POLITECHNIKA KRAKOWSKA
im. T. Kościuszki
Wydział Inżynierii Materiałowej i Fizyki

Katedra Fizyki

Kierunek studiów: .....Fizyka...Techniczna.....
Specjalność: .....modelowanie..komputerowe.....

STUDIA STACJONARNE

PRACA DYPLOMOWA

INŻYNIERSKA

Jakub Jurczak

Uczenie maszynowe w mechanice kwantowej

Machine Learning in Quantum Mechanics

Promotor:
Dr Radosław Kycia

Kraków, rok akad. ...2020..../.....2021....

List of Figures

Abstract

The main goal of the thesis is to show how deep learning, a subset of machine learning, can be used to predict quantities from quantum mechanics i.e. atomization energies by using specific representation of molecules known as Coulomb Matrices which can be used as the input data for feed-forward neural network.

Physical analysis of quantum mechanical properties gives plethora of information about the microscopic world but the standard methods of solving the Schrödinger equation have large computational complexity and require heavy numerical calculations to finish. On the other hand, experimental workings take too much time to set up and can be financially expensive. It would be highly beneficial to leverage modern power of artificial neural networks.

Machine learning has been more and more popular due to increase in computational power and very useful tools from statistical methods. It is natural then to use methods offered by ML to adapt them into problems in Physics that could result in optimal solutions to given tasks.

Abstrakt

Głównym celem pracy było ukazanie jak metody uczenia głębokiego, które są podzbiorem uczenia maszynowego, mogą być użyte do predykcji wielkości fizycznych z mechaniki kwantowej t.j. energii atomizacji, używając odpowiedniej reprezentacji molekuł jako macierzy Coulomba, które mogą być użyte jako dane wejściowe w głębokiej sieci neuronowej.

Analiza fizyczna wielkości kwantowo-mechanicznych jest w stanie ukazać ogrom informacji na temat świata mikroskopowego aczkolwiek standardowe metody rozwiązywania równania Schrödinger'a są złożone obliczeniowo i wymagają trudnych numerycznych metod w celu uzyskania poprawnych wyników. Z drugiej strony, podejście eksperymentalne zajmuje dużo czasu i może być również kosztowne finansowo. Byłoby wysoce efektywne wykorzystać nowoczesne metody sztucznych sieci neuronowych.

Uczenie maszynowe staje się ocraz bardziej popularne w związku ze zwiększeniem się mocy obliczeniowej komputerów oraz wsparciem ze strony metod statystycznych. Jest zatem zrozumiałe aby użyć metod uczenia maszynowego by zaadoptować je do problemów w fizyce, które mogą skutkować optymalnym rozwiązaniem rozważanego problemu.

Contents

0.1 Aim of the thesis

This thesis is focused on showing that using machine learning, we can build an AI-based regression model that can predict the quantum parameters of chemical compounds. We will focus on the prediction of the atomization energies, but the method is versatile and it can be used to predict other parameters (e.g molecular spectra). The author will check convergence and error estimates of the algorithm as well as compare real calculated values to the predicted ones showing how well the AI learns the energies from matrices.

0.2 Scope of the thesis

The thesis is organized as follows: The first part will describe the necessary theory for how we can use machine learning techniques to make a regression model that takes as an input the Coulomb's matrix of a molecule and predicts the atomization energy. A brief introduction to chemical compounds and some concepts from quantum mechanics will be provided. Then we will proceed to describe software tools that will come handy when we would like to build and implement a machine learning model. In the second part it will be shown how to actually build such regression model using a neural network with hidden layers in Python programming language. The necessary code will be provided as well as author's comments in a program's code.

0.3 Methodology

Keras - a library for machine learning in Python - will be used to achieve the results in training the AI. The whole implementation will be based on a dataset that contains Coulomb matrices of molecules as well as atomization energies of these molecules.

Part I

Theoretical part

Chapter 1

Introduction

Machine learning (often shortcutted to ML) is a concept that has emerged recently in mainstream media but has been in use long before modern supercomputers and information technology development. It is the study of application and science of algorithms that make sense of data. We live in very fast times, where billions of data are handled each day, it was necessary to build a set of tools that could give us answers to patterns in those large data chunks. Machine learning exists in our everyday lives, even if sometimes we don't necessarily see it immediately. Machine learning has a vast number of applications e.g. spam filtering, navigating, personal assistant with voice commands and chess engines.

We can actually be more precise and give more general definition [6]: 'Machine Learning is the field of study that gives computers the ability to learn without being explicitly programmed'. One can reduce the abstraction of the term to give a slightly practical definition of this (ML) science: 'A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.'[6]

It was not a big surprise that machine learning methods gained a lot of popularity in Physics, mainly because more and more physicists started to see that ML could help them solve some scientific problems. Physics itself is highly connected with statistical methods of learning starting from the field of Statistical Mechanics and ending on collecting experimental data. Machine learning methods have helped physicists to provide many answers by examining high-dimensional data in a different way that it has been done before. Physicists equipped with ML algorithms are able to extract information from large amounts of data that they generate in simulations or experiments. Then, they discover physical principles and underlying patterns in this data, so they can predict how the natural phenomena can be modelled and explained. [13]

This thesis is focused on showing that by using certain techniques from machine learning, we could build an AI-based regression model that will allow us to predict the atomization energies of molecules. The author will show and describe how a neural network - an object within the subfield of ML - can obtain useful results on quantum properties of molecules, specifically their atomization energy, using Coulomb's matrix. The method should predict other parameters e.g. molecular spectra.

Chapter 2

Quantum mechanics and chemical compounds

2.1 Quantum mechanics

When Isaac Newton prepared his first experiments to study the nature of light between the XVII-XVIII century, he concluded one thing that he took for certain: Light is made up of tiny particles called 'corpuscles' having negligible mass. Newton had so much authority that he completely overwhelmed Christiaan Huygens's wave approach to light. The consequence of Newton's domination was that Huygen's idea was not much popular and sometimes it was visibly negated. In the beginning of the XIX century, Augustin-Jean Fresnel and Thomas Young made their famous and groundbreaking experiments: diffraction and interference. Those two concepts definitely help to stop praising Newton's idea about light and scientific community started accepting wave description of light. Fresnel developed Huygens' theory and as a consequence, he helped to make wave theory of light more common. In the second half of XIX century, James Clerk Maxwell discovered that it is possible to derive the wave equation describing light from electromagnetism. Heinrich Hertz's experiments basically finished a fundamental studies of light. There was one concept that was really hard to comprehend for Physicists in the end of the XIX century. Wilhelm Wien in 1889 and Lord Rayleigh in 1900 tried to provide theoretical explanations of the blackbody spectrum. An idealized blackbody is a material object that absorbs all of the radiation falling on it. As it turned out, the Wien's formula gave correct predictions for high-frequency data but it miserably failed fitting for low frequencies. On the other hand, Rayleigh's attempt was correct except for low frequencies. Those results were so profound that history gave this phenomenon its own name: The ultraviolet catastrophe.

In the 1900, Max Planck gave corrections to theoretical model for black body radiator. He had to revolutionize the whole theory by introducing a concept of light quanta - photons. Thanks to light quanta postulate, the correct formula for a black body could be derived such that it was coherent with experimental data. In the 1905, Albert Einstein used Planck's concept of a photon (light corpuscle) to explain photoelectric effect which gave him a Nobel Prize. In 1924, Arthur Compton made his famous scattering experiment which proved that light can act as a stream of corpuscles. The idea of wave-particle duality was slowly beginning to born. Light in some cases behaves like electromagnetic wave with angular frequency ω and wave vector $\mathbf{k}$. But it also can behave differently when it interacts with matter, it behaves like a stream of particles, photons to be precise, and for an electromagnetic wave with frequency $\nu = \frac{\omega}{2\pi}$ and length $\lambda = \frac{c}{\nu}$ the following relations hold:

- E - energy of photon;
- h - Planck's constant, $6.62607015 \cdot 10^{-34} \cdot J \cdot (Hz)^{-1}$;
- ν - electromagnetic frequency;
- $\hbar$ - reduced Planck's constant;
- ω - angular frequency of electromagnetic wave;
- p - momentum of a particle (magnitude);
- k - wavenumber;
- π - mathematical constant;

$$E = h\nu = \hbar\omega, \tag{2.1}$$

$$p = \hbar k, \tag{2.2}$$

where $k = \frac{2\pi}{\lambda}$. Those relations are nothing but Louis de Broglie's hypothesis.[12].

The quantity of $\frac{h}{2\pi}$ appears in Quantum Mechanics frequently so we almost always work with $\hbar$ defined as follows:

$$\hbar = \frac{h}{2\pi}.$$

2.1.1 The Schrödinger equation

Every physical theory is based on a set of postulates, they are the nature's laws, they cannot be derived from other principles because they are indeed fundamental in their own nature. They can presented on a mathematical level as axioms e.g Newton's laws of motion and Maxwell's equations in Electrodynamics. Both of those sets of laws are the nature's laws. They have been verified by numerous experiments and daily usage.

In classical mechanics, physical state is described by generalized momenta and positions. In the case of only one particle, we have the position vector $\mathbf{x}_{clas}(t)$ and momentum vector $\mathbf{p}_{clas}(t)$. The generalized momenta and position vectors give us the definite trajectory of a particle. When we come to the microscopic world, every classical idea or law is not relevant to describe anything that happens on a microscale. But we can actually use de Broglie's hypothesis to deduce what is the behaviour of a particle in a microworld. De Broglie's hypothesis tells us that to every particle with energy E and momentum p we assign a wave with angular frequency $\omega = 2\pi\nu$ and wavenumber k that obeys relationships (2.1) and (2.2). The wavelength is related to wavenumber with:

$$\lambda = \frac{2\pi}{k} = \frac{2\pi\hbar}{p} = \frac{h}{p}. \tag{2.3}$$

Now focusing on a single particle and following de Broglie we accept the postulate below:

Postulate 1 *We postulate that the physical system containing one particle is fully described with so called wave function written as $\psi(\boldsymbol{r}, t)$. In other words, the state of a system is given by $\psi(\boldsymbol{r}, t)$.*

Let us briefly recall some statements about the wave function:

- The wave function can also be a function of other parameters. This relation is dependent on the type of physical system we want to describe;
- The wave function is a field, i.e., complex valued function of space ($\mathbf{r}$) and time (t);
- The wave function has to be normalizable, it describes the probability of finding a particle in some point in space. In the language of abstract vector spaces this means that $\psi(\mathbf{r}, t) \in \mathbf{L}^2$[30]. The $\mathbf{L}^2$ space implies that ψ has to obey normalization condition: $\int_{-\infty}^{\infty} |\psi(x,t)|^2 dx = 1$;

