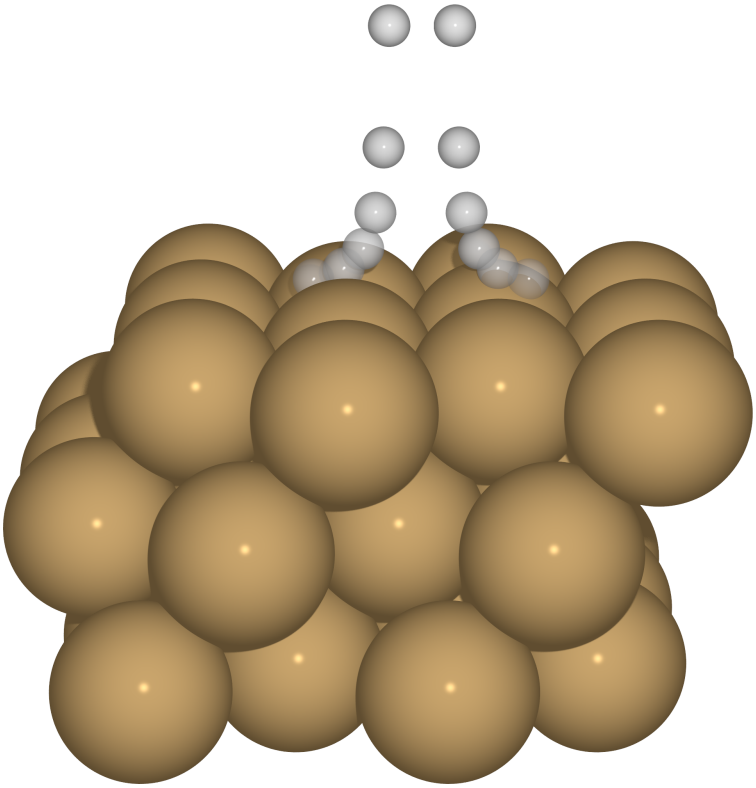
CHALMERS
UNIVERSITY OF TECHNOLOGY



Neural Network Potentials for Molecule-Surface Interactions

Master's thesis in Physics

Nicklas Österbacka

Department of Physics
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Master's thesis 2019

Neural Network Potentials for Molecule-Surface Interactions

NICKLAS ÖSTERBACKA

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Physics
Division of Chemical Physics
Chalmers University of Technology
Gothenburg, Sweden 2019

Neural Network Potentials for Molecule-Surface Interactions
NICKLAS ÖSTERBACKA

© NICKLAS ÖSTERBACKA, 2019.

Supervisor and Examiner: Anders Hellman, Dept. of Physics

Master's Thesis 2019
Department of Physics
Division of Chemical Physics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: An illustration of hydrogen dissociation over a Cu(111) surface.

Typeset in LaTeX
Gothenburg, Sweden 2019

Neural Network Potentials for Molecule-Surface Interactions

NICKLAS ÖSTERBACKA
Department of Physics
Division of Chemical Physics
Chalmers University of Technology

Abstract

A method of approximating the potential energy surface (PES) of molecules interacting with crystalline surfaces was used to find an analytic representation of the interactions between molecular hydrogen and a solid copper surface. The potential energy of the H_2/Cu(111) system was sampled using density functional theory with the Perdew-Burke-Ernzerhof exchange-correlation functional. The machine learning framework TensorFlow was used to generate neural network representations of the energy surface from this data set.

The final representation accurately reproduces the features of the underlying PES around potential entry channels for hydrogen dissociation, which agrees well with other studies. The model also accurately predicts the zero-point energy of the H_2 molecule.

In addition to examining the energy landscape around these entry channels the dissociative sticking probability at different translational kinetic energies for the system is estimated using classical trajectory methods. It is found that this produces sticking probabilities that disagree with comparable experimental results; they are in agreement with similar computational estimates, however, and reflect that the sticking probability increases with the kinetic energy in qualitative agreement with experiments.

Keywords: density functional theory, artificial neural networks, potential energy surfaces, molecular dynamics.

Acknowledgements

First and foremost I would like to thank my supervisor, Anders Hellman, for suggesting this topic and for all the time taken to discuss the project with me. I am incredibly grateful for the help you have given me. It has proven impossible to leave your office without feeling encouraged.

All of my friends and family have been a great support during the work, and there is simply not enough room to thank each and every one of you. However, I would like to especially thank the following people, in no particular order:
Petter Rosander, without whom I would never have made it through my Master's studies with my sanity relatively intact.
Daniel Brunnsåker, without whom this thesis would remain perpetually unfinished.
Matilda Chöler, who has been (un)fortunate enough to live with me through all of this.

The density functional theory computations and *ab initio* molecular dynamics simulations were performed on resources at Chalmers Centre for Computational Science and Engineering provided by the Swedish National Infrastructure for Computing.

Nicklas Österbacka, Gothenburg, August 2019

Contents

1
Introduction

Machine learning (ML) is the name given to a group of related techniques used to build models directly from data [1], rather than explicitly coding software for the specific task; the algorithm essentially *learns* by studying the data fed to it. Such techniques have been successfully applied in many fields, but has been given a lot of attention particularly in data analysis and image recognition. For instance, YouTube uses neural networks, i.e. a way of facilitating ML, to sort through the vast amounts of data they gather from their users and base their video recommendations on that [2] and Nvidia has used a special neural network structure to allow for the control and steering of self-driving cars using a single front-facing camera [3].

ML techniques are clearly of interest to the software and automotive industries, but efforts have been made to find applications in the physical sciences as well; for instance, researchers have investigated the use of ML techniques to speed up the evaluation of energies for atomic systems by essentially bypassing the need to solve the computationally intensive Kohn-Sham equations in density functional theory [4], and others have been successful in using neural networks to model the interactions between atoms in a wide variety of systems [1], resulting in a so-called neural network potential (NNP).

NNPs make direct use of a fundamental property of neural networks: they are *universal approximators* [5, p. 167], giving them the ability to approximate essentially any function to arbitrary accuracy. The final approximation is furthermore both analytic and differentiable. This does not require the underlying function itself to have an analytical representation; solutions of the Schrödinger equation, for instance, can only feasibly be found using numerical methods for most systems, but a neural network trained on points of an atomic system's potential energy surface functions as an analytic approximation of the Schrödinger equation.

When studying the dynamics of a chemical reaction, for instance, it is often necessary to probe it many times over under slightly different initial conditions; doing this from first principles, i.e. by calculating energies and forces using numerical methods like density functional theory in each simulation step, is rather expensive [1]. Using an analytical potential for the system instead makes such investigations more feasible.

One reaction that has been intensively investigated both experimentally and computationally is hydrogen dissociation over metal surfaces [6]. For instance, Rettner

et al probed the reaction over a Cu(111) surface through both associative desorption and molecular beam experiments [7] in 1994; the results of the molecular beam experiments were later recreated computationally by Smeets *et al* [8].

In this work we follow in their footsteps by finding a NNP description of the H_2/Cu(111) system, and then use it to probe the dissociation reaction using molecular dynamics techniques.

1.1 Objectives

The purpose of this work is to find a neural network approximation for the H_2/Cu(111) system from *ab initio* data and use it to approximate the sticking probability in the dissociative regime.

The DFT calculations are to be carried out using the software package GPAW, and the neural networks will be optimized using the machine learning framework TensorFlow. Sticking probability estimation will be carried out by probing the dissociative reaction using molecular dynamics on the final neural network description of the potential energy surface.

2
Theory

This chapter contains the theoretical background for the computational techniques used in this work; density functional theory, molecular dynamics, and neural networks.

2.1 Quantum mechanics and simulations

For brevity, all equations are presented in atomic units. This means that the electron mass m_e, elementary charge e, and reduced Planck's constant $\hbar$ are set to 1.

2.1.1 The Schrödinger equation

Of central importance to quantum mechanics is the (nonrelativistic) time-independent Schrödinger equation, which may be written

$$\mathcal{H}\Psi = E\Psi. \tag{2.1}$$

Here, $\mathcal{H}$ is the *Hamiltonian* of the system, Ψ its total many-body wavefunction and E its total energy. For a molecular system this takes into account energy contributions from both the nuclei and electrons therein. Taking into account all these contributions is nontrivial, but is simplified somewhat by the *Born-Oppenheimer approximation* (or, in this context, the *clamped nuclei approximation* [9, p. 25]). The mass of an electron is several orders of magnitudes smaller than that of a nucleus, so the electrons respond to any external influence much quicker - the nuclei can essentially be considered static from the point of view of the electrons. This allows the decoupling of the problem into two systems, one nucleic and one electronic.

Consider the *electronic* Schrödinger equation,

$$\mathcal{H}_e\Psi_e = E_e\Psi_e, \tag{2.2}$$

where Ψ_e is the electronic many-body wavefunction of the system, describing all electrons therein, and the nuclear coordinates are considered external parameters in $\mathcal{H}_e$. From here on out we consider only the electronic Schrödinger equation, so we drop the index e. The Hamiltonian may be written [10, p. 9]

$$\mathcal{H} = -\frac{1}{2}\sum_{i=1}^{N}\nabla_i^2 + \sum_{i=1}^{N}V_n(\mathbf{r}_i) + \sum_{i=1}^{N}\sum_{j<i}U(\mathbf{r}_i,\mathbf{r}_j), \tag{2.3}$$

where the terms define kinetic energy, electron-nuclei interactions and electron-electron interactions. Solving Equation (2.2) is still no easy feat. A system of N electrons yields a $3N$-dimensional differential equation for a $3N$-dimensional wave function. Finding solutions is nevertheless possible, but in practice approximations are made and numerical methods often have to be employed. One such method is *density functional theory* (DFT) and the rest of this section is concerned with presenting its theoretical foundation.

2.1.2 The Hohenberg-Kohn theorems

As was mentioned previously, dealing with wavefunctions for real molecular systems is cumbersome due to the large amount of dimensions involved. Hohenberg and Kohn [11] helped alleviate this by reformulating the problem in terms of the electron density rather than wavefunctions.

