Machine learning in solid-state physics and statistical physics

Vrček, Lovro

UNIVERSITY OF ZAGREB
FACULTY OF SCIENCE
DEPARTMENT OF PHYSICS

Lovro Vrček

Machine learning in solid-state physics and statistical physics

Master Thesis

Zagreb, 2018.

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO-MATEMATIČKI FAKULTET
FIZIČKI ODSJEK

Lovro Vrček

Strojno učenje u fizici čvrstog stanja i statističkoj fizici

Diplomski rad

Zagreb, 2018.

UNIVERSITY OF ZAGREB
FACULTY OF SCIENCE
DEPARTMENT OF PHYSICS

INTEGRATED UNDERGRADUATE AND GRADUATE UNIVERSITY
PROGRAMME IN PHYSICS

Lovro Vrček

Master Thesis

Machine learning in solid-state physics and statistical physics

Advisor: Vinko Zlatić, dr. sc.

Co-Advisor: Ivo Batistić, prof. dr. sc.

Master Thesis grade: _____

Committee: 1. _____

2. _____

3. _____

Master Thesis defence date: _____

Zagreb, 2018.

I would like to express special appreciation and thanks to Dr. Ivor Lončarić on guidance through this project and hours spent discussing all the problems we have encountered and ideas to overcome these problems.

Also, I would like to thank Dr. Vinko Zlatić for his expert advice and encouragement throughout the project, as well as Dr. Predrag Lazić on help regarding parallel computing.

Furthermore, I would like to thank Dr. J.I. Juaristi and Dr. M. Alducin for providing the data used in this project.

Finally, I would like to thank all my family and friends, especially my parents Vesna and Neven, brother Janko, and girlfriend Kristin, for all the support throughout the years.

Strojno učenje u fizici čvrstog stanja i statističkoj fizici

Sažetak

U ovom radu proučavamo alternativni pristup simuliranju molekularne dinamike velikih sustava preko dugih vremenskih perioda; korištenje strojnog učenja umjesto DFT-a. Proučavani sustav sastoji se od površine rutenija koja međudjeluje s atomima vodika. Za konstruiranje regresijskog modela koristimo neuronske mreže te treniramo nekoliko različitih arhitektura mreža kako bismo našli optimalnu. Kao ulaz koristimo Gaussove deskriptore koji kartezijeve koordinate atoma pretvore u oblik pogodniji za opis sustava. U ovom slučaju, postoji 20 deskriptora za svaku vrstu atoma, što znači da u ulaznom sloju neuronske mreže imamo 40 čvorova. Pokazano je da optimalna arhitektura neuronske mreže sadrži tri skrivena sloja, od kojih prvi ima 50 čvorova, drugi 30, a treći 10. Osim toga, pokazano je kako se taj regresijski model ponaša ovisno o broju koraka prilikom treniranja, analizirana je važnost korištenih deskriptora te je proučeno ponašanje modela u slučaju korištenja Zernike deskriptora umjesto Gaussovih, ili mijenjanja polumjera obuhvaćanja atoma.

Ključne riječi: DFT, molekularna dinamika, neuronske mreže, deskriptori

Machine learning in solid-state physics and statistical physics

Abstract

In this work, we study an alternative approach to simulating molecular dynamics of large systems over long time periods; the one using machine learning instead of DFT. Studied system is a ruthenium surface which is interacting with hydrogen atoms. We use neural networks to obtain the regression model for the studied system, and train several different architectures of neural networks in order to find the optimal one. For input we use Gaussian descriptors which take Cartesian coordinates of the atoms and translate them in a more suitable form. In this case, there are 20 descriptors for each type of atoms, meaning that input layer of neural network has in total 40 nodes. Optimal architecture was found to be the one with three hidden layers with 50, 30, and 10 nodes, respectively. It was shown how our regression model behaves depending on number of training steps, importance of used descriptors was analyzed, and it was shown how model behaves if Zernike descriptors are used instead of Gaussian, or if cutoff radius is altered.

Keywords: DFT, molecular dynamics, neural networks, descriptors

Contents

1 Introduction

In recent years, we have witnessed the fast development of artificial intelligence and impact it has on our everyday lives. With each day we more and more often hear about data science that needs machine learning methods to analyze huge datasets, self-driving cars, optical character recognition and many other aspects of modern world. All these problems are undergoing intense study in computer science research and regularly produce some new discoveries that could help not only in the mentioned problems, but also in many other branches of science.

Meanwhile, molecular dynamics is an important field of research in physics and chemistry with many applications in material science, biology and medicine. For example, molecular dynamics is used for simulating the growth of thin films [1], refining structures of proteins and computer-assisted drug design [2]. Systems that are studied typically consist of large number of atoms for which it is impossible to use accurate quantum-mechanical methods and usually semi-empirical force fields are used.

Transition metals are particularly hard to model accurately in this way, but on the other hand, they are some of the most common elements researched in physics and chemistry, and will also be topic of this work. As defined by IUPAC, transition metals are "*elements whose atom has partially filled d sub-shell, or which can give rise to cations with an incomplete d sub-shell*", [3]. Most of these elements have large range of complex ions in various oxidation states and catalytic properties either as the elements or as ions. Although some of them, such as platinum and ruthenium, are mostly inert, they are extremely important as catalysts for the industrially and ecologically most important reactions.

A simple theory that describes some of the properties of metals is Drude-Sommerfeld model in which electron-electron interaction and periodic potential of the lattice are neglected. This means that electrons in the metal move as in infinite potential well.

In such case, result of solving Schrödinger equation are plane waves

$$\psi_{\vec{k}}(\vec{r}) \propto e^{i\vec{k}\vec{r}}, \tag{1.1}$$

with energies

$$E_{\vec{k}} = \frac{\hbar^2 \vec{k}^2}{2m_e}, \tag{1.2}$$

where $\vec{r}$ is a position vector, $\vec{k}$ is wave vector, $\hbar$ is Planck's constant and m_e is mass of electron. Here we have used periodic Born-von Karman boundary conditions

$$\psi_{\vec{k}}(\vec{r} + N_i\vec{a}_i) = \psi_{\vec{k}}(\vec{r}), \tag{1.3}$$

where $\vec{a}_i$ is a unit vector describing the lattice and N_i is number of unit cells in the direction of $\vec{a}_i$. These boundary constraints allow only quantized wave numbers, meaning that

$$\vec{k} = \alpha_1\vec{b}_1 + \alpha_2\vec{b}_2 + \alpha_3\vec{b}_3, \tag{1.4}$$

$$\alpha_i = \frac{n_i}{N_i}, \qquad n_i = 0, \pm 1, \pm 2, ... \tag{1.5}$$

where $\vec{b}$ are unit vectors in reciprocal space.

However, even though Drude-Sommerfeld model can give us basic understanding of metals, most of the properties still remain unexplained. The reason for that is electronic structure of the material is not taken into account. More importantly, this model can't treat molecules, covalent bonds, etc. While there is still no computationally cheap universal method for calculating the electronic structure for all possible materials, the density functional theory (DFT) is applicable for most systems and is widely used today as its computational cost is modest. DFT is a computational quantum mechanical modeling method used in physics, chemistry and material science which provides us the ground state properties of the system. The foundation of the theory of electronic structure of matter is non-relativistic Schrödinger equation for the many-electron wavefunction Ψ. If Born-Oppenheimer approximation is applied, heavy nuclei are fixed in space while electrons are free to move, and this yields the

many-electron time-independent Schrödinger equation [4]:

$$\hat{H}\Psi = \left(-\frac{\hbar^2}{2m_e} \sum_j \nabla_j^2 - \sum_{j,l} \frac{Z_l e^2}{|\vec{r_j} - \vec{R_l}|} + \frac{1}{2} \sum_{j \neq j'} \frac{e^2}{|\vec{r_j} - \vec{r_{j'}}|} \right) = E\Psi, \tag{1.6}$$

where $\hat{H}$ is the Hamiltonian, $\vec{r_j}$ are positions of electrons, $\vec{R_l}$ and Z_l are positions and atomic numbers of the nuclei, e is elementary charge and is the total E is energy of the system. First term in equation (1.6) is kinetic energy $\hat{T}$, second term is the potential energy from the external field due to positively charged nuclei $\hat{V}$, and the last one is the electron-electron interaction energy $\hat{U}$. It is easy to see that, for any larger number of particles, solving equation (1.6) is a troublesome task. Luckily, DFT gives us a way around. Crucial fact for DFT is that many physical properties, like electron density $n(\vec{r})$, depend only on $\vec{r}$, whereas Ψ depends on $\vec{r_1}, \vec{r_2}$, ... $\vec{r_n}$. For normalized wave function, electron density which plays a key role in DFT is obtained by tracing over all other variables:

$$n(\vec{r}) = N \int d^3 r_2 \ldots \int d^3 r_N \Psi^*(\vec{r}, \vec{r_2}, \ldots, \vec{r_N}) \Psi(\vec{r}, \vec{r_2}, \ldots, \vec{r_N}). \tag{1.7}$$

The Hohenberg-Kohn theorems state that the total energy is a unique functional of the electron density, and that the functional that delivers the ground state energy of the system gives the lowest energy if and only if the input density is true ground state density n_0 [5]. With this in mind, and by knowing n_0, it is possible to calculate ground-state expectation of any other observable $\hat{O}$:

$$O[n_0] = \left\langle \Psi[n_0] \middle| \hat{O} \middle| \Psi[n_0] \right\rangle. \tag{1.8}$$

In particular, for total energy that would mean:

$$E_0 = E[n_0] = \left\langle \Psi[n_0] \middle| \hat{T} + \hat{V} + \hat{U} \middle| \Psi[n_0] \right\rangle, \tag{1.9}$$

where the contribution of the external potential $\left\langle \Psi \middle| \hat{V} \middle| \Psi \right\rangle$, in case of more general density function, can be written as:

$$V[n] = \int V(\vec{r}) n(\vec{r}) \mathrm{d}^3 r. \tag{1.10}$$

The functional $V[n]$ differs from system to system, while $T[n]$ and $U[n]$ are universal. For a specified system, we have to minimize the functional

$$E[n] = T[n] + U[n] + \int V(\vec{r})n(\vec{r})\mathrm{d}^3r \tag{1.11}$$

with respect to $n(\vec{r})$, which will yield ground-state electron density n_0, from which we can obtain all other ground-state observables. Problem of minimizing the functional in (1.11), first we consider an energy functional that does not explicitly have an electron-electron interaction term:

$$E_s[n] = \left\langle \Psi_s[n] \left| \hat{T} + \hat{V}_s \right| \Psi_s[n] \right\rangle, \tag{1.12}$$

where $\hat{V}_s$ is an external effective potential in which the particles are moving so that $n_s(\vec{r}) = n(\vec{r})$. After that, we solve the Kohn-Sham equation [6]:

$$\left[-\frac{\hbar^2}{2m}\nabla^2 + V_s(\vec{r}) \right] \phi_i(\vec{r}) = \epsilon_i \phi_i(\vec{r}), \tag{1.13}$$

where ϕ_i are orbitals and ϵ_i their corresponding energies. From solving this equation we obtain the orbitals ϕ_i that reproduce the density of the original many-body system, $n(\vec{r})$:

$$n(\vec{r}) = n_s(\vec{r}) = \sum_i^N |\phi_i(\vec{r})|^2. \tag{1.14}$$

Since the first appearance of the DFT in 1964 [5], there have been many improvements of the method, such as in [7], but it still faces difficulties while describing intermolecular interactions, which is why improving the DFT is still an active research topic.

In this work, we will show how machine learning methods can be used in statistical and solid-state physics, and compare accuracy, speed and adaptability of these methods with some other frequently used techniques, such as DFT. Specifically, we will study the system consisting of hydrogen atoms which are interacting with a slab of ruthenium. Ruthenium is an element with symbol Ru and atomic number 44, belonging to the group 8 and period 5 of the periodic table. It is a transition metal belonging to the platinum group of the periodic table of elements, and is inert to

most other chemicals. Commercially, it is obtained from a sulfide of nickel and iron called pentlandite, and is unaffected by air, acids and water. However, it does react with molten alkali and halogen, oxidizes explosively and is suspected to be carcinogen. So far, it has found its application in electronic industry for manufacturing chip resistors and in chemical industry for use as anodes for chlorine production in electrochemical cells [8].

The reason for focusing on ruthenium is that it is one of the most important catalysts in the Fischer-Tropsch process. Fischer-Tropsch process is a collection of chemical reactions for making hydrocarbon fuels from carbon monoxide and hydrogen. The process was first introduced in 1933. by Franz Fischer and Hans Tropsch and used by Germany in World War II. to produce replacement motor fuel [9]. It is important in coal liquefaction, gas-to-liquids technology and many other chemical processes aimed at producing compounds based on hydrocarbon chains [10], and could potentially be used for obtaining low-sulfur diesel fuel [11]. Fischer-Tropsch process occurs only in the presence of a certain metal catalysts, most common being ruthenium, cobalt and iron. Advantage of ruthenium over other catalysts is that it is the most active catalyst and works at the lowest temperatures. It acts as a pure metal providing the simplest catalytic system of Fischer-Tropsch synthesis, but also produces the hydrocarbons of highest molecular weight. However, industrial applications of ruthenium are limited by its low world resources and, hence, its high price [12].

Since the aim of this work is to not just study static systems using machine learning, but to also apply it to molecular dynamics, it is important to note that temperature of this system would no longer be zero, as is the case of the static system. This means it is possible to study the system in microcanonical or canonical ensemble, and calculate its thermal properties, such as entropy and Gibbs energy, which are computationally difficult to obtain using DFT.

2 Machine learning

Machine learning is a field of computer science that uses statistical techniques to give computers the ability to learn with data, without being explicitly programmed [13]. It evolved from the study of pattern recognition and computational learning theory, and explores the study and construction of algorithms that overcome following strictly static program instructions by making data-driven predictions or decisions [14]. With this ability it has become crucial in many aspects of everyday life, such as web-page ranking and email filtering, but also in some state-of the art research fields, like computer vision [15] and speech recognition [16]. These tasks can usually be divided into three categories, which will be explained in the following subsection.