When we will accept the above postulate and its consequences, we can proceed to unravel the fundamental equation of non-relativistic Quantum Mechanics:

Postulate 2 *We postulate that the wave function $\psi(\boldsymbol{r}, t)$ obeys the following equation:*

$$i\hbar \frac{\partial}{\partial t}\psi(\boldsymbol{r}, t) = -\frac{\hbar^2}{2m}\nabla^2\psi(\boldsymbol{r}, t) + V(\boldsymbol{r}, t)\psi(\boldsymbol{r}, t). \tag{2.4}$$

The above equation is credited to Erwin Schrödinger. We construct the following operator:

$$\hat{H} = -\frac{\hbar^2}{2m}\nabla^2 + V(\mathbf{r}, t), \tag{2.5}$$

where $\hat{H}$ is Hamilton's operator also known as Hamiltonian. It is produced by substituting in the classical hamiltonian momentum with momentum operator, namely:

$$-i\hbar\nabla \rightarrow \overrightarrow{\mathbf{p}}. \tag{2.6}$$

Then, we can write Schrödinger equation in a more compact form[12]:

$$i\hbar \frac{\partial}{\partial t}\psi(\mathbf{r}, t) = \hat{H}\psi(\mathbf{r}, t). \tag{2.7}$$

Just as mentioned, the above equation allows us to find any quantum physical state (assuming non-relativistic limit). It is not a surprise that this equation (to be precise: Schrödinger equation for multi-particle systems) carries a great value to atomic structures and grant us knowledge about energies of molecules. If we are examining states for which the phase dependency is presented as $\exp[iEt]$ and we substitute this formula to (2.5), then we obtain the equation for eigenvalues of a Hamiltonian i.e E, which we interpret as the energies of a system.

2.2 Chemical compounds

The atoms - in all substances that contain multiple atoms - are held together by electromagnetic interactions between protons and electrons. Electromagnetic interaction can be repulsive or attractive depending on the type of charged species. If the species are charged oppositely (meaning that there are positive and negative charges), electromagnetic attraction results in a force that causes the species to attract each other. Conversely, electromagnetic interaction between two species with the same charge results in electromagnetic repulsion. Atoms form chemical compounds when the attractive electromagnetic interactions between them are stronger than the repulsive interactions.

2.2.1 Coulomb matrix

Coulomb matrices (CM) were first proposed in [23] as a simple global descriptor of molecules. We can treat CM as mimicking the electromagnetic interaction between nuclei. To calculate Coulomb matrix for some molecule we use the formula below:

- CM_{ij} - (i, j) element of a Coulomb matrix;
- Z_i - nuclear charge of ith atom;
- R_{ij} represents the distance between atom i and j;

$$CM_{ij} = \begin{cases} 0.5 \cdot Z_i^{2.4} & \text{for } i = j, \\ \frac{Z_i Z_j}{R_{ij}} & \text{for } i \neq j. \end{cases} \tag{2.8}$$

The diagonal entries are essentially interactions of an atom with itself, they are a polynomial fit of the atomic energies to the nuclear charge Z_i. To visualize this concept, we present the Coulomb Matrix of methanol below read from [2]:

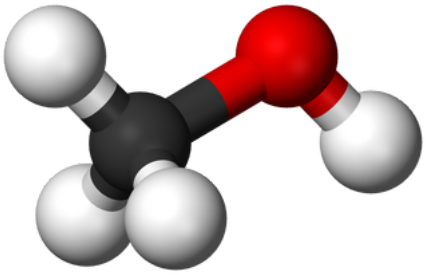

Figure 2.1: Ball and stick model of methanol [2].

$$\begin{pmatrix} 36.9 & 33.7 & 5.5 & 3.1 & 5.5 & 5.5 \\ 33.7 & 73.5 & 4.0 & 8.2 & 3.8 & 3.8 \\ 5.5 & 4.0 & 0.5 & 0.35 & 0.56 & 0.56 \\ 3.1 & 8.2 & 0.35 & 0.5 & 0.43 & 0.43 \\ 5.5 & 3.8 & 0.56 & 0.43 & 0.5 & 0.56 \\ 5.5 & 3.8 & 0.56 & 0.43 & 0.56 & 0.5 \end{pmatrix} \tag{2.9}$$

As we know, the building atoms of methanol are four atoms of hydrogen, one carbon atom and one oxygen atom. The Coulomb matrix is invariant with respect to translation, rotation and permutation. In the matrix above, the first row represents carbon that interacts with all the other atoms. The second row corresponds to oxygen, and the remaining rows correspond to hydrogen. The matrix is perfectly symmetric, meaning that every corresponding row is equal to every corresponding column.

2.2.2 Atomization energy and enthalpy of atomization

Atomization energy is the quantity that accompanies the total separation of all atoms in a chemical substance (either a chemical compound or a chemical element). It is related to a physico-chemical parameter known as enthalpy of atomization which is the enthalpy change of the total seperation of the atoms. It is often represented by the symbol ΔH_{at}. The associated

standard enthalpy is known as standard enthalpy of atomization in units of $\frac{kJ}{mol}$ at 289 K and 1 atm of pressure.

Below is the short description of how one can calculate the atomization energy of a given molecule:

- Calculate the total energy for the isolated atoms of the given molecule;
- Calculate the total molecule energy;
- Subtract the energies of the atoms from the total molecule energy to obtain the atomization energy.

Chapter 3

Machine learning and its principle of working

3.1 Artificial intelligence

In 1950, Alan Turing started wondering: "Can a machine think like a human?". He laid out the fundamental idea and redefined a question for it, to make it much easier to be precise in a language of machines, namely:"Can a machine imitate and convince other human being that it is also a human?". This question has become a really important idea known as Turing's Test. A human judge asks questions to a person or a machine in the next room. If the machine convinces the judge that it is indeed the human, the machine passes the test. In 1943, Warren McCulloch and Walter Pitts studied how the biological brain works, they proposed the first concept of a simplified brain cell, McCulloch-Pitts (MCP) neuron. Biological neurons create a network of interconnected nerve cells and they use them to send electrical and chemical signals, so there is a clear input and a binary output for those signals [15]. The AI research officially started in 1956 at the Dartmouth Conference where the term 'artificial intelligence' was created. It was necessary to combine efforts in cybernetics, automata theory, and complex information processing in order to study how machines think [7].

In 1958, some breakthrough was made by Frank Rosenplatt. He published the first concept of the perceptron learning rule based on the MCP neuron model. His proposition of an algorithm was simple, the weight coefficients are multiplied by features and the decision is made, the neuron either fires or it does not [19]. This gave the birth of study about neural networks.

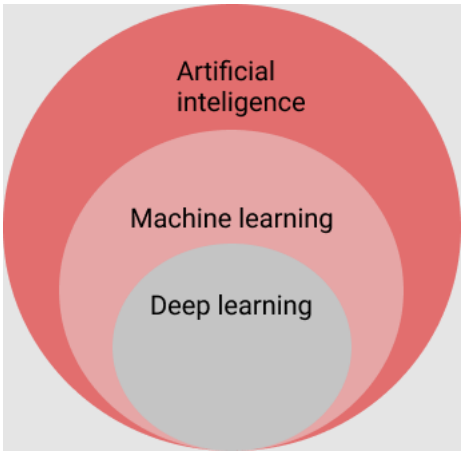

Figure 3.1: AI, machine learning and deep learning as a set.

3.2 Machine learning

Algorithms that are trained in machine learning have a major goal: put the data into machine learning algorithm and train it until it starts to see connection between features and it learns to group them among themselves. There are many possible algorithms that could give accurate predictions, but we are interested in the one that will give the best predictions. That is, why we have to use validation methods to measure how well given ML algorithm performs. This way, we can compare different algorithms that we consider for a given problem and then choose the one that is optimal for our task. Nowadays, it is hard to not experience machine learning around us every day. Powerful recommendation systems are built using ML models when we search for a song that seems interesting to us. Voice assistants can make a call or schedule a meeting if we tell them to do so. Vacuum cleaners learn all the paths to become optimized for cleaning. It is harder and harder for spam to land on our e-mail box. Medicians use image analysis systems to make a precise diagnosis. Not to mention the rise of self-driving cars which can transport us from point A to point B without the need to manually steer the vehicle.

3.3 Types of machine learning

This section will provide the description of four fundamental types of machine learning. When faced with problem, one type might give better results than the others, thus it is important to summarise when we can apply each technique.

3.3.1 Supervised machine learning

Supervised machine learning concentrates on labeled data which are known before we even start to train the algorithm. We try to predict the unseen or future data by feeding known labels to the algorithm. A popular example is how email spam filtering works. If we have data consisting of two labels: a spam message or non-spam message, we can train the algorithm on that data to predict the future spam message and classify them accordingly so our experience of handling e-mails will be much better [19][6].

3.3.2 Unsupervised machine learning

Unsupervised learning focuses on unlabeled data or data with unknown structure. We could take a large pile of data and organize them into subgroups also called clusters. We could study and analyse these and find groups that share some similarity between each other and the ones that are more dissimilar with respect to other groups. Then, we could structure the data and discover new patterns and labels through grouping similar data [19][6].

3.3.3 Reinforcement machine learning

Reinforcement learning is based on the system (an agent) that acts with the environment and is seeking a reward signal. It is similar to supervised learning, but the system does not learn from labels or values, it rather constantly searches for an action that would produce the greatest value by a reward function. Through trial and error, agent learns how it should do a given task to maximize its reward [19][6].

3.3.4 Deep learning

Deep learning is a subset of machine learning (all deep learning is machine learning, yet not all machine learning is deep learning). Deep learning algorithms use an artificial neural network that is designed to learn the way the human brain learns. Deep learning models use data that pass through multiple neurons, then the weights and biases are applied in each successive layer to continually adjust and improve the outcomes [5].

3.4 Machine learning pipeline

In order to construct a machine learning model these are typical steps to be followed [9]:

Selection and preprocessing of a dataset

Machine learning models typically learn on two kinds of dataset that is split before the learning takes place: the training set and the testing set. The training set is necessary in order for a model to learn from its mistakes. The test set is used to see how well the model predicts values of interest previously unseen. The test set offers a clear message when the learning is done, based on the score on the test set we can either state that the given model handles the predition correctly or it does not work for a given problem. Depending on the dataset, it might be labeled, meaning that it has 'tags' which combined with certain features, could give us solid outputs. Other times, the data is unlabeled, meaning that the model will have to extract the hidden patterns from the features itself.

Categorical and numerical data

When we are dealing with the data, there is a distinction between certain types of the data. The distinction can be summarized as:

- Numerical data
  - Continuous numerical data;
  - Discrete numerical data;
- Categorical data

- Ordinal features;
- Nominal features;

The numerical data is data points with exact numbers. It could be subdivided into two forms, continuous: such like weight, temperature, salary etc. and discrete: number of students, number of languages spoken. On the other hand, there is a categorical data which represents characteristics such as a hockey player's position, team, hometown. We can further distinguish the categorical data to ordinal and nominal features: Ordinal features can be sorted or ordered e.g t-shirt sizes, in contrast, nominal features do not imply any order e.g t-shirt color [19][28].

