement with MNDO, certain consistent inaccuracies began to surface. One such instance was the overstated repulsion between two atoms when they are relatively close to each other, which in particular encompassed the approximate range of van der Waals distances. This resulted in exaggerated activation energies. This discrepancy was traced back to an excessively repellent interaction within the core-core potential, and even the parameters ξ in the exponent of the STOs differed for s and p atomic orbitals on the same atom. To rectify these issues, the core-core function underwent a modification where both attractive and repulsive Gaussian functions, centered at internuclear points, were introduced [18, Ch.17]. The implementation of these modifications was anything but simple, involving tremendous patience, extensive computer time, and intricate empirical techniques [19]. This led to the subsequent reparameterization of the entire model, in an effort to better align with experimental observations, culminating in the development of the Austin Model 1 (AM1) method [20], developed at the University of Texas at Austin. The core-core repulsion of the AM1 model is:

$$V_{CC}(A,B) = V_{CC}^{\text{MNDO}}(A,B) + \frac{Z'_A Z'_B}{R_{AB}} \sum_k (a_{kA} e^{-bkA(R_{AB}-c_{kA})^2} + a_{kB} e^{-b_{kB}(R_{AB}-c_{kB})^2})$$

where V_{CC}^{MNDO} is the core-core repulsion of the MNDO method from Equation (4.5). The added term represents the off-center attractive and repulsive Gaussian functions that were introduced. The reasoning behind the use of these Guassian functions comes from them being able to model repelling and attracting forces between atoms. These forces tend to balance each other out and give a more accurate model for atomic interactions hence resolving the problem with MNDO about it failing to capture the repulsion between two atoms at close separation distances. The number of Gaussian functions for each atom,

represented by k, usually ranges from 2 to 4. This variation is determined empirically to achieve the best fit with experimental data for each individual atom. It should be noted that the Gaussian functions were added more or less as patches on to the underlying parameters, which explains why a different number of Gaussians is used for each atom.

AM1 typically offers more accurate predictions for the heats of formation than MNDO, with some exceptions involving Br atoms. Depending on the specific system and the information required, either AM1 or PM3 (see bellow) is often the method of choice for achieving the most accurate results with organic molecules using semiempirical techniques. These methods also tend to provide improved estimates for activation energies compared to MNDO [21].

4.4 Parametric Method 3 (PM3)

The Parametric Method 3 (PM3) method, described in [2–4] is build up nearly the exact same way as the AM1 model, using mostly the same functions, except that only two Gaussian functions were allocated for each atom but with an improved set of parameters. The parameterization of the MNDO and AM1 methods were mostly done by hand as the five parameters g_{ss}, g_{pp}, g_{sp}, g_{p2} and h_{sp} for the one-center two-electron integrals were derived from atomic data, while the remaining parameters were adjusted until they satisfactorily aligned with the observed data. Given that this optimization process was manually executed, it limited the inclusion of reference compounds to a relatively small number. J.J.P. Stewart who authored the PM3 method aimed to automatically optimize these parameters by deriving and implementing formulas for the derivative of a suitable error function with respect to the parameters [2, 3]. This new method allowed for all the parameters to be optimized simultaneously, including even the two-electron terms which were previously taken from atomic data. Stewart also used a remarkably larger training dataset, consisting of over 500 compounds. Therefore, the PM3 method can be viewed as an AM1 with all its parameters fully optimized for a larger set of molecules. Essentially, it could be regarded as an optimal set of parameters that, at the very least, represents a favorable local minimum when compared to the experimental data [7].

The PM3 method is as of now especially popular for organic systems. It is usually considered better than AM1 at finding hydrogen bond angles as well as for calculating heats of formation. Hypervalent molecules are also predicted more accurately as well as energies and bond angles, on average. However, the AM1 has shown to be more accurate in predicting for hydrogen bond energies [21]. Holder et al. have found, in their study [12, p. 479] involving compounds that include H, C, N, O, F, Cl, Br, and I, that the PM3 method yielded an average absolute error in heat of formation of 22 kJ/mol for 408 compounds, a marked improvement over AM1's 27 kJ/mol. Furthermore, Dewar et al. noted that the PM3 method produced an average absolute error in bond lengths of 0.022 over 344 bonds, in comparison to AM1's 0.027. For 146 angles, PM3 had an error of 2.8 degrees versus AM1's 2.3 degrees. Lastly, in terms of dipole moments for 196 compounds, PM3 had an error of 0.40 D, a slight increase compared to AM1's 0.35 D [22].

In 2006, J.J.P. Stewart introduced the PM6 [23] which offer numerous of improvments compared to PM3 and AM1. Unlike previous methods, the PM6 was parameterized from *ab-inito*

data as well as experimental. It was also shown to give better heats of formation than the *ab-initio* DFT method B3LYP/6-31G for a test set of around 1000 compounds [12, Ch.6].

4.5 Extension to include *d*-orbitals

As we have seen, the MNDO, AM1 and PM3 metohd only account for s and p-basis and are without d orbitals.That means that the two-center two-electron integrals can be modeled by multipoles up to the 4th order (quadrupoles). This means that these models can not be applied to most of the transition metal compounds and may be inadequacies for hypervalent compounds of main-group elements given that the significant role of d-orbitals in achieving quantitative precision has been well established through *ab initio* calculations [17]. To account for the presence of these d-orbitals in our calculations, it becomes necessary to extend the modeling of two-center two-electron integrals by including multipoles up to order 16. There has been proposed extensions of the MNDO and PM3 method which include d-orbital.

The MNDO/d [24] extended MNDO to include d orbitals for many second-row and later elements. In this extension, the formalism and parameters for hydrogen, helium, and the first-row elements are unchanged but with added 15 parameters per atom for the following elements (Na, Mg, Al, Si, P, S, Cl, Br, I, Zn, Cd and Hg). In this method, multipoles beyond orer 4 are neglected [7] and has 3 new parameters, U_{dd}, ξ_d and β_d. Out of the now 12 new one-center two-electron integrals, only the g_{dd} is set as a freely-varying parameter while the rest are analytically calculated using pseudo-orbital exponents [7] by a truncation of the semiempirical multipole– multipole interactions [25] at the quadrupole level [17]. The MNDO/d has been applied for the standard. molecules as well as to some compounds of metals and hypercoordinate molecules. It was said to give better results over other semiempirical methods, especially for hypervalent compounds [26, p. 722-725].

There is also an extension og PM3 which is parameterized with d-orbitals. However, the parameters were fitted exclusively using geometric data and not heats of formation, dipole moments or ionization energies [27, 28]. This method is adopted primarily because of the limited availability of dependable energy data for transition metal compounds.

4.6 Parameters

We can now construct a table containing the different parameters in the three semiempirical methods MNDO, AM1 and PM3.

As we have discussed,for MNDO and AM1 the one-center two-electron integrals are normally taken from atomic spectra while optimization techniques are used for the PM3 method. The MNDO method has 12 parameters per atom, including one-electron terms ($U_{ss/pp}$), resonance terms ($\beta_{s/p}$), two-electron terms ($\beta_{s/p}$, g_{ss}, g_{sp}, g_{pp}, g_{p2}, h_{sp}), and the core-core repulsion parameter (α). The AM1 and PM3 methods extend upon MNDO by incorporating additional parameters (a_{kA}, b_{kA}, c_{kA}) that represent the Gaussian multipliers and the Gaussian center radius of atom A. Including the d-orbital for the one-center one-electron integral U_{dd} and for the resonance term β_d will yield additional two parameters for the MNDO and PM3 method. Lastly, for the MNDO and AM1 methods, the one-center two-

Parameter	Units	MNDO	AM1	PM3
U_{ss}, U_{pp}	eV	$\star$	$\star$	$\star$
β_s, β_p	eV	$\star$	$\star$	$\star$
ξ_s, ξ_p	bohr^{-1}	$\star$	$\star$	$\star$
α_A	Å^{-1}	$\star$	$\star$	$\star$
g_{ss}, g_{pp}	eV	$\circ$	$\circ$	$\star$
g_{sp}	eV	$\circ$	$\circ$	$\star$
g_{p2}	eV	$\circ$	$\circ$	$\star$
h_{sp}	eV	$\circ$	$\circ$	$\star$
a_{kA} or K_{kA}	-	X	$\star$	$\star$
b_{kA} or L_{kA}	Å^{-2}	X	$\star$	$\star$
c_{kA} or M_{kA}	Å	X	$\star$	$\star$

Table 4.1: *Parameters in the semiempirical methods MNDO, AM1 and PM3. The $\star$ indicates that the parameter was fitted on experimental data while $\circ$ means the parameters have been obtained from experiments. The X indicates that the semiempirical method does not support the corresponding parameter.*

electron integrals were taken from experiments, unlike the PM3, where they were optimized to better fit the training data.

Chapter 5

Optimization

5.1 Optimization

Now we turn our attention to optimization algorithms, vital in the realm of machine learning for minimizing a specific cost or loss function. This function quantifies the disparity between the model's predictions and the true target values, encapsulating the learning task. Optimization algorithms, with their systematic and efficient procedures, play an indispensable role in navigating the vast parameter space, iteratively updating the model parameters to identify configurations that yield optimal performance [29]. By continually adjusting these parameters, the algorithms enable the model to converge to a state where the cost function is minimized, aligning the model's predictions closely with the ground truth [30, p.271-274]. The optimization process leverages both deterministic consistency and controlled randomness, employing intelligent iterative techniques to explore and exploit the search space in pursuit of the best solutions.

Machine learning problems often entail intricate optimization challenges, given the complex, non-convex and high-dimensional nature of the optimization landscape [31, 32]. Analytical or classical methods may prove insufficient or impractical in such scenarios. Optimization algorithms offer a powerful alternative, leveraging iterative techniques that intelligently explore and exploit the search space in search of optimal solutions. These algorithms employ diverse strategies, ranging from gradient-based approaches, which we will dive into in this section, that rely on derivatives of the cost function to gradient-free methods that employ sampling or heuristic strategies [33]. Deterministic algorithms ensure consistent outcomes given the same initial conditions, while stochastic algorithms introduce controlled randomness to avoid local optima and foster exploration.

The choice of an appropriate optimization algorithm is contingent upon various factors, including the properties of the cost function, the constraints of the problem, the characteristics of the available data, and the desired performance metrics. A well-suited optimization algorithm must strike a balance between efficiency, scalability, reliability, and adaptability to different problem domains. Moreover, it should offer ease of implementation and tuning, allowing practitioners to tailor the algorithm to their specific needs [34–36].

Beyond their practical utility, optimization algorithms also hold immense value in advancing

the frontiers of machine learning research. By continuously developing and refining these algorithms, researchers can explore new models, paradigms, and learning techniques that were once deemed intractable or computationally prohibitive. Optimization algorithms serve as a gateway to unlocking the full potential of machine learning, empowering researchers to tackle complex and real-world problems with enhanced precision and efficiency.

5.2 Gradient Descent

We will start of by discussing the gradient descent (GD) algorithm. GD is a widely utilized technique for optimizing the parameter(s) θ in various optimization problems. Its fundamental concept is rooted in the observation that any given function, denoted as $\mathbf{F}(\mathbf{x})$, experiences the steepest decrease when moving from the current point $\mathbf{x}$ in the direction opposite to the negative derivative $-\nabla\mathbf{F}(\mathbf{x})$ [37].

Mathematically, the iterative update scheme of GD is expressed as follows:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta\nabla\mathbf{F}(\mathbf{x}_k)$$

Here, $\eta > 0$ represents the learning rate, which determines the magnitude of movement in the direction of the negative gradient. By selecting a suitably small value for η, the condition $\mathbf{F}(\mathbf{x}_{k+1}) \leq \mathbf{F}(\mathbf{x}_k)$ can be satisfied, indicating that each iteration k brings us closer to a smaller value.

In the context of machine learning, the objective is to minimize the cost function $\mathcal{C}(\theta)$[1], and thus, the iteration scheme for GD can be expressed as follows:

$$\begin{aligned} v_t &= \eta_t \nabla_\theta \mathcal{C}(\theta_t) \\ \theta_{t+1} &= \theta_t - v_t \end{aligned} \tag{5.1}$$

Here, $\nabla_\theta \mathcal{C}(\theta_t)$ represents the gradient of the cost function with respect to θ at the t-th iteration step, and η_t denotes the learning rate [29]. The optimization process begins with an initial guess for θ, denoted as θ_0, and proceeds iteratively towards a minimum. It is crucial to choose the learning rate, η_t, carefully, as excessively small values may result in a large number of iterations before convergence, while relatively large values may lead to overshooting the minimum.

Despite its wide application, the standard GD method does have several drawbacks that should be considered:

- Sensitivity to the initial condition: GD can be sensitive to the choice of the initial parameter values, which may affect the convergence and the quality of the obtained solution.
- Determinism: Once GD converges to a minimum, it remains at that point, whether it is a global or local minimum, without further exploration.

[1]The advantage here is the fact that the first derivative of $\mathcal{C}(\theta)$ wrt. θ is zero at a minimum whether the minimum is local or global.

- Sensitivity to the learning rate: The performance of GD is highly dependent on the learning rate selection. Choosing an inappropriate learning rate may hinder convergence or lead to suboptimal solutions.
- Uniform treatment of all parameter directions: GD treats all directions in the parameter space equally by using the same learning rate per parameter throughout the optimization process, which may not be optimal for every situation. the same learning rate per parameter

To overcome these limitations, we can use various variants of GD that address specific challenges and provide enhanced optimization capabilities. These variations introduce advanced techniques such as momentum, adaptive learning rates, stochasticity, and methods to balance exploration-exploitation trade-offs.

5.3 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) [38] is a variant of the gradient descent optimization method that provides substantial computational advantages, particularly in scenarios with high-dimensional optimization problems or sizable datasets, which are common in machine learning applications.

SGD takes a unique approach to minimizing the cost function as it recognizes that the cost function can be expressed as a summation over all the data points n. In other words,

$$\mathcal{C}(\beta) = \sum_{i=0}^{n} c_i(\mathbf{x}_i, \beta)$$

where $\mathcal{C}$ is the cost function and the vector $\mathbf{x}_i$ represents the ith data point or input vector.

Therefore, the gradient of this cost function is also a sum over the gradients calculated at each of the n data points,

$$\nabla_\beta \mathcal{C}(\beta) = \sum_{i=0}^{n} \nabla_\beta c_i(\mathbf{x}_i, \beta)$$

Rather than evaluating the gradient of the entire data set, SGD cleverly operates on a subset of the data, known as a "*mini-batch*". Each mini-batch contains randomly selected points from the original data set. The size of the batch, denoted as B, is significantly smaller than n, particularly when handling large data sets. If we denote the total number of these batches as n/B, then each mini-batch can be indexed as B_k, where $k = 1, \ldots, n/B$. The SGD approach modifies the gradient calculation to only include the data points within the current mini-batch B_k where $k \in [1, n/M] \subset \mathbb{Z}^+$ is picked at random with equal probability. We then get:

$$\nabla_\beta \mathcal{C}(\beta) = \sum_i^n \nabla_\beta c_i(\mathbf{x}i, \beta)$$
$$\rightarrow \nabla_\beta \mathcal{C}_{B_k}(\beta) = \sum_{i \in B_k} \nabla_\beta c_i(\mathbf{x}_i, \beta)$$

Each complete iteration over the set of mini-batches (i.e., n/B batches) constitutes an "epoch". Multiple epochs are typically carried out to improve the convergence of the optimization.

One of the major advantages of SGD over the standard GD is the inherent stochasticity. This characteristic decreases the chance of the optimization process getting stuck in a local minimum, hence enhancing the search for a global minimum. Moreover, the use of mini-batches significantly reduces the computational burden associated with gradient calculations.

To ensure a fair representation of data points across epochs and to prevent the repeated selection of overlapping data points, a shuffled index array approach can be introduced. This array, with a length equal to the total number of data points (N), contains shuffled indices and is used to randomly select mini-batches in each epoch.

In practice, this process would look like the pseudocode shown below:

```
indices= [0, ..., N-1]
for epoch in epochs:
random_indices = shuffle(indices)
for b_k in B_k:
batch = random_indices[b_k * B: b_k * B + B]
```

The iterative scheme for SGD can then be described mathematically as:

$$v_t = \eta_t \nabla_\theta E_{B_k}(\theta_t),$$
$$\theta_{t+1} = \theta_t - v_t.$$

In this scheme, v_t represents a running average of recent gradients [29]. With its introduction of stochasticity and computational efficiency through the use of mini-batches, SGD often outperforms standard GD, proving to be an effective tool for addressing large-scale or high-dimensional optimization problems.

5.4 Momentum-based Gradient Descent

The addition of momentum in gradient descent is a significant refinement that takes into account the direction of our movement in the parameter space. This strategy, known as Momentum Stochastic Gradient Descent (Momentum SGD) [39], can enhance the efficiency and stability of the optimization process.

The principle behind the momentum term is to allow the gradient descent update at each step to be influenced not only by the current gradient but also by the preceding update direction. This persistence can be particularly beneficial in several scenarios, for instance, when navigating areas of small gradients or oscillating in high-curvature directions.

The mathematical formulation of the Momentum SGD update rule is given by [40, p. 74] [41]:

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta_t \nabla_\theta E_{B_k}(\theta_t) \\ \theta_{t+1} &= \theta_t - v_t \end{aligned} \tag{5.2}$$

In these equations, γ denotes the momentum parameter, which is a hyperparameter typically chosen in the range [0, 1]. The term γv_{t-1} represents the contribution from the previous update direction, while $\eta_t \nabla_\theta E_{B_k}(\theta_t)$ is the gradient descent update corresponding to the current mini-batch B_k. The parameter update θ_{t+1} is then computed as the difference between the current parameter θ_t and the momentum-adjusted velocity v_t.

The benefits of adding a momentum term are manifold. Firstly, by incorporating the 'memory' of previous update directions, Momentum SGD can accelerate convergence, particularly in areas of the cost function with small gradients. This is advantageous because such areas might otherwise slow down the optimization process, due to the small steps taken in each iteration.

Secondly, the momentum term can effectively reduce oscillations in directions of high curvature, thereby enhancing the stability of the optimization process. This can be crucial in complex optimization landscapes, where naive gradient descent might be prone to erratic movements.

Thus, Momentum SGD leverages the advantages of both SGD and momentum to provide a robust, efficient optimization method that is particularly suitable for challenging machine learning tasks.

5.5 ADAM

Navigating the multi-dimensional parameter space to find the minimum of a cost function can be challenging using standard optimization algorithms like Gradient Descent and Stochastic Gradient Descent (SGD). These methods, while efficient under certain conditions, may not point directly to the minima, often leading to suboptimal paths through the cost function landscape. To improve this, we incorporate the concept of momentum—a weighted addition of previous gradients—ensuring a more valley-aligned progression towards the minima instead of moving side-to-side across the valley.

In our efforts to enhance the optimization scheme further, we also use the Root Mean Squared Propagation (RMSProp) [42, Ch. 4] method. RMSProp adjusts the learning rate individually for each parameter and proves effective for non-convex functions, addressing the common hurdle of choosing an appropriate step length in each dimension. This is achieved by substituting η_t in Equation (5.2) with a a diagonal matrix, G.

Building on these techniques, we employ ADAM (ADAptive Moment estimation) [43] as our primary optimizer. ADAM synergizes the principles of momentum SGD and RMSProp [42, Ch. 4], making it a comprehensive and effective optimization method for complex problems. Specifically, it maintains running averages of both the first moment (akin to momentum) and the second moment (similar to RMSProp) of the gradient. ADAM also incorporates bias corrections to account for the initial zero initialization of the moments, thus offering a more balanced and robust optimization approach.

The iteration scheme in ADAM can be presented as follows:

$$
\begin{aligned}
g_t &= \nabla_\theta E_{B_k}(\theta_t) \\
m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\
s_t &= \beta_2 s_{t-1} + (1-\beta_2) g_t^2 \\
\hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\
\hat{s}t &= \frac{s_t}{1-\beta_2^t} \\
\theta t + 1 &= \theta_t - \eta_t \frac{\hat{m}_t}{\sqrt{\hat{s}_t} + \epsilon}
\end{aligned}
$$

Here, $m_t = \mathbb{E}[g_t]$ and $s_t = \mathbb{E}[g_t^2]$ are the running average of the first and second moment of the gradient. The exponential decay rates β_1 and β_2 determine the memory of the first and second moment, typically set to 0.9 and 0.99, respectively. The value of $\epsilon \sim 10^{-8}$ is a small constant added for numerical stability to prevent division by zero errors.