2.1 Types of machine learning

First type that will be discussed here is *supervised learning*, and it is used to learn a model from labeled training data that allows us to make predictions about unseen or future data [17]. The term supervised refers to a set of samples where the desired output signals are already known. Therefore, in supervised learning we first create and train a predictive model using machine learning algorithms and already labeled data, after which we put new, unlabeled data into the model and obtain the prediction. This can be performed using data with discrete class labels and continuous labels, in which case these subcategories are called classification and regression respectively. For example, predicting whether an email is spam or non-spam is a classification problem, whereas predicting the price of a house based on its size in square meters is a regression problem [18].

Next type is *unsupervised learning*, in which we are dealing with unlabeled data or data of unknown structure. In other words, output data is not provided and goal of algorithms of this type is to find regularities in the input [17]. Often there is a structure to the input space such that certain patterns occur more frequently than others, enables us to extract meaningful information from the input only. Most widely used technique for this is *clustering*, an exploratory data analysis technique that allows us to organize a pile of information into meaningful subgroups called clusters without

having any prior knowledge of their group membership. Companies often use this method to profile their customers when only data available is past transactions of those customers. This allows them to decide which strategy to use on different types of customers.

Last type that will be discussed here is *reinforcement learning*. It is used in systems where output is a sequence of actions. In such case, a single action is not important, the only thing important is that sequence of correct actions reach the goal [17]. Therefore, when the system is in some intermediate state, there is not the best possible action. All those actions that lead to the goal are considered good, and a reinforcement learning algorithm assesses the goodness of policies from past good actions which helps it create a strategy to reach the goal. A popular example of reinforced learning is a chess engine, where the algorithm decides upon a series of possible moves depending on the state of the board and tries to win the game of chess. If it succeeds, the algorithm receives the reward and marks the moves as "good". If some move leads to losing the game, it will be marked as "bad" and will not be repeated. Some games are more complex, in which case reward system might also be more complex than just win or lose, but information about the current state will still include a reward signal [18]. This is the reason why reinforcement learning is sometimes considered as a subfield of supervised learning. However, in reinforcement learning, reward signal of the current state is not some ground truth label or value, but a measure of how well the action was measured by a reward function, and due to this we have observed reinforcement learning separately.

In the following section we will focus ourselves on one of the most common and most powerful computing systems in machine learning - artificial neural networks. Although they can be used for solving all three machine learning problems, they are mostly used for supervised learning, and in this work we have used them for the same purpose.

2.2 Artificial neural networks

Artificial neural networks are computing systems inspired by the biological neural networks found in brains of humans and animals [19]. Such systems can learn to perform tasks by considering examples of that task, but without any strict algorithm telling them what to do. For example, neural networks can tell if there is dog on some picture just by looking at other pictures of dogs, without specifying what dogs look like or in what way they differ from cats. That is, artificial neural networks can solve problems that would be much more difficult with conventional computing techniques, and for that they use the same method that human brain uses at young age: learning solely from experience. More formal definition of a neural network would be, [20]:

"*A neural network is a massively parallel distributed processor made up of simple processing units, which has a natural propensity for storing experiantial knowledge and making it available for use. It resembles the brain in two respects:*

1. *Knowledge is acquired by the network from its environment through a learning process.*

2. *Interneuron connection strengths, known as synaptic weights, are used to store the acquired knowledge.*"

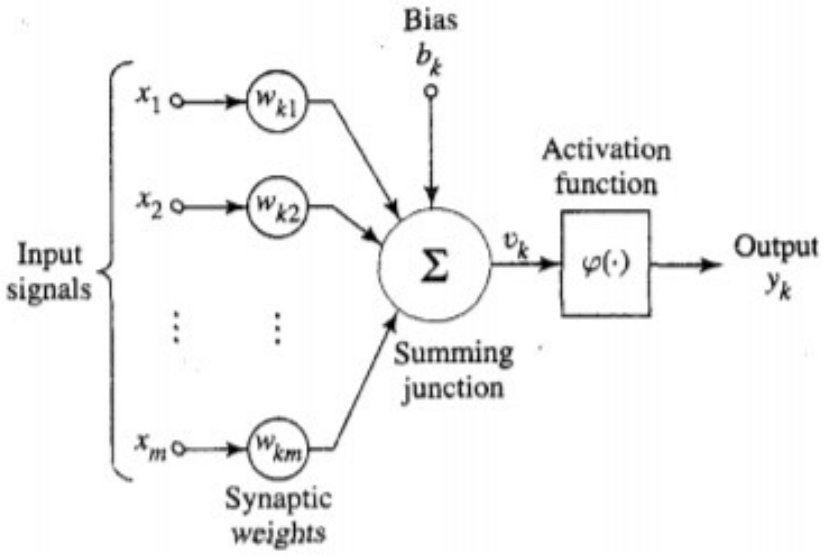

Figure 2.1: A model of a nonlinear artificial neuron. Source: [20].

2.2.1 Structure of neural networks

Artificial neural network is structured as a collection of connected artificial neurons, often called nodes. Artificial neurons are mathematical functions which receive one or more input, each input is separately weighted and then all inputs are summed and passed through a non-linear activation function which produces an output of artificial neuron. The activation function is usually monotonic, increasing, continuous differentiable and bounded. In most cases, activation function imitates a Heaviside step function which is 0 for all $x < 0$ and 1 otherwise. Hence, two most widely used activation functions are hyperbolic tangent and sigmoid function, defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \tag{2.1}$$

but sometimes other functions, such as Gaussian, can also be used.

In most cases, artificial neural networks are aggregated into layers of neurons. The first layer, called input layer, receives "raw" data as input. Output of the neural network is produces in the output layer, which can emit either a single value or an output vector with multiple values. Between these two, there can be one or more hidden layers. In the most simple case of neural networks, a feedforward neural network, connections between neurons do not form a cycle and all values are propagated in only one direction, from the input nodes through the hidden nodes to the output nodes. Each of these neurons performs a series of operation explained in the previous paragraph and sends output either to another neuron or emits it as output of a whole neural network. Except feedforward, there are many other types of neural networks such as recurrent which have cycles between nodes and modular neural networks consisting of series of independent neural networks moderated by some intermediary. Those types of neural networks are not important for this work, and will not be analyzed here.

2.2.2 Learning algorithms

Specifying number of hidden layers and number of nodes (neurons) per layer is not enough for a neural network to work. We still have to find out which weights should be when sending a signal between two neurons. In other words, we have to apply

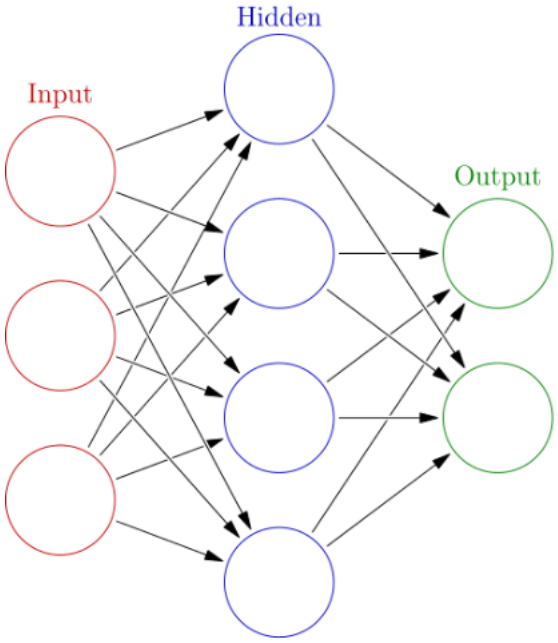

Figure 2.2: A feedforward artificial neural network with one hidden layer. Circles represent neurons and arrows represent connection between neurons. Source: [21].

learning algorithm to train the neural network .

Most popular learning algorithm is *gradient descent*, a first-order iterative optimization algorithm for finding the minimum of a function. In each iteration of the algorithm we take a step proportional to the negative of the function's gradient, which lets us approach the local minimum. The function we want to minimize is the cost function which describes the error of the fitting model. There are many types of cost functions, but the most popular one is mean-squared error function, defined as:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2, \tag{2.2}$$

where m is number of training examples, θ is a parameter vector, h_θ is hypothesis of model, $x^{(i)}$ is the vector of independent variables of i-th training example and $y^{(i)}$ is dependent variable of the same training example. With every step to the minimum of the cost function, we have to modify parameters θ. We have already said that this step is proportional to the negative gradient of the cost function, but we also need to control how quickly we approach the minimum. Therefore, we introduce the learning rate α as factor that multiplies cost function gradient and obtain formula for

updating the parameters θ:

$$\theta_j \to \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \tag{2.3}$$

We see that, if α is too small, it will take a long time for the algorithm to converge, but if it is too large, algorithm may overshoot the minimum and never converge. From equation (2.3) we see that for successful training of our machine-learning model we only need the gradient of the cost function and the suitable learning rate.

When performing gradient descent on neural networks, most frequently used method for finding cost function gradients is by using *backpropagation*. Backpropagation is sometimes also called backward propagation of errors, because the error is calculated at the output and distributed back through the network layers. This algorithm can be described as follows:

First, let us define $a^{(l)}$ as input vector of l-th layer of neural network, $\delta_j^{(l)}$ as "error" of node j in layer l. In neural networks, parameters of the model $\theta_{ij}^{(l)}$ are in fact weights between node i in layer l and node j in layer $l+1$. We also define $\Delta_{ij}^{(l)}$ as "accumulation of error" which will be used to calculate gradient for $\theta_{ij}^{(l)}$. We start with training set of m training examples$(x^{(i)}, y^{(i)})$, and set $\Delta_{ij}^{(l)} = 0$ for all i, j, and l. Then, for each training example we perform the following tasks:

1. Set the current training example as input for neural network, $a^{(1)} = x^{(i)}$
2. Perform forward propagation to compute $a^{(l)}$ for $l > 1$
3. Calculate $\delta^{(L)} = a^{(L)} - y^{(i)}$
4. Calculate errors $\delta^{(l)}$ for $1 < l < L$
5. Calculate $\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^l \delta_i^{l+1}$

After performing this for all m training examples, we easily calculate gradient as:

$$\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta) = \frac{1}{m} \Delta_{ij}^{(l)}. \tag{2.4}$$

With this, we have all that is necessary to successfully train the neural network.

3 Applying machine learning to physics

In solid-state physics, the potential energy surface (PES) is defined as the function yielding the potential-energy of an atomic configuration, if the atomic coordinates are provided [22]. It is one of the most important quantities of chemical systems, and knowing PES we can determine all the other system's properties by different computer simulations. Furthermore, even the reliability of molecular dynamics strongly depends on the accuracy of the calculated PES. The most precise technique for molecular dynamics simulation are *ab initio* methods based on DFT, but those are limited to relatively small number of molecules and are very computationally expensive. Another problem with *ab initio* methods is that information about the PES is not stored, meaning that at each time step the electronic structure problem has to be solved from the beginning. This problem was solved with introduction of empirical PES, in which information from the electronic structure calculations can be stored. That enables us to reduce computational costs and perform simulations with larger length and time scales. However, construction of a reliable empirical potential is a difficult, time-consuming task which often relies on fitting the parameters of a guessed functional form for the interaction potential. Further, numerical accuracy of such calculations is limited by the approximations made while selecting the functional form, and functional form for one system is rarely transferable to another system.

In recent years, a new idea for constructing a DFT-based PESs has emerged. It is based on machine learning methods and using artificial neural networks to construct PESs with *ab initio* accuracy and can describe all types of bonding [23]. Until recently, neural networks have been limited only to low-dimensional PESs [24,25], but by combining neural networks with a PES representation inspired by empirical potentials, this method can be generalized to high-dimensional problems as well. Using this, we can obtain many-body potentials which are functions of all atomic coordinates and can be used on systems of arbitrary size.

In the most simple method which uses neural networks, we start with a set of atom coordinates, send them to neural network with randomly generated weights which calculates total energy of the system. This value is compared to value obtained

with DFT and error is calculated. By repeating this process for many different atomic configuration, we can minimize the error stored in cost function. After this, the neural network is trained and we obtain the optimal weights which give the best prediction of total energy of the model. However, there are several disadvantages in this method, which do not allow using it on high-dimensional systems. The first one is that all weights in neural network are usually different, meaning that order in which the atomic coordinates are fed to the network is not arbitrary. Even if we switch order of coordinates of two identical atoms, different total energy will be obtained. Fixed structure of a neural network also implies that once it is optimized for a certain number of atoms, it can't be used to predict energies of system with another number of atoms. Hence, a new architecture of neural network ought to be built, which is another disadvantage of such model.

3.1 Descriptors

Main idea for solving the mentioned limitations is based on dividing total energy E into atomic contributions E_i [23], as

$$E = \sum_i E_i. \tag{3.1}$$

Same approach is used in empirical potentials. Furthermore, it is easy to see that coordinates of the atoms are not suited to be the input parameters. Hence, instead of coordinates α of atom i $\{R_i^\alpha\}$, descriptors $\{G_i^\mu\}$ are introduced. These descriptors are values of symmetry functions for each atom i, and describe energetically relevant local environment of each atom. They depend on the positions of all atoms, as can be seen by dotted arrows on Figure 3.1, which depicts topology of the new neural network for a system consisting of three atoms and all associated degrees of freedom. Now there is a "standard" neural network S_i for each atom in the system which yields energy contribution E_i. These subnets all have the same structure and weight values, which ensures the invariance of the total energy with respect to interchanging two atoms. Total energy of the system is obtained by summing all the energy contribution, as in Equation (3.1).