Feature scaling

Many of the numerical features are unbalanced, for example, there might be times when we will have to process data with different units such as length in kilometers and mass in kilograms. If we put the data as it is, we risk the high bias from difference in units. We can use feature scaling methods to give our data the properties of a standard normal distribution that is: mean that is equal to zero and unit variance. Standardization shifts the mean of each feature so that it is centered at zero and each feature has a standard deviation of 1 (unit variance). For instance, to standardize the j_{th} feature, we can simply subtract the sample mean, μ_j from every training example, and make a division by standard deviation σ_j. Mathematically it is presented as follows:

$$\mathbf{x}'_j = \frac{\mathbf{x}_j - \mu_j}{\sigma_j}. \tag{3.1}$$

In the equation above $\mathbf{x}_j$ is a vector with j-th feature values and a standardization is applied to each j-th feature.

Choosing the right algorithm

The choice of an algorithm that is going to learn depends on a problem. If we have labeled data, then we can proceed to more popular algorithms such as [19]:

- Regression algorithms: Linear and logistic regression are examples of regression algorithms used to understand relationships in data. Linear regression is used to predict the value of a dependent variable based on the value of an independent variable. Logistic regression can be used when the dependent variable is binary in nature: 1 or 0.
- Decision trees: Making a classification based on set of decisions rules. Its advantage comes from a fact that it does not require features to be scaled beforehand.
- Instance-based algorithms: Such as k-nearest neighbours, a typical 'lazy learner', it does not learn a discriminative function from the training data but memorizes the training dataset instead.

When we have to deal with unlabeled data, we could use for example:

- Clustering algorithms: Clustering focuses on identifying groups of similar records and labeling the records according to the group to which they belong. This is done without prior knowledge about the groups and their characteristics using e.g k-means algorithm
- Association algorithms: Association algorithms find patterns and relationships in data and identify frequent 'if-then' relationships called association rules.

- Neural networks: A deep neural network defines a network with multiple hidden layers, each of which successively refines the results of the previous layer.

Metric

We need a tool to compare our results with our expectations from the data. That is why we use different metrics to compare output from different models to be able to find optimized algorithm for given task. Most popular metrics are mean absolute error (MAE)

$$MAE = \frac{1}{N}\sum_{j=1}^{N}|y_j - \hat{y}_j|, \tag{3.2}$$

and often mean squared error (MSE):

$$MSE = \frac{1}{N}\sum_{j=1}^{N}(y_j - \hat{y}_j)^2, \tag{3.3}$$

where y_j represents the original value and $\hat{y}_j$ represents prediction by a machine learning algorithm. It is worth noting that interpretation of MSE can be influenced by data scaling and the dataset used for learning. It is therefore sometimes advised to use the coefficient of determination (R^2) that is defined as [19]:

$$R^2 = 1 - \frac{MSE}{Var(y)}, \tag{3.4}$$

where $Var(y)$ is a variance of the target variable y.

Training the algorithms

Once all the data is prepared and methodology is chosen, we put variables with the data to the input. Through the iterations, an algorithm is learning to adjust the weights in order to make the best prediction. Once the algorithm finishes the state of learning, it becomes necessary to start validation process - using the test set - in order to check how well the model has learnt to predict the data.

3.4.1 Model usage and its improvement

After the training iterations are done, we could test our trained model on a subset of data that we have put to the test set. If we see that our predictions have a large error, we have to optimize parameters again and adjust the model again. Sometimes this step means to go back to the first step and set new parameters for learning. Below figure can briefly summarise the whole process:

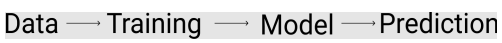

Figure 3.2: Example of a machine learning pipeline [21].

3.5 Artificial neural networks

Artificial neural networks are built on top of artificial neurons. They are inspired by the network of biological neurons found in our brains. Deep learning is a science that investigates various constructions of artificial neural networks. They vary from one model to the other, they are efficacious and can be scaled to face hard machine learning problems such as classifying a large dataset of images (we can think of something like Google Images), controlling speech recognition services (Google Assistant, Apple's Siri, Amazon Alexa), recommending user his favourite videos on YouTube platform or beating an opponent in a chess game.[6][3]. It is valuable to stop for a second and to really think why the interest in artificial neural networks will have a huge impact in the nearby future:

- We live in times of vast data coming from different sources, it is proven in an AI field that ANNs often outmatch other machine learning techniques when it comes to complex problems;
- Thanks to Moore's law (the number of transistors on an affordable CPU would double every two years [10]) training of ANN has become achievable due to increased computers' performance. The gaming industry pressurizes the hardware market to be in a state of constant high-end solutions, what is more, cloud computing allowed many users to experiment on ANNs without the worries about their PC components;
- The training algorithms have been reinforced. They are similar to the previous ones from 90s, but the reinforcements have made a great impact;
- With the increasing knowledge of people about ANNs, there seems to be an increase in research funding, this implies steady progress, thus we shall see new revolutionary products;

ANNs could also be viewed as units that are placed in a particular way in a series of layers, each neuron is connected in the network, and together they compose a system that we know as Artificial Neural Network. Neural networks are commonly composed with input layer, output layer and some hidden layers. The input layer is responsible for loading the data from a dataset that is of our interest. After the data goes through the input, there comes in play hidden layers. It is actually up to the creator of the network to say how many hidden layers will the network have. The hidden layers transform the data for a valuable output for the output layer. Finally, the output layer gives us a response signal for the data that has been put in the input.

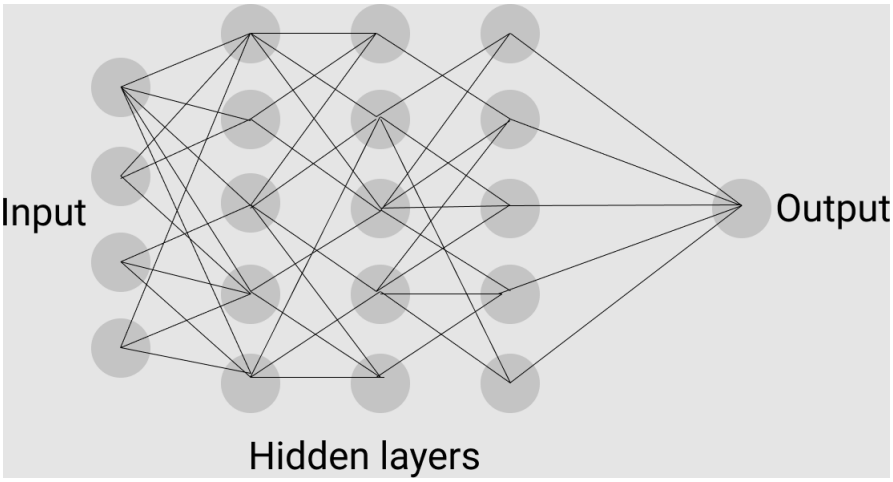

Figure 3.3: Example of a neural network structure [24].

3.6 Single layer neural network

3.6.1 Mathematical principles of Perceptron

We can put a concept of artificial neuron as a simple binary classification task. The first class is 1, and the second -1.We define a decision function $\phi(z)$ that takes a linear combination of input values $\mathbf{x}$ and a weight vector $\mathbf{w}$ and it is written as a scalar product of $\mathbf{x}$ and $\mathbf{w}$, namely:

$$z = \mathbf{w}^T\mathbf{x},$$

A decision function makes its prediction using a threshold θ:

$$\phi(z) = \begin{cases} 1 & \text{if } z \geq \theta, \\ -1 & \text{otherwise.} \end{cases} \tag{3.5}$$

If we rearrange the terms and define a weight-zero as $w_0 = -\theta$ and $x_0 = 1$ we can write z as:

$$z = w_0x_0 + w_1x_1 + ... + w_mx_m = \mathbf{w}^T\mathbf{x}.$$

After this operation we write $\phi(z)$ as:

$$\phi(z) = \begin{cases} 1 & \text{if } z \geq 0, \\ -1 & \text{otherwise.} \end{cases} \tag{3.6}$$

The negative threshold is often called the bias unit[19].

3.6.2 Learning rule of the perceptron

The key principle of how perceptron learns from the data is the update of weights. The simultaneous update of each weight w_j, in the weight vector $\mathbf{w}$, can be written as [19]:

$$w_j := w_j + \Delta w_j, \tag{3.7}$$

where the change in w_j is calculated as:

$$\Delta w_j = \eta(y^{(i)} - \hat{y}^{(i)})x_j^{(i)}, \tag{3.8}$$

where η is called the learning rate, $y^{(i)}$ is a true class label and $\hat{y}^{(i)}$ is the class predicted by perceptron.

The whole idea of a learning rule for perceptron may be summarised in a simple manner:

1. First, generate weights randomly;
2. For every training example $\mathbf{x}^{(i)}$:
   - Compute the output value $\hat{y}$;
   - Update the weights according to update rule (3.8);

3.6.3 ADAline

To understand the concept of multilayer neural networks, let us begin with simple single layer neural network to show from first principles how a neural network behaves on its fundamental level. One can treat Perceptron as a single layer neural network. In this section we will extend the notion of Perceptron, an improvement called Adaline (ADAptive LInearNEuron) published by Bernard Widrow and his doctoral student Tedd Hoff [29].

The main difference between Rosenblatt's perceptron learning rule and Adaline is the update of weights. The weights are updated using a linear function also called activation function (crucial concept for deep learning methods) instead of being updated by a unit step function. The activation function in Adaline is written as $\phi(z)$ which is just:

$$\phi(z) = \phi(\mathbf{w}^T\mathbf{X}) = \mathbf{w}^T\mathbf{X}. \tag{3.9}$$

The threshold function is still used to predict the final outcome (class) as is being done in the Perceptron learning rule (3.6)

3.7 The Multilayer Perceptron and Backpropagation

A multilayer perceptron is created with one input layer, one or more layers of threshold logic unit (a logic unit that computes a weighted sum of its inputs z and then applies a step function to that sum to make a prediction) called hidden layers, and one end layer called the output layer. If an ANN is built with a deep stack of hidden layers, it is called a deep neural network (DNN). It was a problem for many researchers to find a way to train multilayer perceptron. In 1986, Rumelhart, Hinton and Ronald published their paper [22], where they introduced the concept of backpropagation training algorithm which is nowadays used on a regular basis. It is the combination of Gradient Descent Method and a concept of automatic differentiation - automatical computing of gradients.