This adaptive adjustment of the learning rate enables ADAM to handle areas with large gradient norms more efficiently and accelerates convergence in flatter regions. The flexibility and effectiveness of ADAM make it our preferred choice for optimizing the expected energy in our studies.

Chapter 6

Machine Learning

Machine learning is a field of study that lies at the intersection of computer science, statistics, and artificial intelligence. Its core aim is to design algorithms that allow computers to learn from and make predictions or decisions based on data and improve their performance over time [44, p. 13-18]. The data utilized in machine learning typically consists of a collection of examples, with each example described using features, and possibly associated with a label.

The field can be broadly categorized into three types: supervised learning, unsupervised learning, and reinforcement learning. In supervised learning, the algorithm is given a set of input-output pairs (called training data) and learns a function that maps the inputs to the outputs. The goal is to generalize the function to unseen inputs (called test data) and make accurate predictions. Supervised learning problems can be further divided into two subtypes: regression and classification. In regression, the output is a continuous value, such as the temperature of a given day. In classification, the output is a discrete label, such as the type of an animal.

In unsupervised learning, the algorithm is given a set of inputs (called unlabeled data) and learns to discover patterns, structures, or features in the data without any guidance. The goal is to find some hidden or latent representation of the data that captures its essence or meaning. Unsupervised learning problems can be further divided into two subtypes: clustering and dimensionality reduction. In clustering, the algorithm groups similar inputs together into clusters, such as customers with similar preferences or documents with similar topics. In dimensionality reduction, the algorithm reduces the number of features or dimensions of the inputs, such as compressing high-resolution images or minimizing high-frequency words.

In reinforcement learning, the algorithm does not receive any predefined data but instead encompasses an agent interacting with an environment, learning from its actions and feedback (in the form of rewards or penalties). The goal is to find an optimal policy that maximizes the expected cumulative reward over time. Reinforcement learning problems can be modeled as Markov decision processes, where the algorithm observes the state of the environment, chooses an action to perform, receives a reward and a new state, and repeats this process until it reaches a terminal state or a time limit. Reinforcement learning can be used to solve complex and dynamic problems, such as playing games, controlling robots, or

optimizing systems.

6.1 Artificial Neural Networks

Artificial Neural Networks (ANN) is a collection of Machine Learning (ML) algorithms which are inspired and modeled after biological neural networks in the brain. Typically, but not necessarily, an ANN consists of three types of layers, each consisting of their own set of operations, in which the training data $\mathbf{X}$ passes through in order to optimize the learner. These layers consists of *nodes* that are connected together by a set of weights. Usually the layers makeup consists of *input-layers*, *hidden-layers* and *output-layers*. Consequently their set of nodes are referred to as input-nodes, hidden-nodes and output-nodes respectively. The main idea behind ANNs is to calibrate the weights which connect the numerous classes of layers. The way this is done is through a process called *backpropagation*. The calibrated weights are then used to optimize our model $\hat{\mathbf{y}}$.

The simplest type of ANN is the Feed Forward Neural Network (FFNN or NN). In an FFNN, the connections between the nodes, do not form a cycle, unlike recurrent neural networks for example. In other words its input-nodes are only connected to the hidden-nodes and its hidden-nodes only to the output nodes. That means there are no direct connections between the input and output nodes.

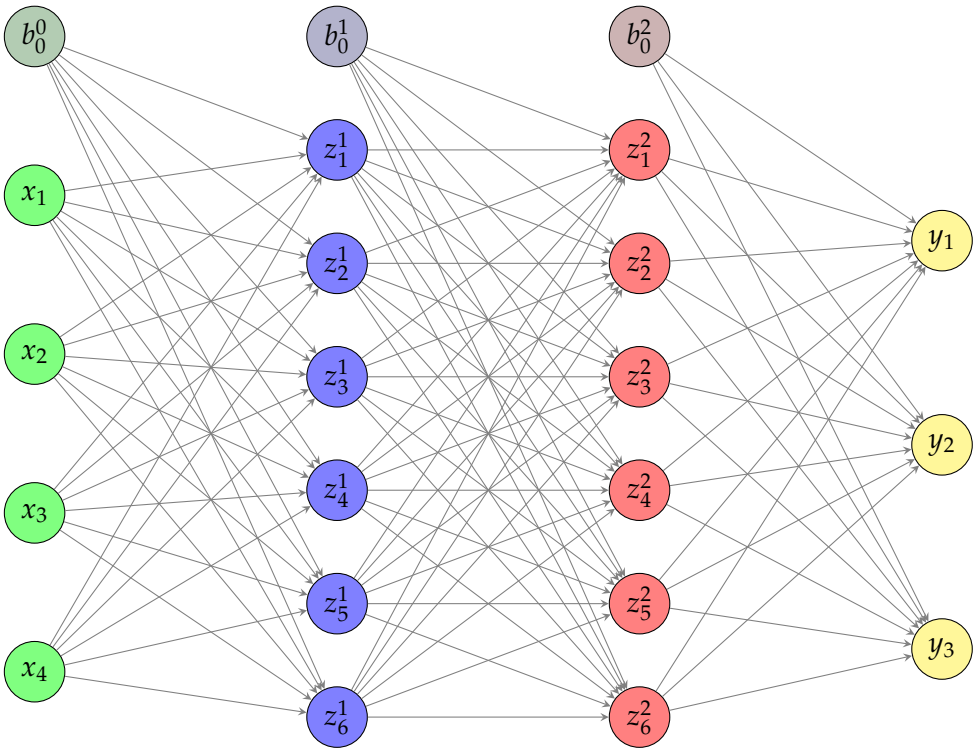

Figure 6.1: *A simple feedforward neural network with an input layer x, two hidden layers z^1 and z^2, an output layer y and bias units distinguished by b.*

The number of nodes in the input-layer is determined by the number of features;

complexity/dimensionality of the input data. The number of nodes in the hidden-layers can be freely chosen and tweaked to achieve optimal results and lastly the number of nodes in the output-layer is determined by the complexity/dimensionality of the targets in our model.

6.2 Forward-propagation

Forward propagation is a reference to the scheme (set of operations) the inputs passes through before arriving to a generated weighted output, that is a fit for our model. The input data is fed to each node in each subsequent layer following the input-layer. For each of these individual nodes in our network, the following computation is performed:

$$z_j^l = \sum_{i=1}^{n_{l-1}} = w_{ij}^l a_i^{l-1} + b_j^l \tag{6.1}$$

$$a_j^l = f(z_j^l) \tag{6.2}$$

where z_j^l corresponds to the input value in layer l for node j. w_{ij}^l corresponds to the matrix-element w_{ij} for layer l in the weights matrix W^l, where w_{ij}^l connects node i in layer $l-1$ to node j in layer l. We also have b_j^l which is the bias-term for node j in layer l. Lastly a_j^{l-1} is the value produced by the *activation function* (defined in section 6.4) defined for z_j^{l-1}.

6.3 Backpropagation

Now we arrive at the backpropagation algorithm which is essential for training an FFNN. In essence, backpropagation is used to update each of the weights and biases of a neural network, in order so that the output $\hat{\mathbf{y}}$ best fits the target $\mathbf{y}$. Hence minimizing the error of the output.

For a regression problem, we might consider the Mean Square Error (MSE) as our cost function

$$\mathcal{C} = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$$

As for classification, consider the Cross Entropy cost function

$$\mathcal{C} = -\sum_{i=1}^{N} \left[y_i \log \hat{y}_i + (1 - y_i) \log \left(1 - \hat{y}_i\right) \right].$$

The theory behind the backpropagation algorithm is to compute the gradient of the cost function wrt. the weights and biases and use the various gradient descent schemes discussed in Chapter 5. This is done via the chain rule for derivation, one layer at a time, iterating backwards from the last layer $l = L$ to the first $l = l_0$. Hence the name backpropagation.

Recall that the gradient of a function is the vector of all its partial derivatives.

$$\nabla \mathcal{C} = \left\{ \frac{\partial \mathcal{C}}{\partial w_{ij}^l}, \frac{\partial \mathcal{C}}{\partial b_i^l} \right\}.$$

For the layer L, using the chain rule, we obtain

$$\frac{\partial \mathcal{C}}{\partial w_{ij}^L} = \frac{\partial \mathcal{C}}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L} \frac{\partial z_j^l}{\partial w_{ij}^L}$$

and we can rewrite this as

$$\frac{\partial \mathcal{C}}{\partial w_{ij}^L} = \frac{\partial \mathcal{C}}{\partial a_j^L} \frac{df}{dz_j^L} a_i^{L-1}. \tag{6.3}$$

As for the derivative with respect to the bias b_j^L, we obtain

$$\frac{\partial \mathcal{C}}{\partial b_j^L} = \frac{\partial \mathcal{C}}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L} \frac{\partial z_j^L}{\partial b_j^L}$$

which again can be rewritten as

$$\frac{\partial \mathcal{C}}{\partial b_j^L} = \frac{\partial \mathcal{C}}{\partial a_j^L} \frac{df}{dz_j^L}.$$

Considering what we've arrived to until now we can define the error at layer l for node j as

$$\delta_j^l = \frac{\partial \mathcal{C}}{\partial z_j^l} = \frac{\partial \mathcal{C}}{\partial a_j^l} \frac{\partial a_j^l}{\partial z_j^l} = \frac{\partial \mathcal{C}}{\partial a_j^l} \frac{df}{da_j^l}. \tag{6.4}$$

Equation (6.3) can then easily be rewritten using δ_j^L

$$\frac{\partial \mathcal{C}}{\partial w_{ij}^L} = \delta_j^L a_j^{L-1}$$

The error δ_j^l gauges the contribution from node j in changing the cost function at layer l. Building up on what we obtained we can generally define the error δ_j^l for j-th node at a layer l in terms of the "prior" layer $l+1$ in the backpropagation scheme.

$$\delta_j^l = \sum_{\mathrm{k}} \frac{\partial \mathcal{C}}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} \tag{6.5}$$

Where

$$\frac{\partial \mathcal{C}}{\partial z_k^{l+1}} = \delta_k^{l+1}$$

Solving for the second derivative term wrt. z_j

$$\frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{jk}^{l+1} \frac{df}{dz_j^l}$$

and putting this back into Equation (6.5), we get

$$\delta_j^l = \sum_{\mathrm{k}} \delta_k^{l+1} w_{jk}^{l+1} \frac{df}{dz_j^l} \tag{6.6}$$

So for the complete backprogation scheme we start with computing δ_j^L and then iterate backwards starting with layer $l = L - 1$. For each iteration over a layer l we compute δ_j^l using Equation (6.6) and update the weights and biases in accordance to the gradient descent scheme in use. The gradient descent in its simplest form would follow

$$w_{ij}^l = w_{ij}^l - \eta \delta_j^l a_i^{l-1}$$
$$b_j^l = b_j^l - \eta \delta_j^l$$

here η is the learning rate.

6.4 Activation functions

Activation functions play a crucial role in neural networks by introducing nonlinearities into the network's computations. These functions are applied to the input value at each node, transforming the node's weighted sum into an output value. The choice of activation function can greatly influence the network's learning behavior, with various functions offering unique strengths and weaknesses. The most common activation functions used are:

- **Sigmoid:** A smooth, s-shaped curve defined by:

$$f(z) = \frac{1}{1 + e^{-z}}$$

- **ReLU (Rectified Linear Unit):** A simple piecewise linear function that allows positive values to pass through unchanged, while mapping negative values to zero:

$$f(z) = \begin{cases} z & \text{if } z \geq 0 \\ 0 & \text{elsewhere} \end{cases}$$

- **Leaky ReLU:** A variation of ReLU that includes a small slope for negative values, controlled by a parameter $\alpha = 0.01$:

$$f(z) = \begin{cases} z & \text{if } z \geq 0 \\ \alpha \cdot z & \text{elsewhere} \end{cases}$$

- **Softmax:** Often used in the output layer for classification tasks, Softmax normalizes the input into a probability distribution over multiple classes:

$$f(z) = \frac{e^z}{\sum_i e^{z_i}}$$

These activation functions enable the network to model complex patterns and provide flexibility in tuning the network's behavior. Their implementation within the network aligns with the computations performed for individual nodes, as detailed in the forward-propagation section.

Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a class of artificial neural networks which explicitly introduce a temporal dimension in their computation model. RNNs are tailored to handle sequential data, where the sequence's order and length are crucial for the underlying task. Making them well adopted for tasks which include to recognize patterns in sequences of data, such as text, genomes, handwriting recognition [45, 46], or speech analysis and text-to-speech synthesis[47–49], as well as time-series analysis, and other sequence-oriented applications. Unlike feed-forward neural networks, which we looked at until now, RNNs have 'cyclic' connections making them powerful for modeling sequences. They take the temporal sequence of the input data into account by having a loop in the network, which acts as a 'memory state' of the network.

RNNs propagate information both in the spatial dimension, from input to hidden layers and then to output, as well as in the temporal dimension, from one sequence element to the next. To accommodate the latter, an RNN includes recurrent connections that feed the current hidden layer's activations back into the same layer at the next time step. This design enables the network to maintain a form of internal state, capturing information about the sequence up to the current time step.

Formally, consider an RNN with a size N input layer, one hidden layer of size H, and K output units. It can be represented as follows: for each timestep t, the activation a_t of the hidden layer and the output y_t is given by:

$$\begin{aligned} a_t^j &= \sigma_h \left(\sum_{k=1}^{H} w_{kj}^l a_{t-1}^k + \sum_{i=1}^{N} w_{ij}^l x_t^i + b_j^l \right) \\ y_t &= \sigma_y \left(\sum_{j=1}^{K} w_{jk}^{l+1} a_t^j + b_k^{l+1} \right) \end{aligned}$$

where $w_{kj}^l \in \mathbb{R}^{H \times H}$ and $w_{ij}^l \in \mathbb{R}^{N \times H}$ are the weight from the ith neuron in layer $l-1$ to the jth neuron in layer l and from the kth neuron in layer l to the jth neuron in layer $l+1$, respectively. The input $x_t^i \in \mathbb{R}^N$ is the value of input i at time t and a_t^j be the activation of the

jth neuron in the hidden layer at time t. Further, σ_h and σ_y are activation functions [1]. Lastly, $b_j^l \in \mathbb{R}^H$ and $b_k^{l+1} \in \mathbb{R}^K$ are the bias terms for the hidden layer and output layer, respectively..

The RNN performs its computations in a recurrent process which can be unfolded in time. One of the main issues of standard RNNs is the so-called vanishing and exploding gradient problem [50], which hampers learning long-range temporal dependencies.

The issue arises due to the nature of the backpropagation through time algorithm used to train these networks. In the backpropagation process, we need to calculate the gradients of the cost function with respect to the weights to update them. But, for long sequences, these gradients tend to become very small (vanish) or very large (explode), which leads to long training times, poor performance, and unpredictable results [51–53].

To address this problem, sophisticated RNN architectures have been introduced such as Long Short-Term Memory (LSTM)[54] networks and Gated Recurrent Unit (GRU)[55] networks.

An LSTM network includes a memory cell that can maintain information in memory for long periods of time. A typical LSTM unit is composed of a cell, an input gate, an output gate, and a forget gate (called LSTM with a forget gate [56]). The cell remembers values over arbitrary time intervals, and the three gates regulate the flow of information into and out of the cell.

LSTM introduces the cell state $c_t \in \mathbb{R}^H$ which can be formulated as:

$$
\begin{aligned}
f_t &= \sigma_h \left(\sum_{k=1}^{H} w_{kj}^l h_{t-1}^k + \sum_{i=1}^{N} w_{ij}^{l-1} x_t^i + b_j^l \right) \\
i_t &= \sigma_h \left(\sum_{k=1}^{H} w_{kj}^l h_{t-1}^k + \sum_{i=1}^{N} w_{ij}^{l-1} x_t^i + b_j^l \right) \\
\tilde{c}_t &= \sigma_c \left(\sum k = 1^H w_{kj}^l h_{t-1}^k + \sum_{i=1}^{N} w_{ij}^{l-1} x_t^i + b_j^l \right) \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \\
o_t &= \sigma_h \left(\sum_{k=1}^{H} w_{kj}^l h_{t-1}^k + \sum_{i=1}^{N} w_{ij}^{l-1} x_t^i + b_j^l \right) \\
h_t &= o_t \odot \sigma_i(c_t)
\end{aligned}
$$

In these equations:

[1] σ_h is usually chosen to be ReLU or the hyperbolic tangent, while the activation function chosen for σ_y depends on the task, but usually sigmoid for binary-classification tasks and softmax for multi-class classification as it outputs a probability distribution over the classes. For regression tasks, we usually do not apply the activation function to keep the output a continuous value.

- $f_t, i_t, o_t \in (0,1)^H$ are the activation vectors of the forget gate, input (or update) gate, and output gate, respectively.
- $h_t, \tilde{c}_t \in (-1,1)^H$ are the hidden state vector (also known as the output vector of the LSTM unit) and the cell input activation vector, respectively.
- $x_t^i \in \mathbb{R}^N$ is the input vector (input i at time t) to the LSTM unit.
- The initial values of the cell state and hidden state are $c_0 = h_0 = 0$.
- $\odot$ denotes the Hadamard product, which is an element-wise multiplication.
- The superscript H refers to the number of hidden units, and N refers to the number of input features.
- The weights $w_{kj}^l \in \mathbb{R}^{H \times H}$ and $w_{ij}^{l-1} \in \mathbb{R}^{N \times H}$ represent the connections between hidden states and between input features and hidden states, respectively.
- The bias $b_j^l \in \mathbb{R}^H$ is part of the real-valued vector space.
- σ_h is usually the sigmoid activation function, while σ_c and σ_i both represent the hyperbolic tangent function[2]. In the case of the 'peephole' LSTM [56, 57], $\sigma_i(z) = z$.

The GRU, on the other hand, regulates the flow of information similar to the LSTM but does so without using a separate memory unit. It uses a simplified gating system, combining the forget and input gates of the LSTM into a single 'update' gate [58]. This results in a lighter model architecture, while still providing effective control over the hidden content.

It is expressed as follows:

$$
\begin{aligned}
z_t &= \sigma_h \left(\sum_{k=1}^{H} w_{kj}^l h_{t-1}^k + \sum_{i=1}^{N} w_{ij}^{l-1} x_t^i + b_j^l \right) \\
r_t &= \sigma_h \left(\sum_{k=1}^{H} w_{kj}^l h_{t-1}^k + \sum_{i=1}^{N} w_{ij}^{l-1} x_t^i + b_j^l \right) \\
\tilde{h}_t &= \sigma_c \left(\sum_{k=1}^{H} w_{kj}^l (r_t \odot h_{t-1})^k + \sum_{i=1}^{N} w_{ij}^{l-1} x_t^i + b_j^l \right) \\
h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t
\end{aligned} \tag{6.7}
$$

where z_t is the update gate which controls how much information the previous hidden state is carried over to the current step, r_t is the reset gate which is used to decide how much of the past information (hidden state) needs to be forgotten, and $\tilde{h}_t$ is the candidate hidden state which contains the new memory to be stored in the hidden state. The final hidden state h_t at time t is then a blend of the previous hidden state h_{t-1} and the candidate hidden state $\tilde{h}_t$, controlled by the update gate z_t. Specifically, each dimension of the hidden state is updated to an extent determined by the corresponding dimension in the update gate.

[2]The use of different activation functions in an LSTM reflects the fact that they perform different roles: the sigmoid function for determining the amount of information flow, and the hyperbolic tangent for creating potential state updates.

The performance of GRU has been observed to be comparable to that of the Long Short-Term Memory (LSTM) model across a broad range of applications and tasks, and some studies have documented instances where the GRU outperforms LSTM, despite its relative computational efficiency [59].