The first of the *Hohenberg-Kohn theorems* state that the ground-state energy $E_0 = E_0[n(\mathbf{r})]$ is a unique functional[1] of the system's electronic density, and the second theorem states that the density that minimizes $E_0[n(\mathbf{r})]$ is the ground-state density $n_0(\mathbf{r})$ [12, p. 11]. An important implication of this result is that any properties of the system may be determined from the density alone.

2.1.3 The Kohn-Sham approach

The Hohenberg-Kohn theorems are important and lay the theoretical foundation for modern DFT, but make no progress towards a viable computational scheme. The dimensional reduction from $3N$ to 3 certainly helps, but in the end just trades one problem for another. Fortunately, the Kohn-Sham approach to DFT was developed shortly after the Hohenberg-Kohn theorems were formulated [13]. The idea is to consider a system of fictitious non-interacting electrons whose pseudo-wavefunctions yield the same ground-state density as the real system.

Consider the electronic contribution to an atomic system's energy. According to the Hohenberg-Kohn theorems, this may be expressed as a functional of the electron density,

$$E = E[n(\mathbf{r})]. \tag{2.4}$$

Kohn and Sham expressed this in the form

$$E[n(\mathbf{r})] = \int d\mathbf{r} n(\mathbf{r}) V_{\mathrm{n}}(\mathbf{r}) + \frac{1}{2} \int\int d\mathbf{r} d\mathbf{r}' \frac{n(\mathbf{r}) n(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} + T[n(\mathbf{r})] + E_{\mathrm{XC}}[n(\mathbf{r})], \tag{2.5}$$

where the terms represent, in order, the influence of the *external potential* from the nuclei present, electron-electron Coulomb interaction under the *mean-field approximation* (sometimes referred to as *Hartree energy*[2]), the kinetic energy of a system

[1] A functional in this context is a function of a function; $g[f(t)]$, where $f(t)$ is some arbitrary function of the variable t.

[2] Hartree is also the name of an energy unit, and one of the founders of quantum chemistry.

of noninteracting Kohn-Sham electrons of density $n(\mathbf{r})$ and *exchange-correlation energy*. E_{XC} includes all quantum mechanical effects and lacks a known analytical form.

From the second of the Hohenberg-Kohn theorems it is known that $n_0(\mathbf{r})$ minimizes E, so

$$\left.\frac{\partial E[n]}{\partial n}\right|_{n_0} = 0. \tag{2.6}$$

Through this *variational principle* it is possible to find an equation for a set of wavefunctions $\phi_i(\mathbf{r})$ that may be used to construct $n(\mathbf{r})$. By requiring these wavefunctions to be orthonormal one ends up with the *Kohn-Sham equations* [9, p. 40], which have the form

$$\left[\hat{T} + \hat{V}_{\mathrm{n}}(\mathbf{r}) + \hat{V}_{\mathrm{H}}(\mathbf{r}) + \hat{V}_{\mathrm{XC}}(\mathbf{r})\right]\phi_i(\mathbf{r}) = \varepsilon_i\phi_i(\mathbf{r}). \tag{2.7}$$

This is very similar to the Schrödinger equation as it is usually presented, with two important differences: $\phi_i(\mathbf{r})$ are *one-electron* Kohn-Sham wavefunctions and the electronic system they represent is nonphysical; it acts as a useful computational abstraction but does not necessarily have a physical significance beyond yielding the true ground-state density [10, p. 49].

The operators in Equation (2.7) are defined as

$$\hat{T} = -\frac{\nabla^2}{2}, \tag{2.8a}$$

$$\hat{V}_{\mathrm{n}} = -\sum_I \frac{Z_I}{|\mathbf{r} - \mathbf{R}_I|}, \tag{2.8b}$$

$$\hat{V}_{\mathrm{H}} = \int d\mathbf{r}' \frac{n(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|}, \tag{2.8c}$$

$$\hat{V}_{\mathrm{XC}} = \left.\frac{\partial E_{\mathrm{XC}}[n(\mathbf{r})]}{\partial n}\right|_{n_0}. \tag{2.8d}$$

The formalism of DFT has been in principle exact up to this point, but V_{XC}, just like E_{XC}, lacks known exact functional form for the general case. Many approximative representations, *exchange-correlation functionals* (XC functionals), have been derived; this is discussed in more detail in the next subsection. Once an appropriate XC approximation has been chosen for the system of interest the problem may be solved self-consistently, i.e. iteratively solved until the current solution agrees with the previous one. Starting from a guess for the electron density Equation (2.7) is solved; a new density is computed from the resulting set of Kohn-Sham wavefunctions via

$$n(\mathbf{r}) = \sum_i |\phi_i(\mathbf{r})|^2. \tag{2.9}$$

The old and new densities are then compared. If they differ by more than some threshold, Equation (2.7) is solved again using the new density and the process is repeated until the new and old densities converge, i.e. until the procedure reaches self-consistency. The ground-state energy of the system may then be found through

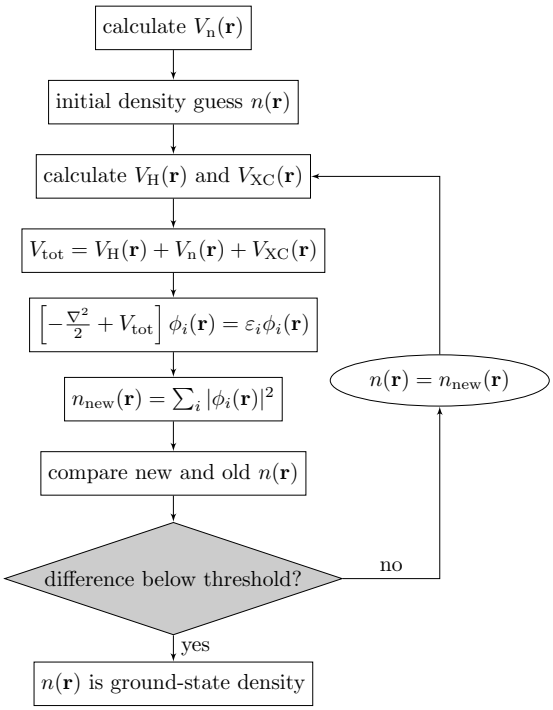


Figure 2.1: A flowchart of the computational scheme commonly employed in the Kohn-Sham approach to DFT.

Equation (2.5).

The Kohn-Sham DFT scheme is illustrated in Figure 2.1.

2.1.4 Exchange-correlation functionals

XC functionals can be categorized based on what the approximations take into account. This is sometimes referred to as the *Jacob's ladder* of DFT [14] after the biblical story of a ladder leading to heaven; in this case, heaven is chemical accuracy. Each rung on the ladder makes use of more properties of the system, but only the bottom two will be touched upon here.

On the lowest rung the *Local Density Approximation* (LDA) is found, which takes into account only the local density $n(\mathbf{r})$. The known exchange-correlation functional of a homogeneous electron gas with density $n(\mathbf{r})$ is used as a stand-in for the true expression [12, p. 14],

$$E_{\mathrm{XC}}^{LDA}[n(\mathbf{r})] = \int d\mathbf{r} \varepsilon_{\mathrm{XC}}^{\mathrm{hom}}[n(\mathbf{r})] n(\mathbf{r}), \tag{2.10}$$

where $\varepsilon_{\mathrm{XC}}^{\mathrm{hom}}$ is the exchange-correlation energy per electron of the homogeneous electron gas. This approach was proposed by Kohn and Sham in their original paper on DFT [13]. LDA can be generalized to also include electron spin, resulting in the *Local Spin Density Approximation* (LSDA).

On the second rung are *generalized gradient approach* (GGA) functionals, making use the gradient of the density, $\nabla n(\mathbf{r})$, in addition to $n(\mathbf{r})$ [15, p. 154]:

$$E_{\mathrm{XC}}^{GGA}[n_\uparrow, n_\downarrow] = \int d\mathbf{r}\varepsilon_{\mathrm{XC}}[n_\uparrow, n_\downarrow, \nabla n_\uparrow, \nabla n_\downarrow]n(\mathbf{r}). \tag{2.11}$$

As they normally take electron spin into account, they can be viewed as extensions of LSDA. There are several GGA functionals readily available for use in DFT software packages, such as the Perdew-Burke-Ernzerhof (PBE) functional [16] used in this work.

The approach taken by Perdew *et al.* in deriving the functional is entirely theoretical; they apply corrections to the LSDA-level theory through an ansatz that fulfills certain known properties of the exact exchange-correlation energy. A convenient way of representing this is through an enhancement factor F_{XC} over local exchange,

$$E_{\mathrm{XC}}^{GGA}[n_\uparrow, n_\downarrow] = \int d\mathbf{r}\varepsilon_{\mathrm{XC}}^{\mathrm{hom}}[n(\mathbf{r})]F_{\mathrm{XC}}(r_s, \zeta, s)n(\mathbf{r}). \tag{2.12}$$

r_s is the local Seitz radius[3], defined via $n = 3/4\pi r_s^3$, ζ the relative spin polarization[4], $(n_\uparrow - n_\downarrow)/n$, and $s = \frac{|\nabla n|}{2k_F n}$ a reduced density gradient, describing how the density varies on the scale of the local Fermi wavelength $2\pi/k_F = 2\pi/(3\pi^2 n)^{1/3}$ [17, p. 36]. Plots of F_{XC} for the non-spin-polarized ($\zeta = 0$) and fully spin-polarized ($\zeta = \pm 1$) cases are shown in Figure 2.2; the details of the functional as well as a derivation of the enhancement factor may be found in Appendix A. At $s = 0$ PBE is equivalent to LSDA, which has no s-dependence, so corresponding LSDA plots would simply be horizontal lines at the $s = 0$ PBE values.

2.1.5 DFT in practice and the Projector-Augmented Wave Method

To perform numerical calculations on realistic atomic systems it is necessary to define the positions of the particles within it, but wavefunctions are spread out in space rather than localized to single points. This necessitates the use of a three-dimensional computational cell which encompasses more space than is spanned by just the atoms [9, p. 246]. The exact nature of this cell, i.e. its dimensions and boundary conditions, will vary depending on the system being modelled. In this work periodic boundary conditions are used, since the topic at hand concerns extended solid surfaces.