3.7.1 Gradient Descent

The main problem for a machine learning algorithm is a defined objective function which we want to optimize while the algorithm is training. This objective function is often a cost function subject to minimization procedure. Let us define the cost function J, to learn the weights as the sum of squared errors (SSE):

$$J(\mathbf{w}) = \frac{1}{2}\sum_i (y^i - \phi(z^{(i)}))^2. \tag{3.10}$$

In comparison to the single layer perceptron, this cost function is differentiable and it is convex. All these implies that we could use a powerful algorithm called gradient descent that we conceptually visualize, in figure 3.4:

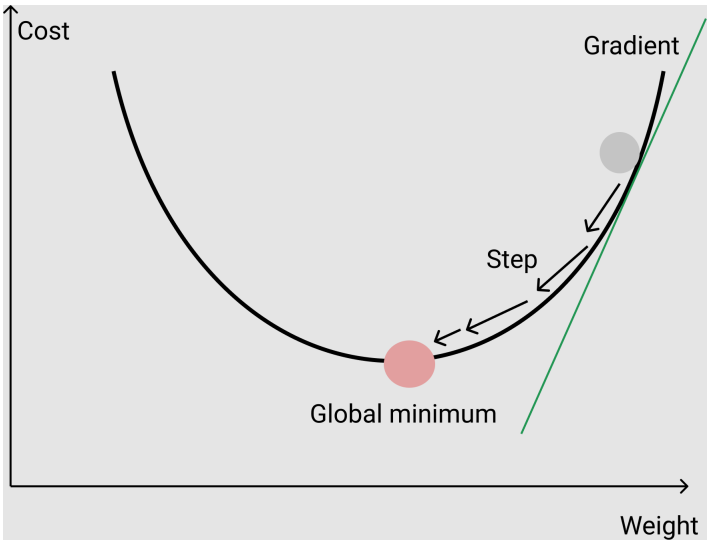

Figure 3.4: Gradient descent [4].

After each iteration, we want to go in the direction of descent pointed out by the gradient of J, where the step size is coming from the learning rate and the slope of the function. The rule of weight update is:

$$\mathbf{w} := \mathbf{w} + \Delta\mathbf{w}, \tag{3.11}$$

where:

$$\Delta\mathbf{w} = -\eta\nabla J(\mathbf{w}). \tag{3.12}$$

Now let us quickly derive the gradient descent for SSE [19]:

$$\begin{aligned}
\nabla J(\mathbf{w}) = \frac{\partial J}{\partial w_j} &= \frac{\partial}{\partial w_j}\frac{1}{2}\sum_i (y^{(i)} - \phi(z^{(i)}))^2 \\
&= \frac{1}{2}\frac{\partial}{\partial w_j}\sum_i (y^{(i)} - \phi(z^{(i)}))^2 \\
&= \frac{1}{2}\sum_i 2(y^{(i)} - \phi(z^{(i)}))\frac{\partial}{\partial w_j}(y^{(i)} - \phi(z^{(i)})) \\
&= \sum_i \left(y^{(i)} - \phi(z^{(i)})\right)\frac{\partial}{\partial w_j}\left(y^{(i)} - \sum_i (w_j^{(i)} x_j^{(i)})\right) \\
&= -\sum_i (y^{(i)} - \phi(z^{(i)}) \cdot x_j^{(i)}.
\end{aligned} \tag{3.13}$$

3.7.2 Stochastic Gradient Descent

If we have a large piles of data, it is hard to use Gradient Descent Method, especially because it can be computationally heavy to reevaluate the whole training dataset each time we take a step toward the global minimum.

There is a popular alternative for GD, Stochastic Gradient Descent, sometimes also called iterative or online gradient descent. The difference is this: Instead of updating the weights based on the sum of the accumulated errors over all training examples, $\mathbf{x}^{(i)}$:

$$\Delta \mathbf{w} = \eta \sum_i (y^{(i)} - \phi(z^{(i)}) \cdot \mathbf{x}^{(i)}, \tag{3.14}$$

we just update the weights for each example:

$$\eta(y^{(i)} - \phi(z^{(i)}) \cdot \mathbf{x}^{(i)}. \tag{3.15}$$

The strength of stochastic gradient descent is reaching convergence much faster than the 'regular' gradient descent, after all, we update the weights more frequently. Each gradient is evaluated on a single example, therefore, the error surface is noisier than in GD which has its advantage when it comes to escaping local minima. It is worth mentioning that to get accurate results one must shuffle the dataset for every iteration to prevent cycles and one must present the data in a random order. [19]

3.7.3 Backpropagation

Now, we can proceed to the heart of the ANNs - backpropagation algorithm. To sum it up - it uses Gradient Descent with automatic differentiation, meaning that: there are two phases of data flow (two passes through the network one forward and one backward). The backpropagation computes the gradient of an error with regard to every single model parameter. Then it performs a Gradient Descent and the whole process is continued until the network converges. Below, we list the process of backpropagation:

- The gradient descent goes over all the training set multiple times. Each pass is called an epoch;
- Each mini-batch is sent to the input layer and then to the first hidden layer. Next, there is a computation of neurons in this layer, the result is transported to another layer, the output is computed, and then it is sent to another layer and so on until the output layer is finally reached. This is actually a process of making predictions, but intermediate results are upheld because they will be needed for the backward pass;
- After forward pass, the algorithm measures how far from prediction it was, the loss function compares predicted values from the network with actual values and then it returns some measure of the error;
- Then, the chain rule is used to estimate how much each output connection contributed to the error;
- The algorithm works backward using chain rule again to estimate how much of error contributions came from connection in the layer below (backpropagation);
- On the last step, the algorithm uses Gradient Descent to make corrections to weights, using the gradient that it has calculated previously [6];

3.8 Activation function

Activation functions are used to decide if a neuron should be activated or not. Essentially, they transform input signals to output signals. Below, are presented common activation functions used heavily in the industry as well as in research:

- One of the more popular activation function is ReLU (rectified linear unit). Given an element x, the function is defined as the maximum of that element and 0, $ReLU(x) = \max(x, 0)$;
- The sigmoid function $f(x) = \frac{1}{1+\exp(-x)}$ is defined as $f : \mathbb{R} \to (0, 1)$. It is commonly used in a more classic machine learning algorithms such as logistic regression;
- Similar to sigmoid activation function, the $\tanh(x)$ function exhibits point symmetry about the origin of the coordinate system;

Chapter 4

Software

4.1 Software tools

The purpose of this section is to show what types of tools where used when writing the implementation of a theory. First, the author describe Python programming language and its momentum in the world of science, and then some details about libraries that were used in a code will be given.

4.1.1 Python

Python [18] is a programming language created in 1991 by Guido van Rossum. It is not a compiled language, rather it has an interpreter that helps it execute instructions without a need for previous compiling into a machine-language instructions. It is a multi-paradigm language, meaning that it does not make a programmer use a specified programming paradigm, one can actually focus on solving a given problem instead of worrying about certain paradigm's rules. Since Python is object-oriented, it organizes its structure not by using logic and functions but it treats everything as an object. An object can be defined as a data field that has unique attributes and behavior. The strength of a Python language comes from the fact that it has a really easy syntax making a point of readability and a large number of general and highly specialized libraries. A natural implication from this fact stems that one can focus on writing code efficiently and quickly in comparison to other programming languages. It is not surprising that Python gained a lot of popularity in the scientific world. A scientist does not have to worry about some unpleasant things when creating their software e.g manual memory management, they just strictly start to implement a given problem. The community of Python constantly tries to improve the language and serves its support for new users.

4.1.2 NumPy

NumPy [16] is a library made for scientific computing in Python. It provides a multidimensional array object, a special type of matrix objects and variety of different implementations of fast operations on arrays like linear algebra methods, discrete Fourier Transforms, statistical operations, random simulations and more. At the top of hierarchy in NumPy library is object called *ndarray*. This encapsulates n-dimensional arrays of homogeneous data types, with many operations being performed in compiled code for performance. The strength of NumPy comes from a fact that it does all of its operations on vectors 'behind the scenes' using C programming language. A vectorized code has its advantages such that:

- It is easier to read;

- It is possible to achieve same outcomes with fewer lines of code so there is less probability for a mistake;
- It maintains reasonable computational complexity by not using loops;

4.1.3 Keras

Keras [11] is an API (Application Programming Interface) written in Python for a much popular machine learning platform - TensorFlow [27]. It gives a kind of interface for TensorFlow to quickly and accurately implement ideas and machine learning solutions. It was created with a concept of fast experimentations on data in opposite to TensorFlow which gives much more power to a user in developing highly efficient algorithms. We might think of Keras as a higher layer of TensorFlow: It gives us specific tools and options to quickly create and develop a model using exceptionally fast and powerful iterations of automatic differentiation. Keras can be run on TPU (Tensor Processing Unit) or on large clusters of GPUs (Graphics Processing Unit) and it supports exports to run models in a browser or on a mobile device. The structure of Keras is divided between layers and models. The simplest type of model is the Sequential model, a linear stack of layers.

4.1.4 TensorFlow

TensorFlow [27] is an open source library for developing and training machine learning models. It was created by Google company using Python as an API for execution code written in C++ for maximum performance. It allows developers to create graph structure, where data flows through graphs and each graph represents a mathematical operation. The graphs are connected by a multidimensional arrays which are treated as a tensor. All the values in this tensor hold identical data type with a known shape (dimensions). The power of TensorFlow comes from its pipeline when solving a given problem using machine learning, meaning:

- Preprocessing of the data;
- Building the model;
- Training and estimating the model;

Finally, a significant feature of TensorFlow is the TensorBoard. The TensorBoard enables to monitor graphically and visually what TensorFlow is doing.

4.1.5 Scikit-learn

Scikit-learn [25] is an open source machine learning library for Python programming language as well as a simple yet powerful data analysis tool. It was created as a layer between NumPy, Matplotlib and SciPy focusing on classical ML methods rather than reinforcement learning. It contains many tools helpful in solving regression, classification, clustering and dimensionality reduction problems. The project was started in 2007 as a Google Summer of Code project by David Cournapeau. Later that year, Matthieu Brucher started working on this project as a part of his thesis. Scikit-learn has a nice infrastructure in composing the pipeline of project, it has many built-in machine learning algorithms and models called estimators; each estimator can be fitted to some data using its fit method. There is also a place for transformers, objects used for preprocessing the data before they go to a predictor. Together, transformers and estimators give nicely shaped structure for producing a fully developed model in just a few lines of code.