6.5 Graph Representation Learning

Now we will go one step forward and start to look at a fairly new branch of Machine Learning involves predicting information or properties from graph data which is a way of representing data as mathematical graphs (which will be introduced later). This is called Graph Representation Learning. Specifically we will look at Graph Neural Networks (GNN) and Message Passing Neural Networks (MPNNs) which is a specific framework or representation of GNNs.

Since Graph Representation Learning is focused on learning meaningful representation of graph data, it is there Most of this section will be taken from the book "Graph Representation Learning" by William L. Hamilton [60] and "Geometric Deep Learning Grids, Groups, Graphs, Geodesics, and Gaugesas" by Michael M. Bronstein et al. [61] as they give an excellent introduction and cover most of the material we will need on this topic.

6.6 Overview

As mentioned, GNNs are a category of neural networks formulated specifically to work with graph-structured data. Traditional neural networks are usually structured to process grid-like data (e.g., images processed by Convolutional Neural Networks) or sequential data (e.g., text processed by Recurrent Neural Networks). In contrast, GNNs are engineered to operate on the graph structure, considering attributes associated with nodes or edges, or both. This unique capability renders GNNs as an ideal choice for tasks where both the structural connectivity and attribute data of the graph are essential.

The theoretical underpinning of GNNs is rooted in their capacity to execute computations on graphs via defined update and aggregation functions. These functions constitute the central elements of a structure known as a Graph Network Block [62], a fundamental building unit in the architecture of GNNs. A Graph Network Block ingests a graph, carries out computations on its components, and outputs a graph. Specifically, it updates the attributes of nodes, edges, and the global state (if present) in an independent manner. Concurrently, the aggregation functions facilitate the interchange of information among edges, nodes, and the global state. This mechanism, where nodes exchange messages with their connected neighbors, is commonly referred to as message passing in the field.

The sequence of computational steps in a Graph Network Block, as seen in a typical formulation of GNNs, includes:

1. Updating the attributes of the edges.
2. Aggregating the edge updates at each node.
3. Updating the attributes of the nodes.

4. Aggregating the edge updates for the global state, if applicable.
5. Aggregating the node updates for the global state, if applicable.
6. Updating the global state, if applicable.

GNNs aim to create representations of nodes influenced by both the topology of the graph and any existing feature information. This methodology contrasts with shallow embedding techniques that optimize a standalone embedding vector for each node. GNNs extend beyond this, offering a method to construct intricate encoder models that encapsulate the multifaceted structure and features present in graph data.

6.7 Neural Message Passing

Since Graph Representation Learning is focused on learning meaningful representation of graph data, it is to have a basic grasp of graph theory. For readers who are not familiar with graph theory, the book "Discrete Mathematics and Its Applications" by Kenneth H. Rosen [63, Ch. 10] gives a great introduction and covers all the necessary topics.

In this section, we will analyze GNN arcitectures which are *permutation equivariant* functions $\mathbf{F}(\mathbf{X}, \mathbf{A})$, also called a GNN layer, where $\mathbf{X} \in \mathbb{R}^{d \times |\mathcal{V}|}$ is a set of node features of a graph and $\mathbf{A}$ is its corresponding adjacency matrix. The functions $\mathbf{F}$ are constructed by applying shared *permutation invariant* functions $\phi(\mathbf{x}_u, \mathbf{X}_{\mathcal{N}_u})$ over closest neighbours where $\mathcal{N}_u$ is the neighbourhood for the node $u \in \mathcal{V}$. Usually, the local function ϕ is reffered to as *message passing*. The study of GNN layers is an active research area. However, there are three "types" of GNN layers which are commonly used and extensively written about in the literature. These three types of GNN layers govern how the message passing function ϕ ffects and modifies the neighboring features during the transformation process.

In order to keep permutation invariance, the node features $\mathbf{X} \in \mathbb{R}^{d \times |\mathcal{V}|}$ are *aggregated* with a permutation-invariant function $\bigoplus$. This function is a nonparametric operation[3] such as a element-wise sum, mean or maximum[4].

Since GNNs learn a representation for a graph based on its node features where the nodes don't have a natural ordering, permitting the nodes of the graph won't change the graph itself, therefore the function used to aggregate the node features must be permutation invariants. Neural networks can independently process each node in the graph, yielding a set of node embeddings. These embeddings are then combined using permutation-invariant operations to obtain a representation of the entire graph.

[3] In this context, the term "nonparametric operation" refers to an operation that does not rely on any specific parametric form or distributional assumptions of the underlying data. This means that nonparametric operations make fewer assumptions about the data and are not tied to specific parameterized models. These operations are also permutation-invariant because their results do not change if the order of the elements in the input set changes.

[4] For example, we can find the sum of the set $S = \{1, 2, 3\}$ as $\sum_{s \in S} s = 6$, the mean $\mu(S) = 2$ and the maximum $\max S = 3$. These do not change on the permutation of the input.

6.8 GNN layers

As briefely mentioned, GNN layers are usually classified in three categories, sometimes refered to as the differnt GNN flavours. Each of these flavours has its unique way of aggregating information from neighbouring nodes in a graph. Let's delve into each one, keeping in mind that the mathematical symbols used here have been defined in previous sections.

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph with the node set $\mathcal{V}$ and the edge set E. Firstly, we have the convolutional layer [64–66]. In this approach, the features of the neighbouring nodes are directly aggregated with fixed weights. This can be represented as:

$$\mathbf{h}_u = \phi\left(\mathbf{x}_u, \bigoplus_{v \in \mathcal{N}_u} c_{u,v}\psi\left(\mathbf{x}_v\right)\right).$$

Here, $c_{u,v}$, denotes the importance node v has to the representation of neighbouring node u, and $\mathbf{x}_v$ is the features of node $v \in \mathcal{V}$. It's a constant that often directly depends on the entries in $\mathbf{A}$, which represents the structure of the graph. What's interesting here is that when the aggregation operator $\bigoplus$ is chosen to be the sum operator to combine information from different nodes, it's like spreading or 'diffusing' information from one node to its neighbours. This process is a broader version of a concept called convolution, which is a mathematical operation on two functions, used to produce one by 'blending' the two together. In the context of GNNs, it's used to blend information from a node and its neighbours. However, the concept is generalized to work with data structured as a graph, which is a bit more complex than the standard use of convolution.Lastly, $\mathbf{h}_u$ is the output of one or more layers for each node $u \in \mathcal{V}$ in the graph. Moving on from the convolutional layer, we encounter the attentional layer [67–69] which is defined as follows:

$$\mathbf{h}_u = \phi\left(\mathbf{x}_u, \bigoplus_{v \in \mathcal{N}_u} a\left(\mathbf{x}_u, \mathbf{x}_v\right)\psi\left(\mathbf{x}_v\right)\right).$$

We see that $c_{u,v}$ is replaced by the "learnable" self-attention mechanism a, which computes the importance coefficients, or weights, $a\left(\mathbf{x}_u, \mathbf{x}_v\right)$[5]. These coefficients determine the level of "attention" or importance that one node should give to another during the information aggregation process. Specifically, $a\left(\mathbf{x}_u, \mathbf{x}_v\right)$ calculates the importance of node v's information to node u. These coefficients are often softmax-normalised across all neighbours. This ensures that the coefficients are positive and sum to one, which can be interpreted as the probability distribution of attention over all the neighbours. Again, if $\bigoplus$ is the sum operator, the aggregation will be a linear combination of the neighbourhood node features where the weights are feature-dependent. However, unlike in the convolutional flavour, these weights are not fixed but are determined by the learnable attention mechanism. This means that the importance assigned to each neighbour's features can dynamically adjust based on the features of the nodes themselves. In other words, the model can pay more 'attention' to certain neighbours based on their features, hence the term 'attentional' flavour."

[5]It is learnable because it is capable of adjusting these coefficients during the training procedure based on the input features of the nodes. This allows the model to learn which connections between nodes are more important for a given task.

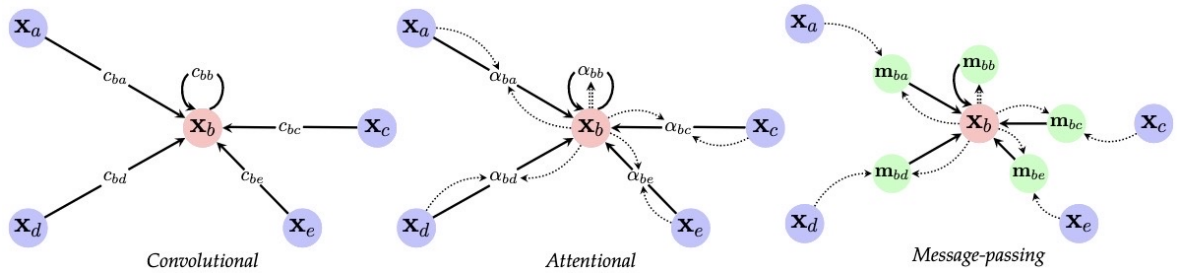

Figure 6.2: *From [61, p. 78]. This visualization depicts the three types of GNN layers.*

Lastly, we have the the message-passing layer [5, 62] which computes arbitrary vectors (also called "messages") across edges:

$$\mathbf{h}_u = \phi\left(\mathbf{x}_u, \bigoplus_{v \in \mathcal{N}_u} \psi\left(\mathbf{x}_u, \mathbf{x}_v, \mathbf{e}_{u,v}\right)\right).$$

In this scenario, ψ is now a learnable differnetiable function, like a neural network, computing v 's vector sent to u, and the aggregation can be considered as a form of message passing on the graph. In particular, we refer to ϕ as the *update* function and ψ as the *message* function.

As briefly mentioned, an attention layer with fixed weights (not learnable) essentially becomes a convolutional layer. In fact, both attention and convolutional GNNs can be considered specific cases of message-passing GNNs as they both involve a node transmitting its own features to its neighbors, but they differ in how these transmitted features are weighted. This is simply shown by setting the message function $\psi\left(\mathbf{x}_u \mathbf{x}_v\right) = c_{u,v}\psi\left(\mathbf{x}_v\right)$ for convolutional GNNs and $\psi\left(\mathbf{x}_u, \mathbf{x}_v\right) = a\left(\mathbf{x}_u, \mathbf{x}_v\right)\psi\left(\mathbf{x}_v\right)$ for attentional GNNs.

While message-passing GNNs offer flexibility, they can be challenging to train and memory-intensive due to the need for computing vector-valued messages across edges. For many naturally-occurring graphs, where edges signify class similarity[6] (like the case when u and v are likely to share the same output for an egde (u, v)), convolutional aggregation often outperforms in terms of regularisation and scalability. Attentional GNNs strike a balance, enabling complex neighbourhood interactions modelling while only computing scalar-valued quantities across edges, enhancing scalability compared to message-passing GNNs.

6.9 Iterative Process of GNN Layers

The iterative process in GNNs, often referred to as the message-passing process, begins with initializing node vector representations as node attributes: $\mathbf{h}_v^{(0)} \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$. Then, at each message-passing iteration k, a hidden representation $\mathbf{h}_u^{(k)}$ corresponding to each node $u \in \mathcal{V}$ is updated. This update is based on the information aggregated from u 's graph neighbourhood $\mathcal{N}_u$. The iterative update can be expressed as follows:

[6] Such graphs are often called homophilous

$$\mathbf{h}_u^{(k+1)} = \phi^{(k)}\left(\mathbf{h}_u^{(k)}, \bigoplus_{v \in \mathcal{N}_u} \psi^{(k)}\left(\mathbf{h}_v^{(k)}\right)\right) \tag{6.8}$$

$$= \phi^{(k)}\left(\mathbf{h}_u^{(k)}, \mathbf{m}_{\mathcal{N}_u}^{(k)}\right) \tag{6.9}$$

where $\phi^{(k)}$ and $\psi^{(k)}$ are arbitrary differentiable functions (i.e., neural networks) of layer k, and $\mathbf{m}_{\mathcal{N}(u)}^{(k)}$ is the "message" that is aggregated from u 's graph neighbourhood $\mathcal{N}_u$ where each individual message from a neighbour v to u is computed by the function $\psi^{(k)}\left(\mathbf{h}_v^{(k)}\right)$. We use superscripts to distinguish the embeddings and functions at different iterations of the process.

The message-passing process involves three steps: message computation, aggregation, and update. For each pair of nodes $(u, v) \in \mathcal{E}$, a message $\mathbf{m}_{vu}^{(k)}$ is computed using the current embeddings of the nodes. Then, for each node $v \in \mathcal{V}$, all the incoming messages from its neighbours are aggregated to form $\mathbf{a}_v^{(k)}$. Finally, the aggregated message is used to update the current embedding of the node. The procedure looks like this [70, p. 71]:

$$\begin{aligned}
\text{Message: } \mathbf{m}_{vu}^{(k)} &\leftarrow \psi^{(k)}\left(\mathbf{h}_v^{(k-1)}, \mathbf{h}_u^{(k-1)}\right), \forall (u, v) \in \mathcal{E}, \\
\text{Aggregation: } \mathbf{a}_v^{(k)} &\leftarrow \bigoplus_{u \in \mathcal{N}v} \left(\left\{\mathbf{m}_{vu}^{(k)} \mid u \in \mathcal{N}_v\right\}\right), \forall v \in \mathcal{V}, \\
\text{Update: } \mathbf{h}_v^{(k)} &\leftarrow \phi^{(k)}\left(\mathbf{h}_v^{(k-1)}, \mathbf{a}_v^{(k)}\right), \forall v \in \mathcal{V}.
\end{aligned} \tag{6.10}$$

This iterative process, allows GNNs to propagate and process information across the graph over multiple steps or 'layers'. It is what enables GNNs to capture the complex, hierarchical patterns in the graph data, which is key to their powerful performance on a wide range of tasks. Since we want to set ψ to be an NN or affine transformation, e.g., $\psi^{(k)}(\mathbf{x}_1, \mathbf{x}_2) = \sigma(\mathbf{x}_1\mathbf{W}_1 + \mathbf{x}_2\mathbf{W}_2)$ where $\mathbf{x}_1, \mathbf{x}_2$ are node features, $\mathbf{W}_1, \mathbf{W}_2$ are learnable weights and $\sigma(\cdot)$ is an activation function (6.4).

After the GNN has produced the representation, $\mathbf{h}_v^{(K)}$, where K is the total number of message-passing layers, we can make a prediction on for example, classify a set of nodes $S \subseteq \mathcal{V}$. This is done by using a readout function. A readout function is another permutation-invariant operator plus an NN. We then get

$$\text{Readout: } \mathbf{r}_S \leftarrow \sigma(\mathrm{R}(\mathbf{h}_v^{(K)}, v \in S \subseteq \mathcal{V})).$$

In essence, the readout function is responsible for aggregating the features of all the nodes in the graph to form a graph-level representation. It provides a summarized and fixed-size output that encapsulates the essential information about the entire graph.

Part III

Methods

Chapter 7

QM9 dataset

7.1 General description

Since neural networks learn from data, it is natural to assume that for a neural network to be trained to predict molecular properties, it would need to learn from a dataset consisting of molecules as features and properties as targets.

Luckly, the chemical universe GDB-17[71] contains data about 166 billion organic molecules, in SMILES (Simplified Molecular Input Line Entry System) format. But even with such an immense chemical space, the challenge remains to provide high-quality quantum chemical property calculations for a substantial and relevant subset.

The QM9 dataset [1] aims to address this, being a part of the quantum machine project [72] along with the QM7 [73, 74], QM7b [73, 75] and QM8 [76] datasets. It is a carefully curated set of around 134k molecules taken from GDB-17 chemical universe. These molecules, each containing up to nine heavy atoms, serve as a relevant, consistent, and comprehensive exploration of small organic molecule space.

For each molecule in the QM9 dataset, a wide range of quantum chemical properties have been reported. These include the the equilibrium geometric configurations of each molecule (xyz-coordinates of each atom of the molecule in the euclidean space), as well as their corresponding harmonic frequencies, dipole moments, and polarizabilities. Moreover, energetic properties such as the ground-state energy, total energy, enthalpies, and free energies of atomization are also provided for each of the molecules in the dataset[1, 77] . All of these properties have been calculated using the B3LYP/6-31G(2df,p) [78, 79] algorithm which is a method based Density Functional Theory (DFT) [80].

Although the QM7, QM7b, and QM8 datasets share similarities with QM9, they are inherently smaller in size and scope. QM7 and QM7b consist of only 7,165 molecules each with up to seven atoms, while QM8 contains 22k molecules, each featuring up to eight atoms. In contrast, the QM9 dataset, encompassing 134k molecules each with up to nine heavy atoms, provides a more extensive sampling of the chemical space. Consequently, a machine learning model trained on the QM9 dataset can traverse a broader landscape, enhancing its capability to locate more optimal local minima and potentially improving its prediction

No.	Property	Unit	Description
1	tag	—	'gdb9' string to facilitate extraction
2	i	—	Consecutive, 1-based integer identifier
3	A	GHz	Rotational constant
4	B	GHz	Rotational constant
5	C	GHz	Rotational constant
6	μ	D	Dipole moment
7	α	a_0^3	Isotropic polarizability
8	ϵ_{HOMO}	Ha	Energy of HOMO
9	ϵ_{LUMO}	Ha	Energy of LUMO
10	ϵ_{gap}	Ha	Gap ($\epsilon_{LUMO} - \epsilon_{HOMO}$)
11	$\langle R^2 \rangle$	a_0^2	Electronic spatial extent
12	zpve	Ha	Zero point vibrational energy
13	U_0	Ha	Internal energy at 0 K
14	U	Ha	Internal energy at 298.15 K
15	H	Ha	Enthalpy at 298.15 K
16	G	Ha	Free energy at 298.15 K
17	C_v	$\frac{cal}{molK}$	Heat capacity at 298.15 K

Table 7.1: *From the original QM9 paper, shows the calculated properties for each of the molecules. Properties are stored in the order given by the first column. In column 1, 'gdb9' idicates that the molecule is taken from a subset of the GDB-17 chemical universe with up to nine heavy atoms. Energies are given in Hartrees.*

performance.

7.1.1 File format

Given that the SMILES strings of QM9 molecules originate from the GDB-17 chemical universe, it became known that translating from string-based chemical representations like SMILES or InChI to Cartesian coordinates is not a straightforward task and can be prone to implementation-specific artifacts. This non-bijective mapping is not invariant to small geometric perturbations in the molecular structure, such as slight variations in bond angles, dihedral angles, and interatomic distances, which can potentially lead to different string representations for molecules with the same topology. The sensitivity of the transformation process to these nuances results in potential variations in the resulting molecular geometries. Furthermore, the reverse mapping is also complex, as there can be multiple valid 3D structures that correspond to the same SMILES or InChI string, due to the rotational freedom around single bonds and other flexible parts of the molecule.

Because of these transformation issues, 3,054 out of the 133,885 molecules in the QM9 dataset were found to have discrepancies between their SMILES (or InChI) representations and the Cartesian coordinates [1]. These molecules are often excluded training sets using the QM9

data [81], which results in 130,831 molecules.

```
5
gdb 1 157.7118 157.70997 157.70699 0. 13.21 -0.3877 0.1171 0.5048 35.3641
0.044749 -40.47893 -40.476062 -40.475117 -40.498597 6.469
C -0.0126981359  1.0858041578  0.0080009958 -0.535689
H  0.002150416 -0.0060313176  0.0019761204  0.133921
H  1.0117308433  1.4637511618  0.0002765748  0.133922
H -0.540815069  1.4475266138 -0.8766437152  0.133923
H -0.5238136345  1.4379326443  0.9063972942  0.133923
1341.307 1341.3284 1341.365 1562.6731 1562.7453 3038.3205 3151.6034 3151.6788
3151.7078
C C
InChI=1S/CH4/h1H4 InChI=1S/CH4/h1H4
```

Figure 7.1: *Example of a CH4 ((methane)) molecule in the QM9 dataset*

where the formatting is is described in Table 7.2

Line Content	Description
1	Number of atoms n_a
2	Scalar properties (see Table 7.1)
$3, \cdots, n_a + 2$	Element type, coordinate (x, y, z, in Å), Mulliken partial charges (in e) on atoms
$n_a + 3$	Harmonic vibrational frequencies ($3n_a - 5$ or $3n_a - 6$, in cm^{-1})
$n_a + 4$	SMILES strings from GDB-17 and from B3LYP relaxation
$n_a + 5$	InChI strings for Corina and B3LYP geometries

Table 7.2: *From the original QM9 paper. The molecules in the dataset are stored in a XYZ-like file format for molecular structure and properties, however, unlike traditional XYZ-format, it contains information on molecular properties and string representation.*

That means that Internal energy at 0 K, $\mathcal{U}_0$, for the CH4 molecule above is $\mathcal{U}_0 = -40.45893$ which is the 13th (or 4th in 3rd line) number in line number 2. The next 5 lines give us the positions of the individual atoms in the molecule as well as their partial charges, which is given by the last number in these 5 lines. However, for our study, only the $\mathcal{U}_0$, atom types and their position in space is relevant.