The wavefunctions themselves also pose problems. They are continuous functions that can be hard to find representations for. In practice, one normally expands them in some *basis set*, a set of functions whose sum approximates the full wave function [12, p. 22]. Periodic structures lend themselves well to using periodic basis

[3] A sphere of radius r_s contains, on average, one electron.

[4] In this expression $n_\uparrow$ and $n_\downarrow$ are the density of electrons in either of the two spin states and n the total electron density.

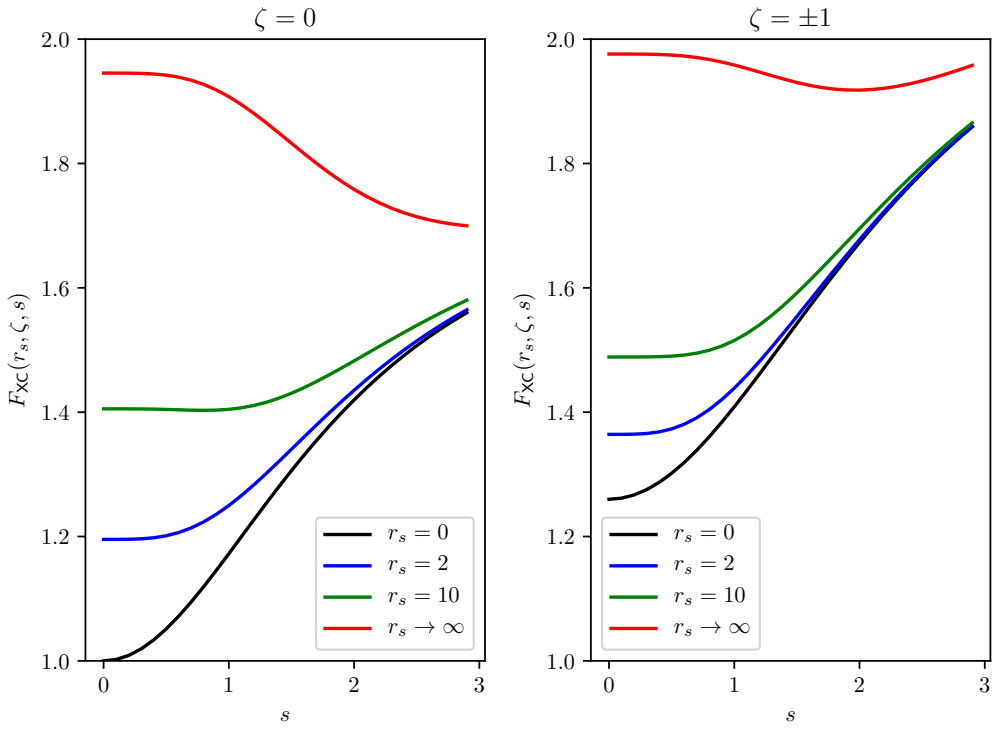


Figure 2.2: Enhancement factor over local exchange F_{XC} for the PBE functional.

sets. One way this can be achieved is using a planewave expansion, where the Kohn-Sham wavefunctions are represented as [9, p. 250]

$$\phi_i(\mathbf{r}) = \sum_{\mathbf{G}} c_i(\mathbf{G}) \exp(i\mathbf{G} \cdot \mathbf{r}), \tag{2.13}$$

where $\mathbf{G}$ are reciprocal lattice vectors. This is in essence a Fourier series and is thus an infinite sum. This is clearly impractical! To reduce the set of functions to be summed over a *cutoff energy* is defined as

$$E_{\mathrm{cut}} = \frac{|\mathbf{G}_{\mathrm{max}}|^2}{2}, \tag{2.14}$$

and only planewaves satisfying $|\mathbf{G}|^2/2 < E_{\mathrm{cut}}$ are included in the summation [9, p. 251].

Another phenomenon to consider is the fact that electronic wavefunctions tend to oscillate strongly close to the nuclei. This behaviour requires many Fourier components to represent, i.e. a high planewave cutoff energy. Chemical properties of molecules and materials are primarily determined by valence electrons, however, so the precise treatment of core electrons is often not strictly necessary [9, p. 259]. A common approach is to approximate the behaviour of core electrons by using *pseudopotentials* in place of the system's full potential; this essentially replaces the electron density of the core electrons inside of a cutoff radius with a smoothed-out density that captures some of the important properties of the system [12, p. 64], but circumvents the problem of oscillating wavefunctions. The proper behaviour of the

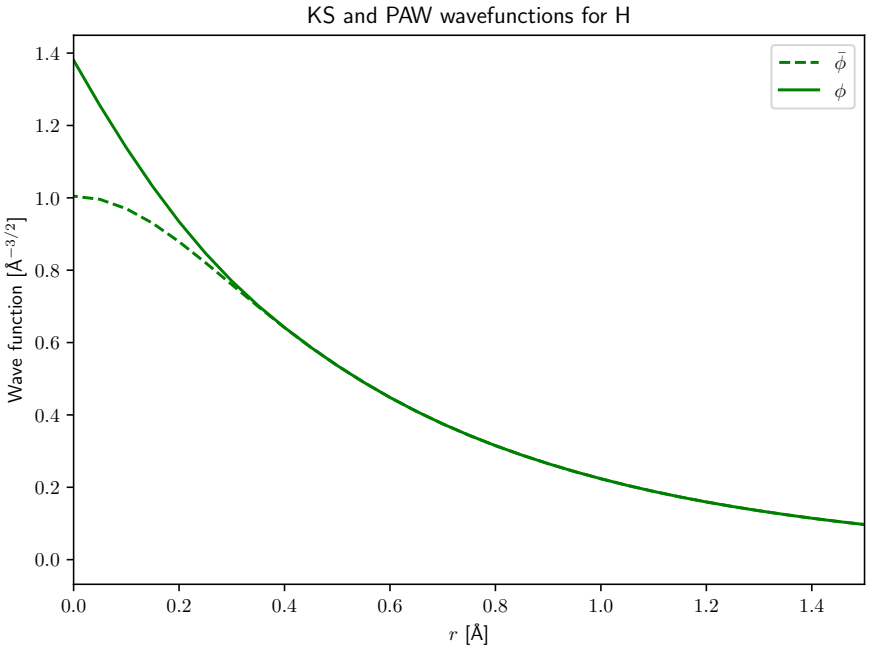


Figure 2.3: Kohn-Sham wavefunction ϕ and auxilliary smooth PAW wavefunction $\tilde{\phi}$ for a hydrogen atom.

wavefunctions is retained outside of the cutoff radius [9, p. 260].

An extension of the pseudopotential approach (and the augmented plane wave method) is the *projector augmented wave* (PAW) method [18]. The idea is to define a linear transformation between the Kohn-Sham pseudo-wavefunctions ϕ_i and corresponding auxilliary smooth wavefunctions $\tilde{\phi}_i$,

$$\phi_i = \hat{\mathcal{T}}\tilde{\phi}_i. \tag{2.15}$$

The Kohn-Sham equations are transformed in an equivalent manner [19],

$$\hat{\mathcal{T}}^\dagger \mathcal{H} \hat{\mathcal{T}} \tilde{\phi}_i = \varepsilon_i \hat{\mathcal{T}}^\dagger \hat{\mathcal{T}} \tilde{\phi}_i, \tag{2.16}$$

and solved. The transformation operator $\hat{\mathcal{T}}$ is defined in such a way that it only modifies wavefunctions inside of atom-centered augmentation spheres, defined as regions around nuclei within a cutoff radius. The end result of this is that wavefunctions may be treated separately inside and outside of the augmentation spheres using appropriate representations for smooth and oscillating parts, while providing a way to regain the full Kohn-Sham wavefunctions if so desired. A plot of both the PAW wavefunction and the corresponding full KS wavefunction for a hydrogen atom is shown in Figure 2.3 as a function of distance from the atom's center r.

2.1.6 Molecular dynamics and the potential energy surface

Molecular dynamics (MD) techniques allow for the study of the dynamics of many-particle systems. By numerically integrating the relevant equations of motion numerically it is possible to simulate how systems develop over time [20, p. 197]. For

this to be possible one needs to know how the particles of interest interact.

The energetics of these interactions are described by the system's *potential energy surface* (PES) [20, p. 263]: potential energy as a function of geometry, i.e. of nucleic positions $\mathbf{R}$; $E_{\text{tot}} = E_{\text{tot}}(\mathbf{R}_1, \ldots, \mathbf{R}_N)$ for a system with N nuclei. The forces acting upon nucleus A can be calculated as the negative gradient of the PES with respect to $\mathbf{R}_A$ [20, p. 266],

$$\mathbf{F}_A = -\nabla_{\mathbf{R}_A} E(\mathbf{R}_1, \ldots, \mathbf{R}_N). \tag{2.17}$$

While the motion of atoms is treated classically, the forces may be calculated from electronic structure theory. This can be done on-the-fly during simulations in *ab initio* MD [12, p. 198], though the computational cost to do so is high, or via some analytical function describing the interactions between atoms such as a Lennard-Jones pair potential [20, p. 177].

Some practical considerations need to be made; what is the goal of the simulation and what properties, if any, are to be extracted from it? To this end an *ensemble* is chosen, which puts constraints on certain quantities of the system. The one relevant to this work is the *microcanonical ensemble*, or the *NVE ensemble*, which keeps the number of particles N, the volume V and the total energy E of the system constant [20, p. 171].

The choice of ensemble also restricts the types of computational algorithms that can be used. For the NVE ensemble the *velocity-Verlet* algorithm may be employed. A timestep h is chosen, which needs to be large enough for the simulation to be feasibly run on a computer and small enough to capture the relevant dynamics of the system. If h is chosen to be too large errors will be introduced, and the results inaccurate [20, p. 200].