It is still important to determine symmetry function necessary to calculate de-

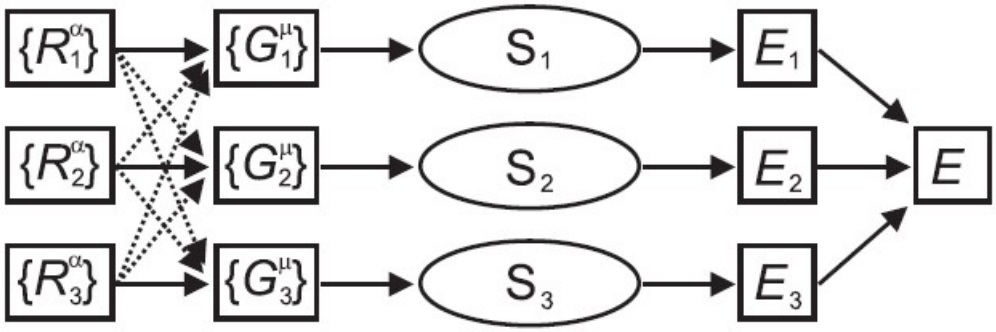

Figure 3.1: Structure of a neural network proposed for generalization to high-dimensional systems. The Cartesian coordinates of atom i are given by $\{R_i^\alpha\}$, and are translated into a set of μ descriptors $\{G_i^\mu\}$ which enter the subnet S_i yielding the energy contribution E_i. Energy contributions are then summed into total energy E. Source: [23].

scriptors. Although there exist some other types of symmetry functions [24], the most popular ones with following features [23]:

- descriptors obtained from identical structures of atoms should yield the same energies,
- symmetry functions should invariant with respect to the translation or rotation of the system,
- number of symmetry functions must be independent of the coordination of the atom, due to possibility of coordination number to change during the molecular dynamics.

One example of such symmetry function is a Gaussian symmetry function introduced by Behler [22, 23]. Prior to defining them, we introduce cutoff function f_c which defines energetically relevant local environment of the interatomic distance R_{ij}. There are also several possible types of cutoff functions, one used by Behler [22, 23]:

$$f_c(R_{ij}) = \begin{cases} 0.5 \cdot \left[cos\left(\frac{\pi R_{ij}}{R_c}\right) + 1 \right] & \text{for } R_{ij} \leq R_c, \\ 0 & \text{for } R_{ij} \geq R_c, \end{cases} \tag{3.2}$$

where R_c is a cutoff radius, which has to be sufficiently large to include several

nearest neighbors. The other, more general one [26, 27]:

$$f_c(R_{ij}) = \begin{cases} 1 + \gamma(R_{ij}/R_c)^{\gamma+1} - (\gamma+1)(R_{ij}/R_c)^{\gamma} & \text{for } R_{ij} \leq R_c, \\ 0 & \text{for } R_{ij} \geq R_c, \end{cases} \tag{3.3}$$

where γ is user-specified parameter that determines the rate of decay of the cutoff function as it extends from $R_{ij} = 0$ to $R_{ij} = R_c$. There are also other possible types of cutoff functions, but in order to have a continuous force-field, the cutoff function and its first derivative should be continuous in R_{ij} [27], such as those defined in Equations (3.2) and (3.3). We can also see that both cutoff functions yield a value one for $R_{ij} = 0$ and zero for $R_{ij} \geq R_c$, meaning that distanced atoms are less energetically relevant.

Finally, we can define Gaussian symmetry functions that were mentioned previously. Radial symmetry functions are constructed as sum of Gaussian with the parameters η for width and R_s for mean value:

$$G_i^1 = \sum_{j \neq i} e^{-\eta(R_{ij} - R_s)^2/R_c^2} f_c(R_{ij}). \tag{3.4}$$

These descriptors capture the interaction of atom i with all atoms j as the sum of Gaussians with width η and center at R_s. Summing over all j ensures the independence of coordination number. Angular terms describe three-atom interactions, and are constructed for all triplets of atoms i, j and k by summing over the cosine values of the angles $\theta_{ijk} = \cos^{-1}\left(\frac{\vec{R}_{ij} \cdot \vec{R_{jk}}}{R_{ij} R_{jk}}\right)$ centered at atom i, with $\vec{R}_{ij} = \vec{R}_i - \vec{R}_j$:

$$G_i^2 = 2^{1-\zeta} \sum_{j,k \neq i} (1 + \lambda \cos\theta_{ijk})^{\zeta} e^{-\eta(R_{ij}^2 + R_{ik}^2 + R_{jk}^2)/R_c^2} f_c(R_{ij}) f_c(R_{ik}) f_c(R_{jk}), \tag{3.5}$$

with parameters $\lambda = \pm 1$, η and ζ. These descriptors can be seen on Figure 3.2, where they are plotted for a few different parameters. Three cutoff functions and the Gaussian ensure smooth decay to zero in case of large interatomic separations. Gaussian descriptors are the ones that are most commonly used, but some other descriptors exist as well, such as three-dimensional Zernike [28] and four-dimensional bispectrum [29] descriptors.

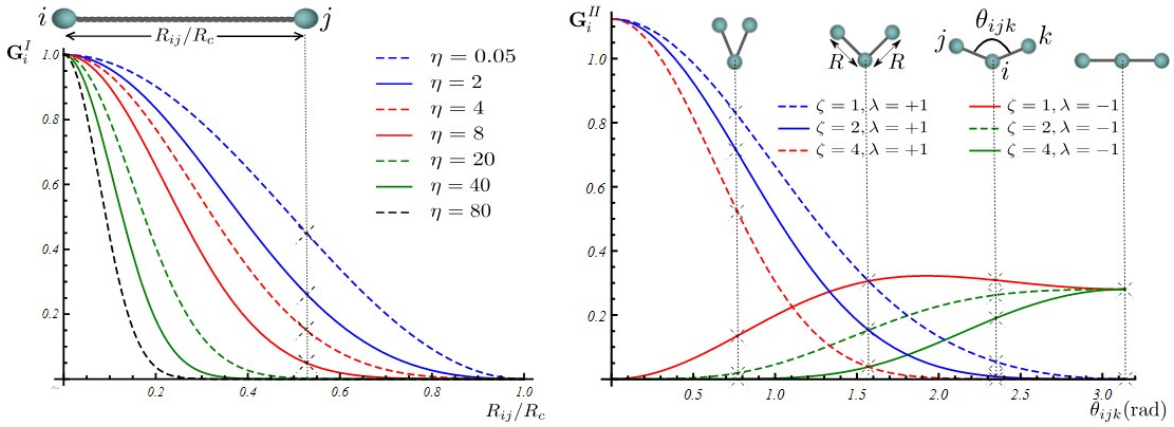

Figure 3.2: Radial (left) and angular (right) Gaussian descriptors for a few different parameters. Source: [27].

Zernike descriptor components for each integer degree are defined as norm of Zernike moments with the same corresponding degree, while Zernike moments are products between spherical harmonics and Zernike polynomials. For calculating Zernike moments, the density function $\rho_i(\vec{r})$ is defined for each atomic level environment as:

$$\rho_i(\vec{r}) = \sum_{j \neq i} \eta_j \delta(\vec{r} - \vec{R}_{ij}) f_c(\|\vec{R}_{ij}\|_2). \tag{3.6}$$

This density function represents local chemical environment of atom i, and η_j is a weight parameter. This type of descriptors can be seen on Figure 3.3. Bispectrum of four-dimensional spherical harmonics are invariant under rotation of local atomic environment, as has been suggested by Bartok et al. [29]. Hence, it was shown that bispectrum of mapping the atomic distribution in Equation (3.6) to the four-dimensional unit sphere can also be used as descriptor.

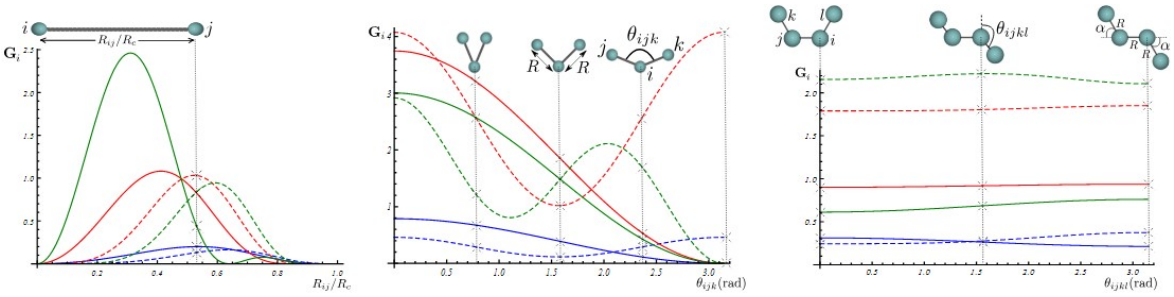

Figure 3.3: Dependence of Zernike descriptors on pair-atom distance (left), three-atom angle (middle), and four-atom dehidral angle (right). Source: [27].

Finally, this method with neural networks and Gaussian descriptors was tested in [23] on a silicon bulk. For training the neural network the DFT energies were used and obtained root-mean-square error of about 5 meV. It was also shown that number of DFT energies needed for optimizing the neural network was large in comparison to empirical potentials, but once trained neural networks can be used without modifications even if new DFT data is provided. While this method appears to be general, meaning it can applied to crystals, liquids, surfaces, etc., it lacks extrapolation abilities to structures different than those in training set. Neural networks are also easily parallelized which enables them to calculate energies much faster than the DFT, while the accuracy seems to be limited only by the accuracy of DFT energies used in training set. All this results seem like a giant breakthrough in studying the molecular dynamics of large systems on long scales, and provide a motivation for further research of application of neural networks and other machine learning techniques in physical and chemical problems that have not been successfully solved by now.

4 Data and software used

4.1 Data

The system we will be studying consists of 48 ruthenium atoms and 16 hydrogen atoms. Ruthenium atoms are arranged in a slab with dimension $4 \times 4 \times 3$, while hydrogen atoms are added on the surface. This system can be seen on Figure 4.1. The importance of interaction between ruthenium and hydrogen, and their application in Fischer-Tropsch process has already been debated in introduction. Due to this, systems such as that seen on Figure 4.1, are undergoing thorough research, both experimentally [30] and with help of computer simulations [31]. In [30], Denzler et al. investigated a mechanism of recombinative desorption of hydrogen from a Ru(0001) surface, induced by a femtosecond-laser excitation. This mechanism was then compared to thermally initiated desorption. It was shown that, for the laser driven process, the hot substrate electrons mediate the reaction within a few hundred femtoseconds resulting in a huge isotope effect between H_2 and D_2 in the desorption yield. In mixed saturation coverages, it was shown that this ratio crucially depends on the proportions of H and D. Denzler et al. have also propsed a concentration dependent rate constant k which accounts for the faster excitation of H versus D, but also state that a crucial test for that proposition would be *ab initio* calculation.

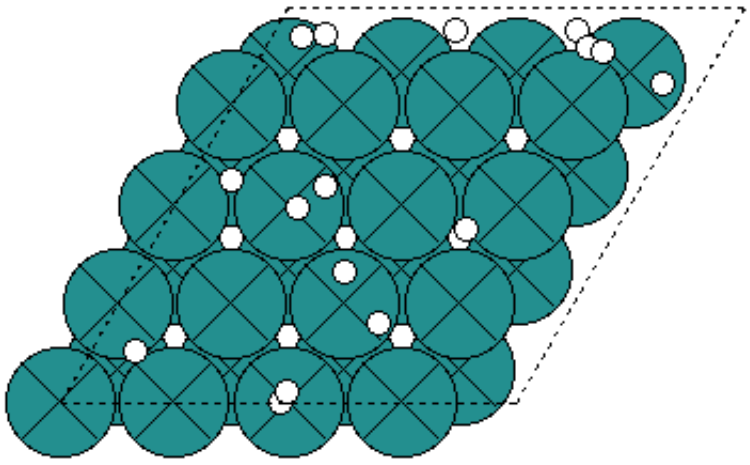

Figure 4.1: A depiction of a slab of 48 ruthenium atoms (blue circles) and 16 hydrogen atoms (white circles).

This same system describing desorption of H_2, D_2 and HD from a H:D-saturated Ru(0001) surface was also simulated using *ab initio* molecular dynamics extended with electronic friction by Juaristi et al. in [31]. While performing *ab initio* molecular dynamics, DFT was used at every step to obtain energy and forces acting in the system. This way, they obtained more than 1 million DFT energy points stored in OUTCAR files (339 trajectory files, each containing 8000 points). Data from one trajectory file can be seen on Figure 4.2. This is the same dataset we have started with. In order to train the neural network and find optimal architecture, we have divided the initial dataset into three parts: training set, validation set and test set. Training set included data from 240 trajectory files, validation set from 50 files, and test set from the remaining 50 files. However, we didn't take all points from trajectory files, but only every 100th point. The reason for this is that we didn't want to have correlated data, since the points are obtained from molecular dynamics simulation and neighoboring steps would have very similar energies which is unfit for training the neural network. Furthermore, for each dataset we will use, first 2000 points from the OUTCAR files were read only from the OUTCAR file first used for obtaining the dataset, while afterwards they were skipped. This is due to the fact that first 2000 steps are almost identical in every trajectory file, and by including them every time we would get too many similar points in the used datasets. Finally, we have obtained three datasets which we will use, training set containing $\sim$ 11000 points, validation set of $\sim$ 3000 points and test set of also $\sim$ 3000 points.

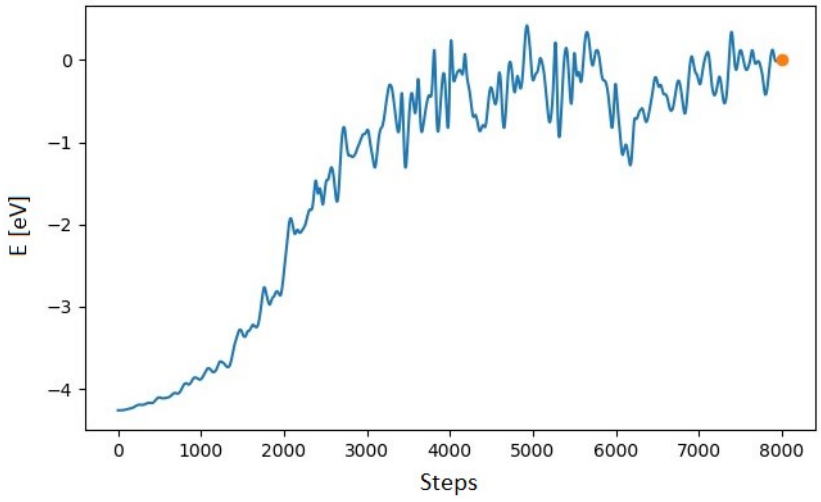

Figure 4.2: A plot showing total energy of the system E for each step in trajectory file. Plot obtained using [35].