7.1.2 Graph Representation of the data

As the QM9 data is is in ".xyz" format and is displaying geometric coordinates as well as atom types, as shown in Figure 7.1, we have to do some modifications to effectively map the data into a graph representation of itself. To effectively achieve this, we want to capture both properties of the atoms as well as the bonds they participate in.

Let a molecule be represented by a set of atoms A and a set of bonds B. For each atom $a \in A$, several features are captured and transformed into a representation suitable for

GNNs. These attributes include:

- Atom Type: Denoted by $x_1(a)$, this function maps the chemical symbol (H, C, N, O or F) of atom a to an integer index and subsequently one-hot encodes[1] it into a vector.
- Atomic Number: Represented by a scalar $z(a)$.
- Aromaticity: A binary value $\text{arom}(a)$ indicating whether atom a is aromatic.
- Hybridization States: Denoted by three binary values $\text{sp}(a)$, $\text{sp2}(a)$, and $\text{sp3}(a)$, representing the hybridization states.
- Number of Bonded Hydrogens: Calculated as $h(a)$, this value counts hydrogen atoms bonded to non-hydrogen atom a.
- 3D Coordinates: The geometric coordinates are represented as a vector $p(a)$.

The final feature vector for the atoms, $x(a)$, is formed by concatenating $x_1(a)$ and other scalar functions.

Edges in the graph represent the bonds between atoms. For each bond $b \in B$, the following features are defined:

- Bond Type: Represented by a vector $E_t(b)$, where bonds are mapped to integer indices reflecting single, double, triple, and aromatic types, and subsequently one-hot encoded into $E_a(b)$.
- Adjacency Matrix: We define an adjacency matrix A to represent the connections between atoms, where $A_{ij} = 1$ if there is a bond between atom i and atom j, and $A_{ij} = 0$ otherwise.

7.2 Automatic differentiation

Automatic differentiation (autodiff or AD) [82, 83] is a set of techniques that enable a computer program to compute the partial derivatives of a function with high precision and efficiency which are fundamental to the optimization procedures in machine learning.

This is achieved by leveraging the fact that every computer program, regardless of its complexity, executes a sequence of elementary arithmetic operations (addition, multiplication, etc) and functions, such as $\sin(x)$, $\exp(x)$ and $\log(x)$. By repeatedly applying the chain rule of calculus to these operations, AD can compute derivatives of arbitrary order automatically, with a high degree of accuracy, limited only by the computer precision, and a computational complexity that is only a small constant factor greater than that of the original program.

Unlike symbolic differentiation, which struggles with the conversion of a computer program into a single mathematical expression and can lead to inefficient code as it can grow exponentially in computational complexity, or numerical differentiation, like finite-differences, which can introduce round-off errors and cancellation as it suffers from numerical instability, AD provides a robust and efficient method for derivative computation. This is particu-

[1] One-hot encoding is a process where an integer index is represented by an array in which all elements are zero except for the index position, which is set to one.

larly crucial when calculating partial derivatives of a function with respect to many inputs, as is often required in gradient-based optimization algorithms.

The fundamental principle underpinning autodiff is the chain rule of calculus. Given a composite function $f : \mathbb{R}^2 \to \mathbb{R}$ where $f(g(x,y), h(x,y))$ for $g, h : \mathbb{R}^2 \to \mathbb{R}$, the partial derivative of f with respect to x is given by:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial x} + \frac{\partial f}{\partial h} \cdot \frac{\partial h}{\partial x}$$

With $x, y \in \mathbb{R}$. This means that the rate at which f changes with respect to x is a combination of how much f changes with g and h, and how much g and h change with x.

AD applies this principle iteratively to compute the partial derivatives of functions composed of many nested functions, which is a common scenario in machine learning models. This is particularly important in the context of neural networks, where we often need to compute the gradient of a loss function with respect to high-dimensional parameters.

There are two types of AD. These are forward [84] and backward mode [85] (or accumulation). In forward mode, the chain rule is applied following the natural order of computation, propagating derivatives from the input variable to the output variable. In contrast, the reverse mode applies the chain rule in the reverse order of computation, starting from the output and and propagating back to the inputs. This procedure involves two 'passes' through the function: a forward pass to compute the value of the function, and a backward pass to compute the partial derivatives.

The reverse mode is particularly of great interest in machine learning, where it has recently been applied in the backpropagation algorithm to compute the gradients of the loss function with respect to the weights and biases of the NN. By utilizing the reverse mode of AD, it allows for faster and more efficient training of complex models. The ability to accurately calculate the partial derivatives of functions is crucial in backpropagation, as the derivative of a function can be determined by finding the derivative of every operation in the function and using the chain rule to calculate the derivative. The backpropagation algorithm leverages this by reversing all the operations, allowing for efficient computation of the required gradients.

7.3 Continuous Kernel-Based Convolutional Operator

We develop further from the work by Gilmer et al., 2017 in their paper "Neural Message Passing for Quantum Chemistry" [5] who used the a MPNN trained on the QM9 dataset to predict a wide range of molecular properties. They used the continuous kernel-based convolutional operator, which is also the edge-conditioned convolution from the paper Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs [86].

In the field of graph convolutional neural networks, the continuous kernel-based convolutional operator, is a convolutional operator which addresses the intricate challenging task of incorporating edge attributes effectively into the graph convolutional process, providing the ability to handle both directed and undirected graphs, thus enabling more complex

modeling.

Consider a directed or undirected graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a finite set of vertices with $|\mathcal{V}| = n$[2] and $\mathcal{E}$ is a subset of $\mathcal{V} \times \mathcal{V}$ comprising of the edges with $|\mathcal{E}| = m$. The graph is presumed to be labelled on both vertices and edges. Let $\mathbf{h}_v^{(0)} : \mathcal{V} \mapsto \mathbb{R}^{d_0}$ signify a function that assigns labels (features) to each vertex where $d_0 = \dim(V)$ is the dimension of the vector features V and $K : \mathcal{E} \mapsto \mathbb{R}^s$ allocate labels (attributes) to each edge, where $s = \dim(E)$ is the dimensionality of edge attributes, E, in the graph. Further, let $\mathcal{N}_u = v; (v, u) \in \mathcal{E} \cup u$[3] designate the neighbourhood of vertex u, including u itself, making a self-loop.

The Continuous Kernel-Based Convolutional Operator is then defined as:

$$\mathbf{h}_u^{(k+1)} = \mathbf{W}\mathbf{h}_u^{(k)} + \bigoplus_{v \in \mathcal{N}_u} \mathbf{h}_v^{(k)} \cdot \psi^{(k)}(\mathbf{e}_{u,v}) \tag{7.1}$$

where:

- $\mathbf{h}_u^{(k+1)}$ is the updated feature of the node u in the $(k+1)$-th layer.
- $\mathbf{h}_u^{(k)}$ is the feature vector of node u in the k-th layer.
- $\mathbf{W}$ is a learnable matrix that transforms the node features.
- $\mathbf{h}_v^{(k)}$ is the feature vector of a neighboring node v in the k-th layer.
- $\mathcal{N}_u$ is the set of neighboring nodes to node u.
- $\mathbf{e}_{u,v}$ is the edge feature between node u and node v.
- $\psi^{(k)}$ is a neural network (in practice, often a Multi-Layer Perceptron or MLP) that maps the edge features to an output matrix that's used in the convolution.

In the Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs paper, the authors used the $\bigoplus = \sum$ as their permutation invariant aggregation operator.

The core of this operator is the edge-conditioned component $\mathbf{h}v^{(k)} \cdot \psi^{(k)}(\mathbf{e}u, v)$, where the edge attributes directly influence the transformation applied to the neighbouring node features. The function $\psi^{(k)}$ generates a unique transformation matrix for every edge, thus encapsulating the edge attributes into the learning process.

[2] In graph theory, $|\cdot|$ is a way to denote the cardinality (or size) of a set/graph. See appendix

[3] This notation includes all vertices v such that there exists an edge from v to u in the set of edges $\mathcal{E}$. This is essentially the set of all vertices that are connected to u by an edge. The $\cup u$ part means that the vertex u itself is included in its set of neighbors. This is common in self-loop scenarios where each node is considered to be connected to itself.

Chapter 8

The Model

In this chapter, we will delve into the core model of this thesis, the architecture that integrates the MPNN with the PM3 method. Here's how the model functions:

- MPNN Component: The MPNN model serves as a learner capable of predicting the parameters for the PM3 method. Unlike traditional approaches that rely on fixed tabular parameters, this model dynamically adapts the parameters to better suit the molecule. The iterative prediction process enables a more tailored response to each specific molecule.
- Loss Function: The accuracy of the model's predictions is evaluated using a loss function that calculates the difference between the total energy derived from the QM9 and the predictions made by the PM3 method with MPNN predicted parameters.
- Hybrid Approach: Unlike other machine learning trained parameter sets, this model presents a hybrid approach. The trained MPNN network is not a separate entity but an integrated part of the computation itself. This integration offers a bridge between the computational rigor of the PM3 method and the adaptive learning capabilities of the MPNN, creating a dynamic system with potentially enhanced performance.

All the code used in this project can be found at my GitHub [87] or simply by following this link: https://github.com/aleksda/Thesis.

8.1 Implementation

Having established the theoretical framework and methodological approach for our project, we now turn our focus towards the computational implementation. In this realm, we have chosen Python as our programming language due to its robust capacity for handling scientific and mathematical programming tasks pertinent to our research. Python's primary strengths lie in its comprehensive suite of libraries and tools tailor-made for scientific programming and data analysis, specifically within the domain of computational chemistry and deep learning. Python's high-level, clear, and readable syntax further facilitates the communication and replication of complex scientific concepts and research methodologies, making it an optimal choice for our study.

For deep learning, Python includes powerful libraries such as PyTorch [88], Tensorflow [89], and JAX [90]. For this research, we employed PyTorch for general machine learning while PyTorch Geometric (PyG) [91] was utilized for handling GNNs. PyG provides a wide range of pre-implemented GNN operators and facilitates the handling of irregular structured data. Specifically for our project, we employ the QM9 dataset, which is readily available in PyG already in its graph representation format. Furthermore, the 3k molecules with discrepancies between their string representations and the Cartesian coordinates are already filtered out. PyTorch also features Automatic Differentiation, as well as GPU acceleration. This feature becomes especially relevant when dealing with large datasets and computationally intensive models, like in this work, as it can potentially enhance computational speed and efficiency.

When compared to other potential languages and libraries for this project, such as MATLAB, Julia [92], or C/C++, Python's advantages are noteworthy. MATLAB, although a powerful tool for numerical computation, lacks thespecialized libraries for deep learning, GNNs and computational chemistry. While it does offer GPU computing capabilities, they are not as user-friendly or deeply integrated as Python's. Julia, while an emerging contender in the field of scientific computing with strong support for automatic differentiation, currently lacks robust, mature libraries for GNNs, and its ecosystem for GPU-accelerated computing is not as comprehensive as Python's. As for C/C++, despite their high computational performance, they also lack specialized libraries for GNNs, and their syntax does not provide the readability and ease of use required for expressing complex models such as GNNs. Furthermore, none of these alternatives offer as straightforward a path to GPU acceleration as Python with PyTorch.Lastly, PyTorch's automatic differentiation is a considerable advantage over these alternatives, streamlining the computation of gradients for backpropagation in neural networks, a process that is typically more manual and error-prone in MATLAB, Julia, or C/C++.

8.2 Theoretical description

The reasons to train a GNN on the QM9 dataset (or other data with a graph representation) over a traditional NN, are mainly due to these factors:

1. **Handling Variable Input Length:**
   GNNs can manage variable input lengths, a necessary feature for dealing with the QM9 dataset where the size of the data (number of atoms and bonds) varies. Traditional NNs require fixed-size inputs, which is not ideal for such a dataset.

2. **Maintaining Structural Information:**
   GNNs are capable of processing graph-structured data, preserving the essential structural relationships between atoms in a molecule. In contrast, traditional NNs, due to their need for vector or matrix inputs, can lose some of this important structural information.

3. **Invariance to Input Order and Graph Isomorphism:**
   GNNs have the inherent property of being invariant to the order of input nodes and graph isomorphism. This means that the output remains consistent even when the arrangement of atoms is altered but the bonding structure is kept the same. Traditional NNs do not have this property, making GNNs more suitable for molecular data.

4. **Efficiency with Large Graph Data:**
   GNNs are designed to handle large and complex graph-structured data efficiently. They scale well with the size of the input graph, primarily due to the shared node and edge update functions. This efficiency is vital when processing large molecular datasets like QM9.

Following Gilmer et al., 2017 , we will construct a GNN with the Continuous Kernel-Based Convolutional Operator as our message-passing layer.

Following their steps , we will use a neural network as the message function, ψ, and gated recurrent unit as update function, ϕ. We also chose sum as our permutation invariant operator.

The procedure from Equation (6.10) then becomes:

$$\begin{aligned}
\text{Message: } \mathbf{m}_{vu}^{(k)} &\leftarrow \sigma\left(\text{NN}\left(\mathbf{h}_v^{(k-1)}, \mathbf{h}_u^{(k-1)}\right)\right), \forall (u,v) \in \mathcal{E}, \\
\text{Aggregation: } \mathbf{a}_v^{(k)} &\leftarrow \sum_{u \in \mathcal{N}_v} \left(\left\{\mathbf{m}_{vu}^{(k)} \mid u \in \mathcal{N}_v\right\}\right), \forall v \in \mathcal{V}, \\
\text{Update: } \mathbf{h}_v^{(k)} &\leftarrow \text{GRU}\left(\mathbf{h}_v^{(k-1)}, \mathbf{a}_v^{(k)}\right).
\end{aligned}$$

where $\sigma(\cdot) = \text{ReLU}(\cdot)$ is the activation function. The choice of using the Gated Recurrent Unit as the update function comes from its inherent capability to handle sequential data and their ability to model complex dependencies. since the nodes are updated iteratively, the state of a node at the kth iterations depends on its state at the $(k-1)$th iteration (as we see above) and the information aggregated from its neighborhood. This makes this a sequential task where the sequence is the iterations of update.

As an RNN, the GRU helps control the flow of information between consecutive iterations.This allows the model to capture both short-term and long-term dependencies in the data by choosing what information to keep and what to discard at each step. This is helpful when dealing with large graphs or when the objective is to aggregate information from nodes that are not directly connected but can be reached by traversing through multiple intermediate nodes.

In the context of molecular analysis on the QM9 dataset, the incorporation of a GRU in a GNN framework proves to be especially beneficial. It facilitates the inclusion of information from individual atoms within a molecule that are located multiple bonds away from a reference atom, thereby providing a global perspective of the molecule instead of a merely local one. For instance, consider the Methane (CH4) molecule from Figure 7.1. The Carbon atom has four Hydrogen atoms in its immediate local environment, or one bond away. However, for more complex molecules like Ethane (C2H6), there are Hydrogen atoms two bonds away from each Carbon atom, reachable by traversing through an adjacent Carbon atom. Such an arrangement exemplifies how interactions that span multiple bonds (two, three, or more) can impact the properties of the atom under consideration, and underscores the utility of the GRU in capturing these extended influences for a more accurate molecular modeling.

Continuing with the GNN implementation, we will employ a readout function after the a

readout function node and edge update functions. This readout phase is represented by:

$$\text{Readout: } \mathbf{r}_S \leftarrow \sigma(\mathrm{R}(\mathbf{h}_v^{(K)}, v \in S \subseteq \mathcal{V})).$$

where K is the total number of message-passing layers. Distinct from the approach of Gilmer et al., the readout in our model doesn't directly serve as the GNN's output. Instead, it provides 'corrections' to the PM3 parameters of the effective Hamiltonian. This is done by taking the already previously fitted PM3 parameters from MOPAC [2, 4, 93] and adding them with the corrections from the GNN entry-wise which will then yield the updated PM3 parameters for the effective Hamiltonian. This is then repeated until the model converges or we reach satisfactory results. Therefore, ultimately, the GNN is trained to predict corrections to the PM3 parameters.

Introducing the corrections directly into the PM3 method without adjusting the fixed PM3 parameters can disrupt the SCF loop, which solves the Hartree-Fock equations. The SCF loop iteratively refines the electronic wave function until reaching a self-consistent solution. Abruptly altering key parameters—such as those defining the effective Hamiltonian—drastically changes the set of equations being solved and reshapes the potential energy surface of the molecule. This may hinder the SCF procedure's ability to locate a stable minimum, leading to potential convergence failure or a sharp increase in time complexity as more iterations will be needed to reach convergence. In contrast, gradually 'tuning' the fixed PM3 parameters using corrections derived from the GNN enables the model to adapt incrementally to the patterns and correlations within the data. This gentle, iterative refinement is less disruptive to the SCF loop's convergence, while it enhances the PM3 method's predictive capability.

This all means that the output, $\hat{y}$, of the network is what gets predicted by the PM3, not after the readout phase. That means

$$\hat{y} = E^{\text{tot}} = E^{\text{elec}} + \sum_{i<j} E_{i,j}^{\text{nuc}} \tag{8.1}$$

where E^{elec} is the sum of electronic energy and $E_{i,j}^{\text{nuc}}$ is the nuclear interaction between the ith and jth atom calculated by the MPNN-PM3 model. The energy is from Equation (4.5) can be written on the form

$$E^{\text{elec}} = \frac{1}{2}\,\mathrm{Tr}[\mathbf{D} \times (h_0 + \mathbf{F}(\mathbf{D}))]$$

where $\mathbf{F}(\mathbf{D}) = h_0 + \mathbf{C}(\mathbf{D})$ is the Fock matrix, $\mathbf{D}$ is the single-particle density matrix, $\mathbf{C}$ is the Coulomb matrix and h_0 is the core Hamiltonian.

This means that our MPNN is directly trained to generate accurate predictions of the core Hamiltonian h_0, the Coulomb matrix $\mathbf{C}$ and the nuclear energy E^{elec}. That gives us

$$\mathbf{r}_S : \mathcal{V} \to \{\hat{h}_0, \widehat{\mathbf{C}}, \widehat{E}^{\text{elec}}\}$$

where again, the readout, $\mathbf{r}_S$ is the final transformation we receive from our MPNN after the message, aggregation and update steps. The quantities $\hat{h}_0$, $\widehat{\mathbf{C}}$ and $\widehat{E}^{\text{elec}}$ are the MPNN's prediction of the core Hamiltonian,

Coulomb matrix and the nuclear energy given by the input data $\mathcal{V}$.
The predicted values are then used to make corrections to the existing PM3 parameters as used in the MOPAC software package. Hence, the updated parameters for the effective Hamiltonian and the Coulomb matrix are calculated as follows:

$$h_0' = h_0 + \Delta\hat{h}_0,$$
$$\mathbf{C}' = \mathbf{C} + \Delta\widehat{\mathbf{C}}$$

and

$$E^{\text{elec}} = \widehat{E}^{\text{elec}} + \Delta\widehat{E}^{\text{elec}}$$

where $\Delta\hat{h}_0$, $\Delta\widehat{\mathbf{C}}$ and $\Delta\widehat{E}^{\text{elec}}$, are the corrections predicted by the MPNN while h_0 and $\mathbf{C}$ are the core Hamiltonian and Coulomb matrix of the PM3 method with the standard MOPAC parameters.

This procedure is repeated iteratively, with the MPNN adjusting the corrections it provides based on the updated output. As the MPNN continues learning from the updated data, it gradually refines the approximations to h_0, $\mathbf{C}$ and $\Delta\widehat{E}^{\text{elec}}$, thereby enhancing the PM3 method's predictive capability. This dynamic refinement ensures the SCF loop's stability, preventing any abrupt changes that could disrupt its convergence process.