4.1.6 SciPy

The SciPy [26] library is one of the core packages that make up the SciPy stack. It provides many user-friendly and efficient numerical routines, such as routines for numerical integration, interpolation, optimization, linear algebra, and statistics. It creates an ecosystem for making scientific calculations easier and it splits between four different packages:

- Python;
- NumPy;
- SciPy library;
- Matplotlib;

4.1.7 Matplotlib

Matplotlib [14] [8] is a library for making fully customized plots in Python either 2D or 3D. Although historically the library was written purely in Python, nowadays it is fully equiped with NumPy vectorized arrays. This combination assures that bars, plots, charts, animations, etc. can always be handled despite of relatively large data chunks. In just a few lines of code, one can simply code a clear and concise plots or they may experiment with richness of modules, functions and classes to present fancy architectures. The architecture of Matplotlib is roughly divided in three parts:

- the pylab interface is the set of functions provided by matplotlib.pylab which allow the user to create plots;
- the set of classes that do the heavy lifting, creating and managing figures, text, lines, plots and so on;
- the backends are device-dependent drawing devices, aka renderers, that transform the frontend representation to hardcopy or a display device;

4.1.8 Pandas

Pandas [17] is an open source data analysis and manipulation tool, built on top of the Python programming language. It has been in development since 2008 and is *de facto* standard in industry with a strong position to be one of the best managed open source projects. Pandas has everything a data scientist needs in their job. Pandas comes with a DataFrame object that allows a user to manipulate data with integrated indexing. It has an in-built data structures like CSV and text files, Microsoft Excel, SQL databases, and the fast HDF5 format for saving or reading data. If there is a necessity for an optimized, fast and efficient code, Pandas uses critical code paths written in Cython or C. Python with pandas is used in a wide variety of academic and commercial domains, including Finance, Neuroscience, Economics, Statistics, Advertising, Web Analytics, and more. Pandas aims to be the fundamental high-level building block for doing practical, real world data analysis in Python.

Part II

Applications

Chapter 5

Implementation of the ML algorithm

5.1 Dataset

The dataset that has been used for implementation of a network was taken from [20].The whole credit of using this data goes to [1] [23]. It is a subset of a much broader data set, namely GDB-13 which contains close to one billion stable and synthetically accessible organic molecules. It is important to note that QM7 dataset has molecules that do not exceed 23 atoms in accordance to their structure. The whole dataset has 7165 molecules with their Coulomb's matrices calculated beforehand. The atomization energies are in SI units of kcal/mol in the range between -2000 to -800 kcal/mol.The structure of QM7 is as follows:

- Array X: It represents the molecules and their respective Coulomb's matrices having the shape (7165 x 23 x 23), this array is treated as an input for neural network;
- Array T: It represents the atomization energies of the molecules having the shape (7165);

5.2 Google colaboratory platform

Since deep learning methods can be highly sensitive to hardware of a computational unit, the author will use Google Colaboratory (Colab), the free platform for typing and executing Python code along with necessary libraries. Using Colab will have an advantage of being portable, one can open a web browser and does not have to worry about performance speed and computing resources. Strengths of Colab come from those facts:

- It has pre-configured environment so a user does not have to manually preset every setting connected to Python ecosystem;
- It gives free (up to 12 hours) access to using GPU (Graphical Processing Unit) so a user does not have to have a PC that comes from high-end technology;
- It is portable, the whole code can be saved and downloaded as .ipynb file, so it can be run later on any device with Python on the board;

Below we can see how the code can be compiled using Google environment:

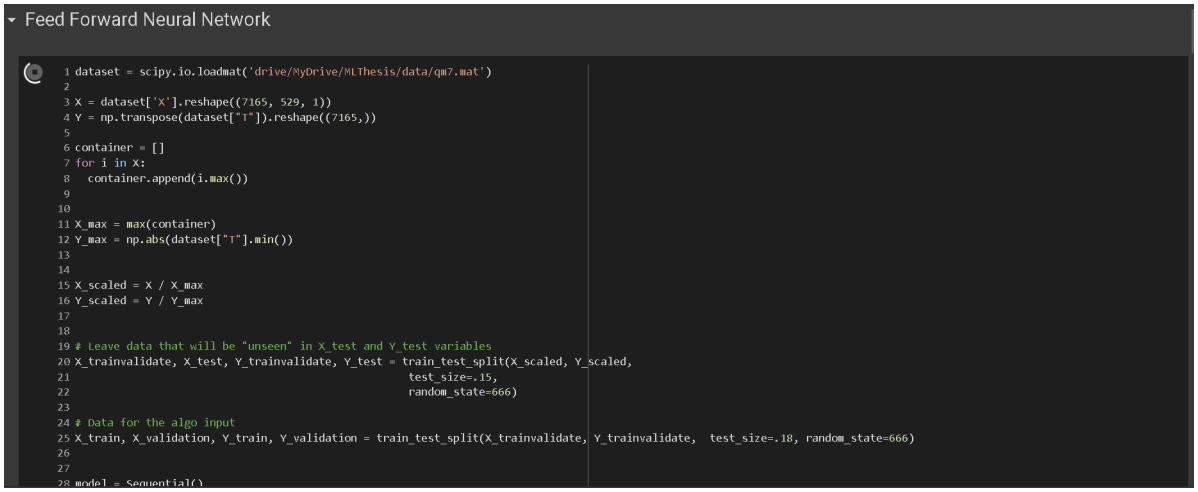

Figure 5.1: Google Colaboratory Platform - screenshot I.

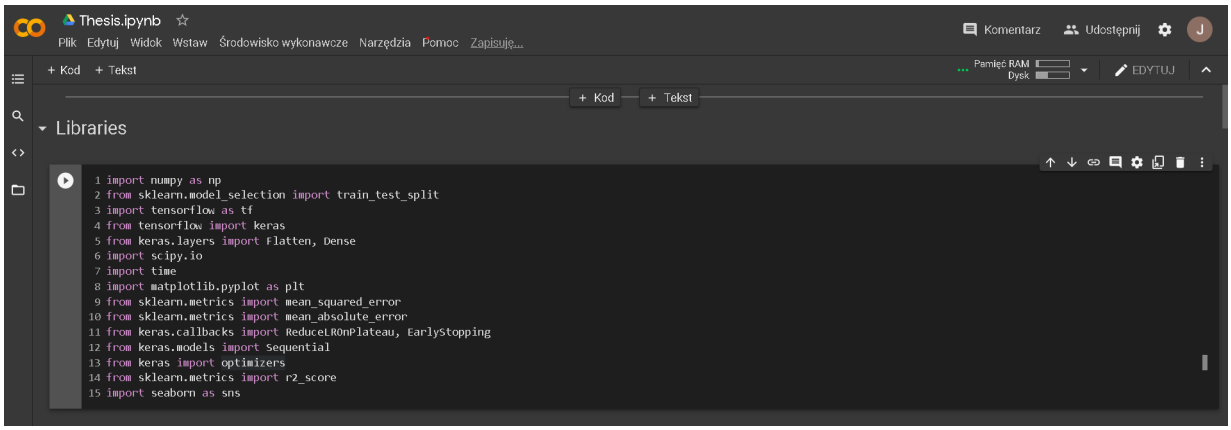

Figure 5.2: Google Colaboratory Platform - screenshot II.

5.3 Sketch of the model (layers and flattening)

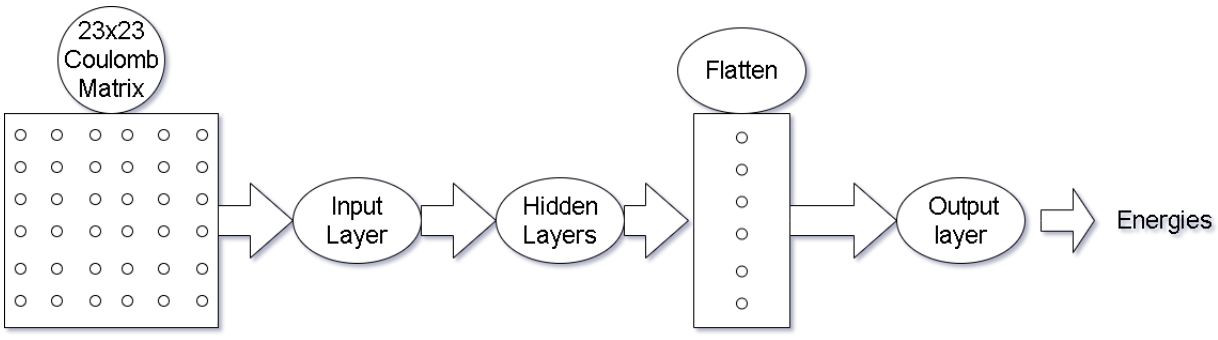

Figure 5.3: Simplified sketch of architecture.

On the figure 5.3 we can see a sketch of the entire model which will be subjected to a finite number of iterations. Figure 5.3 summarizes what will be happening in the code:

- First, load a Coulomb matrix to the input layer;

- Then, run the algorithm and let the data flow through the hidden layers;
- Just before the final layer, flatten the matrix, i.e. transform it to a 1-D vector;
- Finally, get predicted energies from the output;

5.4 Pipeline

5.4.1 Importing libraries

This step is self explanatory. The imports of libraries are necessary to run the program:

```
[language=Python]

import numpy as np
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow import keras
from keras.layers import Flatten, Dense
import scipy.io
import time
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from keras.callbacks import ReduceLROnPlateau, EarlyStopping
from keras.models import Sequential
from keras import optimizers
from sklearn.metrics import r2_score
import seaborn as sns
```

5.4.2 Dataset loading

We use sci-kit learn library to load our dataset. It is written as .mat file (meaning that it was generated in MATLAB environment). By using loadmat() method we convert the data to numpy array:

```
[language=Python]

dataset = scipy.io.loadmat('drive/MyDrive/MLThesis/data/qm7.mat')
```

5.4.3 Data preview

By using the below code we can preview first records of the data that will be put into the MLP algorithm:

```
X = dataset["X"]
y = dataset["T"].T

X[:3]
y[:3]
```

and see how the data look like in the array for y - dependent variable:

```
array([[-417.96],
   [-712.42],
   [-564.21]], dtype=float32)
```