In velocity-Verlet MD positions $\mathbf{r}$ and velocities $\mathbf{v}$ for atom A are calculated as[5] [20, p. 202]

$$\mathbf{r}_A(t+h) = \mathbf{r}_A(t) + \mathbf{v}_A(t)h + \frac{h^2}{2M_A}\mathbf{F}_A(t), \tag{2.18a}$$

$$\mathbf{v}_A(t+h) = \mathbf{v}_A(t) + \frac{1}{2M_A}\left[\mathbf{F}_A(t) + \mathbf{F}_A(t+h)\right]h, \tag{2.18b}$$

where M_A is the mass of atom A. Positions and velocities at $t = 0$ are given as initial conditions, while positions at $t = h$ are calculated as

$$\mathbf{r}_A(h) = \mathbf{r}_A(0) + h\mathbf{v}_A(0) + \frac{h^2}{2M_A}\mathbf{F}_A[\mathbf{r}_A(0)]. \tag{2.19}$$

In this work MD simulations are not used to extract any properties of the system under study; it is used simply as a tool to generate *trajectories* for atoms, i.e. the path they take after a number of iterations for a set of initial conditions. The

[5] The source cited assumes that $M_A = 1$, while we make no such assumptions here.

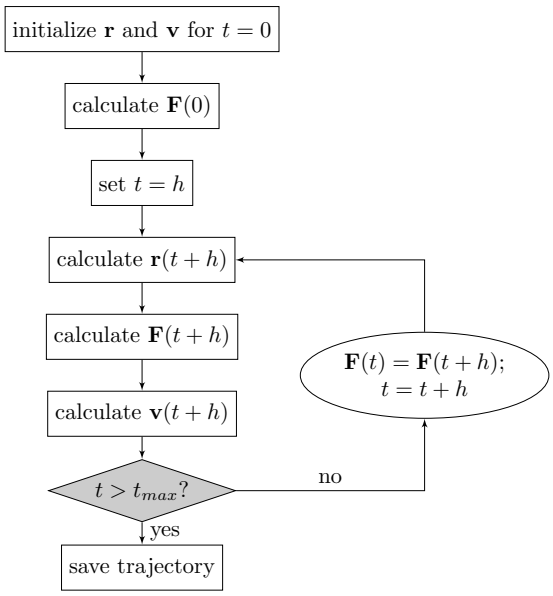


Figure 2.4: A flowchart of the velocity-Verlet MD algorithm as used in this work.

velocity-Verlet algorithm is illustrated in Figure 2.4.

One problem with the MD approach to atomic interactions is that it fails to capture their quantum mechanical properties. A quantum mechanical system can never fall below its *zero-point energy*, i.e. its energy in the lowest quantum state. For a free hydrogen molecule this means that its vibrational energy cannot reach zero, but this is no problem classically [9, p. 134]. One way to bridge this gap between the quantum and classical descriptions is to apply a *quasiclassical* approach. In quasiclassical MD the zero-point energy is taken into account to some extent [21, p. 82].

2.2 Neural networks

Neural networks (NNs) are computational algorithms that are commonly used to facilitate *machine learning*; such algorithms are not explicitly programmed to perform a certain task, but are rather taught to do so by adapting to examples fed to them [5, p. 2].

2.2.1 The structure of neural networks

NNs consist of several simple computational units, *neurons*, which are divided into interconnected layers. While many different structures and variants exist the present work is concerned only with *multilayer feed-forward* NNs [5, p. 22], where the neurons are connected to every neuron in the neighboring layers.

A schematic of a simple NN is shown in Figure 2.5. This particular network has two inputs, one hidden layer with three neurons, and one output neuron. The grey

neurons are *bias* neurons and always take on a value of 1. The neuron-neuron connections are shown as arrows and the strength of this connection is described by a weight W [5, p. 11], or by a bias b for the bias neurons; the set of weights and biases belonging to a particular neural network are here denoted *network parameters*.

The number of input and output neurons is determined by what the network is built to do, while any number of hidden layers with an arbitrary number of neurons may be present. The network can be described mathematically as $\mathcal{O}(\boldsymbol{\mathcal{I}})$, i.e. an output $\mathcal{O}$ as a function of some inputs $\boldsymbol{\mathcal{I}}$. The hidden layers have the purpose of manipulating data fed through them in some useful manner [5, p. 22].

The input neurons simply accept external data, while the hidden and output neurons calculate the quantity [5, p. 130]

$$\chi_j^L = g_L\left(\sum_i W_{ij}\chi_i^{L-1} + b_j\right). \tag{2.20}$$

Here χ_j^L is the value of the j'th neuron of the L'th layer, g_L the *activation function* of the layer, W_{ij} the weight of the connection between the two neurons, and b_j the bias of the j'th neuron. Information propagates from the input towards the output, so Equation 2.20 is computed in sequence from the hidden layer closest to the inputs to the output layer. Many different activation functions may be used for different purposes [5, p. 13], but in this work the hyperbolic tangent function is used for the hidden layers, while the identity function is used for the output layer in order to not restrict the output to be in $(-1, 1)$.

NNs have an important property; they are *universal approximators* [5, p. 167]. As long as the activation function g is nonconstant, bounded and monototonically increasing there exists a set of parameters such that

$$F(\mathbf{x}) = \sum_{i=1}^{m_1} \alpha_i b\left(\sum_{j=1}^{m_0} W_{ij}x_j + b_i\right) \tag{2.21}$$

is an approximation of $f(\mathbf{x})$ [5, p. 167], i.e. $\mathcal{O}(\boldsymbol{\mathcal{I}}) \approx f(\boldsymbol{\mathcal{I}})$.

2.2.2 Training procedure

The universal approximation theorem is merely an existence theorem; it says there *is* a set of parameters such that $\mathcal{O}(\boldsymbol{\mathcal{I}}) \approx f(\boldsymbol{\mathcal{I}})$, but provides no way of finding them. This task can be formulated as an optimization problem, however. This is done by generating a set of input-output pairs [5, p. 34] $\{(\boldsymbol{\mathcal{I}}_1, f(\boldsymbol{\mathcal{I}})_1), \ldots, (\boldsymbol{\mathcal{I}}_n, f(\boldsymbol{\mathcal{I}})_n)\}$, evaluating the performance of the current network parameters and modifying the weights and biases so that $\mathcal{O}(\boldsymbol{\mathcal{I}})$ approaches $f(\boldsymbol{\mathcal{I}})$ as the procedure commences. The goal of this *training* procedure is to produce a NN that has the ability to generalize; to accurately approximate the points used as input-output pairs, but also accurately predict the output in the space spanned by the inputs [5, p. 164].

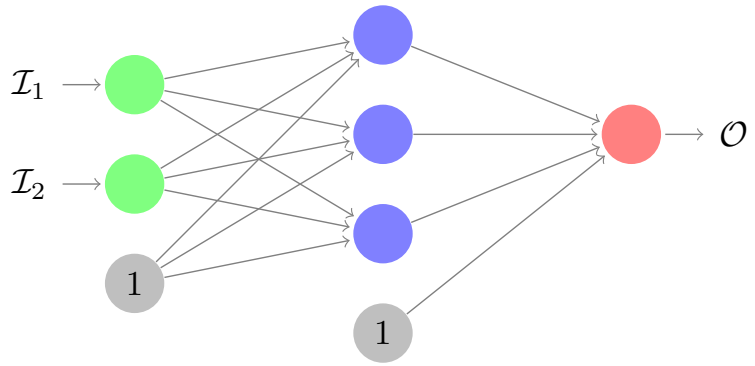


Figure 2.5: A feed-forward neural network with two input nodes (green), one hidden layer with three nodes (blue), and one output layer node (red). Biases are shown in grey.

The optimization methods employed in this work are all based on *gradient descent* (GD), commonly referred to as *backpropagation* algorithms in the context of NNs; they use the gradient of the error function with respect to the parameters of the NN to determine how weights are updated. The algorithms can be roughly divided into two categories: adaptive optimizers, which adapts the step size taken on a parameter-by-parameter basis, and nonadaptive, which uses the same step size for all parameters. A description of the three adaptive optimizers used in this work (RMSProp, Adam, and Nadam) may be found in Sebastian Ruder's review article on GD optimizers [22], while an explanation of the three nonadaptive ones are found later in this section.

There are three ways to handle the training data [22]: stochastic GD, where they are used one-by-one, minibatch GD, where the data is split into subsets, and batch GD, where the entire data set is used at once. This is relevant for the *cost function* (or, in some literature, *error function*), used to evaluate the performance of the network during training. The specific choice of this function is application dependent, but in this work the Mean Square Error (MSE) is used, defined as [23]

$$\varepsilon = \frac{1}{N}\sum_{i=1}^{N}(\mathcal{O}_i - \mathcal{O}_i^{\text{ref}})^2, \tag{2.22}$$

where N is the number of examples in the training subset and $\mathcal{O}^{\text{ref}}$ the reference output in the input-output pairs. The training procedure essentially minimizes ε using an optimization scheme.

The simplest nonadaptive optimization algorithm used in this work is the gradient descent (GD) optimizer. In every iteration a subset of the training data set is fed through the network. ε is evaluated, and the network parameters are updated according to the *update rule* [22, p. 3]

$$\nu_t = \eta \nabla_{\mathcal{P}} \varepsilon(\mathcal{P})|_{\boldsymbol{\mathcal{I}}}, \mathcal{P} = \mathcal{P} - \nu_t, \tag{2.23}$$

where t denotes the current batch, $\nabla_{\mathcal{P}}$ the gradient with respect to the network parameters $\mathcal{P}$ and η the *learning rate*, essentially the step size taken in the direction of the gradient; η may be held constant throughout the training procedure, or lowered as training commences through learning rate decay (or learning rate annealing, as it is sometimes called in literature) [5, p. 115].

A variant of the GD algorithm is gradient descent with momentum (GD+M), which incorporates a term roughly analogous to physical momentum. A ball rolling down a hill accelerates in the direction of the slope, gaining momentum, and the effect is similar in this context. This is realized through a modified update rule [22, p. 4],

$$\nu_t = \gamma\nu_{t-1} + \eta\nabla_{\mathcal{P}}\varepsilon(\mathcal{P})|_{\boldsymbol{\mathcal{I}}}, \mathcal{P} = \mathcal{P} - \nu_t, \tag{2.24}$$

where γ a momentum parameter normally set to around 0.9. The effect is that the algorithm is accelerated in the direction of the gradient.