8.2.1 Iterative network parameter refinement via Backpropagation

Since the PM3 method is an integral part of the overall learning model, it plays a central role in providing the predicted value of the energy. This predicted energy is not just an intermediate result; it is directly employed in the loss function, forming a bridge between the forward prediction and the correction mechanism during training. This interplay means that after every iteration, the backpropagation algorithm needs to find the gradient with respect to specific variables in the model. But here's where the intricacy lies: the gradient calculation doesn't stop at the neural network's parameters. Since the model employs backward accumulation, this gradient computation must traverse through the PM3 model itself. This process forms a continuous chain where the derivatives must be carefully handled, respecting the PM3's computational structure and properties. In essence, the integration of the PM3 method adds layers of complexity in the backward pass.

Automatic differentiation has been applied to numerical methods for quantum chemistry in the past for computing accurate derivatives of electronic structure methods [94], to make excising libraries automatically differentiable [95], as well as to reoptimize basis sets [96]. However, for our study, we will use PySEQM [97] which enables computations on GPUs through the use of automatic differentiation.

Part IV

Results

In the experiments conducted in this section, a consistent set of parameters was utilized across all tests, incorporating three message-passing layers. The only variable among the tests was the batch size. The training aimed to minimize the Mean Absolute Error (MAE) using the ADAM optimizer, with an adaptive learning rate that commenced at 0.0001 and subsequently decreased to 0.00001 throughout the optimization process. Early stopping was employed to prevent overfitting and reduce computation time. The data were partitioned into training, test, and validation sets. The training set comprised 1100 molecules, while both the test and validation sets included 100 molecules each. These specific quantities were chosen as they represent 1% of the entire QM9 dataset.

8.3 Comparing different models

Starting with a batch size of 128,

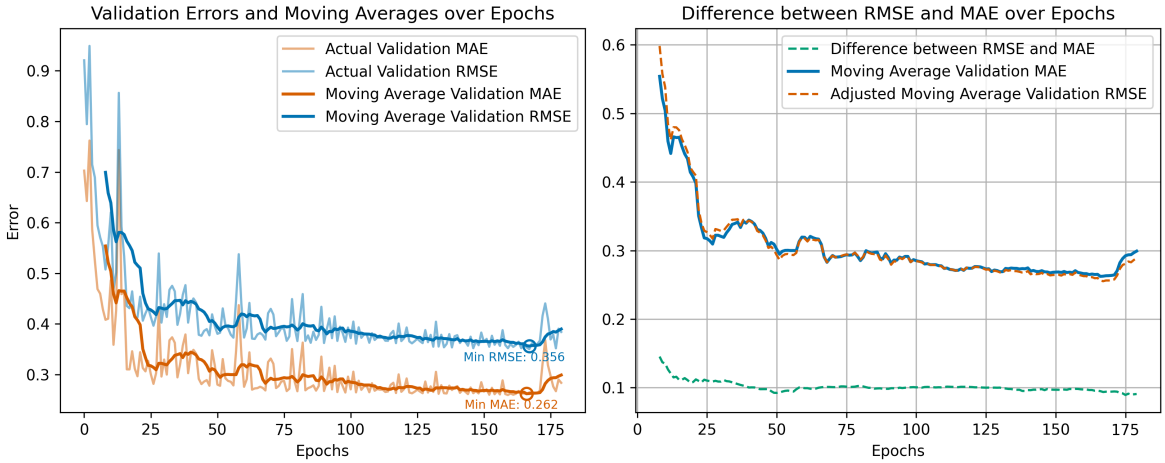

Figure 8.1: *Plot of the validation MAE and RMSE for the MPNN-PM3 hybrid model trained to reduce MAE with a batch size of 128 trained on 1000 molecules. (a) Shows the validation curves with their moving averages while (b) shows the comparison of the MAE and RMSE with removed bias as well as the proportional difference and constant offset between the MAE and RMSE, respectively.*

as we see from Figure 8.1, the validation error is oscillating quite a lot although it is following a descending trend. We also notice that we see almost the exact patterns in the MAE and the Root Mean Squared Error (RMSE) graph with their proportional difference being low. However, we also see that the error kept steadily decreasing from around after 100 epochs to 170 epochs. This indicates that it was slowly approaching a minimum before exponential decay, indicating that it had passed a local minimum and started climbing uphill.

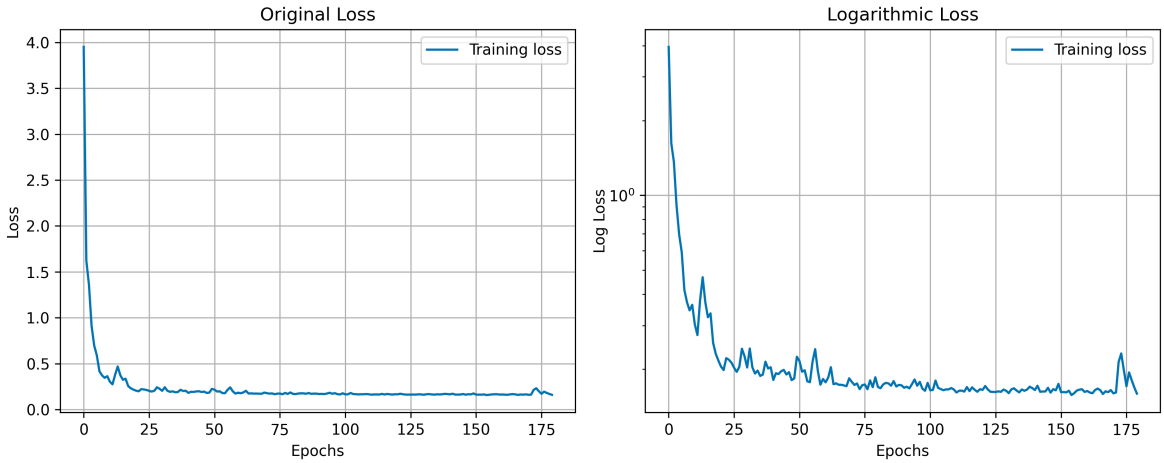

Figure 8.2: *Plot of the loss curve for the MPNN-PM3 hybrid model with a batch size of 128 trained on 1000 molecules. Plot (b) shows the the loss curve with a base 10 logarithmic scale for the y-axis and a linear scale for the x-axis.*

Figure 8.3 shows us the loss curve. As we see, a loss cliff as the loss rapidly falls from around 4.0 to around 2.6. After that, we see the curve remains mostly constant while slowly decreasing. By looking at the exponential scaling we see that the optimization algorithm appears to have identified a promising direction in the loss surface and is rapidly descending towards a local minimum. The exponential rate of decrease suggests that the algorithm is effectively navigating the parameter space. Since we see this behaviour in the initial stages of the training, it suggests the model is rapidly learning to notice 'patterns' in the data. Lastly, we again see the optimizer overshoots after around 170 epochs, this is in line with the observation from figure 8.1 and indicates the optimizer skipped the local minimum, hence reaching early stopping after around 180 epochs.

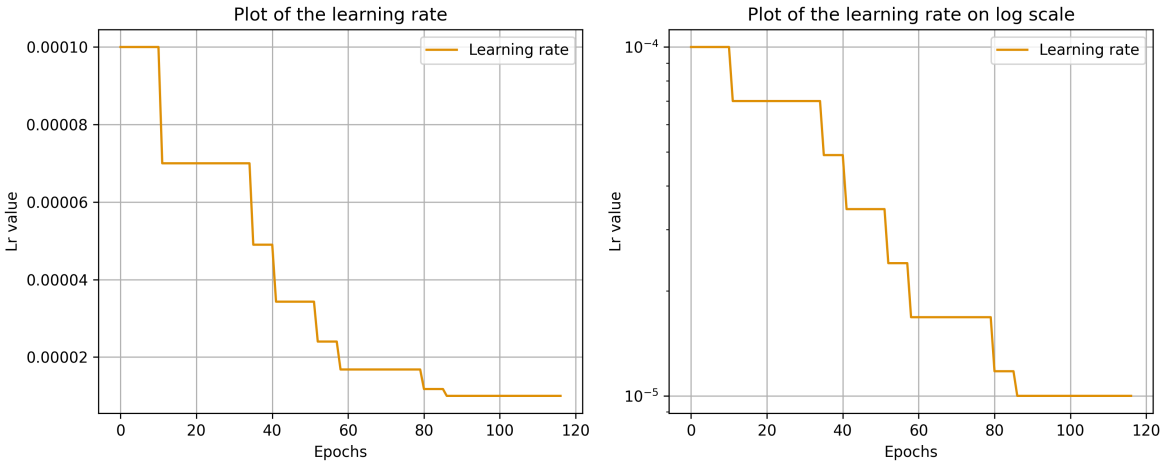

Figure 8.3: *Plot of the step function for the MPNN-PM3 hybrid model with batch size=*128 *trained on 1000 molecules. The figure shows a plot of the linear scaling (a) as well as the base 10 logarithmic scaling on the y-axes (b).*

Looking at Figure 8.3, we can see that after around 25 epoch, from Figure 8.1 we see that this

was right after the optimizer overshot the minimum. This means that the model escaped the local minima due to the relatively high learning rate. The subsequent decrease in learning rate serves as an adaptive strategy to facilitate more precise optimization and enhance the likelihood of convergence to the minimum. We also notice a couple more instances where it overshoots before decreasing in learning rate.

Overall, these results show that the model is working and is managing to navigate the landscape to effectively tune the parameters. But a batch size of 128 may not be the optimal choice, especially if GPU acceleration is not used. When using a batch size of 16, the following observations were observed:

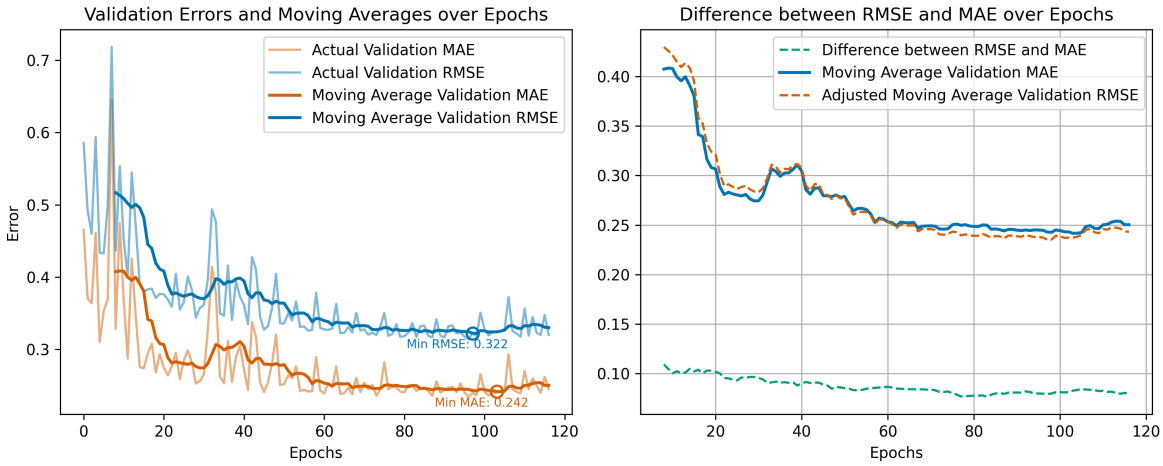

Figure 8.4: *Plot of the validation MAE and RMSE for the MPNN-PM3 hybrid model trained to reduce MAE with a batch size of 16 trained on 1000 molecules. (a) Shows the validation curves with their moving averages while (b) shows the comparison of the MAE and RMSE with removed bias as well as the proportional difference and constant offset between the MAE and RMSE, respectively.*

The first noticeable difference in Figure 8.4 is that the MAE and RMSE error start lower when compared to the same test but with a batch size of 128 as in Figure 8.1 and that it converged to a lower minimum with a MSE $= 0.242$ needing just bellow 120 epochs.

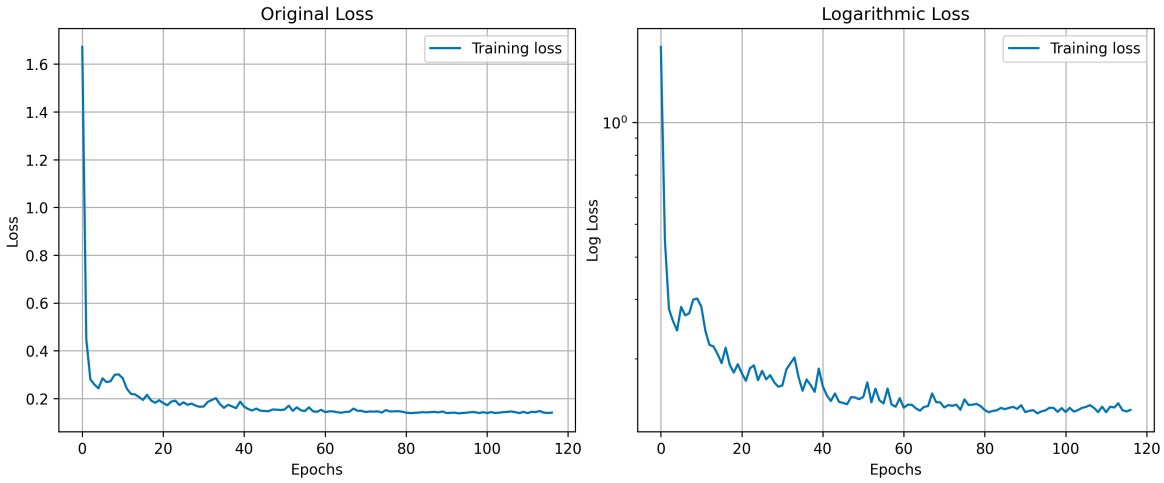

Figure 8.5: *Plot of the loss curve for the MPNN-PM3 hybrid model with a batch size of 16 trained on 1000 molecules. (b) show the the loss curve with a base 10 logarithmic scale for the y-axis and a linear scale for the x-axis.*

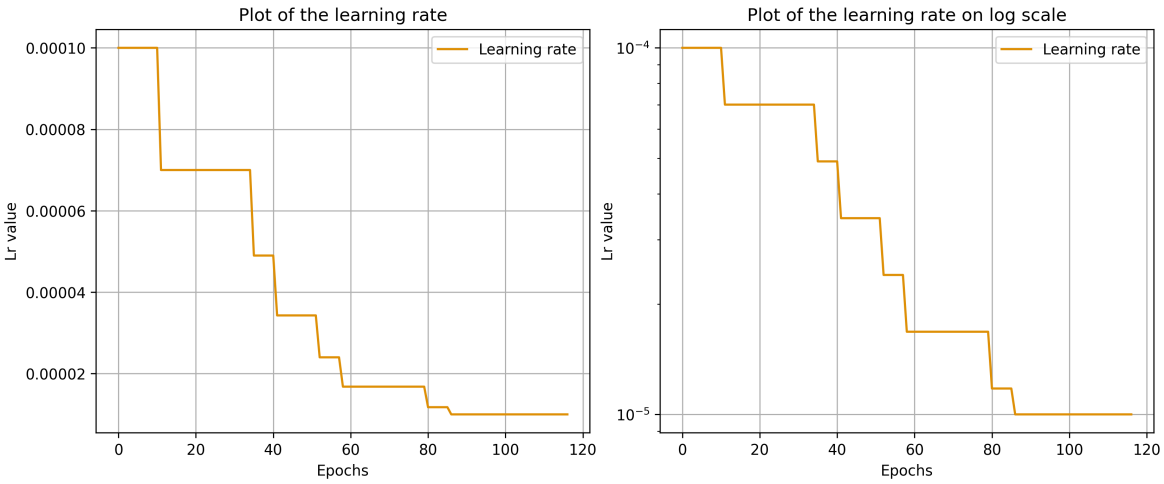

Figure 8.6: *Plot of the step function for the MPNN-PM3 hybrid model with batch size=16 trained on 1000 molecules. The figure shows a plot of the linear scaling (a) as well as the base 10 logarithmic scaling on the y-axes (b).*

When looking at the loss curve in 8.5, we see that the low starting error is followed by a lower starting loss starting at over 1.67. We also again get an exponential decay when logarithmic scaling is used on the y-axis, again showing that the model is approaching a minimum. From Figure 8.6 (b), we see that from between around 30 to 60 epochs, the learning rate was steadily decreasing every 5 to 10 epochs, while it took around 20 epochs for the second drop and second to last drop.

Continuing further, using a batch size of 1 gave the following results

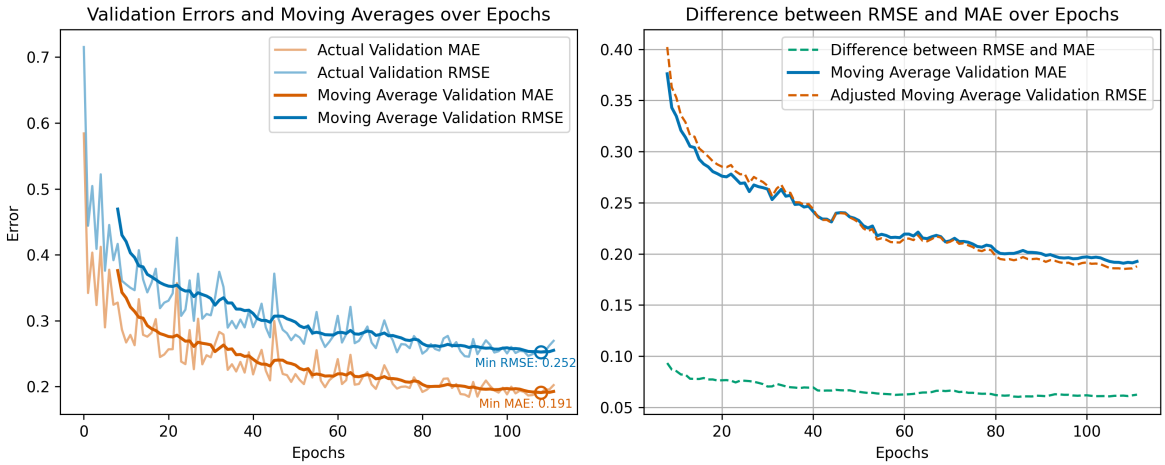

Figure 8.7: *Plot of the validation MAE and RMSE for the MPNN-PM3 hybrid model trained to reduce MAE with a batch size of 1 trained on 1000 molecules. (a) Shows the validation curves with their moving averages while (b) shows the comparison of the MAE and RMSE with removed bias as well as the proportional difference and constant offset between the MAE and RMSE, respectively.*

With a batch size of 1, we start again with a smaller error, however relatively smaller compared to what we see it Figure 8.4 for batch size 16. We also got a less oscillation during training as seen by looking at the moving average of both the MAE and RMSE. An MAE of 0.191 was achieved as well as an RMSE of 0.252 which is lower than the MAE of 0.262 achieved using a batch size of 128 as shown in Figure 8.1.