4.2 Software

In this work, all codes have been written in Python, an object-oriented, interpreted programming language [32]. Aside from standard Python functionality, use has also been made of NumPy package for numerical computing and Matplotlib. NumPy package is fundamental for scientific computing in Python, which contains many useful tools for linear algebra, random number generating, integration with C/C++ and Fortran codes, and also efficient implementations of high dimensional arrays [33]. Matplotlib is a 2D plotting library that can easily produce high-quality plots, histograms, scatterplots, bar charts etc, and it provides you with full control over line styles, fonts and other features. However for analyzing DFT data, creating neural networks and training them, we needed packages designed specifically for that purpose. Thus, we have used ASE and Amp packages for Python.

4.2.1 Atomic Simulation Environment

The Atomic Simulation Environment (ASE) is an open-source set of command-line tools and Python modules for setting up, manipulating, running, visualizing and analyzing atomistic simulations [35]. These command-line tools can be used to obtain graphical user interface, manipulate ASE-database, build molecular crystals and run simple calculations. Central object in ASE package is `Atoms`, which defines a collection of atoms and specifies their positions, atomic numbers, velocities, masses, charges etc., but also unit cell of a crystal and boundary conditions. To `Atoms` object we can attach a `Calculators` object with purpose of calculating energies and forces on the atoms. ASE provides many different calculators, some of which are based on DFT, such as Abinit [36], NWChem [37] and Gaussian [38], while some others like ASAP [39] are based on empiricial potentials. In this work, the DFT data was obtained with VASP [40] calculator and read with help of ASE package.

4.2.2 Atomistic Machine-learning Package

Atomistic Machine-learning Package (Amp) is another open source package for Python designed to easily bring machine-learning to atomistic calculations [27]. This allows us to predict calculations on the potential energy surface. First step is to make Amp calculator learn from any other calculator on a training set of atomic images. Af-

terwards, Amp calculator is trained and can make predictions on energy and forces with arbitrary accuracy, approaching that of the original calculator. One of the greatest features of Amp is that it was designed to integrate closely with the ASE package, meaning that any calculator that works in ASE can easily be used as the parent method in Amp. Thus, we can use Amp calculator as a substitute for some other calculator in molecular dynamics, global optimization, phonon analyses etc.

Idea behind Amp is that, as discussed in previous section, potential energy of an atomic configuration is specified solely by the coordinates of the nuclei. While DFT calculations provide good approximation to the ground-state potential energy, these calculations are expensive for large systems and/or long dynamics. Thus, Amp takes a different approach and, given enough example calculations from any electronic structure calculator, approximates potential energy with a much faster regression model:

$$\mathbf{R} \xrightarrow{\text{Regression}} E(\mathbf{R}). \tag{4.1}$$

Another advantage of Amp is that most of its tasks are suitable for parallel processing, which gives a great performance boost if running calculations on multi-core processors or computer clusters. This is, for example, really suitable for training neural networks, which are the central regression model in Amp package.

5 Results

First part of this work consists of training multiple neural networks with different architectures, in order to find the optimal one. Here we use Gaussian descriptors, with cutoff radius of 6.5 Å, and feed-forward neural networks. There are 40 descriptors in total used as input for neural networks, 20 for each type of atom. Role and importance of these descriptors will be discussed in second part of the work. In third and final part of the work, we will investigate some alternative possibilities for training the neural networks, such as using Zernike descriptors and lowering cutoff radius.

5.1 Finding the optimal neural network

After specifying what calculator will be used, we have to decide on an architecture of a neural network. it is almost impossible to know *a priori* which architecture will yield the best results, so it is necessary to train several neural networks on test set and check their performance on validation set to see which one produces the smallest error. After we find the optimal one using validation set, we run it one more time on the test set, and obtain final error.

First we have trained neural networks with two hidden layers and equal number of nodes in each hidden layer. Number of nodes ranged from 2 per layer to 50 per layer. Results of these calculations can be seen on Figures 5.1 and 5.2. We can see how at low number of nodes per layer root-mean-square (RMS) error is much higher than at other architectures. This is a sign of the underfitting problem, meaning that architecture of neural network is too simple and cannot reproduce a function that would properly fit the training data. Interesting fact is that for higher number of nodes per layer, error doesn't change much.

Afterwards, we have examined neural networks with three hidden layers, again with equal number of nodes per layer. Again, for each architecture we have calculated maximal and RMS error, both total and per hydrogen atom. These calculations can be seen on Figures 5.3 and 5.4. Once again, we notice how both maximal and RMS error barely change with increase in number of nodes per layer. However, underfitting is visible for 2 nodes per hidden layer, and for 50 nodes per hidden layer we

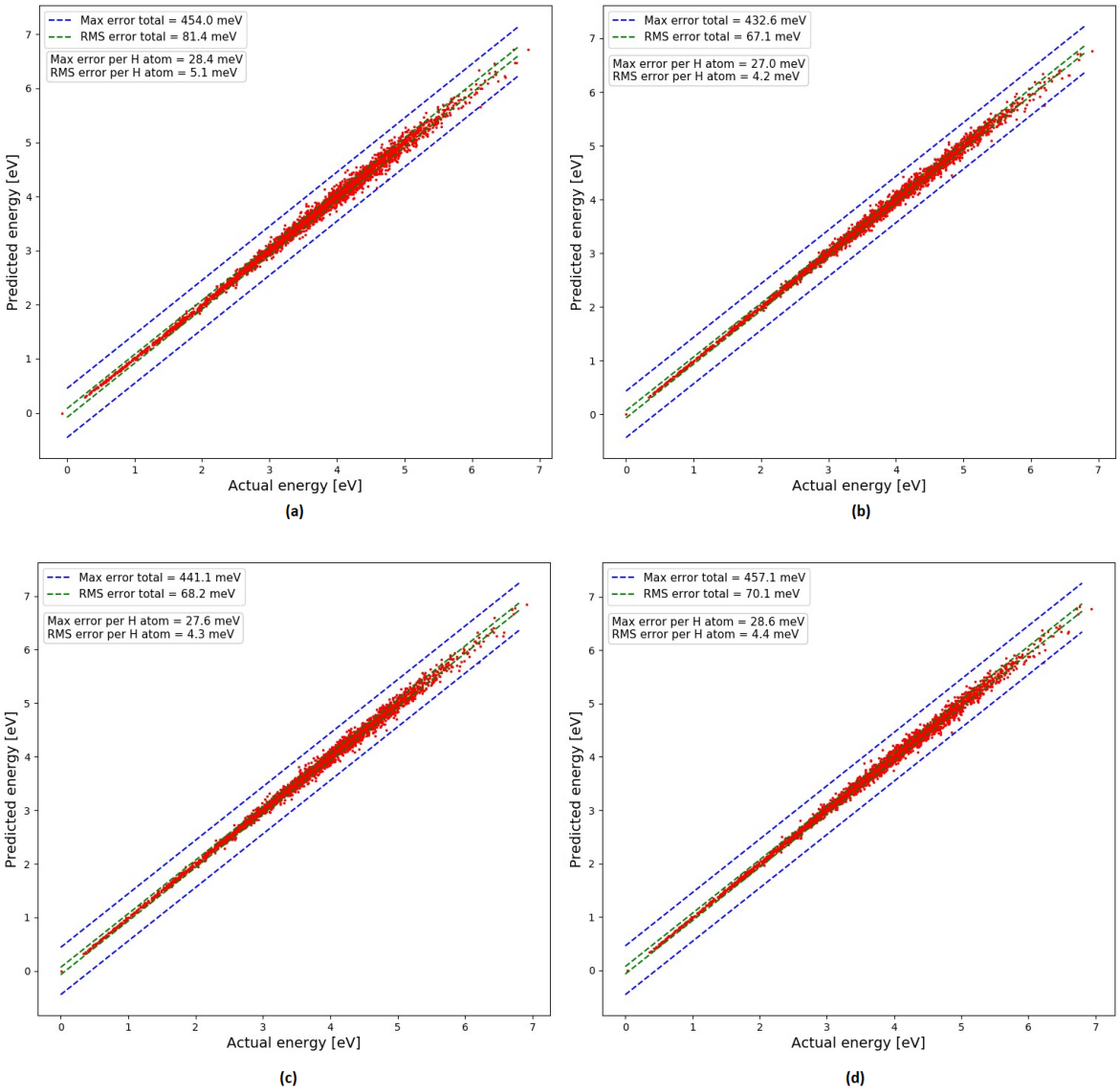

Figure 5.1: Plots showing how total energy of the system predicted by neural networks differs from actual energy obtained with DFT, for different neural network architectures: (a) 2-2, (b) 5-5, (c) 10-10, (d) 20-20.

see signs of overfitting, as both maximal and RMS error are much higher than before.

Some other architectures, with different number of nodes per layer, have also been tested and it was found that the optimal configuration is the one with three hidden layers and 50 nodes in the first layer, 30 in the second, and 10 in the third. This is reasonable, because to the neural network we feed the previously mentioned descriptors, meaning that input layer has 40 nodes. Output, however, is just the total energy of the system, meaning that output layer has only one node. Thus, we need enough nodes in the first hidden layer to successfully process all the information stored in the descriptors, but need just a few nodes in the last hidden layer in order

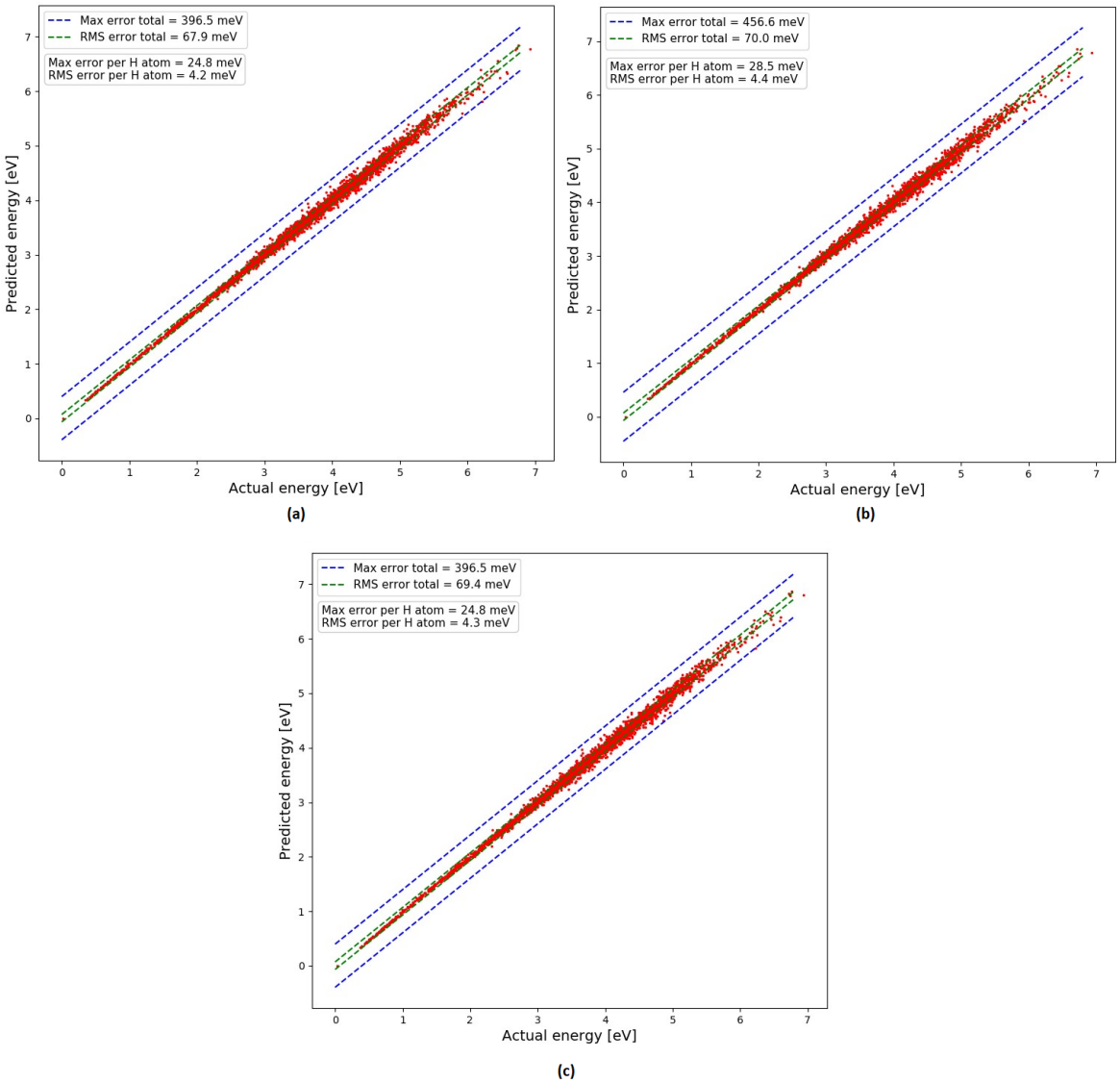

Figure 5.2: Plots showing how total energy of the system predicted by neural networks differs from actual energy obtained with DFT, for different neural network architectures: (a) 30-30, (b) 40-40, (c) 50-50.

to produce a valid single-value output. On validation set, errors per hydrogen atom obtained are following:

$$\text{Max error per H atom} = 23.6 \text{ meV}, \tag{5.1}$$

$$\text{RMS error per H atom} = 4.3 \text{ meV}. \tag{5.2}$$

From Table 5.1, where one can systematically see errors per hydrogen atom obtained for all previously mentioned architectures of neural networks, it is easy to see these are the lowest errors obtained on the analyzed architectures. After running this