A variant of the GD+M algorithm incorporates *Nesterov accelerated gradient* (GD+N), which essentially gives the algorithm some notion of where the next update should be; instead of evaluating the gradient at the *current* parameter values $\mathcal{P}$ they are evaluated at an approximation of their next values, $\mathcal{P} - \gamma\nu_{t-1}$. The update rule becomes [22, p. 4]

$$\nu_t = \gamma\nu_{t-1} + \eta\nabla_{\mathcal{P}}\varepsilon(\mathcal{P} - \gamma\nu_{t-1})|_{\boldsymbol{\mathcal{I}}}, \mathcal{P} = \mathcal{P} - \nu_t. \tag{2.25}$$

No matter which optimizer that is chosen, there are two challenges that must be overcome: local minima in ε and *overfitting*.

Local minima

The error function ε is a function of the NN's parameters, i.e. its weights and biases, and is in general not guaranteed to be globally convex [22, p. 3]. It thus has local minima in which the optimization procedure may get stuck. Batch GD is especially susceptible to this, since the parameters are adjusted directly towards the error function's negative gradient for the entire training dataset.

Minibatch and stochastic GD are less susceptible to this, since the functional form of ε differs from iteration to iteration. The algorithm may, but is not guaranteed to, find minima with lower total errors than batch GD. The choice of optimizer may also affect the quality of the fit.

Overfitting

Overfitting occurs when the NN parameters adapt to the specific characteristics of the training data set, such as noise, rather than to the underlying function that was sampled from [5, p. 164]. This leads to situations where the error function approaches zero for the entire data set, but the NN loses the ability to properly generalize.

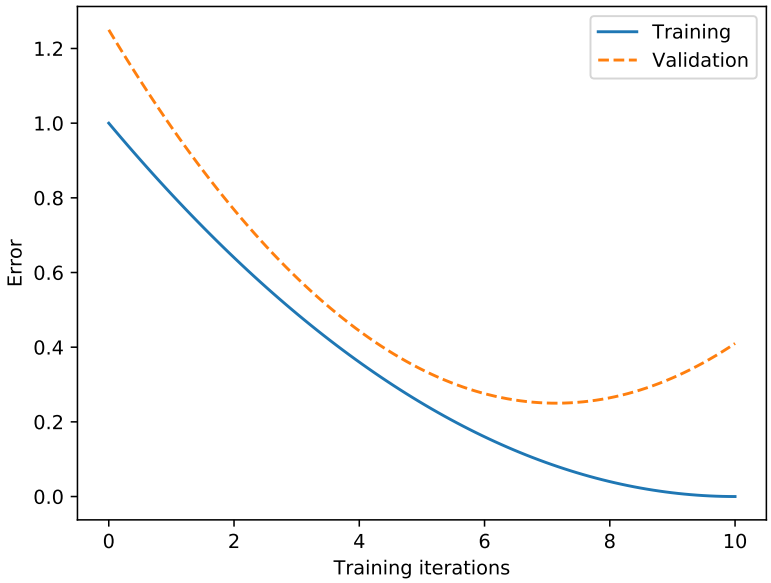


Figure 2.6: An idealized illustration of overfitting.

The simplest way to combat this is *early stopping* [5, p. 173]; simply stop the optimization procedure before overfitting occurs. To accomplish this the data set is split into a the training set, used for the training procedure, and a validation set, used to check for overfitting. The idea is illustrated in Figure 2.6; the training set error keeps decreasing, but the validation error starts increasing at a certain point. The NN is thus overfitting, since it cannot generalize to data sampled from the same source. The ideal stopping point is right when the validation error starts increasing.

In this work, a *test set*[6] is used in addition to the training and validation sets. Despite the latter only being indirectly involved in the training, the fact that the validation set error is used as a measure of when the training procedure is considered finished means that there is no guarantee that the network is able to generalize beyond the training and validation sets [5, p. 171]. The test data set, whose error is evaluated only after training is complete, can be used as a measure of the true accuracy of the network.

Another way of reducing overfitting is *dropout* [24]. In each training iteration nodes are removed from the NN with per-node probability p. At test time, the outgoing weights of nodes that had dropout applied to them are multiplied by p. This has the effect of forcing neurons to adapt to working with randomly chosen subsets of other units, ideally leading to each individual neuron contributing to better predictions on its own rather than just coadapting and developing an overreliance on other neurons [25].

[6] In much of the available literature on this subject the validation set is called test set, and the test set in this context is absent completely.

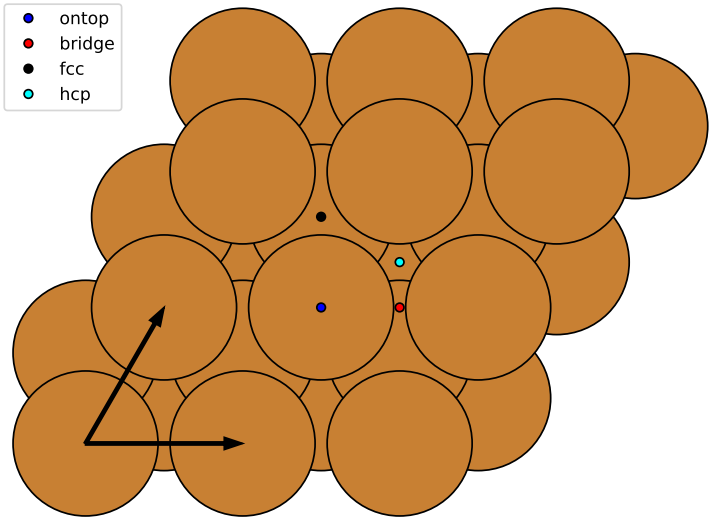


Figure 2.7: An illustration of the Cu(111) surface, its surface lattice vectors and its adsorption sites.

2.3 Surfaces and sticking probabilities

Many bulk solids exhibit a crystalline structure; copper, which we are concerned with in this work, has a face-centered cubic crystal structure [26, p. 175]. By cutting a bulk crystalline solid along some plane a surface facet is formed; the choice of plane results in different surface geometries.

Here we focus on the (111) facet of copper, i.e. Cu(111); a 3-by-3 three-layer slab of this surface is shown in Figure 2.7. Also shown are the locations of commonly *adsorption sites*, which are locations on the surface where adsorbates tend to attach [26, p. 179] as well as the surface lattice vectors [26, p. 172]. These adsorption sites are on top of the surface atoms (*ontop*), in between two surface atoms (*bridge*), and in the space between several surface atoms (*hollow*). The Cu(111) surface has two distinct kinds of hollow sites, denoted by hcp and fcc in Figure 2.7; the former is situated above a second-layer atom, while the latter is above a third-layer atom.

The *sticking probability* s, i.e. the probability that an impinging molecular entity sticks to a surface, is simply the fraction of impinging events that result in sticking. It may be found experimentally through e.g. molecular beam experiments [27, p. 202], where molecules are fired towards the surface under study with a well-defined kinetic energy, or computationally through MD techniques [21, p. 81], which in this context are often called *classical dynamics* or *classical trajectory* methods.

3
Methods

This chapter is concerned with explaining the specifics of how DFT was used to gather data, how to adapt the use of neural networks to the specific task of finding representations for the H_2/Cu(111) potential energy surface and using it to estimate the dissociative sticking probability of the system.

3.1 DFT and data collection

The DFT calculations were performed using GPAW [28, 29] 1.3.0 along with the Atomic Simulation Environment (ASE) [30] 3.15.0 package in Python. ASE is a tool used to define, handle and manipulate atomic systems as Python objects, while GPAW handles the grunt work of DFT calculations. ASE is also used to facilitate MD calculations via the velocity-Verlet method using both first principles and NNPES for force calculations.

The DFT calculations were carried out using the PBE exchange-correlation functional and a plane-wave basis set, using an energy cutoff of 300 eV. The real-space grid spacing was 0.21 Å and the Brillouin zone was sampled using Monkhorst-Pack sampling on an (8,8,1) grid. The computational supercell was chosen to be a 2-by-2 three-layer slab with 10 Å of vacuum on each side.

The computational parameters were converged with respect to the bulk lattice constant, but were not completely converged with respect to the potential energy. A decision was made not to go further to reduce the computational cost of generating a dataset for training neural networks on, as fairly many points are necessary for this.

Potential energies for the hydrogen-copper systems were defined relative to the free surface and hydrogen molecule. If E_{tot} is the system's total energy, the energy relative to these quantities is

$$E_{\mathrm{rel}} = E_{\mathrm{tot}} - E_{\mathrm{H_2}} - E_{\mathrm{surf}}. \tag{3.1}$$

The H_2/Cu(111) system was sampled through a mix of manually placing the hydrogen atoms around adsorption sites and *ab initio* molecular dynamics. For the former, the hydrogen molecule's center-of-mass was fixed at different sites while varying the height to the surface's topmost layer, the separation between the two atoms and the angle the molecular axis makes to the surface. MD sampling was done by randomly picking an impact site on the surface and then randomly picking the height to the

surface between $[3, 5]$ Å, the hydrogen-hydrogen separation between $[0.6, 0.8]$ Å, the translational kinetic energy in the direction of the surface normal of each atom between $[0.08064, 1.134]$ eV, and the molecular axis randomly orientated in space. MD simulations were done with a timestep of 1 fs.

Finally, any data points with a potential energy exceeding 8 eV according to the definition of Equation 3.1 were discarded. The resulting final dataset consisted of 5342 DFT points, of which roughly 2/3 came from *ab initio* MD.

Note: The original idea was to allow the surface slab to relax, which in this case makes it contract slightly in the z direction. However, this procedure was accidentally *not* done for the data points collected with MD.

3.2 Neural network optimization

The training procedure is done using TensorFlow [31] 1.12.0 along with the high-level API Keras [32] 2.2.4 to construct the networks, and to facilitate early stopping as well as learning rate decay.

In addition to the network parameters (i.e. weights and biases) discussed in Section 2.2 there are also *hyperparameters* that affect the network's performance; these are set before the training procedure begins and includes the number of hidden layers, the number of neurons in each hidden layer, the learning rate, as well as optimizer-specific quantities such as momentum. The choice of optimization algorithm, activation function and weight initialization procedure may also affect the performance. In this work a few different optimizers are used and compared to each other, the choice of activation function is the hyperbolic tangent as has been done in previous work [23], and network parameters initialized using the Glorot initialization scheme [33], in which the biases are set to zero and initial values for the weights of each layer are drawn from a uniform distribution between $[-M, M]$, where $M = \sqrt{\frac{6}{N_{\text{in}}+N_{\text{out}}}}$. N_{in} and N_{out} are the numbers of connections to and from the layer, respectively.