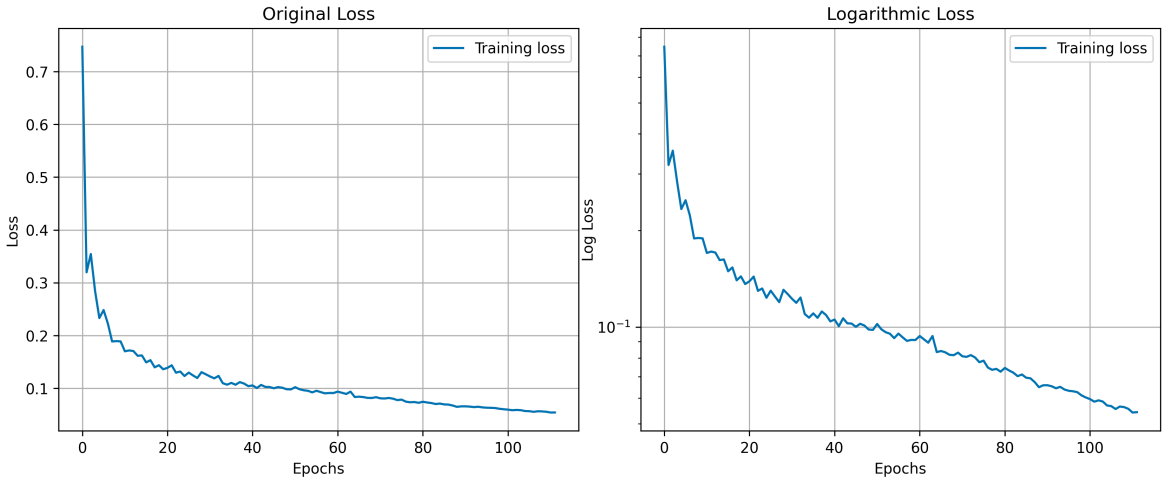

Figure 8.8: *Plot of the loss curve for the MPNN-PM3 hybrid model with a batch size of 1 trained on 1000 molecules. (b) show the the loss curve with a base 10 logarithmic scale for the y-axis and a linear scale for the x-axis.*

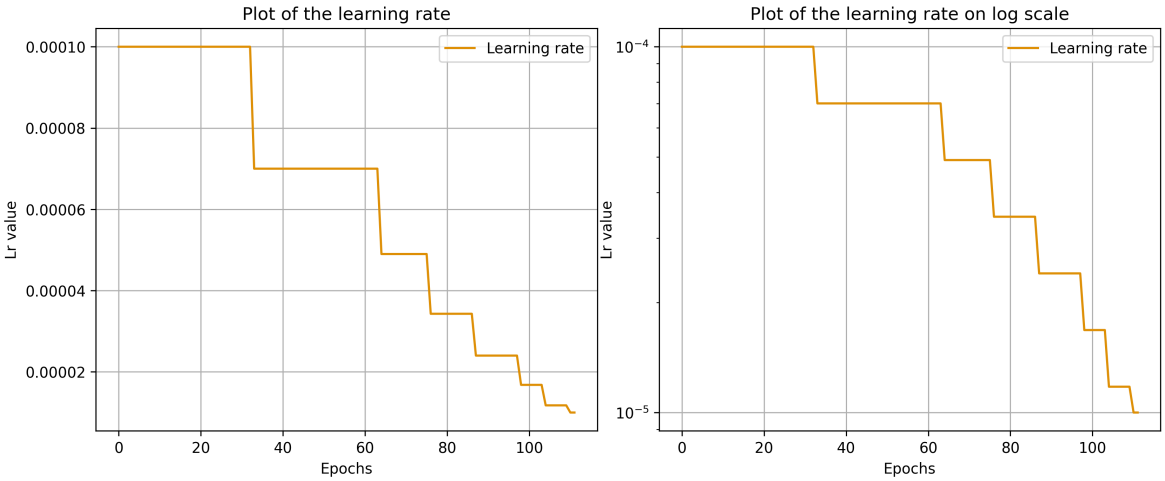

Figure 8.9: *Plot of the step function for the MPNN-PM3 hybrid model with a batch size of* 1 *trained on 1000 molecules. The figure shows a plot of the linear scaling (a) as well as the base 10 logarithmic scaling on the y-axes (b).*

There is a relatively smooth loss curve as seen in Figure 8.8. We can also see that the optimizer overshot a decent amount of times but managed to get to get back on track relatively quickly. There is seems to be close to continious decrees after 80 epoch until the stopping criteria was achieved. Also noticeable is that the "distance" or intervals between learning rate reductions (the steps) is decreasing over time.

8.4 Comparing with PM3

In this section, all the tests were performed on a validation set consisting of 10000 molecules which were not used for training. Further, all the results pertaining to the MPNN-PM3 hybrid method were obtained using a the model with batch size equal to 1.

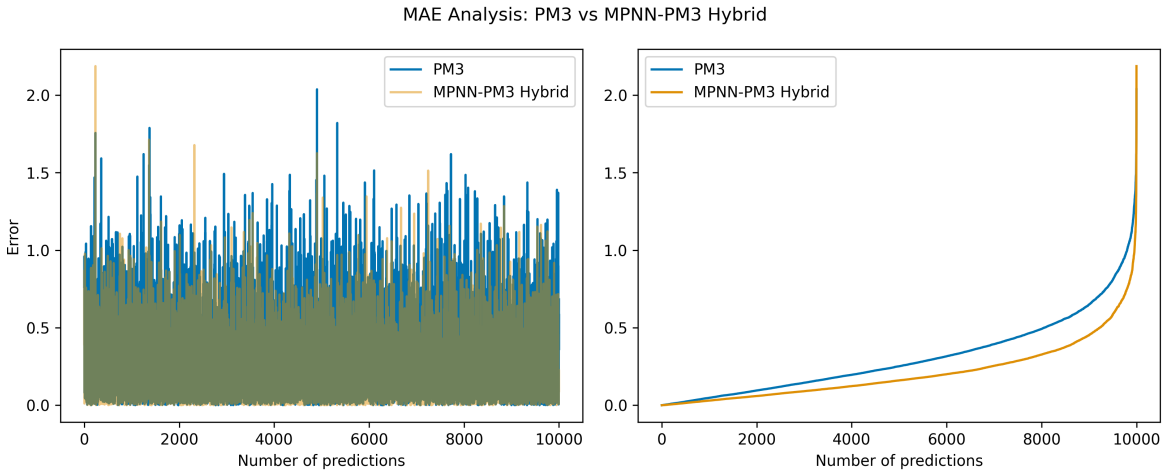

Figure 8.10: *Plot (a) of MAE obtained for each molecule using PM3 method and comparing with the MPNN-PM3 hybrid method. Plot (b) is the same plot but with the errors sorted from lowest to highest*

From Figure 8.10, we see a distribution of the MAE of the PM3 and the MPNN-PM3 hybrid model for each of the molecules in the validation set. We can see that on average, the MAE per molecule is considerably lower for the MPNN-PM3 model. However, there are some instances where the MAE is higher for the hybrid model than for the PM3 method. The highest MAE was also achieved by the hybrid model with an MAE of 2.187, while the highest MAE achieved by the PM3 method was 2.037

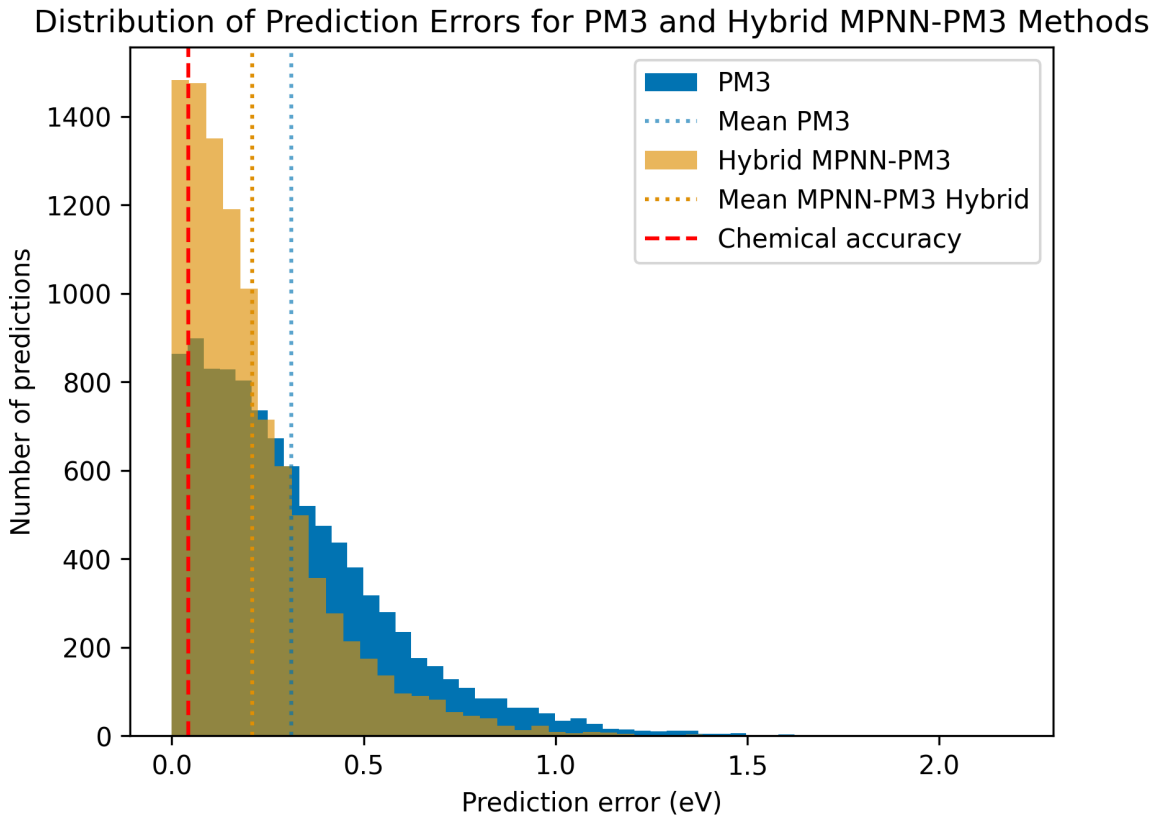

Figure 8.11: *Histogram of the relationship between the models and the target from the QM9 dataset. Plot (a) shows the relationship with the PM3 model, while plot (b) depicts the relationship with the MPNN-PM3 hybrid model. The red line in each plot represents a linear model that best describes the relationship, as determined by linear regression analysis*

Figure 8.12: *Histogram of the prediction errors over all molecules. The blue dotted line shows the mean error for the PM3 method while the orange line shows the mean error for the MPNN-PM3 Hybrid model. The are to the left of the red dashed line is the chemical accuracy range.*

Figure 8.12 shows the prediction error over all the molecules as a histogram. From it, we see that the error for the hybrid model is generally lower than the error for the PM3 model. As indicated by the dotted lines, the mean error for the MPNN-PM3 model was 0.209 eV while it was 0.311 for the PM3 method. The hybrid model predicted 1433 molecules within chemical accuracy while the PM3 method calculated 895 within acceptable error for chemical accuracy.

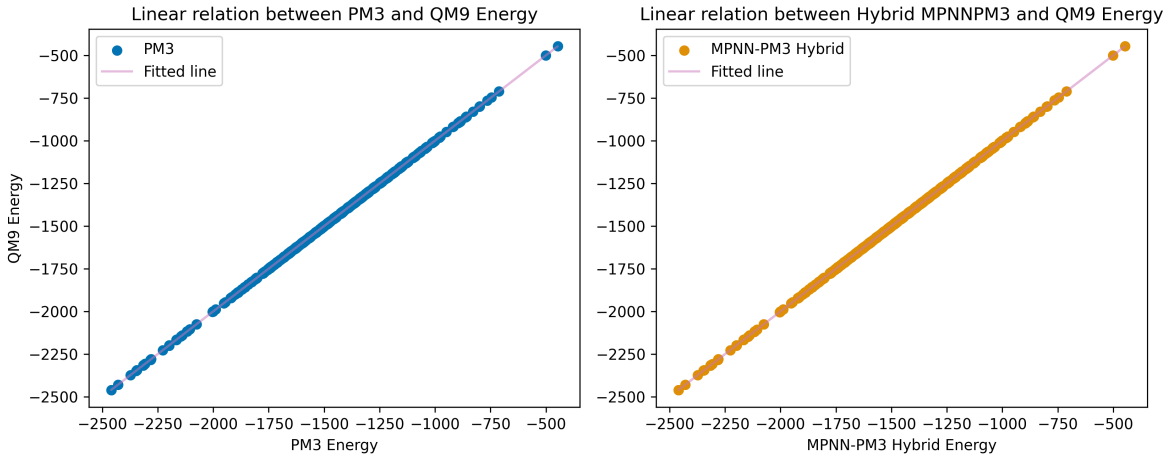

Figure 8.13: *Correlation plot which shows the linear relation between the models and the QM9 target. The red line is a linear model which best describes the relationship as determined by linear regression analysis. Plot (a) shows the linear relation between PM3 and the target while plot (b) shows the relation between the MPNN-PM3 hybrid model and the target.*

Continuing, with Figure 8.13 which shows two scatter plots where the the energy of the PM3 and hybrid model are on the x-axis and the target energy on the y-axis. Each point in the plot represents a single molecule and its positioning shows the energy obtained of the two methods and the target. If the models agree completely with the target, all the points should fall along the diagonal line $y = x$. We can see that this is almost the case for both methods. The faded red line shows the linear relation between calculated/predicted and expected values and was achieved using linear regression. The slope value tells us that of the two fitting show that the model differ with a factor of 2.10994×10^{-5}, making the hybrid model has better linear relationship with the target. The R^2-values were both close to 1 with the R^2 value for the MPNN-PM3 method being slightly higher. This typically indicates that the models have strong predictive power.

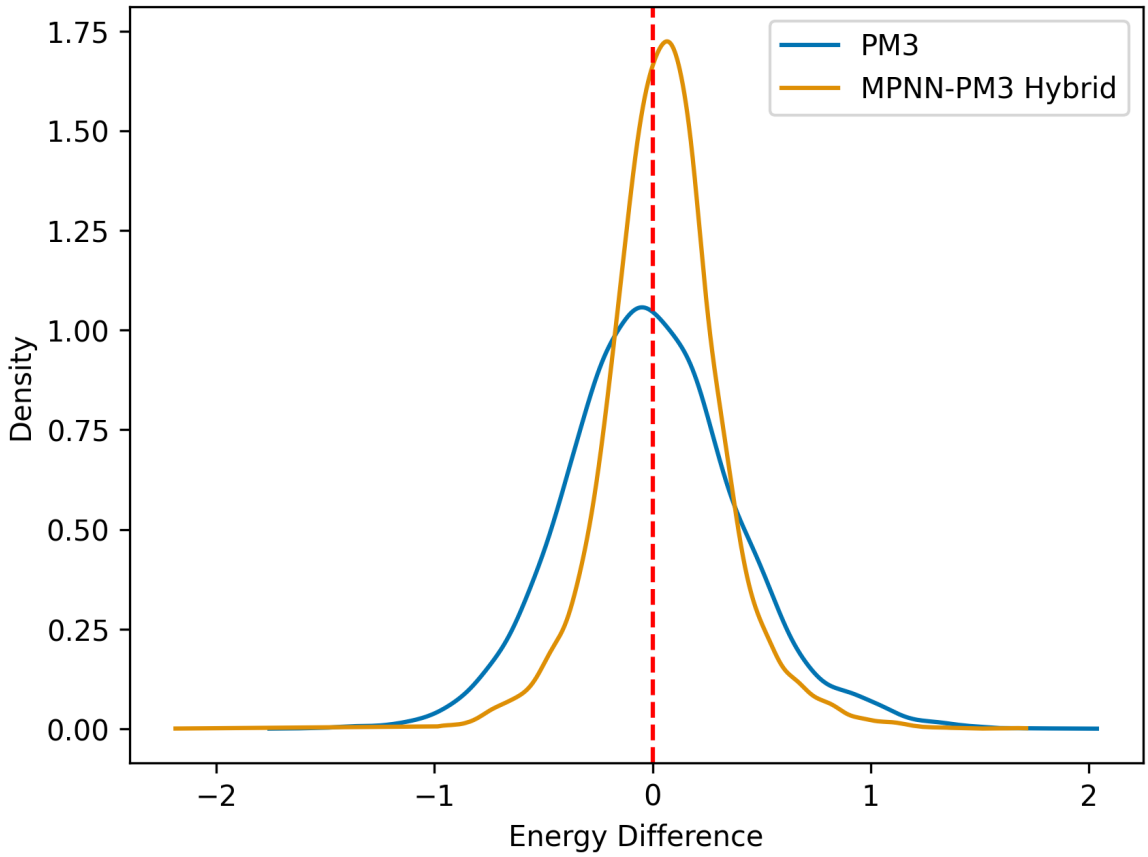

Figure 8.14: *Kernel Density Estimation of the histograms of the energy differences between the QM9 target and the models.*

The Kernel Density Estimation (KDE) of the two models is shown in Figure 8.14. From it, we can see that the PM3 method has a mean energy of -0.004 closer to the target mean when compared against the hybrid model with a mean of 0.055. This also shows that the PM3 has a tendency to calculate energies which are lower than their true value while the hybrid model tends to predict energies higher than their true value (assuming the target value is the true value). However, the standard deviation of the hybrid model is at 0.275 compared to 0.4 for the PM3 method.

8.4.1 Time analysis

In this section, the training time comparison of the differnet MPNN-PM3 hybrid models. All tests were conducted on a Macbook pro with the M1 chip.

Batch Size	Epochs	Time (seconds)	Time per Epoch (seconds)
128	180	2.50×10^4	1.39×10^2
16	117	1.00×10^4	8.55×10^1
1	112	2.80×10^4	2.50×10^2

Table 8.1: *Training itme comparison of the different MPNN-PM3 Hybrid models.*

Table 8.1 shows the training time for the different models as well as their average time per epoch. As we can see from the table, the hybrid model with a batch size of 16 the most time efficient. Even though the model required almost the same amount of epoch to reach early stopping as the model with batch size 1, the time per epoch was over 3 times as fast. The model with a batch size of 128 was around 1.8 times as fast per epoch as the model with batch size 1, but it needed 180 epochs to converge, compared to the 112 needed by the model with batch size 1.

Model Name	**Time (seconds)**
PM3	4.78×10^2
MPNN-PM3	6.54×10^2

Table 8.2: *Comparison of the time the the PM3 method and the MPNN-PM3 Hybrid model (batch size = 1) took to compute the total energy for 10000 molecules*

Table 8.2 shows the time it took to find the total energy for the two models on 10000 molecules. What we see is that MPNN-PM3 is around 36.82% slower than the PM3 method.

Part V

Conclusion

Chapter 9

Discussion

In this work we have implemented a model dubbed MPNN-PM3 hybrid which is a MPNN trained to predict parameters for the PM3 semiempirical method. The model was shown to converge to a minimum during training and was successfully texted and compared against the PM3 model. In the training step, a relatively small learning rate had to be applied as high oscillation of the loss was encountered during training. This behaviour was to be expected as the original PM3 parameters provide a decent optimization of the hybrid model and great deviations in these values, which would have been caused by a higher learning rate would have made the model more unpredictable. It was also observed that the RMSE of the hybrid model decreased at the same rate throughout the training as the MAE. This behaviour is also to be expected as the MAE and the RMSE are highly related and a decrease in one would lead to a decrease in the other.

The MAE and RMSE errors for the first epochs of the hybrid model decreased with the batch size. The reason for this is likely due to the model needing to fit more parameters for higher batch sizes, increasing the risk of poorly fitting certain parameters, at least in the early stages of training when the model is still adapting to the data. On the other hand, the model also converged to lower local minima for smaller batch sizes. The reason for this is not straight-forward. What was however particularly interesting was that that the model with batch size equal to 1 converged to the lowest error. From my experience, this is unusual as training on a small batch size introduces more noise into the gradient updates. This increases the variability and uncertainty in the estimates of the gradients that guide the model's learning process. This can have both positive and negative impacts. The added nose could give the potential capture different characteristics of the overall data distribution as the gradient can vary more widely from one batch to another. This will then help the model to escape local minima in the loss landscape and lead to a better solution. A smaller batch size usually means that more updates are med per epoch which will lead to a slower convergence. This was the case for the MPNN-PM3 hybrid model as training the model took longer even though fewer epochs were needed to reach early stopping.

The MAE of 0.192 which was achieved for the hybrid model with a batch size of 1 is of the same order of magnitude as the MAE achieved by Gilmer et al. [5]. This error reduction is remarkable considering the short time frame for this thesis and the limited scope of training. It is reasonable to expect even better model performance with extensive training.

When comparing the hybrid model with the PM3 model on a validation set of 10000 unseen molecules, a significant reduction in MAE for each molecules was achieved for the wast majority of molecules. The reason for why the MPN-PM3 model had a higher error than the PM3 model for certain molecules could be due to the molecules being relatively different than the ones used in the training set and hence making the parameterization for these molecules suboptimal. The hybrid model also achieved the highest MAE of a single molecule which is likely due to the same reason. But despite this, the mean MAE of the MPNN-PM3 hybrid method for the validation set was 0.201 compared to 0.311 for the PM3 which is a substantial improvement. This is slightly worse than the MAE it achieved during training. However, the difference is small and the model still generalizes well on the new unseen data.