And independent variable X:

```
array([[[36.858105 ,  2.9076326,  2.907612 , ...,  0.       ,
      0.       ,  0.       ],
    [ 2.9076326,  0.5      ,  0.29672  , ...,  0.       ,
      0.       ,  0.       ],
    [ 2.907612 ,  0.29672  ,  0.5      , ...,  0.       ,
      0.       ,  0.       ],
    ...,
    [ 0.       ,  0.       ,  0.       , ...,  0.       ,
      0.       ,  0.       ],
    [ 0.       ,  0.       ,  0.       , ...,  0.       ,
      0.       ,  0.       ],
    [ 0.       ,  0.       ,  0.       , ...,  0.       ,
      0.       ,  0.       ]],

   [[36.858105 , 12.599944 ,  2.9019997, ...,  0.       ,
      0.       ,  0.       ],
    [12.599944 , 36.858105 ,  1.4731166, ...,  0.       ,
      0.       ,  0.       ],
    [ 2.9019997,  1.4731166,  0.5      , ...,  0.       ,
      0.       ,  0.       ],
    ...,
    [ 0.       ,  0.       ,  0.       , ...,  0.       ,
      0.       ,  0.       ],
    [ 0.       ,  0.       ,  0.       , ...,  0.       ,
      0.       ,  0.       ],
    [ 0.       ,  0.       ,  0.       , ...,  0.       ,
      0.       ,  0.       ]],

   [[36.858105 , 14.261827 ,  1.503703 , ...,  0.       ,
      0.       ,  0.       ],
    [14.261827 , 36.858105 ,  2.9250205, ...,  0.       ,
      0.       ,  0.       ],
    [ 1.503703 ,  2.9250205,  0.5      , ...,  0.       ,
      0.       ,  0.       ],
    ...,
```

```
[ 0.       ,  0.       ,  0.       , ...,  0.       ,
  0.       ,  0.       ],
[ 0.       ,  0.       ,  0.       , ...,  0.       ,
  0.       ,  0.       ],
[ 0.       ,  0.       ,  0.       , ...,  0.       ,
  0.       ,  0.       ]]], dtype=float32)
```

This is how the data with coulomb matrix looks like (first entry of the data, 23 x 23 coulomb matrix) as well as the target variable that we want to predict:

```
for i in dataset["X"][0]:
  print(i)
```

```
[36.858105   2.9076326  2.907612   2.9075644  2.9053485  0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.       ]
[2.9076326  0.5        0.29672    0.29671896 0.2966784  0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.        ]
[2.907612   0.29672    0.5        0.29671845 0.29667813 0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.        ]
[2.9075644  0.29671896 0.29671845 0.5        0.29667678 0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.        ]
[2.9053485  0.2966784  0.29667813 0.29667678 0.5        0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.        ]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

Figure 5.4: Screenshot of data - the input variable.

```
[19]
a = dataset["T"].reshape(7165, 1)
for i in a:
  print(i)
  if i == a[10]:
    break
```

```
[-417.96]
[-712.42]
[-564.21]
[-404.88]
[-808.87]
[-677.16]
[-796.98]
[-860.33]
[-1008.49]
[-861.73]
[-708.37]
```

Figure 5.5: Screenshot of data - the target variable.

5.4.4 Scaling and reshaping

After we load the dataset, we first begin by reshaping the input variable X and target variable Y to dimensions acceptable by ANN. Since X is composed of 23×23 matrices we "flatten" every matrix from X to 529×1. We are also reshaping Y so we can represent it as a row vector.

We use one of the simplest methods of data scaling, taking maximum value of both sets and dividing each entry by this maximum value.

```
[language=Python]


X = dataset['X'].reshape((7165, 529, 1))
Y = np.transpose(dataset["T"]).reshape((7165,))

container = []
for i in X:
  container.append(i.max())


X_max = max(container)
Y_max = np.abs(dataset["T"].min())


X_scaled = X / X_max
Y_scaled = Y / Y_max
```

5.4.5 Splitting data to training, validation and test sets

We begin by using the holdout method and we are splitting the data into three parts: a training dataset, a validation dataset, and a test dataset. Test data set vs validation and training data set is split in 80:20 ratio. Same is the split for training and validation.

```
# Leave data that will be "unseen" in X_test and Y_test variables
X_trainvalidate, X_test, Y_trainvalidate, Y_test = train_test_split
    (X_scaled, Y_scaled,  test_size=.20, random_state=666)

# Data for the algo input
X_train, X_validation, Y_train, Y_validation = train_test_split(
    X_trainvalidate,  Y_trainvalidate,
test_size=.20, random_state=666)
```

5.4.6 Sequential model

After the data pre-processing, we can go into the structure of a model using Keras sequential model. It is a perfect fit for a situation where each layer has exactly one input tensor and one output tensor where overall, we have a stack of layers. We could not use the sequential model in a following scenario:

- There would have been multiple inputs or outputs in the model;
- Any layer has multiple inputs or outputs;
- A layer sharing is required for a model to perform well;
- There would be a need for non-linear topology;

We will pass a list of layers to the Sequential constructor contained in variable 'model'. We will also specify a normal distribution for weights generation.

```
model = Sequential()

# 64 -> 128 -> 529 -> Flatten -> 1
model.add(Dense(64, activation='relu', kernel_initializer='he_normal'))
model.add(Dense(128, activation='relu', kernel_initializer='he_normal'))
model.add(Dense(529, activation='relu', kernel_initializer='he_normal'))
model.add(Flatten())
model.add(Dense(1, activation='linear'))
```

After the above is met, we use compile method to configure the model for training. Our loss will be measured using Mean Absolute Error (and additional Root Mean Square Error set in metrics variable) and we choose Adam for our gradient descent optimization algorithm with learning rate set to 0.001.

```
model.compile(loss='mae',
          optimizer=optimizers.Adam(.001),
          metrics=['mae', tf.keras.metrics.RootMeanSquaredError()])
```

Through the usage of "add" method, we create overall 6 layers with different shapes, we use reLU activation function for the first four layers, then we use "Flatten" function to get our output as a 1-D vector using linear activation function on the output.

5.5 Training the model

We begin by using time module to calculate how long it will take for the model to finish training. We use a callback API to make our model more efficient using ReduceLROnPlateau. If the metric stops improving after some specified ("patience" parameter) period of time (epochs), this method will accelerate the convergence of the algorithm by reducing eta parameter by the factor of 0.1.

We set number of epochs to 15 and we initialize training of the model by providing 7 arguments: coulomb matrix train dataset, energies from the train set, batch size of our model, previously specified number of epochs, callback parameter for reducing eta, and validation data. Finally, we print the network summary via summary() method.

[language=Python]

```
start = time.time()


reduce_eta = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=1, verbose=1)

n_epochs = 15

history = model.fit(X_train, Y_train,
                    batch_size=10,
                    epochs=n_epochs,
                    callbacks=[reduce_eta],
                    verbose=1,
                    validation_data=(X_validation, Y_validation))

model.summary()

end = time.time()

print(f"Training time: {end-start}")
print(f"Number of epochs: {n_epochs}")
```

5.6 Results

The algorithm was given 15 epochs until it finished training itself and Mean Absolute Error on test dataset yields 20.701 MAE and reaching the last epoch was obtained after 645.564 seconds.

```
5 print(f"Training time: {end-start}")
6 print(f"Number of epochs: {n_epochs}")

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 529, 64)           128
_________________________________________________________________
dense_1 (Dense)              (None, 529, 128)          8320
_________________________________________________________________
dense_2 (Dense)              (None, 529, 529)          68241
_________________________________________________________________
flatten (Flatten)            (None, 279841)            0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 279842
=================================================================
Total params: 356,531
Trainable params: 356,531
Non-trainable params: 0
_________________________________________________________________
Training time: 645.5643026828766
Number of epochs: 15

[3]  1 approximated_energy = model.predict(X_test)
     2
     3
     4 print(f"Test dataset error (MAE): {Y_max*mean_absolute_error(Y_test, approximated_energy)}")