Hyperparameters are chosen through a trial-and-error procedure. As a starting guess a network with 25 neurons in each layer (i.e. a 25-25 network layout) is chosen, as that was used by Jiang and Guo [34] to demonstrate the efficacy of their algorithm. Initial hyperparameters are those recommended by Sebastian Ruder [22], and optimizer-specific parameters like momentum are not optimized. The initial minibatch size is chosen to be 64.

The optimization procedure was chosen to proceed as follows:

1. Different optimizers are compared and a relevant early stopping criterion established.
2. Learning rates are tried out with a few optimizers that perform well.
3. An appropriate learning rate is used to test $n-n$ networks, i.e. with the same

number of neurons in both hidden layers.
4. The best-performing $n - n$ network is trained with different batch sizes.
5. Learning rates are revisited, both with and without learning rate decay.
6. $n - m$ networks are tried out with the chosen learning rate.

The performance of the networks are checked using the average RMSE on the validation data set.

Finally, the best performers are trained many times and those producing the *lowest* validation set RMSE are selected to produce the final model. The final model is the *ensemble average* of several networks, producing a *committee machine* [35, p. 375]; the output of this committee machine for a given input is simply the average of the outputs from the neural networks it consists of.

Learning rate decay is done via a simple linear scheme, where an initial learning rate l_0 is chosen and then lowered after each epoch until it reaches l_0/f after t epochs, after which is it held constant.

3.3 Neural Network Potential Energy Surfaces

As discussed in Chapter 2.2, NNs may be used to approximate arbitrary functions. The PES of a general atomic system lacks exact analytic representation, since its Schrödinger equation cannot be solved, but an approximation may be found by fitting some appropriate function to discrete points on the PES sampled either experimentally or computationally. Design decisions have to be made with conventional fitting methods; what type of function should be selected? This requires some knowledge of how the potential surface behaves. By instead using a NN no such information is needed, since the universal approximation theorem ensures that they are flexible enough to adapt to any reasonable data set [23, p. 1034].

This work is concerned with hydrogen molecules interacting with periodic frozen solid copper surfaces, so a natural approach would be to use the Cartesian coordinates of the hydrogen atoms as inputs to the NN. A few problems arise: for instance, the PES is invariant to permutations of identical atoms, i.e. exchanging the positions of the hydrogen atoms, and invariant to translations equivalent to multiples of the surface lattice vectors[1] [34]. The neural network potential thus needs to fulfill $\mathcal{O}(\mathbf{x}_1, \mathbf{x}_2) = \mathcal{O}(\mathbf{x}_2, \mathbf{x}_1)$ and $\mathcal{O}(\mathbf{x}_1, \mathbf{x}_2) = \mathcal{O}(\mathbf{x}_1 + n\mathbf{a}, \mathbf{x}_2 + n\mathbf{a})$, where $\mathbf{x} = (x, y, z)$ are the Cartesian coordinates of the hydrogen atoms, n is an integer and $\mathbf{a}$ a surface lattice vector.

These problems could be circumvented by adding additional input-output pairs to the data set: for every $(\mathbf{x}_1, \mathbf{x}_2)$, add $(\mathbf{x}_2, \mathbf{x}_1)$, $(\mathbf{x}_1 + \mathbf{a}, \mathbf{x}_2 + \mathbf{a})$ and so on. This only approximately fulfills the invariance conditions, however. A much more elegant and robust solution is to build the invariances directly into the NN's input space. This

[1] A periodic surface has a periodic PES; every fcc site on a monatomic fcc(111) surface is energetically equivalent, for instance.

may be achieved in several ways, but in this work the approach of Jiang and Guo is used [34].

Through a Fourier series expansion of surface reciprocal lattice vectors two *primitive symmetry functions* that capture the translational periodicity of a surface may be derived. For the fcc(111) surface one may use

$$p_1^{\mathrm{fcc}(111)}(x,y) = \cos\left(\frac{4\pi y}{a\sqrt{3}}\right) + 2\cos\left(\frac{2\pi x}{a}\right)\cos\left(\frac{2\pi y}{a\sqrt{3}}\right), \tag{3.2a}$$

$$p_2^{\mathrm{fcc}(111)}(x,y) = \sin\left(\frac{4\pi y}{a\sqrt{3}}\right) - 2\cos\left(\frac{2\pi x}{a}\right)\sin\left(\frac{2\pi y}{a\sqrt{3}}\right). \tag{3.2b}$$

$a_{\mathrm{fcc}(111)}$ is the surface lattice constant, equal to the closest distance between two atoms on an fcc(111) surface, related to the bulk lattice constant a_{fcc} by $a_{\mathrm{fcc}(111)} = a_{\mathrm{fcc}}/\sqrt{2}$ [34]. To help describe the behaviour of the PES for large atom-surface separation, i.e. for large z, an additional function is used:

$$p_z(z) = \exp(-\lambda z), \tag{3.3}$$

where λ is a positive constant set to unity in this work.

These three functions still lack permutational invariance. They may be used to form *permutational invariant polynomials* (PIPs) that take the property into account, however:

$$G_1 = p_1(x_1,y_1) + p_1(x_2,y_2), \tag{3.4a}$$

$$G_2 = p_1(x_1,y_1)p_1(x_2,y_2), \tag{3.4b}$$

$$G_3 = p_2(x_1,y_1) + p_2(x_2,y_2), \tag{3.4c}$$

$$G_4 = p_2(x_1,y_1)p_2(x_2,y_2), \tag{3.4d}$$

$$G_5 = p_z(z_1) + p_z(z_2), \tag{3.4e}$$

$$G_6 = p_z(z_1)p_z(z_2). \tag{3.4f}$$

To improve the fit an exponential function of the hydrogen-hydrogen separation r is used,

$$G_7 = p_r(r), \tag{3.5}$$

as well as functions of the hydrogen molecule's centre of mass (X,Y,Z):

$$G_8 = p_1(X,Y), \tag{3.6a}$$

$$G_9 = p_2(X,Y), \tag{3.6b}$$

$$G_{10} = p_z(Z). \tag{3.6c}$$

The centre of mass coordinates are simply $X = (x_1 + x_2)/2$ and so on.

Using just these ten functions includes some unphysical symmetries, leading to a lesser fit [34]. To remedy this, three cross terms are added:

$$G_{11} = p_1(x_1,y_1)p_2(x_1,y_1) + p_1(x_2,y_2)p_2(x_2,y_2), \tag{3.7a}$$

$$G_{12} = p_1(x_1,y_1)p_z(z_1) + p_1(x_2,y_2)p_z(z_2), \tag{3.7b}$$

$$G_{13} = p_2(x_1,y_1)p_z(z_1) + p_2(x_2,y_2)p_z(z_2) \tag{3.7c}$$

Together, the thirteen functions G_1 to G_{13} may be used as input space of a NN to fit the PES of H_2 over fcc(111) surfaces.

The dataset gathered as described in Section 3.1 was transformed as described here, and then normalized as Jörg Behler suggests [23, p. 1044] so that each G_i had zero mean and unit variance.

3.3.1 NNPES MD for Sticking Estimation

The NNs, once the optimization procedure is complete, are an analytic approximation of the potential energy surface; as such, the negative gradient of their mathematical expression with respect to the Cartesian coordinates of the hydrogen atoms constitutes an approximation of the forces acting upon them according to Equation (2.17). This has been implemented in a script that allows the calculation of energies and forces at discrete points with ASE; the code may be found in Appendix C. This allows one to use ASE to perform MD simulations using the velocity-Verlet algorithm to approximate the dissociative sticking probability for hydrogen molecules approaching a Cu(111) surface.

The trajectories were chosen to start with the hydrogen center-of-mass at a distance of 5 Å from the surface. The collision point on the surface was chosen randomly and molecules initially randomly orientated in space. The incidence angle was normal to the surface in all cases, with all translational kinetic energy in the direction of the surface normal; the molecule is thus initially nonrotating. Two variants of MD were performed: classical and quasiclassical. In both cases, simulations continue until dissociation occurs, which is discussed below, or until the molecule's center-of-mass exceeds a distance of 5 Å from the surface's topmost layer.

In the classical simulations the zero-point energy of the molecule is neglected; the initial separation was set to be equal to the equilibrium distance between the hydrogen atoms as predicted by the NNPES.

The quasiclassical simulations incorporate the zero-point energy by initially inducing a classical vibration along the molecular axis [36, p. 82]. The phase of this vibration was randomized by starting the molecule off at one of the oscillation's turning points and then integrating the equations of motion using the velocity-Verlet algorithm for a time equal to $t\tau$, where t is a random number in $[0, 1]$ and τ the time it takes for the molecule to go through one full oscillation. The separation and momentum after this time is set to be the initial conditions for the trajectory, after randomly orientating the molecule and adding the translational kinetic energy.

The NNPES description does not allow for the decoupling of the molecule and surface. For this reason the vibrational energies and equilibrium separation of the hydrogen molecule was determined with the molecular axis oriented along the x-axis of the computational cell, with the center-of-mass located at at a z-distance of 5 Å from a top-layer atom of the surface.

The sticking probability at a given translational kinetic energy was calculated as the fraction of trajectories that result in dissociation [36, p. 84]. Determining this requires some sort of criterion to be defined. This was done by examining two-dimensional cuts of the potential energy in regions around a few dissociaton sites; these plots are found in Figures 5.1b, 5.2b, 5.3b, and 5.4b. The molecule was considered to be dissociated if its potential energy and the hydrogen-hydrogen separation would be enough to get over the energy barriers in those figures; the specific values chosen are shown in Equation 5.3.

In addition, a simpler criterion was tried out for the quasiclassical case; the molecule was considered to be dissociated if the hydrogen-hydrogen separation exceeded 2 Å with its center-of-mass z-coordinate within 2.5 Å of the surface's topmost layer.