The MPNN-PM3 method also managed to predict a little over 14% of the molecules withing the chemical accuracy range while the PM3 method calculated just under 9% of the molecules within the chemical accuracy range. By looking at the kernel density estimation, it was noticeable that the PM3 method tends to underestimate the energy of the molecules, calculating an energy which is lower than the actual energy. This observation is in line with what is established about the PM3 method. However, the MPNN-PM3 hybrid tends to overestimate the energy which can also be seen by looking at the distribution of the MAE per molecule. The overestimation could indicate that the model has overfitted as overfitting causes the model to capture noise or outliers in the training data, leading to overpredictions. Overfitting is a common issue in the field of machine learning and could be root of the issue. However, the MPNN-PM3 predictions also has a decently lower standard deviation compared to PM3. This means most of the energies are predicted closer to target value. However, this makes it more unlikely that overfitting is the cause of overestimation as it typically leads to increased variance in predictions on unseen data, which is the opposite of what was observed in this work. The overprediction could then be due to a systematic bias in the model, or more likely due to the model poorly parameterizing the PM3 method for molecules less similar to the ones in its training data.

Chapter 10

Future Work

Taking into account the positive results achieved in this work, there are still much more that can be done. Hyperparameter tuning is a process of systematically selecting the best combination of hyperparameters that define a model's architecture and the way it is trained. This process helps in finding the optimal set of hyperparameters that minimize a predefined loss function or maximize the accuracy of the model on a given dataset. Currently, the MPNN has multiple hyperparamets which can be tuned. These include the size of the internal layers, size of the linear layers of the neural network, choice of activation function and the choice of aggregation operator. There is also a hand full of readout functions to choose from and since many works in the literature suggests that the choice of readout function can greatly impact the performance of a model [98–102], it is worth exploring this in greater depth. In this study, the sequence-to-sequence [103] operator was used as the readout function. This operator is used to map a set of node features to a fixed-size vector and it has tunable parameter which is used determine the number of iterations the operator will make over the node features. A higher number of steps can provide a greater representation of the set, while a lower number will reduce computation time. Removing the readout function can also potentially allow for the model to tune different parameters for the same atom type in a molecule as it won't return a fixed-size representation, but more research has to be done. The ADAM optimizer, which was used to navigate parameter space also has multiple parameters which can be tuned to achieve better results. It is also worth considering other optimization algorithms as SGD as it has shown to perform better than ADAM on certain tasks. More test runs should also be performed to further strengthen and get a more complete picture of the results as one sample run is often not enough for a comprehensive overview.

Further benchmark tests should also be performed to get a broader understanding of how the MPNN-PM3 hybrid model works. The most obvious one would be to train the model on the entire QM9 dataset then run the same experiments to see how it compares with the current model. This could help answer the open questions arrising from the current results. Further tests on how the model predicts other molecular properties it was not trained on and how it compares with PM3 is also of great interest. Besides using the model to predict sets of molecular properties, what should also be done in the future is to use a multi-task learning approach to train the model to correctly predict multiple molecular properties, instead of just training it predict energies as is done so far. This is typically achieved by training the

model to minimize a joint loss function, which is often a weighted sum of the individual loss functions for each task. The weights can be used to balance the importance of different tasks, depending on the specific problem and objectives. It would also be beneficial to see how the MPNN-PM3 hybrid model compares against more sophisticated semiempirical methods like the PM6 [23], PM7 [104] or the orthogonalization-corrected OMx [105] methods which add additional interactions into the Fock matrix and has been shown yield improvements for ground-state and excited-state properties. It is also worth comparing the model against Density-Functional Tight Binding (DFTB) methods or the highly effective extended Tight-Bindind (xTB) [106] methods.

Other than using these methods as comparison against the MPNN-PM3 hybrid model, they could also be used as part of the MPNN architecture by replacing the currently used PM3 method. For example, the PM6 method contains a parameter which controls the rate of which the core-core repulsion between two atoms decay with distance. However, finding and developing automatic differentiable versions of other semiempirical methods is a research project in itself.

Since the PM3 method was optimized to compute properties of the molecules in its training set, the current set of parameters which is standard for the PM3, contribute to a good local minimum for that particular dataset. However, it is quite probable that they are not a local minimum for the QM9 data. Therefore, training the MPNN-PM3 hybrid model to find deviations for these parameters is likely not optimal as we are limiting the model to freely explore the entire landscape by forcing it to not deviate too far from the standard PM3 parameters. A likely better approach would have been to re-optimize the PM3 method for the entire QM9 dataset, or parts of it, and then train a MPNN model to find deviations in the re-optimized parameters.

Extending the model to include *d*-orbitals for transition metals is also a further extension of the project. In its time, the PM3(tm) model was optimized on geometrical data as accurate and reliable energetic data was not available for transition metal compounds. Today, the tmQM dataset [107] contains geometries and properties of over 86,000 large transition metal compounds.

Going back to the machine learning model, there is also other message-passing layers to experiment or implement Equivariant Graph Neural Networks (EGNNs) [108]. These models are equivariant to rotations, translations, reflections and permutations. Such networks can work well on molecular data when a molecule can be represented in various orientations and permutations. Besides that, the implementation of batch normalization [109], which is a technique used tp normalize the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation. This process helps in stabilizing and accelerating the training of deep neural networks. Dropout is another technique where under training, some randomly selected neurons are dropped, setting their contribution to zero. This method can ensure that the model does not become overly reliant on a single neuron, encouraging a more distributed and robust representation.

Lastly, Automatic differentiation has recently also been applied to HF and DFT in the work by Kasim et al. [96] which allows for basis set optimization. Here the authors used automatic differentiation with respect to the the basis set to optimize system-specific basis sets. They effectively reoptimized the cc-pVDZ basis [110] using just a small training set and their

results lead to a decrease in the total energy of all the molecules they used in the test set. The authors also used automatic differentiation to perform alchemical perturbation studies to predict molecular properties without simulating them. A MPNN or other type of network can be used for this training process, or for entirely new use cases in quantum chemistry which utilize automatic differentiation.

Bibliography

[1] Raghunathan Ramakrishnan et al. 'Quantum chemistry structures and properties of 134 kilo molecules'. en. In: *Scientific Data* 1.1 (Aug. 2014). Number: 1 Publisher: Nature Publishing Group, p. 140022. ISSN: 2052-4463. DOI: 10.1038/sdata.2014.22. URL: https://www.nature.com/articles/sdata201422 (visited on 16/06/2023).

[2] James J. P. Stewart. 'Optimization of parameters for semiempirical methods II. Applications'. en. In: *Journal of Computational Chemistry* 10.2 (1989). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/jcc.540100209, pp. 221–264. ISSN: 1096-987X. DOI: 10.1002/jcc.540100209. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/jcc.540100209 (visited on 19/06/2023).

[3] James J. P. Stewart. 'Optimization of parameters for semiempirical methods II. Applications'. en. In: *Journal of Computational Chemistry* 10.2 (1989). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/jcc.540100209, pp. 221–264. ISSN: 1096-987X. DOI: 10.1002/jcc.540100209. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/jcc.540100209 (visited on 10/08/2023).

[4] James J. P. Stewart. 'Optimization of parameters for semiempirical methods, Part 3: extension of PM3 to Be, Mg, Zn, Ga, Ge, As, Se, Cd, In, Sn, Sb, Te, Hg, Tl, Pb, and Bi'. In: (Apr. 1991). DOI: 10.1002/jcc.540120306. URL: https://zenodo.org/record/1229233 (visited on 18/07/2023).

[5] Justin Gilmer et al. *Neural Message Passing for Quantum Chemistry*. arXiv:1704.01212 [cs]. June 2017. DOI: 10.48550/arXiv.1704.01212. URL: http://arxiv.org/abs/1704.01212 (visited on 16/06/2023).

[6] M. Born and W. Heisenberg. 'Zur Quantentheorie der Molekeln'. de. In: *Original Scientific Papers Wissenschaftliche Originalarbeiten*. Ed. by Walter Blum, Helmut Rechenberg and Hans-Peter Dürr. Werner Heisenberg Gesammelte Werke Collected Works. Berlin, Heidelberg: Springer, 1985, pp. 216–246. ISBN: 978-3-642-61659-4. DOI: 10.1007/978-3-642-61659-4_16. URL: https://doi.org/10.1007/978-3-642-61659-4_16 (visited on 08/08/2023).

[7] Frank Jensen. *Introduction to computational chemistry*. en. Third edition. Chichester, UK ; Hoboken, NJ: Wiley, 2017. ISBN: 978-1-118-82595-2 978-1-118-82598-3.

[8] W. Pauli. 'Über den Zusammenhang des Abschlusses der Elektronengruppen im Atom mit der Komplexstruktur der Spektren'. de. In: *Zeitschrift für Physik* 31.1 (Feb. 1925), pp. 765–783. ISSN: 0044-3328. DOI: 10.1007/BF02980631. URL: https://doi.org/10.1007/BF02980631 (visited on 08/08/2023).

[9] David J. Griffiths and Darrell F. Schroeter. *Introduction to Quantum Mechanics*. en. 3rd ed. Cambridge University Press, Aug. 2018. ISBN: 978-1-316-99543-3 978-1-107-18963-8. DOI: 10.1017/9781316995433. URL: https://www.cambridge.org/highereducation/product/9781316995433/book (visited on 09/08/2023).

[10] J. C. Slater. 'The Theory of Complex Spectra'. In: *Physical Review* 34.10 (Nov. 1929). Publisher: American Physical Society, pp. 1293–1322. DOI: 10.1103/PhysRev.34.1293. URL: https://link.aps.org/doi/10.1103/PhysRev.34.1293 (visited on 09/08/2023).

[11] A. Szabo and N.S. Ostlund. *Modern Quantum Chemistry: Introduction to Advanced Electronic Structure Theory*. Dover Books on Chemistry. Dover Publications, 2012. ISBN: 978-0-486-13459-8. URL: https://books.google.no/books?id=KQ3DAgAAQBAJ.

[12] Errol G. Lewars. *Computational Chemistry*. en. Cham: Springer International Publishing, 2016. ISBN: 978-3-319-30914-9 978-3-319-30916-3. DOI: 10.1007/978-3-319-30916-3. URL: http://link.springer.com/10.1007/978-3-319-30916-3 (visited on 31/07/2023).

[13] J. C. Slater. 'Atomic Shielding Constants'. In: *Physical Review* 36.1 (July 1930). Publisher: American Physical Society, pp. 57–64. DOI: 10.1103/PhysRev.36.57. URL: https://link.aps.org/doi/10.1103/PhysRev.36.57 (visited on 31/07/2023).

[14] David C. Young. *Computational Chemistry: A Practical Guide for Applying Techniques to Real World Problems*. Wiley, 2001. ISBN: 9780471333685. URL: https://www.wiley.com/en-hk/Computational+Chemistry%3A+A+Practical+Guide+for+Applying+Techniques+to+Real+World+Problems-p-9780471333685.

[15] [First name] Offenhartz. *Atomic Habits and Molecular Orbital Theory*. McGraw-Hill, 1970.

[16] Michael J. S. Dewar and Walter Thiel. 'Ground states of molecules. 38. The MNDO method. Approximations and parameters'. In: *Journal of the American Chemical Society* 99.15 (June 1977). Publisher: American Chemical Society, pp. 4899–4907. ISSN: 0002-7863. DOI: 10.1021/ja00457a004. URL: https://doi.org/10.1021/ja00457a004 (visited on 01/08/2023).

[17] Walter Thiel. 'Semiempirical quantum–chemical methods'. en. In: *WIREs Computational Molecular Science* 4.2 (Mar. 2014), pp. 145–157. ISSN: 1759-0876, 1759-0884. DOI: 10.1002/wcms.1161. URL: https://onlinelibrary.wiley.com/doi/10.1002/wcms.1161 (visited on 01/08/2023).

[18] Ira N. Levine. *Quantum chemistry*. en. Seventh edition. Boston: Pearson, 2014. ISBN: 978-0-321-80345-0.

[19] Richard C. Bingham, Michael J. S. Dewar and Donald H. Lo. 'Ground states of molecules. XXV. MINDO/3. Improved version of the MINDO semiempirical SCF-MO method'. In: *Journal of the American Chemical Society* 97.6 (Mar. 1975). Publisher: American Chemical Society, pp. 1285–1293. ISSN: 0002-7863. DOI: 10.1021/ja00839a001. URL: https://doi.org/10.1021/ja00839a001 (visited on 01/08/2023).

[20] Michael J. S. Dewar et al. 'Development and use of quantum mechanical molecular models. 76. AM1: a new general purpose quantum mechanical molecular model'. In: *Journal of the American Chemical Society* 107.13 (June 1985). Publisher: American Chemical Society, pp. 3902–3909. ISSN: 0002-7863. DOI: 10.1021/ja00299a024. URL: https://doi.org/10.1021/ja00299a024 (visited on 19/06/2023).

[21] *Computational Chemistry: A Practical Guide for Applying Techniques to Real World Problems | Wiley*. en-hk. URL: https://www-wiley-com.ezproxy.uio.no/en-hk/Computational+Chemistry%3A+A+Practical+Guide+for+Applying+Techniques+to+Real+World+Problems-p-9780471333685 (visited on 10/08/2023).

[22] Michael J. S. Dewar, Caoxian Jie and Jianguo Yu. 'SAM1; The first of a new series of general purpose quantum mechanical molecular models'. en. In: *Tetrahedron* 49.23 (June 1993), pp. 5003–5038. ISSN: 0040-4020. DOI: 10.1016/S0040-4020(01)81868-8. URL: https://www.sciencedirect.com/science/article/pii/S0040402001818688 (visited on 10/08/2023).

[23] James J. P. Stewart. 'Optimization of parameters for semiempirical methods V: Modification of NDDO approximations and application to 70 elements'. en. In: *Journal of Molecular Modeling* 13.12 (Dec. 2007), pp. 1173–1213. ISSN: 0948-5023. DOI: 10.1007/s00894-007-0233-4. URL: https://doi.org/10.1007/s00894-007-0233-4 (visited on 09/08/2023).

[24] Walter Thiel and Alexander A. Voityuk. 'Extension of MNDO to d Orbitals:XXXX Parameters and Results for the Second-Row Elements and for the Zinc Group'. In: *The Journal of Physical Chemistry* 100.2 (Jan. 1996). Publisher: American Chemical Society, pp. 616–626. ISSN: 0022-3654. DOI: 10.1021/jp952148o. URL: https://doi.org/10.1021/jp952148o (visited on 02/08/2023).

[25] Michael J. S. Dewar and Walter Thiel. 'A semiempirical model for the two-center repulsion integrals in the NDDO approximation'. en. In: *Theoretica chimica acta* 46.2 (June 1977), pp. 89–104. ISSN: 1432-2234. DOI: 10.1007/BF00548085. URL: https://doi.org/10.1007/BF00548085 (visited on 02/08/2023).

[26] Walter Thiel. 'Perspectives on Semiempirical Molecular Orbital Theory'. In: vol. 93. Mar. 2007, pp. 703–757. ISBN: 9780470141526. DOI: 10.1002/9780470141526.ch10.

[27] WJ Hehre, J Yu and E Adei. 'Semi-empirical molecular orbital methods for organometallic reaction mechanisms.' In: *ABSTRACTS OF PAPERS OF THE AMERICAN CHEMICAL SOCIETY*. Vol. 212. AMER CHEMICAL SOC 1155 16TH ST, NW, WASHINGTON, DC 20036. 1996, 92–COMP.

[28] Warren J Hehre, Jianguo Yu and Philip E Klunzinger. *A guide to molecular mechanics and molecular orbital calculations in Spartan*. Wavefunction, 1997.

[29] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. arXiv:1609.04747 [cs]. June 2017. DOI: 10.48550/arXiv.1609.04747. URL: http://arxiv.org/abs/1609.04747 (visited on 15/06/2023).

[30] Ian Goodfellow, Yoshua Bengio and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[31] Prateek Jain and Purushottam Kar. 'Non-convex Optimization for Machine Learning'. In: *Foundations and Trends® in Machine Learning* 10.3-4 (2017). arXiv:1712.07897 [cs, math, stat], pp. 142–336. ISSN: 1935-8237, 1935-8245. DOI: 10.1561/2200000058. URL: http://arxiv.org/abs/1712.07897 (visited on 09/08/2023).

[32] Yann Dauphin et al. *Identifying and attacking the saddle point problem in high-dimensional non-convex optimization*. arXiv:1406.2572 [cs, math, stat]. June 2014. DOI: 10.48550/arXiv.1406.2572. URL: http://arxiv.org/abs/1406.2572 (visited on 09/08/2023).

[33] Shiliang Sun et al. *A Survey of Optimization Methods from a Machine Learning Perspective*. arXiv:1906.06821 [cs, math, stat]. Oct. 2019. DOI: 10.48550/arXiv.1906.06821. URL: http://arxiv.org/abs/1906.06821 (visited on 18/06/2023).

[34] Quoc V. Le et al. 'On optimization methods for deep learning'. In: *Proceedings of the 28th International Conference on International Conference on Machine Learning*. ICML'11. Madison, WI, USA: Omnipress, June 2011, pp. 265–272. ISBN: 978-1-4503-0619-5. (Visited on 08/08/2023).

[35] Xin-She Yang et al. 'A framework for self-tuning optimization algorithm'. en. In: *Neural Computing and Applications* 23.7 (Dec. 2013), pp. 2051–2057. ISSN: 1433-3058. DOI: 10.1007/s00521-013-1498-4. URL: https://doi.org/10.1007/s00521-013-1498-4 (visited on 09/08/2023).

[36] Hilde J. P. Weerts, Andreas C. Mueller and Joaquin Vanschoren. *Importance of Tuning Hyperparameters of Machine Learning Algorithms*. arXiv:2007.07588 [cs, stat]. July 2020. DOI: 10.48550/arXiv.2007.07588. URL: http://arxiv.org/abs/2007.07588 (visited on 09/08/2023).

[37] Léon Bottou, Frank E. Curtis and Jorge Nocedal. 'Optimization Methods for Large-Scale Machine Learning'. In: *SIAM Review* 60.2 (Jan. 2018). Publisher: Society for Industrial and Applied Mathematics, pp. 223–311. ISSN: 0036-1445. DOI: 10.1137/16M1080173. URL: https://epubs-siam-org.ezproxy.uio.no/doi/abs/10.1137/16m1080173 (visited on 09/08/2023).

[38] Herbert Robbins and Sutton Monro. 'A Stochastic Approximation Method'. In: *The Annals of Mathematical Statistics* 22.3 (1951). Publisher: Institute of Mathematical Statistics, pp. 400–407. ISSN: 0003-4851. URL: https://www.jstor.org/stable/2236626 (visited on 09/08/2023).

[39] David E. Rumelhart, Geoffrey E. Hinton and Ronald J. Williams. 'Learning representations by back-propagating errors'. en. In: *Nature* 323.6088 (Oct. 1986). Number: 6088 Publisher: Nature Publishing Group, pp. 533–536. ISSN: 1476-4687. DOI: 10.1038/323533a0. URL: https://www.nature.com/articles/323533a0 (visited on 09/08/2023).

[40] Ilya Sutskever. 'Training recurrent neural networks'. AAINS22066 ISBN-13: 9780499220660. phd. CAN: University of Toronto, 2013.

[41] Ilya Sutskever et al. 'On the importance of initialization and momentum in deep learning'. In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1139–1147. URL: https://proceedings.mlr.press/v28/sutskever13.html.

[42] *Fundamentals of Deep Learning [Book]*. en. ISBN: 9781491925614. URL: https://www.oreilly.com/library/view/fundamentals-of-deep/9781491925607/ (visited on 09/08/2023).

[43] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. arXiv:1412.6980 [cs]. Jan. 2017. DOI: 10.48550/arXiv.1412.6980. URL: http://arxiv.org/abs/1412.6980 (visited on 09/08/2023).

[44] *Introduction to Machine Learning*. en-US. URL: https://mitpress-mit-edu.ezproxy.uio.no/9780262043793/introduction-to-machine-learning/ (visited on 09/08/2023).