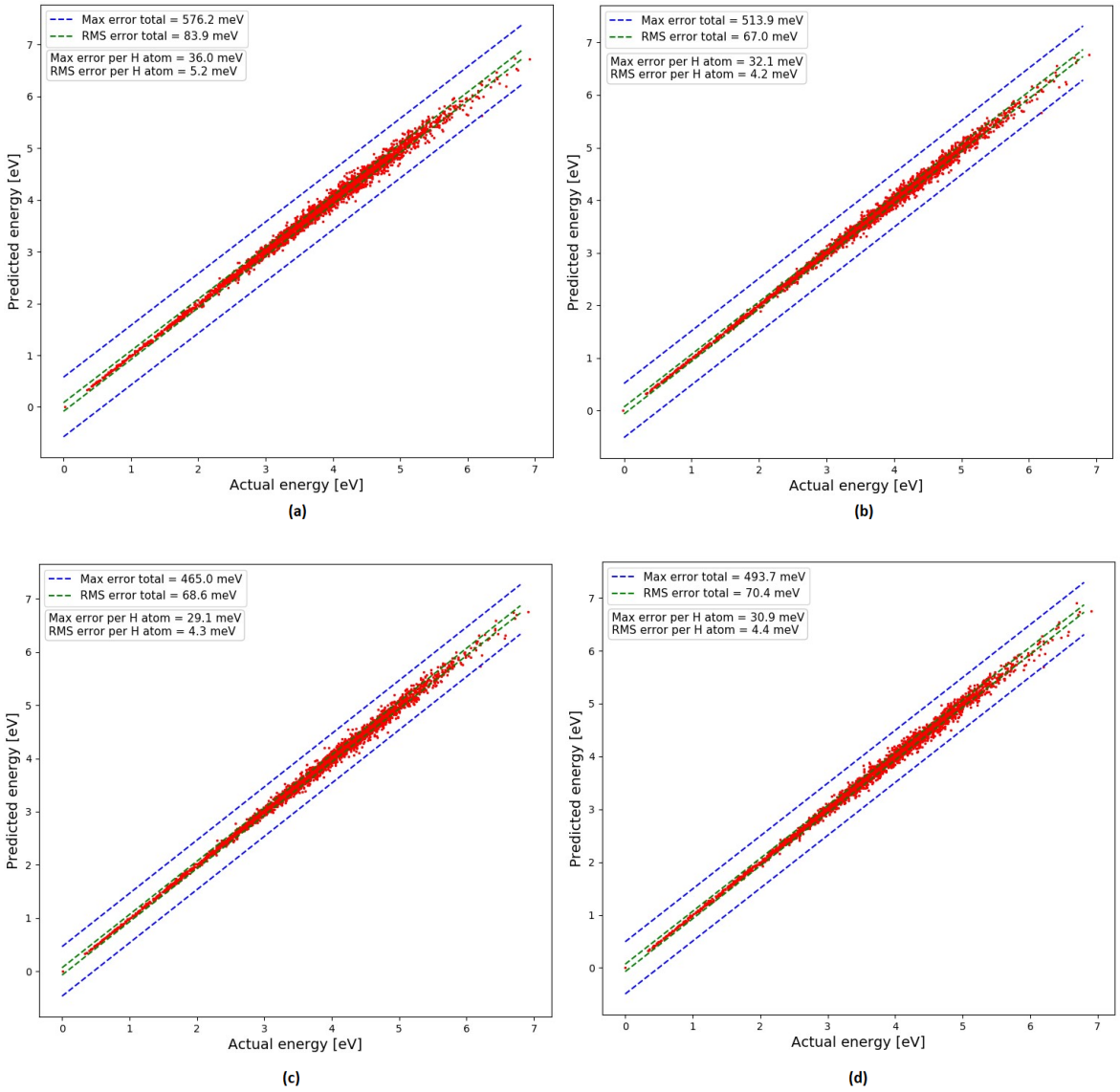

Figure 5.3: Plots showing how total energy of the system predicted by neural networks differs from actual energy obtained with DFT, for different neural network architectures: (a) 2-2-2, (b) 5-5-5, (c) 10-10-10, (d) 20-20-20.

configuration on the test set, we obtain the total errors:

$$\text{Max error total} = 438.8 \text{ meV}, \tag{5.3}$$

$$\text{RMS error total} = 55.3 \text{ meV}. \tag{5.4}$$

and errors per hydrogen atom:

$$\text{Max error per H atom} = 27.4 \text{ meV}, \tag{5.5}$$

$$\text{RMS error per H atom} = 3.5 \text{ meV}. \tag{5.6}$$

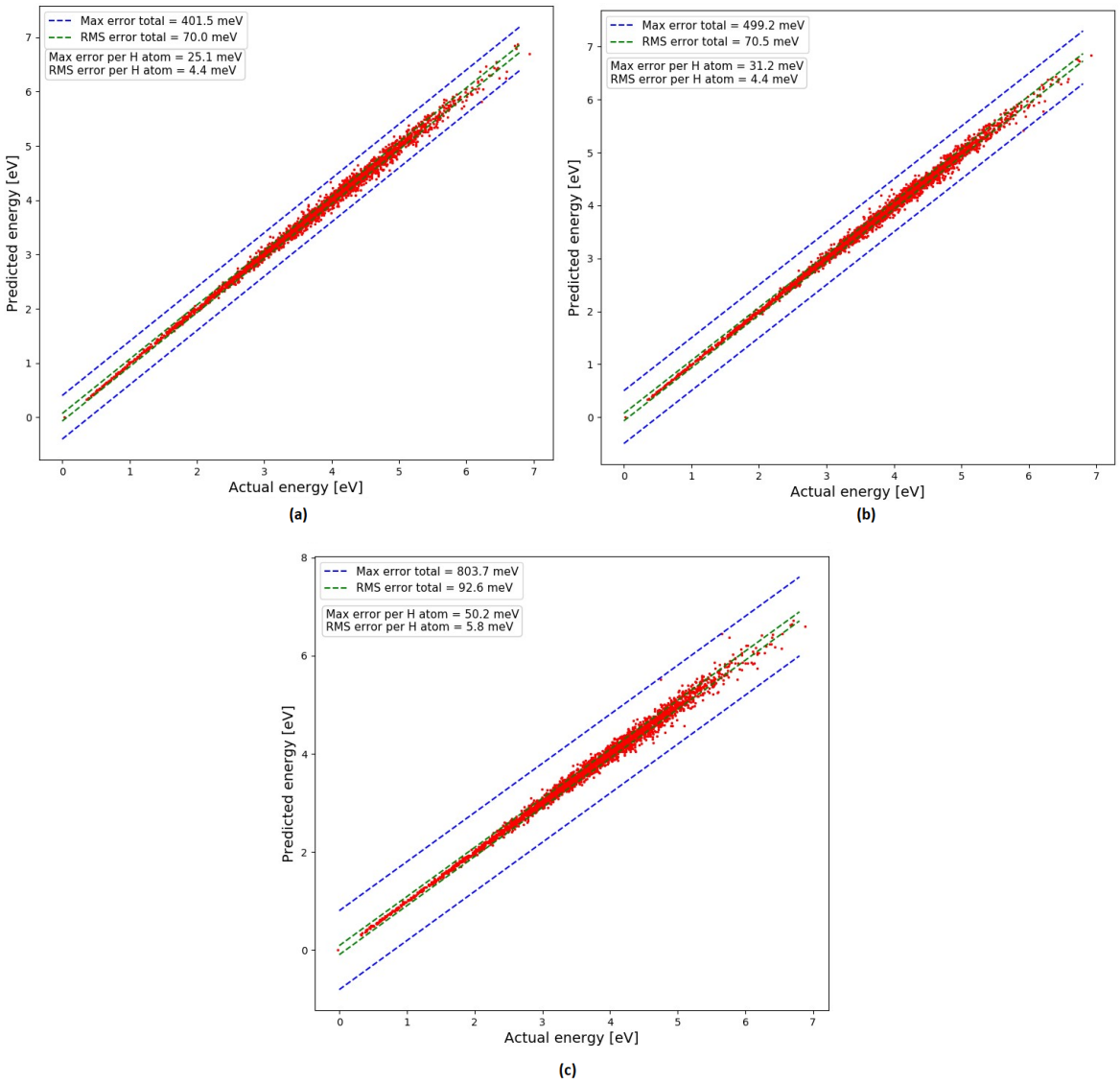

Figure 5.4: Plots showing how total energy of the system predicted by neural networks differs from actual energy obtained with DFT, for different neural network architectures: (a) 30-30-30, (b) 40-40-40, (c) 50-50-50.

Errors per hydrogen atom are of interest to us because they are the only moving parts of the system; ruthenium atoms are fixed. Hence we could, in principle, construct a system with arbitrary number of atoms and perform the same analysis as here. This way, if we change number of hydrogen atoms, we can no longer compare goodness of fit based on the total error, but must use scaled errors. Results of fitting the regression model to the test set can also be seen on the Figure 5.5.

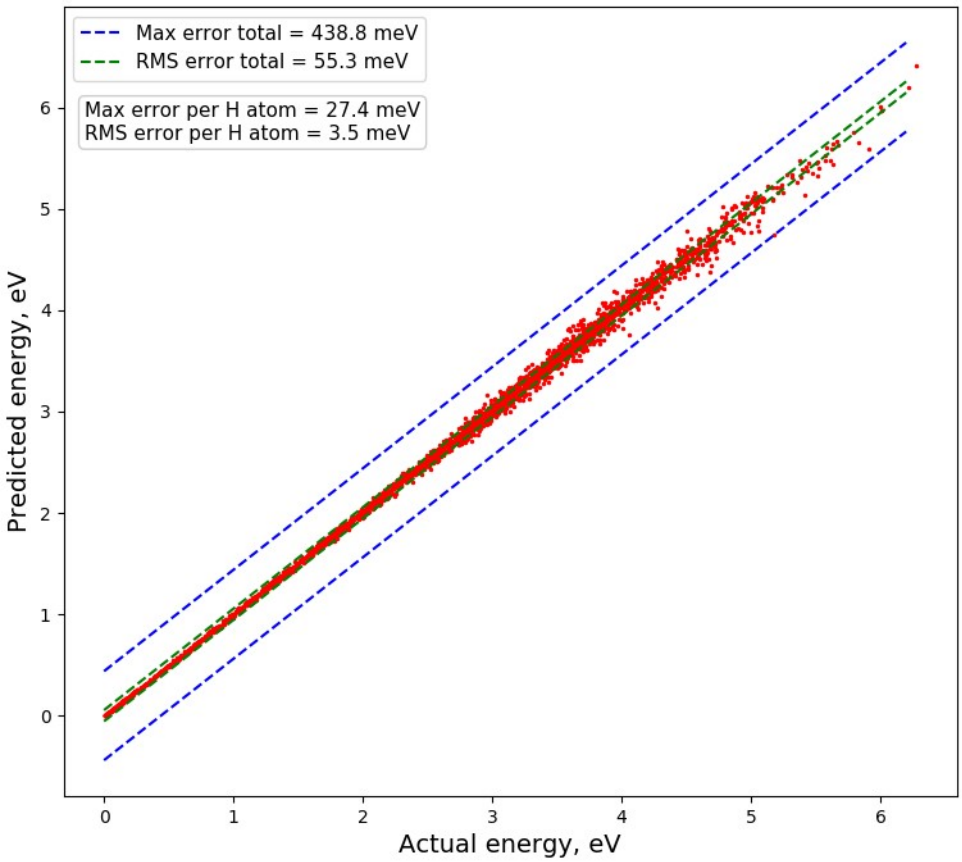

Figure 5.5: Results of fitting the regression model obtained from training the 50-30-10 neural network on the test set.

Neural network	Max error per H atom [meV]	RMS error per H atom [meV]
2-2	28.4	5.1
5-5	27.0	4.2
10-10	27.6	4.3
20-20	28.6	4.4
30-30	24.8	4.2
40-40	28.5	4.4
50-50	24.8	4.3
2-2-2	36.0	5.2
5-5-5	32.1	4.2
10-10-10	29.1	4.3
20-20-20	30.9	4.4
30-30-30	25.1	4.4
40-40-40	31.2	4.4
50-50-50	50.2	5.8
50-30-10	23.6	4.3

Table 5.1: Maximal and RMS errors for neural networks which were used in this problem. All calculations in the table were performed on the validation set.

5.2 *Analyzing the optimal configuration*

Now when we have found optimal neural network for the problem we are solving, we want to investigate properties of this network. For example, one aspect worth looking into are descriptors and what role they play in this problem. As previously stated, there are 20 descriptors per type of atom. Descriptors for hydrogen atoms can be seen on Figure 5.6, while those for ruthenium can be seen on Figure 5.7.

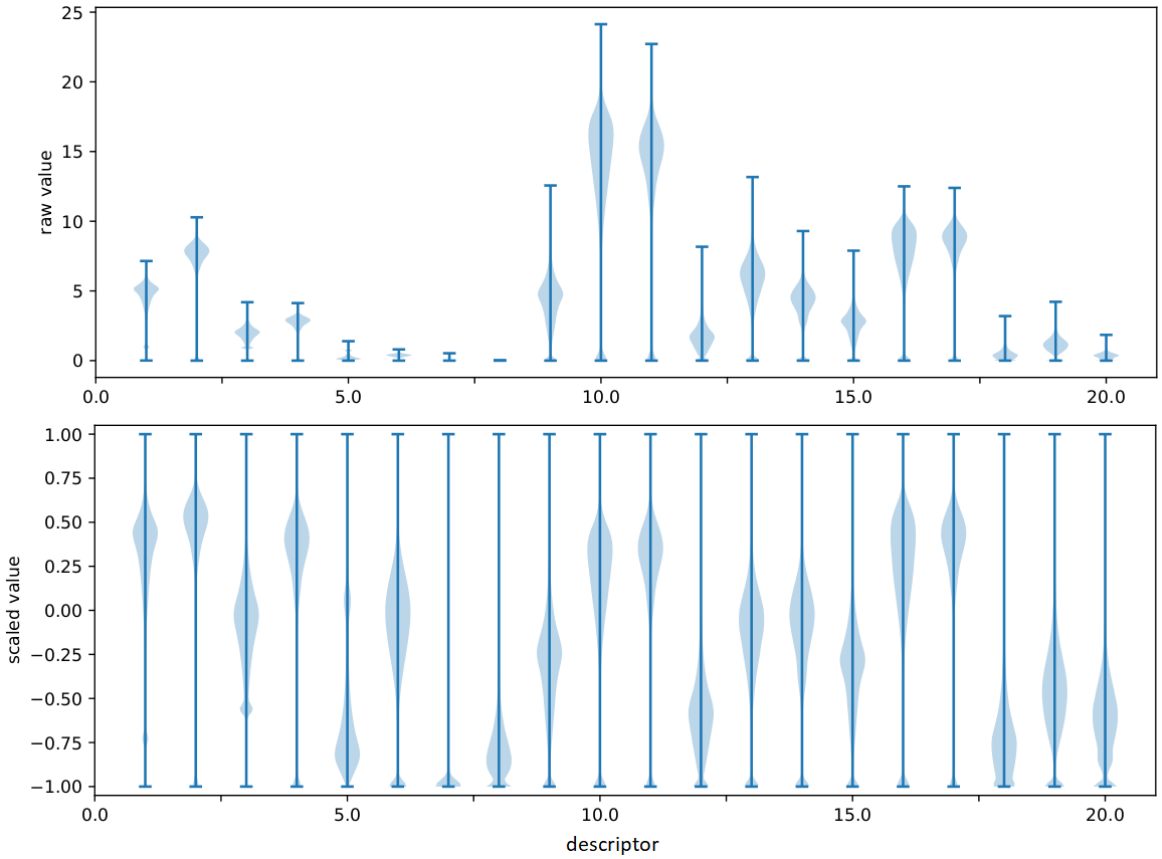

Figure 5.6: Plots showing how total energy of the system predicted by neural networks differs from actual energy obtained with DFT, for different neural network architectures: (a) 2-2-2, (b) 5-5-5, (c) 10-10-10, (d) 20-20-20.