The sticking probability estimation is in essence a Monte Carlo method [36, p. 82]; as such, error estimates for the sticking probability can be calculated as the standard error, defined as [37, p. 306]

$$\sigma_{\overline{x}} = \frac{\sigma}{\sqrt{N}}, \tag{3.8}$$

where N is the number of trajectories (i.e. samples) at a particular incident kinetic energy and σ the standard deviation of the sticking probability based on those N samples.

4
Optimization of neural networks

In this chapter the results of the optimization procedure is described step-by-step.

4.1 Network parameter optimization

Due to the number of hyperparameters being relatively large, combined with the inherent stochasticity of the methods, finding the global minimum (i.e. the ideal neural network for the task at hand) is near impossible. One can tune the parameters to get close, however. Weight initialization is one such parameter, but the Xavier initialization scheme will be used to narrow down the number of hyperparameters. The optimization procedure is divided into five steps: a rough selection of learning rates, a rough selection of the number of hidden layer neurons, a selection of the batch size to use when training the neural networks, a finer search of the learning rate and, finally, a finer search of the network layout.

Early stopping criterion and optimizer comparison

The six optimization algorithms[1] were run 5 times each for 150000 epochs at recommended settings with a batch size of 64, both to get a rough idea of how the algorithms perform on the dataset and to investigate appropriate early stopping criteria. For runs exhibiting minima only after close to 150000 epochs an additional 100000 epochs were run. The average validation RMSE, based on the lowest value of each run, is plotted for each algorithm in Figure 4.1, giving a rough idea of how each algorithm performs at the recommended parameter values.

Figure 4.2 shows a plot of how the validation set RMSE behaved in one run of each nonadaptive optimizer. Due to the oscillatory behaviour of this value, each point on the curves has been averaged over 100 epochs. The behaviour resembles that of the idealized illustration of Figure 2.6 - a clear downward trend that eventually reaches a plateau. An early stopping criterion of 5000 epochs was enough to capture the observed minima during every run of each optimizer.

Figure 4.3 shows the validation set RMSE for one run each of the adaptive optimizers. Each point on the curves has been averaged over 100 epochs due to oscillations, just like in Figure 4.2. The RMSProp and Nadam curves are volatile, fluctuating up

[1] Gradient descent (GD), gradient descent with momentum (GD+M), Nesterov-accelerated gradient descent (GD+N), RMSProp, Adam, and Nadam.

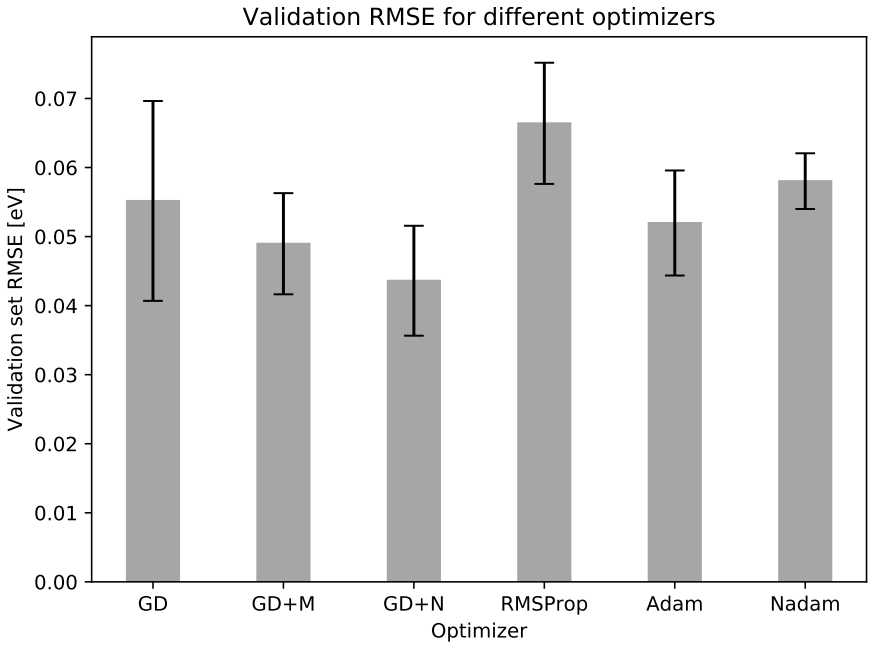


Figure 4.1: Mean validation set RMSE for neural networks with 25 neurons per hidden layer trained with different optimizers. Results were averaged over five runs. Error bars represent the standard deviation.

and down significantly. A different approach than the early stopping scheme may be necessary to efficiently make use of these optimizers.

The adaptive algorithms will be excluded from further studies, due to the difficulty in choosing appropriate early stopping criteria and the fact that GD+M and GD+N appear to perform better on this dataset. The GD optimizer is also excluded as it performed worse and exhibits much slower convergence than the two other nonadaptive algorithms here.

Learning rate

A 25-25 neural network was trained using the GD+M and GD+N optimizers 5 separate times each at different learning rates with an early stopping criterion of 5000 epochs and a batch size of 64. Mean validation set RMSEs for each learning rate are plotted in Figure 4.4, where the data for $l_0 = 10^{-2}$ is taken from Figure 4.1.

For the GD+M optimizer the results are clear; a learning rate of 10^{-2} performs the best here. 10^{-3} yielded slightly lower RMSE with GD+N than 10^{-2}, but the difference is nearly negligible.

Hidden layer neurons

The GD+M and GD+N optimizers were used to train $n - n$ neural networks, i.e. networks with the same number of neurons in both hidden layers, five times each

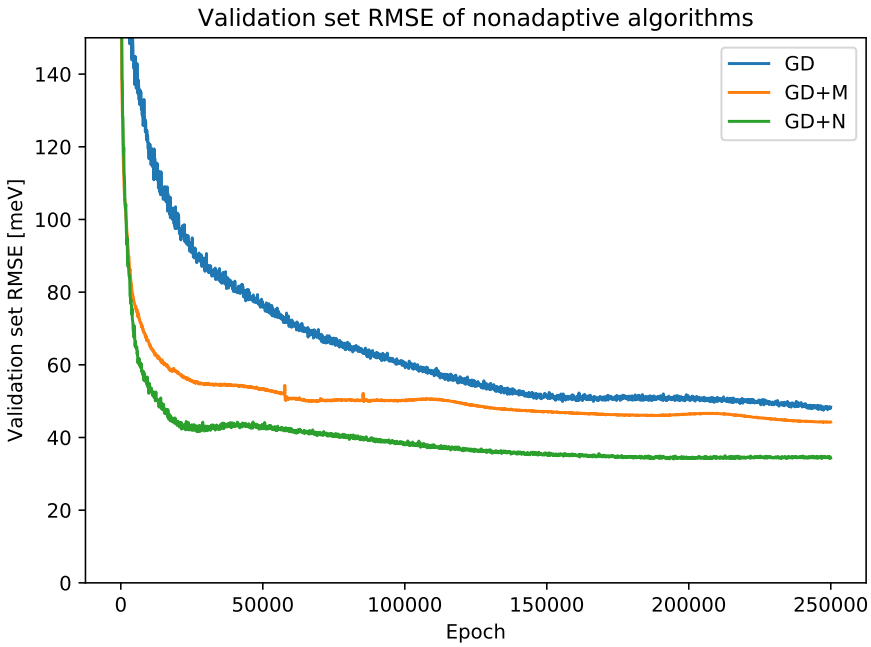


Figure 4.2: Validation RMSE as a function of epoch for one run each of the GD, GD+M and GD+N optimizers. Each point on the curve represents an average over 100 epochs.

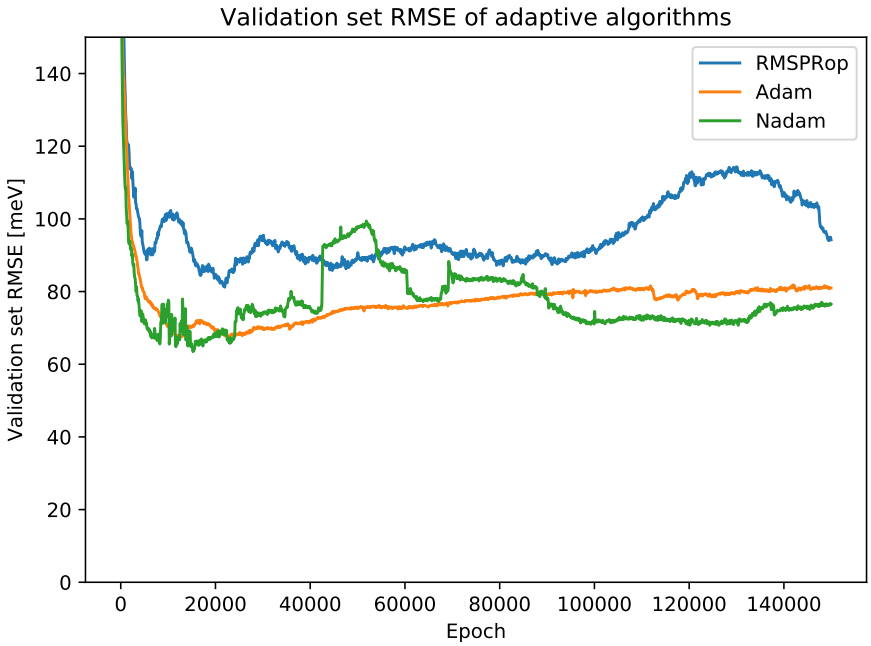


Figure 4.3: Validation RMSE as a function of epoch for one run each of the RMSProp, Adam and Nadam optimizers. Each point on the curve represents an average over 100 epochs.