Test dataset error (MAE): 20.701087951660156
```

Figure 5.6: Summary of the model after all iterations have passed.

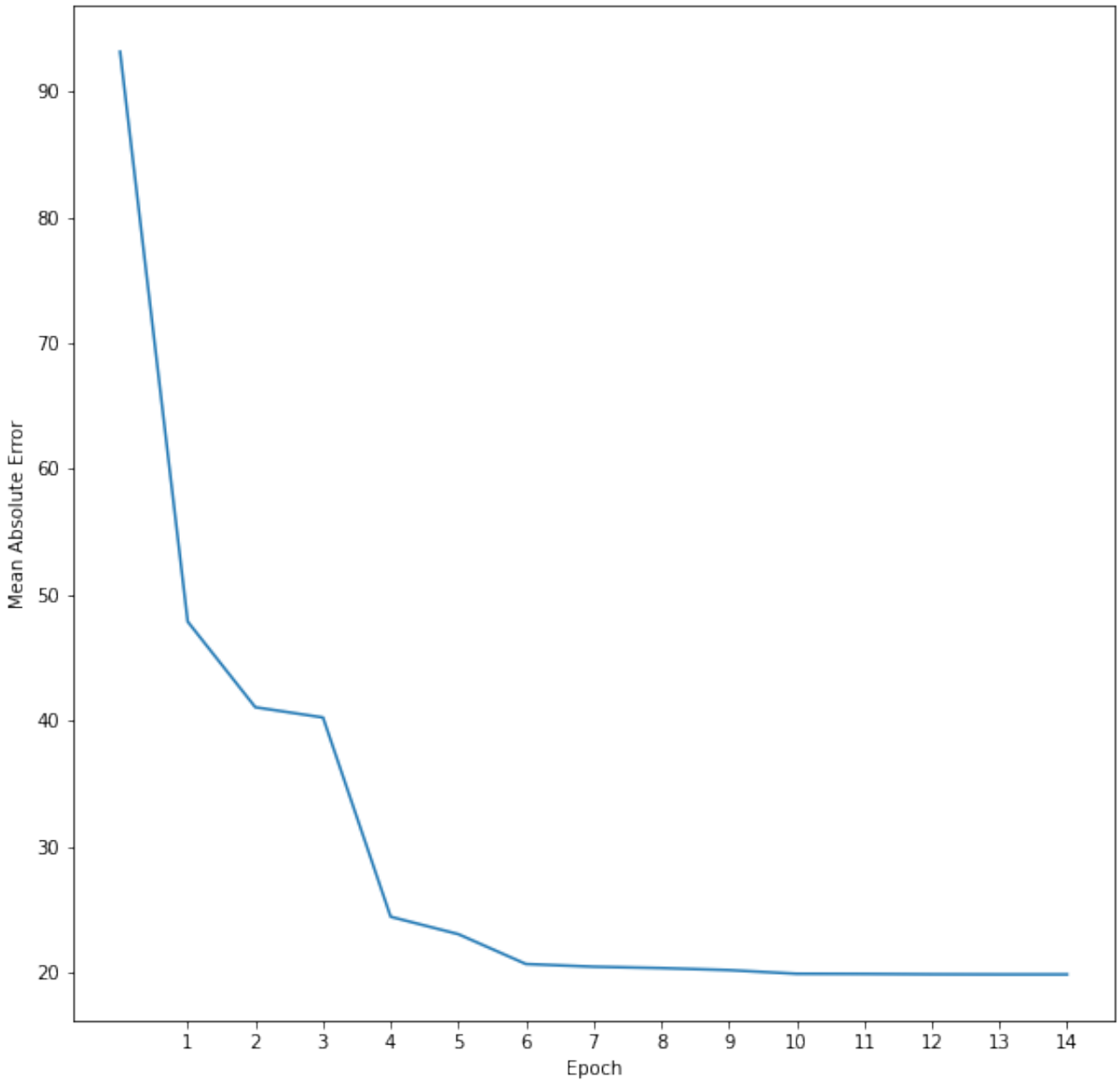

Figure 5.7: Plot of loss using Mean Absolute Error on a training dataset.

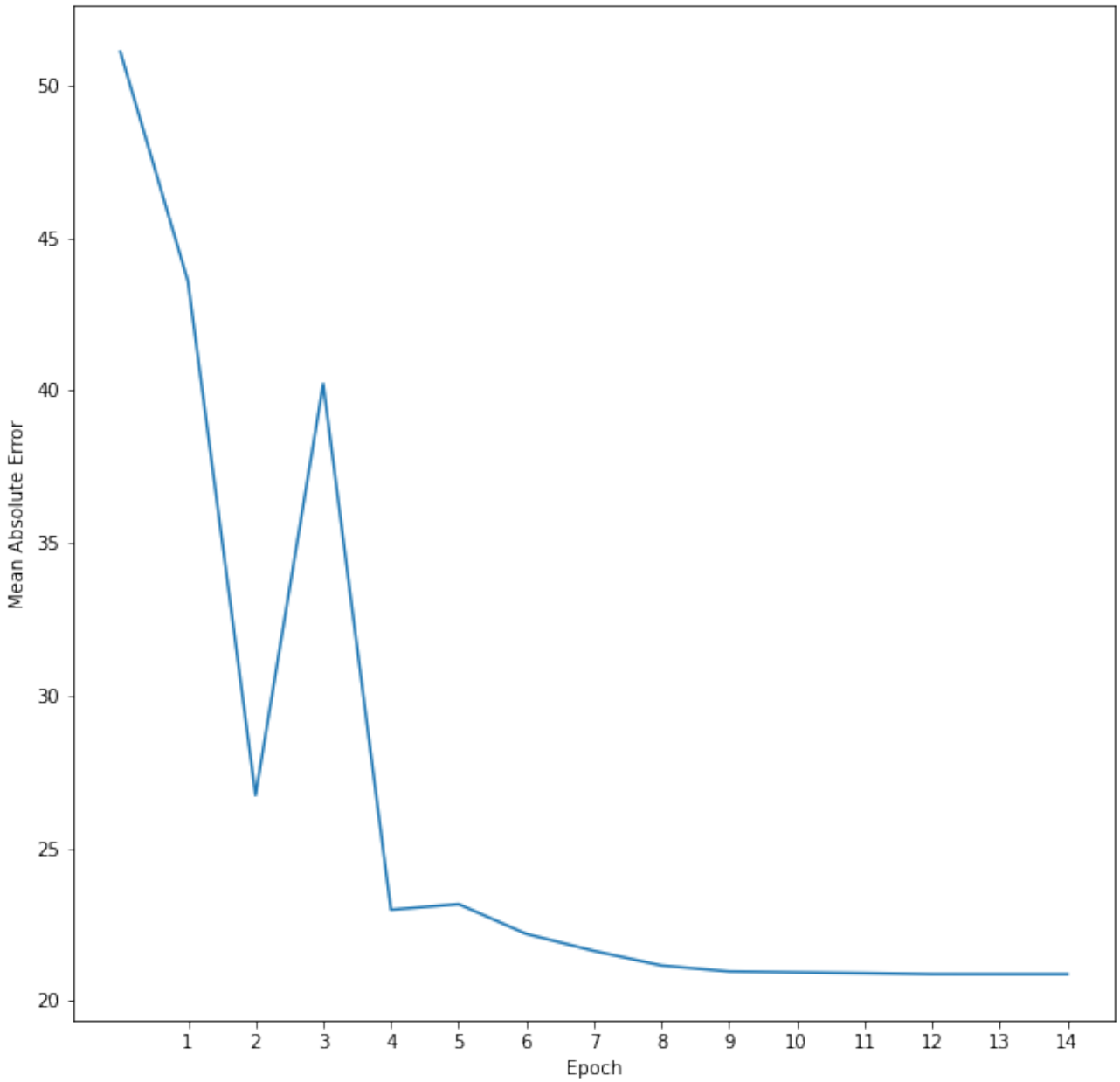

Figure 5.8: Plot of loss using Mean Absolute Error on a validation dataset.

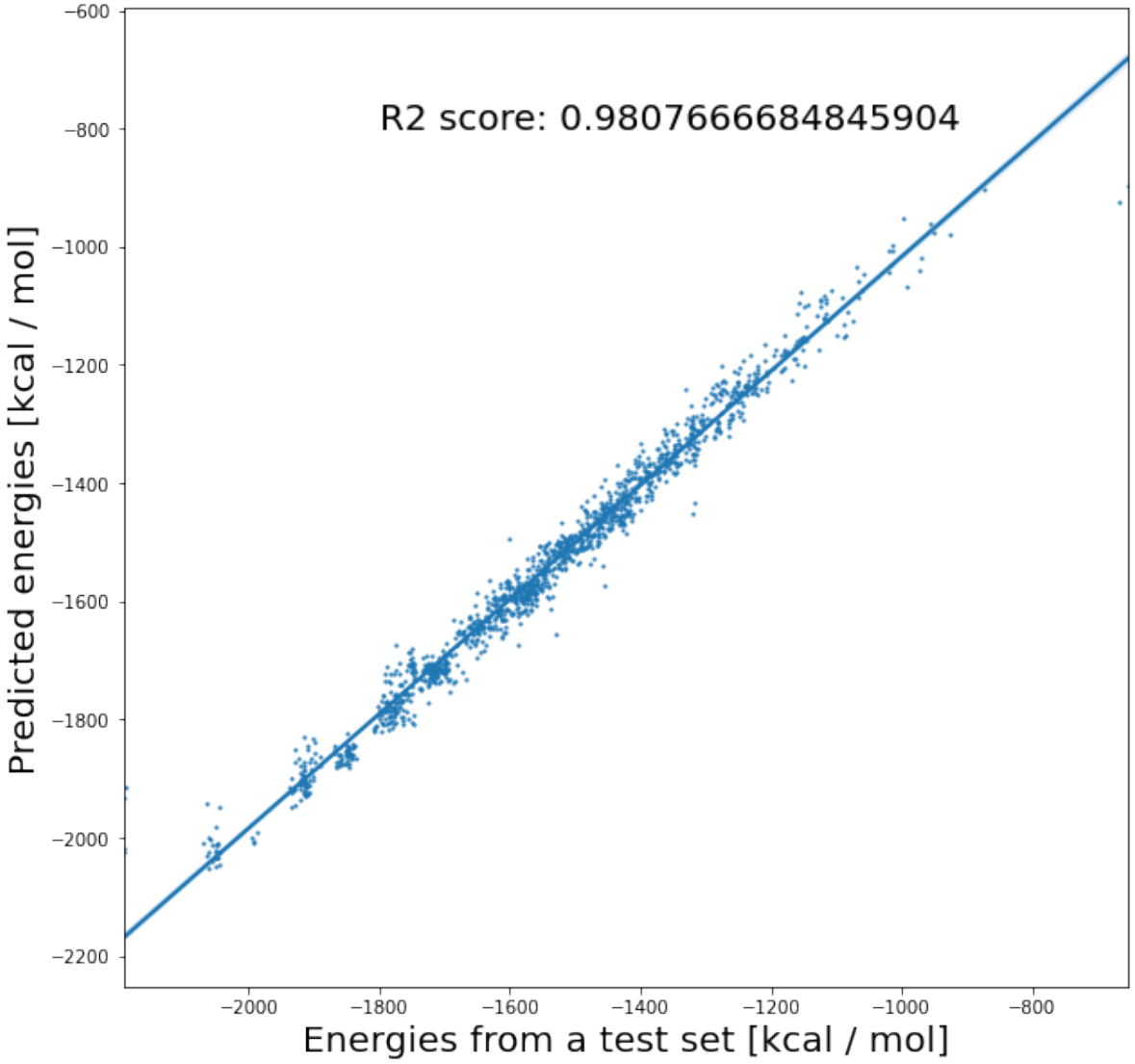

Figure 5.9: R^2 score for test dataset.

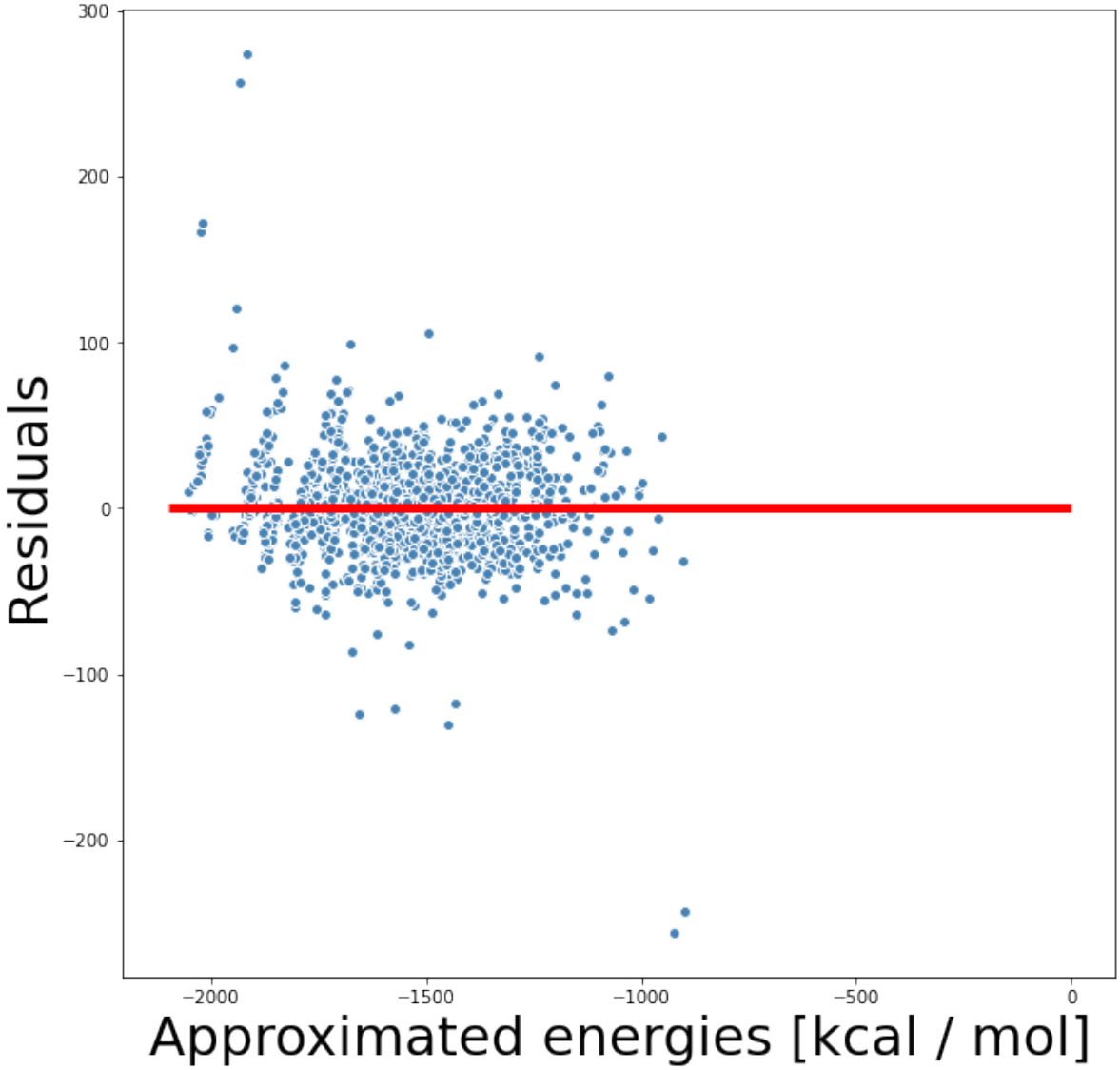

Figure 5.10: Residual plot for predicted energies.

5.7 Conclusions

The thesis has described the quantum principles that have been used to calculate chemical quantum parameters such as atomization energy. Then, a brief description of machine learning and deep learning was provided to explain how one can use ML algorithms to predict atomization energy. By leveraging Python language and its libraries, it has been shown how one can construct a whole machine learning model - a feed-forward neural network - which is able to give predictions for quantum mechanical parameter. We can see that the network has reached convergence both on training set and validation set implying that it has found the approximated solution (approximated energies) with 20.701 MAE. The R^2 coefficient of determination of the regression analysis gives 0.9808, which implies that the approximated energies were predicted in a way that they closely resemble the real ones. Residual analysis shows that there are only few outliers and there is not a clear visible pattern constructed from points. Therefore, we can say that the model is able to capture some explanatory information.

It has been shown that Deep Learning methods could be used to solve an interesting Physics problem in QM. What remains is the need to improve the validity of the models and closely approach the chemical and physical accuracy that one can achieve by the experiment.

Appendices

Appendix A

Implementation of ANN in Python