On both figures, we can see which descriptors are the most important, and which are irrelevant. In case of hydrogen descriptors, it is easy to see that descriptors with index 10 and 11 are crucial for the observed system, while those with index 7 and 8 could be ignored. Investigating the log file produced by Amp calculator, it is easy to see which descriptor represents which function. Descriptors with indexes from 1 to 8 represent radial Gaussian descriptors representing two-atom interactions, while those from 9 to 20 are angular descriptors which account for three-atom interactions.

By reading parameters from the log file, we can also write descriptors that are of our interest explicitly. Those unimportant descriptors, with indexes 7 and 8, are radial descriptors G_i^1 with parameter η=80.0, where G_7^1 describes interaction between two hydrogen atoms, while G_8^1 describes interaction between hydrogen and ruthenium atoms. Cutoff radius used for these calculations was $R_c = 6.5$ Å. The other two hydrogen descriptors of out interest, G_{10}^2 and G_{11}^2, both have the parameters η=0.005, λ=1.0 and ζ=1.0, whereas G_{10}^2 describes three atom interaction between two hydrogen and one ruthenium atom and G_{11}^2 describes interaction between one hydrogen and two ruthenium atoms.

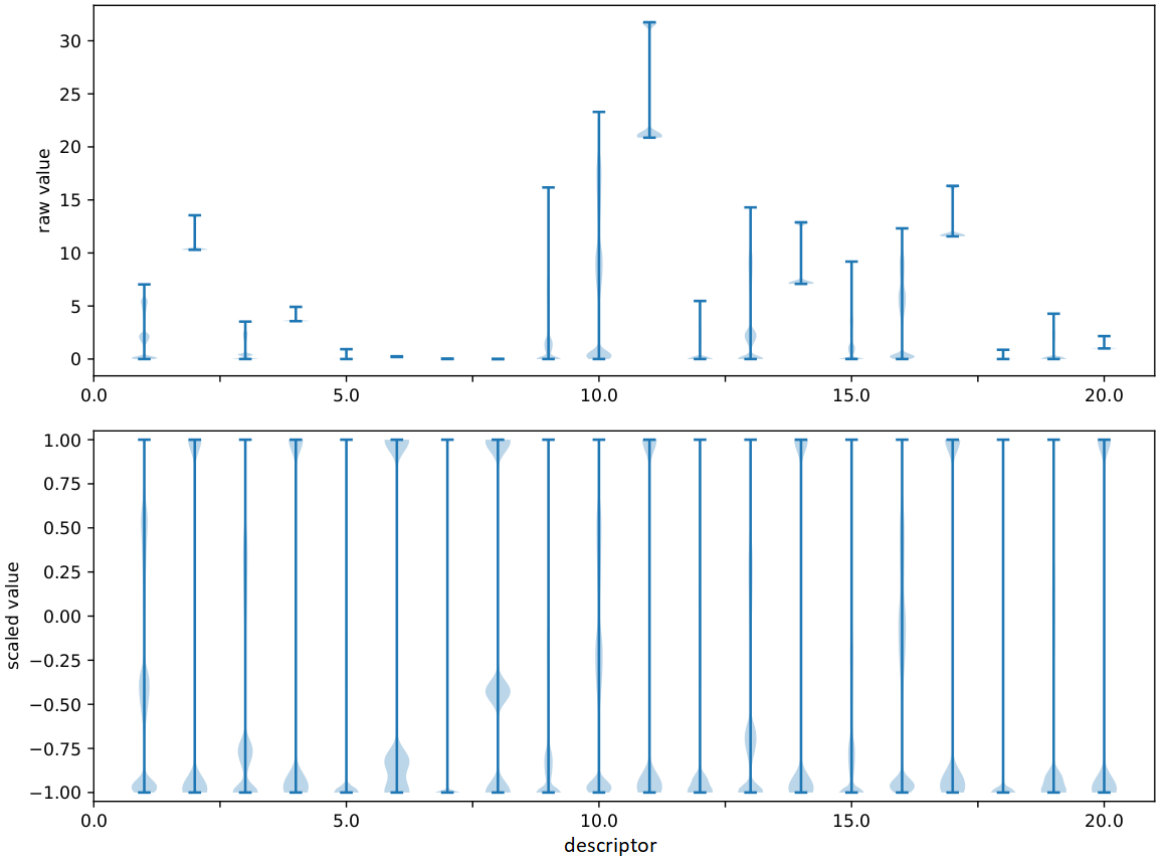

Figure 5.7: Plots showing how total energy of the system predicted by neural networks differs from actual energy obtained with DFT, for different neural network architectures: (a) 2-2-2, (b) 5-5-5, (c) 10-10-10, (d) 20-20-20.

Inspecting the Figure 5.7, we see a large difference with respect to the previous case. Most of the ruthenium descriptors seem like they are insignificant in this problem. This statement is logical, if we take a look at the system we are analyzing. It consists of hydrogen atoms interacting with a slab of ruthenium atoms, which are

all fixed in space. Since all of the ruthenium atoms are fixed, it is reasonable that their descriptors won't be crucial for simulating the system. Even so, we recognize that again descriptors with indexes 10 and 11 play a more important role than the others, especially 5, 6, 7, and 8. Same as before, from 1 to 8 are radial descriptors, while the rest are angular. Descriptors with the index 5 and 6 have parameter η=20.0, and 7 and 8 have $\eta = 80.0$. Descriptors 5 and 7 represent interaction of ruthenium with hydrogen, while 6 and 8 represent interaction between two ruthenium atoms. Descriptors indexed with 10 and 11 have, same as before, parameters η=0.005, λ=1.0 and ζ=1.0. In analogy with the previous case, G^2_{10} represents interaction of ruthenium atom with one ruthenium and one hydrogen atom, while G^2_{11} represents interaction of three ruthenium atoms.

Another property of the model worth looking into is how it converges with the number of training steps. If the model is not good enough, i.e. if our neural network has not enough hidden layers and nodes per layer, it will fail to converge and we will face the problem of underfitting. However, if our model is "too good", with each new training step cost function calculated on the training set will become lower and lower, but if calculated on the validation set, at some point it could start to rise. This means we have entered the regime of overfitting, and should have stopped at some value of cost function, i.e. at some lower number of steps. How cost functions calculated on training and validation set varies with number of training steps can be seen on Figure 5.8.

On this figure we can see that the cost function calculated on the training set, marked with blue dots, continues to fall with number of steps, while cost function calculated on validation set and marked with red crosses, starts to rise somewhere around 2500 steps. This point is marked with black arrow, and represents optimal number of steps required for training the 50-30-10 neural network. Hence, points on the plot left from the optimum correspond to he underfitting regime, while those right of the optimum correspond to the overfitting regime.

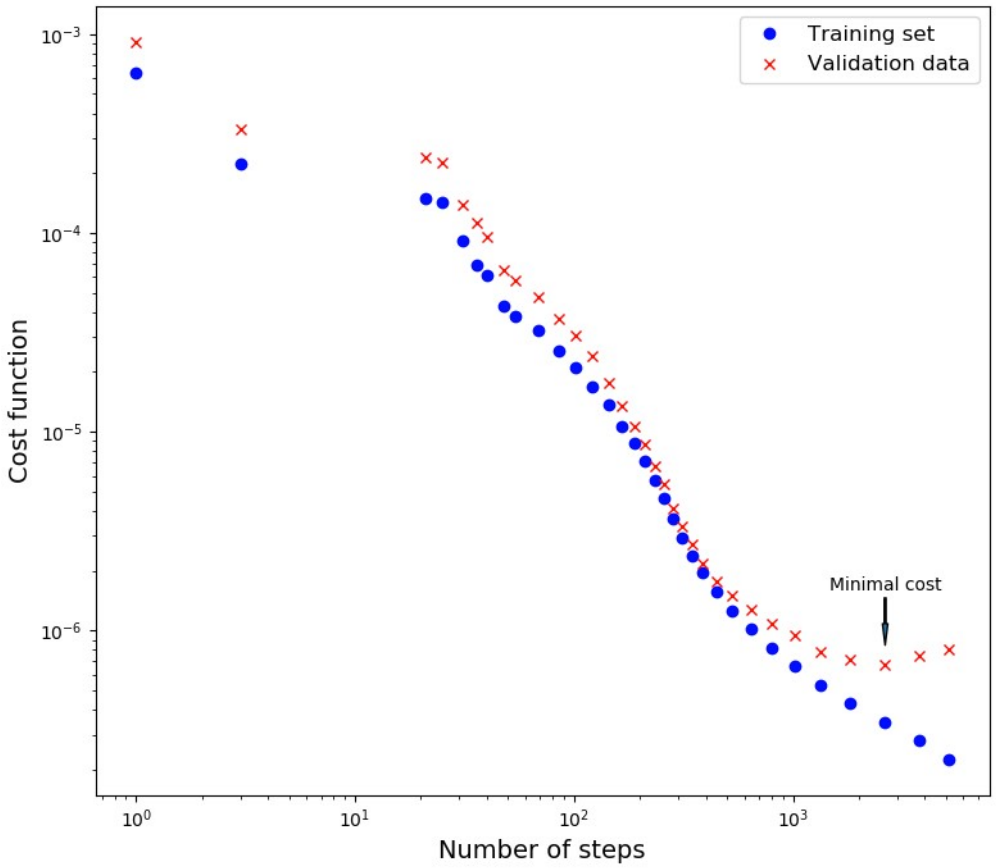

Figure 5.8: Dependence of cost function calculated on the training set (blue) and validation set(red). Arrow marks the spot of minimal cost function calculated on the validation set.

5.3 Alternative approaches

From previous discussion, we see that there are many alternative ways to solve this problem. One of them, and probably the most obvious one, is using different descriptors as input for neural network. In previous simulations, we have used Gaussian descriptors, but now we will solve the same problem with Zernike descriptors instead. Aside from two and three-atom interactions, which are taken into account in Gaussian descriptors, Zernike descriptors calculate four-atom interactions as well. Hence, we expect these calculations to be more accurate, but also to require much more time. Results can be seen on Figure 5.9. Obviously, this is not in agreement with expectations, as errors are huge compared to the errors obtained with Gaussian descriptors:

$$\text{Max error} = 5839.3 \text{ meV} \tag{5.7}$$

$$\text{RMS error} = 290.7 \text{ meV} \tag{5.8}$$

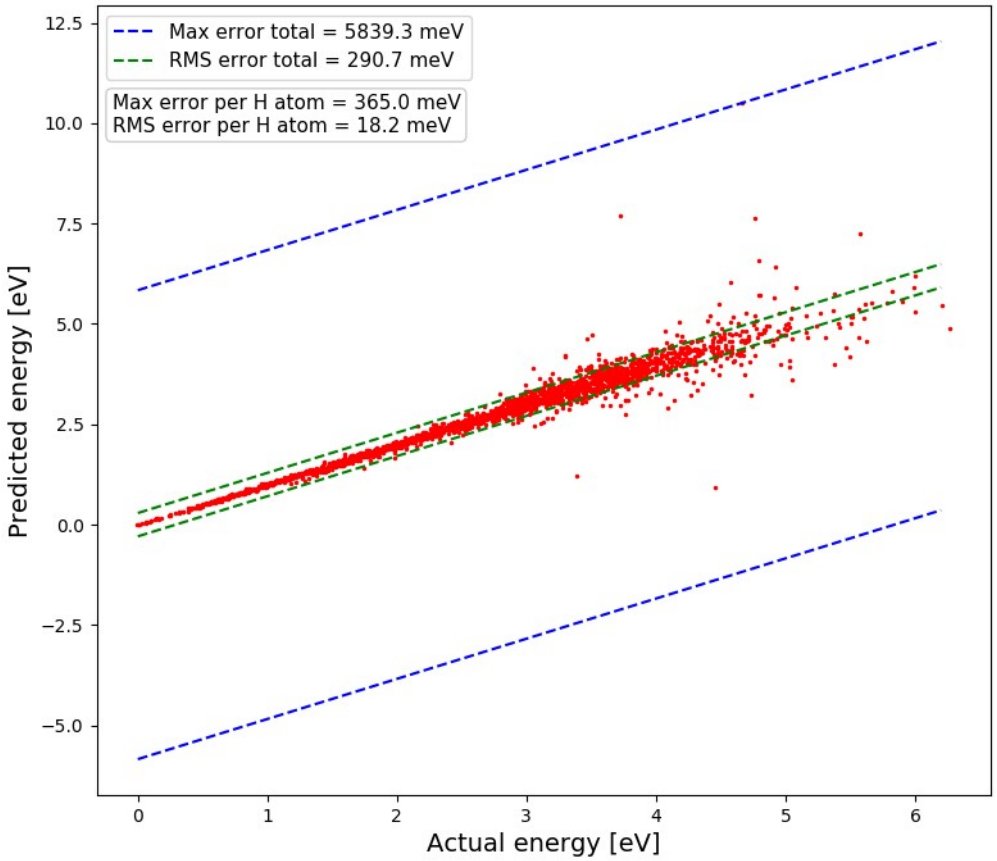

Figure 5.9: Results of fitting the regression model with Zernike descriptors on a test set.