[45] Alex Graves and Jürgen Schmidhuber. 'Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks'. In: *Advances in Neural Information Processing Systems*. Vol. 21. Curran Associates, Inc., 2008. URL: https://papers.nips.cc/paper_files/paper/2008/hash/66368270ffd51418ec58bd793f2d9b1b-Abstract.html (visited on 11/07/2023).

[46] Alex Graves et al. 'A novel connectionist system for unconstrained handwriting recognition'. eng. In: *IEEE transactions on pattern analysis and machine intelligence* 31.5 (May 2009), pp. 855–868. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2008.137.

[47] Santiago Fernández, Alex Graves and Jürgen Schmidhuber. 'An application of recurrent neural networks to discriminative keyword spotting'. In: *Proceedings of the 17th international conference on Artificial neural networks*. ICANN'07. Berlin, Heidelberg: Springer-Verlag, Sept. 2007, pp. 220–229. ISBN: 978-3-540-74693-5. (Visited on 11/07/2023).

[48] Hasim Sak, Andrew W. Senior and Françoise Beaufays. 'Long short-term memory recurrent neural network architectures for large scale acoustic modeling'. In: *INTERSPEECH*. 2014, pp. 338–342.

[49] Xiangang Li and Xihong Wu. *Constructing Long Short-Term Memory based Deep Recurrent Neural Networks for Large Vocabulary Speech Recognition*. arXiv:1410.4281 [cs]. May 2015. DOI: 10.48550/arXiv.1410.4281. URL: http://arxiv.org/abs/1410.4281 (visited on 11/07/2023).

[50] John F. Kolen and Stefan C. Kremer. *A Field Guide to Dynamical Recurrent Networks*. en. Google-Books-ID: NWOcMVA64aAC. John Wiley & Sons, Jan. 2001. ISBN: 978-0-7803-5369-5.

[51] S. Hochreiter. *Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Universität München*. 1991.

[52] S. Hochreiter et al. In: *A Field Guide to Dynamical Recurrent Neural Networks*.

[53] Sunitha Basodi et al. 'Gradient amplification: An efficient way to train deep neural networks'. In: *Big Data Mining and Analytics* 3.3 (Sept. 2020). Conference Name: Big Data Mining and Analytics, pp. 196–207. ISSN: 2096-0654. DOI: 10.26599/BDMA.2020.9020004.

[54] Sepp Hochreiter and Jürgen Schmidhuber. 'Long Short-Term Memory'. In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: https://doi.org/10.1162/neco.1997.9.8.1735 (visited on 11/07/2023).

[55] Kyunghyun Cho et al. *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. arXiv:1406.1078 [cs, stat]. Sept. 2014. DOI: 10.48550/arXiv.1406.1078. URL: http://arxiv.org/abs/1406.1078 (visited on 11/07/2023).

[56] Felix A. Gers, Nicol N. Schraudolph and Jürgen Schmidhuber. 'Learning precise timing with lstm recurrent networks'. In: *The Journal of Machine Learning Research* 3.null (Mar. 2003), pp. 115–143. ISSN: 1532-4435. DOI: 10.1162/153244303768966139. URL: https://dl.acm.org/doi/10.1162/153244303768966139 (visited on 11/07/2023).

[57] F.A. Gers and E. Schmidhuber. 'LSTM recurrent networks learn simple context-free and context-sensitive languages'. In: *IEEE Transactions on Neural Networks* 12.6 (Nov. 2001). Conference Name: IEEE Transactions on Neural Networks, pp. 1333–1340. ISSN: 1941-0093. DOI: 10.1109/72.963769.

[58] F.A. Gers, J. Schmidhuber and F. Cummins. 'Learning to forget: continual prediction with LSTM'. In: *1999 Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470)*. Vol. 2. ISSN: 0537-9989. Sept. 1999, 850–855 vol.2. DOI: 10.1049/cp:19991218.

[59] Dias Satria. 'PREDICTING BANKING STOCK PRICES USING RNN, LSTM, AND GRU APPROACH'. en. In: *Applied Computer Science* 19.1 (Mar. 2023). Number: 1, pp. 82–94. ISSN: 2353-6977. DOI: 10.35784/acs-2023-06. URL: https://ph.pollub.pl/index.php/acs/article/view/3530 (visited on 13/07/2023).

[60] William L Hamilton (2020). 'Graph Representation Learning. Synthesis Lectures on Artificial Intelligence and Machine Learning, Vol. 14, No. 3 , Pages 1-159.' en. In: ().

[61] Michael M. Bronstein et al. *Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges*. en. arXiv:2104.13478 [cs, stat]. May 2021. URL: http://arxiv.org/abs/2104.13478 (visited on 18/06/2023).

[62] Peter W. Battaglia et al. *Relational inductive biases, deep learning, and graph networks*. arXiv:1806.01261 [cs, stat]. Oct. 2018. DOI: 10.48550/arXiv.1806.01261. URL: http://arxiv.org/abs/1806.01261 (visited on 11/08/2023).

[63] Kenneth H. Rosen. *Discrete mathematics and its applications*. en. 7th ed. New York: McGraw-Hill, 2012. ISBN: 978-0-07-338309-5.

[64] Thomas N. Kipf and Max Welling. *Semi-Supervised Classification with Graph Convolutional Networks*. arXiv:1609.02907 [cs, stat]. Feb. 2017. DOI: 10.48550/arXiv.1609.02907. URL: http://arxiv.org/abs/1609.02907 (visited on 11/08/2023).

[65] Michaël Defferrard, Xavier Bresson and Pierre Vandergheynst. *Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering*. arXiv:1606.09375 [cs, stat]. Feb. 2017. DOI: 10.48550/arXiv.1606.09375. URL: http://arxiv.org/abs/1606.09375 (visited on 11/08/2023).

[66] Felix Wu et al. 'Simplifying Graph Convolutional Networks'. In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, Sept. 2019, pp. 6861–6871. URL: https://proceedings.mlr.press/v97/wu19e.html.

[67] Petar Veličković et al. *Graph Attention Networks*. arXiv:1710.10903 [cs, stat]. Feb. 2018. DOI: 10.48550/arXiv.1710.10903. URL: http://arxiv.org/abs/1710.10903 (visited on 11/08/2023).

[68] Federico Monti et al. *Geometric deep learning on graphs and manifolds using mixture model CNNs*. arXiv:1611.08402 [cs]. Dec. 2016. DOI: 10.48550/arXiv.1611.08402. URL: http://arxiv.org/abs/1611.08402 (visited on 11/08/2023).

[69] Jiani Zhang et al. *GaAN: Gated Attention Networks for Learning on Large and Spatiotemporal Graphs*. arXiv:1803.07294 [cs]. Mar. 2018. DOI: 10.48550/arXiv.1803.07294. URL: http://arxiv.org/abs/1803.07294 (visited on 11/08/2023).

[70] Lingfei Wu et al., eds. *Graph Neural Networks: Foundations, Frontiers, and Applications*. en. Singapore: Springer Nature Singapore, 2022. ISBN: 9789811660535 9789811660542. DOI: 10.1007/978-981-16-6054-2. URL: https://link.springer.com/10.1007/978-981-16-6054-2 (visited on 03/07/2023).

[71] Lars Ruddigkeit et al. 'Enumeration of 166 Billion Organic Small Molecules in the Chemical Universe Database GDB-17'. In: *Journal of Chemical Information and Modeling* 52.11 (Nov. 2012). Publisher: American Chemical Society, pp. 2864–2875. ISSN: 1549-9596. DOI: 10.1021/ci300415d. URL: https://doi.org/10.1021/ci300415d (visited on 03/07/2023).

[72] *Quantum Machine*. URL: http://quantum-machine.org/datasets/ (visited on 16/06/2023).

[73] L. C. Blum and J.-L. Reymond. '970 Million Druglike Small Molecules for Virtual Screening in the Chemical Universe Database GDB-13'. In: *J. Am. Chem. Soc.* 131 (2009), p. 8732.

[74] M. Rupp et al. 'Fast and accurate modeling of molecular atomization energies with machine learning'. In: *Physical Review Letters* 108 (2012), p. 058301.

[75] Grégoire Montavon et al. 'Machine learning of molecular electronic properties in chemical compound space'. In: *New Journal of Physics* 15.9 (2013), p. 095003. URL: http://stacks.iop.org/1367-2630/15/i=9/a=095003.

[76] Raghunathan Ramakrishnan et al. 'Electronic spectra from TDDFT and machine learning in chemical space'. In: *The Journal of Chemical Physics* 143.8 (Aug. 2015), p. 084111. ISSN: 0021-9606. DOI: 10.1063/1.4928757. URL: https://doi.org/10.1063/1.4928757 (visited on 05/07/2023).

[77] Zhenqin Wu et al. 'MoleculeNet: a benchmark for molecular machine learning'. en. In: *Chemical Science* 9.2 (Jan. 2018). Publisher: The Royal Society of Chemistry, pp. 513–530. ISSN: 2041-6539. DOI: 10.1039/C7SC02664A. URL: https://pubs.rsc.org/en/content/articlelanding/2018/sc/c7sc02664a (visited on 16/06/2023).

[78] Axel D. Becke. 'Density-functional thermochemistry. III. The role of exact exchange'. In: *The Journal of Chemical Physics* 98.7 (Apr. 1993), pp. 5648–5652. ISSN: 0021-9606. DOI: 10.1063/1.464913. URL: https://doi.org/10.1063/1.464913 (visited on 11/08/2023).

[79] Chengteh Lee, Weitao Yang and Robert G. Parr. 'Development of the Colle-Salvetti correlation-energy formula into a functional of the electron density'. In: *Physical Review B* 37.2 (Jan. 1988). Publisher: American Physical Society, pp. 785–789. DOI: 10.1103/PhysRevB.37.785. URL: https://link.aps.org/doi/10.1103/PhysRevB.37.785 (visited on 11/08/2023).

[80] W. Kohn. 'Nobel Lecture: Electronic structure of matter—wave functions and density functionals'. In: *Reviews of Modern Physics* 71.5 (Oct. 1999). Publisher: American Physical Society, pp. 1253–1266. DOI: 10.1103/RevModPhys.71.1253. URL: https://link.aps.org/doi/10.1103/RevModPhys.71.1253 (visited on 11/08/2023).

[81] Felix A. Faber et al. 'Alchemical and structural distribution based representation for universal quantum machine learning'. In: *The Journal of Chemical Physics* 148.24 (Mar. 2018), p. 241717. ISSN: 0021-9606. DOI: 10.1063/1.5020710. URL: https://doi.org/10.1063/1.5020710 (visited on 06/07/2023).

[82] Richard D. Neidinger. 'Introduction to Automatic Differentiation and MATLAB Object-Oriented Programming'. In: *SIAM Review* 52.3 (Jan. 2010). Publisher: Society for Industrial and Applied Mathematics, pp. 545–563. ISSN: 0036-1445. DOI: 10.1137/080743627. URL: https://epubs-siam-org.ezproxy.uio.no/doi/10.1137/080743627 (visited on 12/07/2023).

[83] Atilim Gunes Baydin et al. 'Automatic Differentiation in Machine Learning: a Survey'. In: *Journal of Machine Learning Research* 18.153 (2018), pp. 1–43. ISSN: 1533-7928. URL: http://jmlr.org/papers/v18/17-468.html (visited on 12/07/2023).

[84] R. E. Wengert. 'A simple automatic derivative evaluation program'. In: *Communications of the ACM* 7.8 (Aug. 1964), pp. 463–464. ISSN: 0001-0782. DOI: 10.1145/355586.364791. URL: https://dl.acm.org/doi/10.1145/355586.364791 (visited on 13/07/2023).

[85] Seppo Linnainmaa. 'Taylor expansion of the accumulated rounding error'. en. In: *BIT Numerical Mathematics* 16.2 (June 1976), pp. 146–160. ISSN: 1572-9125. DOI: 10.1007/BF01931367. URL: https://doi.org/10.1007/BF01931367 (visited on 13/07/2023).

[86] Martin Simonovsky and Nikos Komodakis. *Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs*. arXiv:1704.02901 [cs]. Aug. 2017. DOI: 10.48550/arXiv.1704.02901. URL: http://arxiv.org/abs/1704.02901 (visited on 10/07/2023).

[87] A Davidov. *Improving semiempirical quantum chemistry with graph neural networks (Project code)*. https://github.com/aleksda/Thesis. 2023.

[88] Adam Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. arXiv:1912.01703 [cs, stat]. Dec. 2019. DOI: 10.48550/arXiv.1912.01703. URL: http://arxiv.org/abs/1912.01703 (visited on 14/07/2023).

[89] Martín Abadi et al. *TensorFlow: A system for large-scale machine learning*. arXiv:1605.08695 [cs]. May 2016. DOI: 10.48550/arXiv.1605.08695. URL: http://arxiv.org/abs/1605.08695 (visited on 14/07/2023).

[90] Roy Frostig, Matthew Johnson and Chris Leary. 'Compiling machine learning programs via high-level tracing'. In: 2018. URL: https://www.semanticscholar.org/paper/Compiling-machine-learning-programs-via-high-level-Frostig-Johnson/e399090e4dd760e8f09bd3d4de61b13b057c7740 (visited on 14/07/2023).

[91] Matthias Fey and Jan Eric Lenssen. *Fast Graph Representation Learning with PyTorch Geometric*. arXiv:1903.02428 [cs, stat]. Apr. 2019. DOI: 10.48550/arXiv.1903.02428. URL: http://arxiv.org/abs/1903.02428 (visited on 14/07/2023).

[92] Jeff Bezanson et al. *Julia: A Fast Dynamic Language for Technical Computing*. arXiv:1209.5145 [cs]. Sept. 2012. DOI: 10.48550/arXiv.1209.5145. URL: http://arxiv.org/abs/1209.5145 (visited on 14/07/2023).

[93] James J. P. Stewart. 'MOPAC: A semiempirical molecular orbital program'. en. In: *Journal of Computer-Aided Molecular Design* 4.1 (Mar. 1990), pp. 1–103. ISSN: 1573-4951. DOI: 10.1007/BF00128336. URL: https://doi.org/10.1007/BF00128336 (visited on 18/07/2023).

[94] Teresa Tamayo-Mendoza et al. 'Automatic Differentiation in Quantum Chemistry with Applications to Fully Variational Hartree–Fock'. In: *ACS Central Science* 4.5 (May 2018). Publisher: American Chemical Society, pp. 559–566. ISSN: 2374-7943. DOI: 10.1021/acscentsci.7b00586. URL: https://doi.org/10.1021/acscentsci.7b00586 (visited on 14/07/2023).

[95] Xing Zhang and Garnet Kin-Lic Chan. *Differentiable quantum chemistry with PySCF for molecules and materials at the mean-field level and beyond*. arXiv:2207.13836 [physics]. Aug. 2022. DOI: 10.48550/arXiv.2207.13836. URL: http://arxiv.org/abs/2207.13836 (visited on 13/07/2023).

[96] Muhammad F. Kasim, Susi Lehtola and Sam M. Vinko. 'DQC: a Python program package for Differentiable Quantum Chemistry'. In: *The Journal of Chemical Physics* 156.8 (Feb. 2022). arXiv:2110.11678 [physics], p. 084801. ISSN: 0021-9606, 1089-7690. DOI: 10.1063/5.0076202. URL: http://arxiv.org/abs/2110.11678 (visited on 14/07/2023).

[97] Guoqing Zhou et al. 'Graphics Processing Unit-Accelerated Semiempirical Born Oppenheimer Molecular Dynamics Using PyTorch'. en. In: *Journal of Chemical Theory and Computation* 16.8 (Aug. 2020), pp. 4951–4962. ISSN: 1549-9618, 1549-9626. DOI: 10.1021/acs.jctc.0c00243. URL: https://pubs.acs.org/doi/10.1021/acs.jctc.0c00243 (visited on 11/08/2023).

[98] William L. Hamilton, Rex Ying and Jure Leskovec. *Inductive Representation Learning on Large Graphs*. arXiv:1706.02216 [cs, stat]. Sept. 2018. DOI: 10.48550/arXiv.1706.02216. URL: http://arxiv.org/abs/1706.02216 (visited on 09/08/2023).

[99] Guohao Li et al. *DeeperGCN: All You Need to Train Deeper GCNs*. arXiv:2006.07739 [cs, stat]. June 2020. DOI: 10.48550/arXiv.2006.07739. URL: http://arxiv.org/abs/2006.07739 (visited on 09/08/2023).

[100] Keyulu Xu et al. *How Powerful are Graph Neural Networks?* arXiv:1810.00826 [cs, stat]. Feb. 2019. DOI: 10.48550/arXiv.1810.00826. URL: http://arxiv.org/abs/1810.00826 (visited on 09/08/2023).

[101] Gabriele Corso et al. *Principal Neighbourhood Aggregation for Graph Nets*. arXiv:2004.05718 [cs, stat]. Dec. 2020. DOI: 10.48550/arXiv.2004.05718. URL: http://arxiv.org/abs/2004.05718 (visited on 09/08/2023).

[102] Shyam A. Tailor et al. *Do We Need Anisotropic Graph Neural Networks?* arXiv:2104.01481 [cs]. May 2022. DOI: 10.48550/arXiv.2104.01481. URL: http://arxiv.org/abs/2104.01481 (visited on 09/08/2023).

[103] Oriol Vinyals, Samy Bengio and Manjunath Kudlur. *Order Matters: Sequence to sequence for sets*. arXiv:1511.06391 [cs, stat]. Feb. 2016. DOI: 10.48550/arXiv.1511.06391. URL: http://arxiv.org/abs/1511.06391 (visited on 09/08/2023).

[104] James J. P. Stewart. 'Optimization of parameters for semiempirical methods VI: more modifications to the NDDO approximations and re-optimization of parameters'. In: *Journal of Molecular Modeling* 19.1 (2013), pp. 1–32. ISSN: 1610-2940. DOI: 10.1007/s00894-012-1667-x. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3536963/ (visited on 19/06/2023).

[105] Pavlo O. Dral et al. 'Semiempirical Quantum-Chemical Orthogonalization-Corrected Methods: Theory, Implementation, and Parameters'. In: *Journal of Chemical Theory and Computation* 12.3 (Mar. 2016). Publisher: American Chemical Society, pp. 1082–1096. ISSN: 1549-9618. DOI: 10.1021/acs.jctc.5b01046. URL: https://doi.org/10.1021/acs.jctc.5b01046 (visited on 09/08/2023).

[106] Christoph Bannwarth et al. 'Extended tight-binding quantum chemistry methods'. en. In: *WIREs Computational Molecular Science* 11.2 (2021). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/wcms.1493, e1493. ISSN: 1759-0884. DOI: 10.1002/wcms.1493. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/wcms.1493 (visited on 09/08/2023).

[107] David Balcells and Bastian Bjerkem Skjelstad. 'tmQM Dataset—Quantum Geometries and Properties of 86k Transition Metal Complexes'. In: *Journal of Chemical Information and Modeling* 60.12 (Dec. 2020). Publisher: American Chemical Society, pp. 6135–6146. ISSN: 1549-9596. DOI: 10.1021/acs.jcim.0c01041. URL: https://doi.org/10.1021/acs.jcim.0c01041 (visited on 09/08/2023).

[108] Victor Garcia Satorras, Emiel Hoogeboom and Max Welling. *E(n) Equivariant Graph Neural Networks*. arXiv:2102.09844 [cs, stat]. Feb. 2022. DOI: 10.48550/arXiv.2102.09844. URL: http://arxiv.org/abs/2102.09844 (visited on 09/08/2023).

[109] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. arXiv:1502.03167 [cs]. Mar. 2015. DOI: 10.48550/arXiv.1502.03167. URL: http://arxiv.org/abs/1502.03167 (visited on 10/08/2023).

[110] Thom H. Dunning Jr. 'Gaussian basis sets for use in correlated molecular calculations. I. The atoms boron through neon and hydrogen'. In: *The Journal of Chemical Physics* 90.2 (Jan. 1989), pp. 1007–1023. ISSN: 0021-9606. DOI: 10.1063/1.456153. URL: https://doi.org/10.1063/1.456153 (visited on 10/08/2023).