```
Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/10gPxt_Np-Q7P-
       HBprNad8SVtuS9nQyZ0

##Libraries
"""

import numpy as np
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow import keras
from keras.layers import Flatten, Dense
import scipy.io
import time
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from keras.callbacks import ReduceLROnPlateau, EarlyStopping
from keras.models import Sequential
from keras import optimizers
from sklearn.metrics import r2_score
import seaborn as sns

"""##Feed Forward Neural Network"""

dataset = scipy.io.loadmat('drive/MyDrive/MLThesis/data/qm7.mat')

X = dataset['X'].reshape((7165, 529, 1))
Y = np.transpose(dataset["T"]).reshape((7165,))

container = []
for i in X:
```

```
    container.append(i.max())


X_max = max(container)
Y_max = np.abs(dataset["T"].min())


X_scaled = X / X_max
Y_scaled = Y / Y_max


# Leave data that will be "unseen" in X_test and Y_test variables
X_trainvalidate, X_test, Y_trainvalidate, Y_test = train_test_split
    (X_scaled, Y_scaled, test_size=.20, random_state=666)


# Data for the algo input
X_train, X_validation, Y_train, Y_validation = train_test_split(
    X_trainvalidate, Y_trainvalidate, test_size=.20, random_state
    =666)



model = Sequential()

# 64 -> 128 -> 529 -> Flatten -> 1
model.add(Dense(64, activation='relu', kernel_initializer='
    he_normal'))
model.add(Dense(128, activation='relu', kernel_initializer='
    he_normal'))
model.add(Dense(529, activation='relu', kernel_initializer='
    he_normal'))
model.add(Flatten())
model.add(Dense(1, activation='linear'))


model.compile(loss='mae', optimizer=optimizers.Adam(.001), metrics
    =['mae', tf.keras.metrics.RootMeanSquaredError()])



start = time.time()


reduce_eta = ReduceLROnPlateau(monitor='val_loss', factor=0.1,
    patience=1, verbose=1)

n_epochs = 15
```

```
history = model.fit(X_train, Y_train,
                    batch_size=10,
                    epochs=n_epochs,
                    callbacks=[reduce_eta],
                    verbose=1,
                    validation_data=(X_validation, Y_validation))

model.summary()

end = time.time()

print(f"Training time: {end-start}")
print(f"Number of epochs: {n_epochs}")

"""## Validation"""

approximated_energy = model.predict(X_test)


print(f"Test dataset error (MAE): {Y_max*mean_absolute_error(Y_test
    , approximated_energy)}")
print(f"Test dataset error (RMSE): {Y_max*mean_squared_error(Y_test
    , approximated_energy, squared=False)}")

#plt.figure(figsize=(8,8))

loss = np.array(history.history["loss"]) * Y_max
mae = np.array(history.history["mae"]) * Y_max
rmse = np.array(history.history['root_mean_squared_error']) * Y_max
val_loss = np.array(history.history["val_loss"]) * Y_max
val_mae = np.array(history.history["val_mae"]) * Y_max
val_rmse = np.array(history.history['val_root_mean_squared_error'])
    * Y_max

epochs = [x for x in range(1, n_epochs + 1)]

print(f"Mean absolute error over epochs from test set {mae}")
print(f"Mean absolute error over epochs from validation set {
   val_mae}")
print(f"Root mean square error over epochs from test set {rmse}")
print(f"Root mean square error over epochs from validation set {
   val_rmse}")

plt.figure(figsize=(10,10))
plt.title("Loss on a training set")
plt.xlabel("Epoch")
plt.ylabel("Mean Absolute Error")
plt.xticks(epochs)
plt.plot(loss)
```

```
plt.figure(figsize=(10,10))
plt.title("Loss on a validation set")
plt.xlabel("Epoch")
plt.ylabel("Mean Absolute Error")
plt.xticks(epochs)
plt.plot(val_loss)

R2_scr = r2_score(Y_max * Y_test, Y_max * approximated_energy)

plt.figure(figsize=(10,10))
plt.title("Linear fit", size= 20)
plt.xlabel("Energies from a test set [kcal / mole]", size=20)
plt.ylabel("Predicted energies [kcal / mole]", size=20)
plt.annotate(f"R2 score is {R2_scr}", (-1800.0, -800), size=20)


sns.regplot(Y_test * Y_max, approximated_energy * Y_max, marker="o
    ", scatter_kws={'s':2})

plt.figure(figsize=(10, 10))
plt.scatter(approximated_energy * Y_max, (approximated_energy -
    Y_test.reshape(1075, 1)) * Y_max, c='steelblue', marker='o',
    edgecolor='white', label='Test data')
plt.xlabel("Approximated energies [kcal / mole]", size=30)
plt.ylabel("Residuals", size=30)
plt.title("Residual plot of approximated energies", size=30)
plt.hlines(y=0, xmin= -2100.0, xmax= 0.0, color='red', lw=5)
#plt.xlim([-1.0, -0.20])
```

Bibliography

[1] 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.

[2] *DScribe*. URL: https://singroup.github.io/dscribe/latest/tutorials/coulomb_matrix.html. (accessed: 17.02.2021).

[3] *Erikbern*. URL: https://erikbern.com/2014/11/29/deep-learning-for-chess.html. (accessed: 28.01.2021).

[4] *Github of rasbt*. URL: https://rasbt.github.io/mlxtend/user_guide/general_concepts/gradient-optimization_files/ball.png. (accessed: 28.01.2021).

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

[6] *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2019.

[7] *History of AI Winters*. URL: https://www.actuaries.digital/2018/09/05/history-of-ai-winters/#_edn1. (accessed: 16.01.2021).

[8] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[9] *IBM official website*. URL: https://www.ibm.com. (accessed: 04.02.2021).

[10] *Intel*. URL: https://www.intel.com/content/www/us/en/silicon-innovations/moores-law-technology.html. (accessed: 28.01.2021).

[11] *Keras API website*. URL: https://keras.io/about/. (accessed: 16.01.2021).

[12] Stanisław Kryszewski. *Mechanika Kwantowa*. Gdańsk: Wydawnictwo Uniwersytetu Gdańskiego, 2018.

[13] *Machine Learning for Physics and the Physics of Learning*. URL: http://www.ipam.ucla.edu/programs/long-programs/machine-learning-for-physics-and-the-physics-of-learning/. (accessed: 14.12.2020).

[14] *Matplotlib offical website*. URL: https://matplotlib.org/3.2.1/index.html. (accessed: 16.01.2021).

[15] Walter McCulloch Warren S. Pitts. ""A logical calculus of the ideas immanent in nervous activity"". In: *The bulletin of mathematical biophysics* (1943). DOI: https://doi.org/10.1007/BF02478259.

[16] *Numpy website*. URL: https://numpy.org/. (accessed: 07.03.2021).

[17] *Pandas official website*. URL: https://pandas.pydata.org/. (accessed: 07.03.2021).

[18] *Python language website*. URL: https://www.python.org/. (accessed: 07.01.2021).

[19] *Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn and TensorFlow 2, 3rd Edition*. Packt Publishing, 2019.

[20] *Quantum-machine*. URL: http://quantum-machine.org/datasets/. (accessed: 23.01.2021).

[21] *Ram ai*. URL: https://ram-ai.com/en/. (accessed: 28.01.2021).

[22] David E Rumelhart, Geoffrey E Hinton, and Ronald J. Williams. *Learning internal representations by error propagation*. Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

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

[24] *Schematic of a neural network*. URL: https://knowingneurons.com/2018/04/11/artificial-neural-network/. (accessed: 23.01.2021).

[25] *Scikit-learn website*. URL: https://scikit-learn.org/stable/index.html. (accessed: 16.01.2021).

[26] *SciPy offical website*. URL: https://www.scipy.org/about.html. (accessed: 16.01.2021).

[27] *Tensorflow website*. URL: https://www.tensorflow.org/. (accessed: 16.01.2021).

[28] *towardsdatascience*. URL: https://towardsdatascience.com/data-types-from-a-machine-learning-perspective-with-examples-111ac679e8bc. (accessed: 14.02.2021).

[29] B. Widrow and others. *An Adaptive Adaline Neuron Using Chemical Memistors*. Tech. rep. Stanford, CA, 1960.

[30] *Wolfram Math World*. URL: https://mathworld.wolfram.com/L2-Space.html. (accessed: 11.02.2021).