Reason for this is that, while neural networks work well for interpolation problems, they fail often fail to extrapolate data. Thus, even though we used more advanced form of descriptors, we tried to calculate energies for configurations that were not learned well by the network. One possible solution for this is to gather more data. With larger and more diverse training dataset, more configurations would be learned and our regression model could be more widely used.

Another parameter worth studying is cutoff radius. It determines how close atoms have to be in order to be taken into account while calculating descriptors. Thus, by lowering the cutoff radius we expect results to be worse than before. This was tested on the 50-30-10 neural network with Gaussian descriptors and cutoff radius of 5.5 Å (it can be seen on Figure 5.10(a)), while previously it was 6.5 Å, and can once again, for the sake of comparison, be seen on Figure 5.10(b). Total maximal

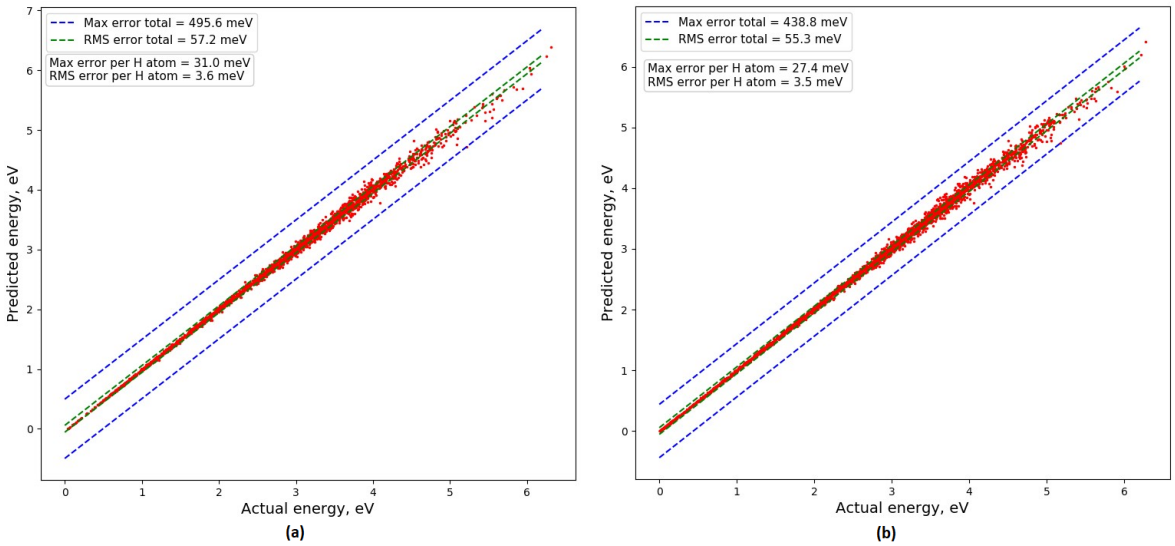

Figure 5.10: Results of fitting the regression model with Gaussian descriptors on a test set, using cutoff radius of (a) R_c=5.5 Å, (b) R_c=6.5 Å.

and RMS errors obtained with cutoff of 5.5 Å are:

$$\text{Max error} = 495.6 \text{ meV} \tag{5.9}$$

$$\text{RMS error} = 57.2 \text{ meV} \tag{5.10}$$

Although they do not differ much from previous results, we see that both errors are slightly higher then with $R_c = 6.5$ Å, which is in agreement with our assumption. However, although results are slightly worse, calculation time was much faster than previously. Thus, in some cases, this could be preferable option.

With all this in mind, it is easy to see how further improvements can be made on this method, and that various modifications are possible, depending on whether we need more speed or accuracy. We have shown that using machine learning can faithfully produce energies of the system, where accuracy is limited only by the accuracy of the training set. With this regression model which yields total energy of the system from its configuration, it is possible to perform molecular dynamics simulations, which is left for future research.

6 Conclusion

For simulating molecular dynamics, the most commonly used technique is density functional theory. However, when it is performed on systems with large number of particles and simulated over long period of time, the simulations become very slow and computationally expensive. Hence, in this work we study an alternative approach to simulating large systems through longer periods. This approach is based on machine learning methods, or more specifically on obtained the regression model by training the neural networks. Crucial point that enables neural networks to solve such problems is introduction of descriptors, functions which take Cartesian coordinates of the atoms in the system and translate them in a form which is more suitable to be used as input for the network.

System upon which we are performing simulations consists of slab of ruthenium interacting with hydrogen atoms. Such system is of great importance in Fisher-Tropsch process which could potentially be used for obtaining low-sulfur diesel fuel, because ruthenium is one of the most promising catalysts for this process. Hence, various research has already been done on this system, both experimentally and theoretically. One such theoretical research is based on simulation using *ab initio* molecular dynamics, in which DFT was used at every step to obtain energy and forces acting in the system. DFT data that was obtained in this research [31], was the same dataset that we have used for training and testing the neural networks. However, one of the great advantages of a method proposed here is that, once the neural networks is trained on the known data, it can easily be used for other similar systems, without need of performing new DFT calculations.

In this work, first we have trained numerous neural networks with Gaussian descriptors and different architectures in order to determine which one suits us best. All these networks were first trained on a training set, their performances were compared on validation set and finally, after finding the optimal one, it was used on test set to obtain the maximal and root-mean-square errors that we could expect when applying this network to some completely new data. Afterwards, we have analyzed what roles do descriptors play in the trained regression model, which are more im-

portant, and how cost functions calculated on the training and the test set behave with respect to number of training steps. Finally, we have studied how changing some properties of the model changes the obtained results.

We have found that optimal neural network architecture for this problem is the one with three hidden layers, with 50, 30, and 10 nodes in first, second and third hidden layer, respectively. Upon analyzing descriptors, we concluded that most of the descriptors for ruthenium atoms are unimportant and could be left out or replaced by some other functions. This is due to the fact that in our system ruthenium atoms remain fixed. Moreover, we have managed to show how cost function calculated on the training set falls with number of training steps, while cost function calculated on the validation set starts to rise at one point. From this we can conclude at which number of steps we enter the overfitting regime and what is the optimal number of steps to perform while training the model. Finally we have shown how accuracy of the network falls with lowering the cutoff radius, and how results were significantly worse when using Zernike descriptors. This was unexpected, but from this we can conclude that we have tried to use the trained model for extrapolation of data and should provide more training data to the model.

To sum things up, we conclude that this method could be used in various types of problems and shows possibilities of great improvement with respect to other, widely used methods, when performing molecular dynamics on large systems. It can be subjected to various modifications allowing it to rise in accuracy and speed, while accuracy of the model is limited only by that of the training set. Therefore, with future improvements and optimizations, it is easily possible that machine learning will become crucial for solving complex problems in solid-state physics, statistical physics and other branches of science as well.

7 Prošireni sažetak

7.1 Uvod

U posljednje vrijeme, svjedočili smo brzom razvoju umjetne inteligencije i učinku na naš svakodnevni život, kao što je recimo znanost o podacima ili prepoznavanje objekata. Takva istraživanja su trenutno goruća tema u računarstvu koja redovito pronalaze nova otkrića. No osim što se ta otkrića mogu koristiti na rješavanju spomenutih problema iz računarstva, također bi mogla naći primjenu u drugim granama znanosti. S druge strane, u fizici se mnogo pažnje posvećuje molekularnoj dinamici koja se može koristiti za simuliranje rasta tankih filmova [1], računalni dizajn lijekova [2], i u brojne druge svrhe. Međutim, takvi se sustavi najčešće sastoje od velikog broja atoma za koje je gotovo nemoguće koristiti precizne kvantno-mehaničke metode.

Elementi koje je naročito teško modelirati na taj način su dosta često metali. Jednostavna teorija kojom se oni opisuju je Drude-Sommerfeldov model, no on ne daje dobre rezultate jer ne uzima u obzir elektronsku strukturu materijala. Jedna od metoda koja omogućuje izračun elektronske gustoće jest teorija funkcionala gustoće (DFT) [4], koja je primjenjiva na većinu sustava te je danas u širokoj upotrebi. Ona se bazira na nerelativističkoj Schrödingerovoj jednadžbi za mnogoelektronsku valnu funkciju Ψ. Ključna činjenica u ovoj teoriji jest da, dok Ψ ovisi o pozicijama svih elektrona, elektronska gustoća $n(\vec{r})$ ovisi o 3 prostorne koordinate. Također, prema Hohenberg-Kohn teoremima [5], uz poznatu elektronsku gustoću u osnovnom stanju n_0, moguće je odrediti očekivanje bilo koje opservable $\hat{O}$ u osnovnom stanju koristeći:

$$O[n_0] = \left\langle \Psi[n_0] \middle| \hat{O} \middle| \Psi[n_0] \right\rangle . \tag{7.1}$$

Unatoč brojnim poboljšanjima od prvog spominjanja ove metode 1964. godine, pomoću DFT-a se još uvijek teško mogu opisati međumolekulske interakcije. Zbog toga se unaprijeđivanje DFT-a i danas aktivno istražuje.

U ovom radu promatrat ćemo sustav koji se sastoji od površine rutenija koja međudjeluje sa 16 atoma vodika te ćemo usporediti preciznost, brzinu i primjenjivost

strojnog učenja, s drugim učestalo korištenim metodama u problemima ovakvog tipa, kao što je DFT. Razlog za promatranje rutenija jest taj što je on jedan od najvažnijih katalizatora u Fischer-Tropsch procesu [9], procesu koji se sastoji omogućuje dobivanje ugljikovodičnih goriva iz ugljičnog monoksida i vodika. Osim toga, budući da cilj ovog rad nije promatranje samo statičnih sustava na temperaturi nula, već i primjena na molekularnu dinamiku u kojoj su temperature veće od nule, mogli bismo konstruirati mikrokanonski ili kanonski ansambl sustava. Time bismo omogućili i računanje termodinamičkih svojstava sustava, kao što su entropija i Gibssova energija.

7.2 Strojno učenje i primjena u fizici

Strojno učenje je grana računarstva koja koristi statističke metode kako bi računalima dala mogućnost da sama uče iz podataka. Koristi se u brojnim aspektima svakodnevnog života, kao što je rangiranje web stranica i detektiranje nepoželjne elektroničke pošte, ali i aktualnim istraživanjima, kao što je recimo računalni vid [15] i prepoznavanje govora [16]. Takvi su zadaci najčešće podijeljeni u tri kategorije: nadzirano, nenadzirano i potpomognuto učenje.

Nadzirano učenje je vrsta učenja u kojem model učimo na temelju označenih podataka, kako bi ga se kasnije moglo koristiti na novim podacima. U nenadziranom učenju nemamo označene podatke te je cilj algoritama nenadziranog učenja da samo pronađu pravilnosti u strukturama podataka te ih razvrstaju u skupine. Potpomognuto učenje se koristi u sustavima u kojima je izlaz sekvenca radnji. U tom slučaju svaka pojedina radnja nije nije važna, već je samo važno da algoritam dođe do željenog cilja. To se postiže na način da algoritam svakoj mogućoj radnji pridijeli parametar koji opisuje koliko je ta radnja dobra, odnosno kolika je vjerojatnost da će nas dovesti do željenog cilja. Tip strojnog učenja kojim se bavimo u ovom rado jest nadzirano strojno učenje; krenut ćemo od poznatih, označenih podataka te na njima istrenirati regresijski model izgrađen pomoću najmoćnijeg sustava strojnog učenja - umjetnih neuronskih mreža.

Umjetne neuronske mreže su računski sustavi inspirirani biološkim neuronskim

mrežama kakve se nalaze u mozgu ljudi i životinja. One mogu rješavati probleme koji bi bili prekompleksni za uobičajene računarske metode, a najčašće su organizirane u slojeve neurona, koji se dijele na ulazni sloj, izlazni sloj, i proizvoljan broj skrivenih slojeva između njih. Pritom se neurone mogu promatrati kao matematičke funkcije s jednom ili više ulaznih varijabli, od kojih se svaki skalira nekim parametrom, zbroje se te proslijede nelinearnoj funkciji aktivacije, koja generira izlaz neurona. Na početku su svi parametri skaliranja u neuronskoj mreži nasumično odabrani te je mrežu potrebno trenirati. To znači da joj proslijedimo označeni set podataka, uspoređujemo izračunate vrijednosti s prije označenim i optimiziramo parametre mreže. Pritom je cilj dobiti što manju funkciju cijene

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2, \tag{7.2}$$

gdje su θ parametri skaliranja u mreži, $x^{(i)}$ su ulazne varijable, $h_\theta(x^{(i)})$ su izračunate vrijednosti, a $y^{(i)}$ su označene vrijednosti.

U ovom ćemo radu iskoristiti neuronske mreže za dobivanje plohe potencijalne energije (PES), funkcije koja za dane koordinate atoma vraća energiju te atomske konfiguracije [22]. Znajući PES, moguće je računalnim simulacijama odrediti sva druga svojstva sustava. O preciznosti dobivene plohe velike ovisi i vjernost simulacije molekularne dinamike. Najpreciznije metode određivanja PES-a su *ab initio* metode bazirane na DFT-u, no njima smo ograničeni na sustave s relativno malim brojem atoma. Osim toga, *ab initio* metode ne čuvaju podatke o PES-u. Ti problemi su riješeni uvođenjem empirijskih potencijala, no da bi oni bili iskoristivi potrebno je puno truda i vremena za njihovo konstruiranje. Stoga ćemo iskoristiti metode strojnog učenja kako bismo premostili te prepreke te konstruirali PES s *ab initio* preciznošću koji opisuje sve tipove veza [23].

Međutim, u slučaju da kao ulaz za neuronsku mrežu koristimo kartezijeve koordinate, brzo uočavamo da čak i ako dva identična atoma zamjene mjesto, mreža će proizvesti drugačiji rezultat. Osim toga, kad jednom istreniramo mrežu na nekom broju atoma, ne možemo ju primijeniti sustav s drugačijim brojem. To rješavamo uvođenjem deskriptora kao ulaza umjesto kartezijevih koordinata. Deskriptori su

rezultati primjene funkcija simetrije na kartezijeve koordinate atoma koji opisuju energetski relevantnu okolinu svakog atoma. Najčešće korišteni deskriptori su Gaussovi deskriptori koje je uveo Behler [22, 23]. Oni se dijele na radijalne koji opisuju dvoatomna međudjelovanja i dani su izrazom:

$$G_i^1 = \sum_{j \neq i} e^{-\eta(R_{ij}-R_s)^2/R_c^2} f_c(R_{ij}), \tag{7.3}$$

i na angularne koji opisuju troatomna međudjelovanja i opisani su s:

$$G_i^2 = 2^{1-\zeta} \sum_{j,k \neq i} (1 + \lambda \cos\theta_{ijk})^\zeta e^{-\eta(R_{ij}^2+R_{ik}^2+R_{jk}^2)/R_c^2} f_c(R_{ij}) f_c(R_{ik}) f_c(R_{jk}). \tag{7.4}$$

Pritom su $\vec{R}_i j$ vektor koji spaja i-ti atom s j-tim, η, λ i ζ su parametri, R_c određuje polumjer obuhvaćanja atoma, a R_s centar Gaussove raspodjele. Funkcija $f_c(R_{ij})$ jest funkcija obuhvaćanja koja opisuje koliko su relevantni atomi koji se nalaze od promatranog i-tog atoma, a θ_{ijk} opisuje kut između tri atoma. Osim Gaussovih, postoje i druge vrste deskriptora, kao što su recimo Zernike deskriptori [28]. Oni uzimaju u obzir i interakcije između četiri atoma. Iako su u fokusu ovog rada Gaussovi deskriptori, na kraju ćemo proučiti i rezultate dobivene pomoću Zernike deskriptora.

7.3 Korišteni podaci i paketi

Promatrani sustav sastoji se od 48 atoma rutenija složenih u konfiguraciju $4 \times 4 \times 3$ i 16 atoma vodika smještenih na površinu rutenijske pločice. Računalne DFT simulacije su provedene na tom sustavu u članku [31] te su nam autori tog rada ustupili podatke pomoću kojih ćemo trenirati neuronske mreže. Ti su podaci dobiveni pomoću *ab initio* molekularne dinamike proširene elektronskim trenjem, pri čemu je u svakom koraku napravljen DFT račun. Time je dobiveno 339 datoteka s trajektorijama, pri čemu je u svakoj 8000 točaka. Iz tog smo seta podataka odbacili početnih 2000 točaka i uzeli svaku stotu točku iz preostalih, s ciljem dobivanja što raznolikijih podataka za treniranje mreže. Također podijelili smo početni set u tri manja: set za treniranje, set za validaciju i set za testiranje neuronskih mreža.

Svi kodovi korišteni u ovom radu pisani su u programskom jeziku Python. Ko-

rišteni su Python paketi NumPy za numeričke izračune i Matplotlib za stvaranje grafova, no daleko najveću ulugu u ovom radu imali su paketi ASE i Amp. Paket ASE (*Atomic Simulation Environment*) je Python paket namjenjen konstruiranju, manipuliranju i vizualiziranju atomskih simulacija. Osim toga, omogućuje korištenje brojnim kalkulatorima kao što je VASP [40], koji je iskorišten za dobivanje ranije spomenutih trajektorija korištenih u ovom radu. Amp (Atomistic Machine-learning Package) je paket koji uvodi strojno učenje u atomske izračune i simulacije. On nam omogućuje da umjesto kalkulatora iz paketa ASE koristimo Amp kalkulator baziran na neuronskim mrežama.

7.4 Rezultati

U prvom dijelu rada treniramo razne arhitekture neuronskih mreža u želji da pronađemo optimalnu. Koristimo Gaussove deskriptore i polumjer obuhvaćanja atoma od 6.5 Å. Istrenirane su razne arhitekture s dva i tri skrivena sloja, a rezultati su prikazani u tablici 5.1. Ispitivanjem na validacijskom setu pronađeno jest da je optimalna konfiguracija ona s tri skrivena sloja i 50, 30, odnosno 10 čvorova u prvom, drugom, odnosno trećem sloju. Od interesa nam je najveća dobivena pogreška i pogreška kvadratne sredine (RMS), obje skalirane s brojem atoma vodika. Razlog zašto skaliramo s brojem atoma vodika jest što bismo, u principu, ovom metodom mogli promatrati sustave s proizvoljnim brojem atoma pa takve pogreške daju bolji uvid u vjerodostojnost modela. Pokretanjem optimalnog 50-30-10 modela nad setom za testiranje dobivamo pogreške:

$$\text{Maksimalna pogreška po H atomu} = 27.4 \text{ meV}, \tag{7.5}$$

$$\text{RMS pogreška po H atomu} = 3.5 \text{ meV}. \tag{7.6}$$

Osim toga, na optimalnoj konfiguraciji analizirana je i uloga pojedinih deskriptora, što je vidljivo na Slikama 5.6 i 5.7. Uočeno je da deskriptori za atome rutenija nisu od velike važnosti za ovaj sustav, što je logično jer su atomi rutenija fiksirani. Također je pokazano da su među deskriptorima za vodik najzastupljeniji oni s indeksima 10 i 11, dok su najmanje zastupljeni s indeksima 7 i 8. Pritom su u oba slučaja deskriptori od 1 do 8 radijalni, a od 9 do 20 angularni.

Nadalje, na Slici 5.8 je pokazano kako pogreška izračunata nad setom za treniranje i setom za validaciju ovisi o broju koraka pri treniranju. Vidljivo je da sa svakim novim korakom pogreška nad setom za treniranje pada, no pogreška nad setom za validaciju u jednom trenu počne rasti. To znači da smo ušli u režim pretreniranosti neuronske mreže te da ona radi izuzetno dobre predikcije samo na podacima na kojima je istrenirana, dok na novim podacima to ne uspijeva. Isto tako, ne lijevoj strani grafa vidljivo je područje podtreniranosti, koje ukazuje da model još nije dovoljno dobar što se uočava u visokim pogreškama na oba seta podataka.

Za kraj su još na optimalnoj 50-30-10 konfiguraciji ispitani neki alternativni pristupi rješevanju ovog problema, kao što je recimo korištenje Zernike deskriptora. Kao što smo rekli, oni uzimaju u obzir i međudjelovanja između četiri atoma, na temelju čega bismo očekivali preczinije rezultate nego za Gaussove deskriptore. Međutim, na Slici 5.9 vidi se kako to nije točno. Izuzetno veliku pogrešku u ovom slučaju opisujemo tome da, ako proučavamo i četveroatomne interakcije, radimo s premalim setom podataka s kojim nismo pokrili dovoljno mogućih slučajeva. Time je naš model ušao u režim ekstrapolacije i podbacio. Stoga bismo za uspješno treniranje ovakvog modela trebali raditi s većim setovima podataka. Zadnja stvar koju smo analizirali jest ponašanje modela s promjenom polumjera obuhvaćanja. Očekujemo da ako smanjimo polumjer obuhvaćanja, manje atoma će se smatrati relevantnima pri konstruiranju deskriptora, te će preciznost modela biti slabija. To je i potvrđeno na Slici 5.10, gdje je za polumjer od 5.5 Å dobivena veća pogreška nego za ranije korišteni polumjer od 6.5 Å.

7.5 Zaključak

Ovim radom smo pokazali kako se pomoću neuronskih mreža mogu vjerno izračunati energije sustava, pri čemu je preciznost ograničena jedino točnošću podataka nad kojima treniramo mrežu. Zaključujemo kako je vjerojatno da će s budućim razvojem znanosti strojno učenje igrati ključnu ulogu u rješavanju kompleksnih problema fizike čvrstog stanja, statističke fizike, ali i brojnih drugih znanstvenih područja.

Bibliography

[1] Zheng, H. Molecular Dynamic Simulation of Thin Film Growth Stress Evolution. Theses and Dissertations. Lehigh University, 2011.

[2] Leach, A. R., Comprehensive Medicinal Chemistry II, Volume 4: Computer-Assisted Drug Design, Elsevier, 2007.

[3] IUPAC, Compendium of Chemical Terminology, 2nd ed. (the "Gold Book"), 1997.

[4] Kohn, W. Nobel Lecture: Electronic structure of matter–wave functions and density functionals, Rev. Mod. Phys. 71, 1253 (1998)

[5] Hohenberg, Pierre; Kohn, W. Inhomogeneous electron gas. // Physical Review. 136 (3B): B864–B871 (1964)

[6] Kohn, W.; Sham, L. J. Self-Consistent Equations Including Exchange and Correlation Effects. // Physical Review. 140 (1965)

[7] Grimme, S. Semiempirical hybrid density functional with perturbative second-order correlation. // Journal of Chemical Physics. 124 (3): 034108, (2006)

[8] Ruthenium (Ru) - Properties, Applications, (13.08.2013), AZo Materials, https://www.azom.com/article.aspx?ArticleID=9275

[9] Generalic, E. Fischer-Tropsch process, (29.08.2017.) Croatian-English Chemistry Dictionary & Glossary. KTF-Split, https://glossary.periodni.com/glossary.php?en=Fischer-Tropsch+process

[10] Höök, M.; Fantazzini, D.; Angelantoni, A.; Snowden, S. Hydrocarbon liquefaction: viability as a peak oil mitigation strategy. // Philosophical Transactions of the Royal Society A. 372, (2014)

[11] Leckel, D. Diesel Production from Fischer-Tropsch: The Past, the Present, and New Concepts. // Energy & Fuels, 23, 2342–2358, (2009)

[12] Schulz, H. Short history and present trends of Fischer-Tropsch synthesis. // Applied Catalysis A: General. 186: 3–12, (1999)

[13] Samuel, A. "Some Studies in Machine Learning Using the Game of Checkers". IBM Journal of Research and Development. 3 (3): 210–229 (1959)

[14] Bishop, C. M. Pattern Recognition and Machine Learning. Springer, ISBN 0-387-31073-8, 2006.

[15] Wernick, M. N.; Yang, Y.; Brankov, J. G.; Yourganov, G.; Strother, S. C. Machine Learning in Medical Imaging.// IEEE Signal Processing Magazine. 27 (4): 25–38, (2010).

[16] Deng, L.; Hinton, G.; Kingsbury, B. New types of deep neural network learning for speech recognition and related applications: an overview // IEEE International Conference on Acoustics, Speech and Signal Processing, 2013.

[17] Raschka, S.; Mirjalili, V. Python Machine Learning, 2nd edition, Birmingham: Packt, 2017.

[18] Alpaydin, E. Introducton to machine learning, 2nd edition, Cambridge: MIT Press, 2009.

[19] van Gerven, M.; Bohte, S. (Editorial) Artificial Neural Networks as Models of Neural Information Processing // Front. Comput. Neurosci. 11:114, (2017).

[20] Haykin, S. Neural Netwroks : A Comprehensive Foundation, 2nd edition, Delhi: Pearson, 2005.

[21] Glosser.ca, Artificial neural network with layer coloring, (28.02.2013), `https://commons.wikimedia.org/wiki/File:Colored_neural_network.svg#metadata`

[22] Behler, J. Neural network potential-energy surfaces in chemistry: a tool for large-scale simulations. // Phys. Chem. Chem. Phys., 13, 17930–17955, (2011)

[23] Behler, J.; Parrinello M. Generalized Neural-Network Representation of High-Dimensional Potential-Energy Surfaces. // Phys. Rev. Lett. 98, 146401, (2007)

[24] Lorenz, S.; Groß, A.; Scheffler, M. Representing high-dimensional potential-energy surfaces for reactions at surfaces by neural networks // Chem. Phys. Lett. 395, 210 (2004).

[25] Blank T.B. et al. Construction of high-dimensional neural network potentials using environment-dependent atom pairs // J. Chem. Phys. 103, 4129 (1995).

[26] Khorshidi, A., Peterson A. A. Amp: A modular approach to machine learning in atomistic simulations. // Comput. Phys. Commun. 207, 310–324 (2016)

[27] AMP Theory, Atomistic Machine-learning Package, https://amp.readthedocs.io/en/latest/theory.html

[28] Novotni, M.; Klein, R. Shape retrieval using 3D Zernike descriptors. // Computer-Aided Design 36(11), 1047–1062 (2004)

[29] Bart'ok, A.P.; Payne, M.C.; Kondor, R.; Csanyi, G. Gaussian approximation potentials: The accuracy of quantum mechanics, without the electrons. // Phys. Rev. Lett. 104, 136403 (2010)

[30] Denzler, D. N. et al. Electronic Excitation and Dynamic Promotion of a Surface Reaction // Phys. Rev. Lett. 91(22), 226102, (2003).

[31] Juaristi, J. I., Alducin, M., Saalfrank, P. Femtosecond laser induced desorption of H 2, D 2, and HD from Ru (0001): Dynamical promotion and suppression studied with ab initio molecular dynamics with electronic friction, // Phys. Rev. B, 95(12), 125439 (2017).

[32] Python programming language, https://www.python.org/

[33] NumPy package, http://www.numpy.org/

[34] Matplotlib plotting library, https://matplotlib.org/

[35] Atomic Simulation Environment, https://wiki.fysik.dtu.dk/ase/index.html#

[36] Abinit, https://www.abinit.org/

[37] NWChem: Open Source High-Performance Computational Chemistry, http://www.nwchem-sw.org/index.php/Main_Page

[38] Gaussian, http://gaussian.com/

[39] ASAP - As Soon As Possible, https://wiki.fysik.dtu.dk/asap

[40] VASP - Vienna Ab initio Simulation Package, https://www.vasp.at/