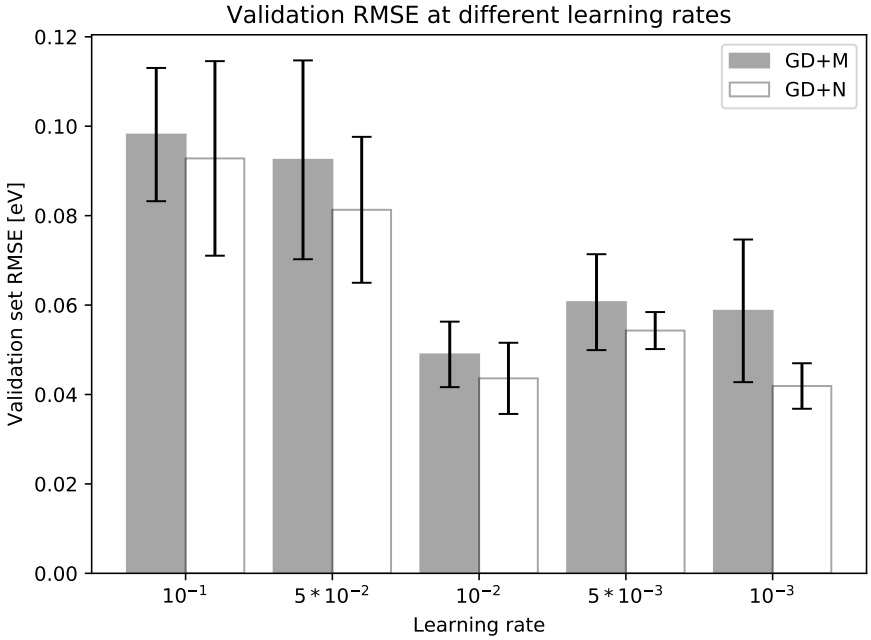


Figure 4.4: Average validation set RMSE for neural networks with 25 neurons per hidden layer at different learning rates. Results were averaged over five runs. Error bars represent the standard deviation.

using a learning rate of 10^{-2} and a batch size of 64. The early stopping criterion was 5000 epochs. Average validation RMSEs are plotted in Figure 4.5, where the bars for the $25 - 25$ configuration are the same as in Figure 4.4.

While the 25-25 configuration yields the lowest mean RMSE for both optimizers, no clear correlation between the error and number hidden layer neurons emerges here. What can be concluded at this point, however, is that the GD+N optimizer performs better GD+M on average; the latter is therefore excluded from further investigation.

Batch size

The effect of different batch sizes on validation RMSE was investigated by using the GD+N optimizer to train neural networks in a 25-25 hidden layer configuration with a learning rate of 10^{-2}. Figure 4.6 shows examples of the behaviour of the validation set RMSE during training; it oscillates considerably more at low batch sizes than large ones. The convergence criterion was chosen to be five times larger than in the previous trials, 25 000 epochs, to prevent early convergence. Average RMSEs over ten separate runs for each batch size are shown in Figure 4.7, where the values for the 25-25 network configuration in Figure 4.5 is reused for the batch size of 64.

In addition to exhibiting more oscillatory behaviour batch sizes below 32 yield higher RMSEs than 64 and 128. A batch size of 128 yields the lowest RMSEs and variances here, though a batch size of 64 is only slightly worse.

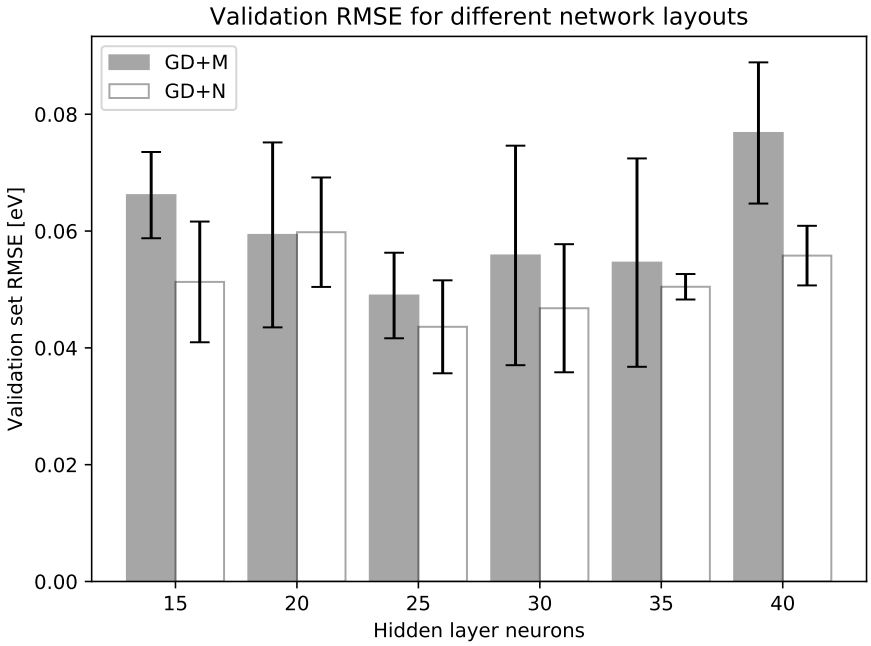


Figure 4.5: Average validation set RMSE for neural networks with equal number of neurons in the first and second hidden layers. The learning rate was 10^{-2} and results averaged over five runs. Error bars represents the standard deviation.

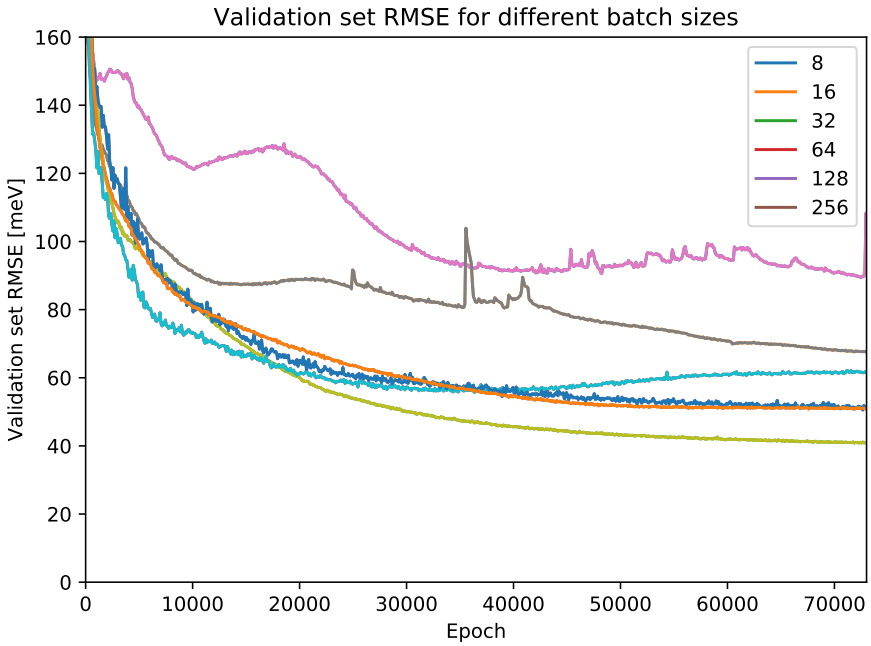


Figure 4.6: Validation RMSE for one run each at different batch sizes. Each point on the curves represents an average over 100 points.

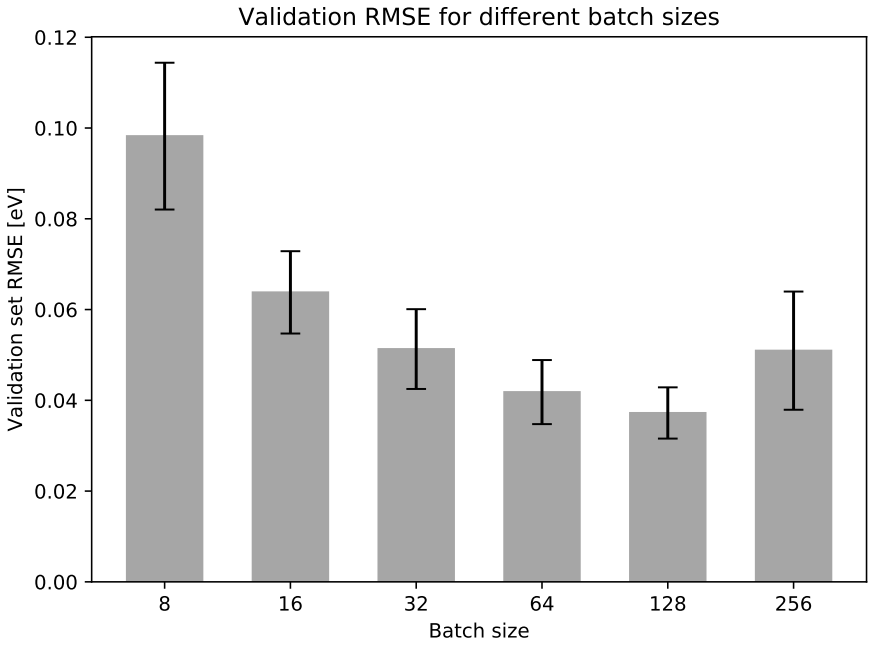


Figure 4.7: Average validation set RMSE for a 25-25 neural network at different batch sizes. The learning rate was 10^{-2} and results were averaged over ten runs. Error bars represents the standard deviation.

Learning rate revisited

Neural networks in a 25-25 hidden layer configuration were trained using GD+N and a batch size of 128 at different learning rates, both with and without decay. A linear decay schedule was used, where the learning rate decays from l_0 initially to l_0/f at epoch t after which it was held constant. An early sticking criterion of 25 000 was used to keep consistency with the batch size optimization.

Figure 4.8 shows the average validation RMSE of 10 runs each at different values of l_0, f and t. The data for $\{l_0 = 10^{-2}, f = 1, t = 0\}$, i.e. without weight decay, is the same as that for the batch size of 128 in Figure 4.7. Applying any decay in this manner has a negative impact on the RMSE for $l_0 = 5 * 10^{-2}$ and $l_0 = 10^{-2}$, but slightly increases the performance in a few other cases. It is clear that a learning rate of 10^{-2} without any decay yields the lowest error of the tried parameters using linear decay.

Hidden layer neurons revisited

The effects of the hidden layer configuration on validation RMSE was investigated by training networks with varying numbers of first and second hidden layer nodes. Networks with a hidden neuron number between 15 and 40 per layer were trained using the GD+N optimizer with a learning rate of 10^{-2} and a batch size of 128. The mean RMSE based on 10 separate runs for these are shown in Figure 4.9, where

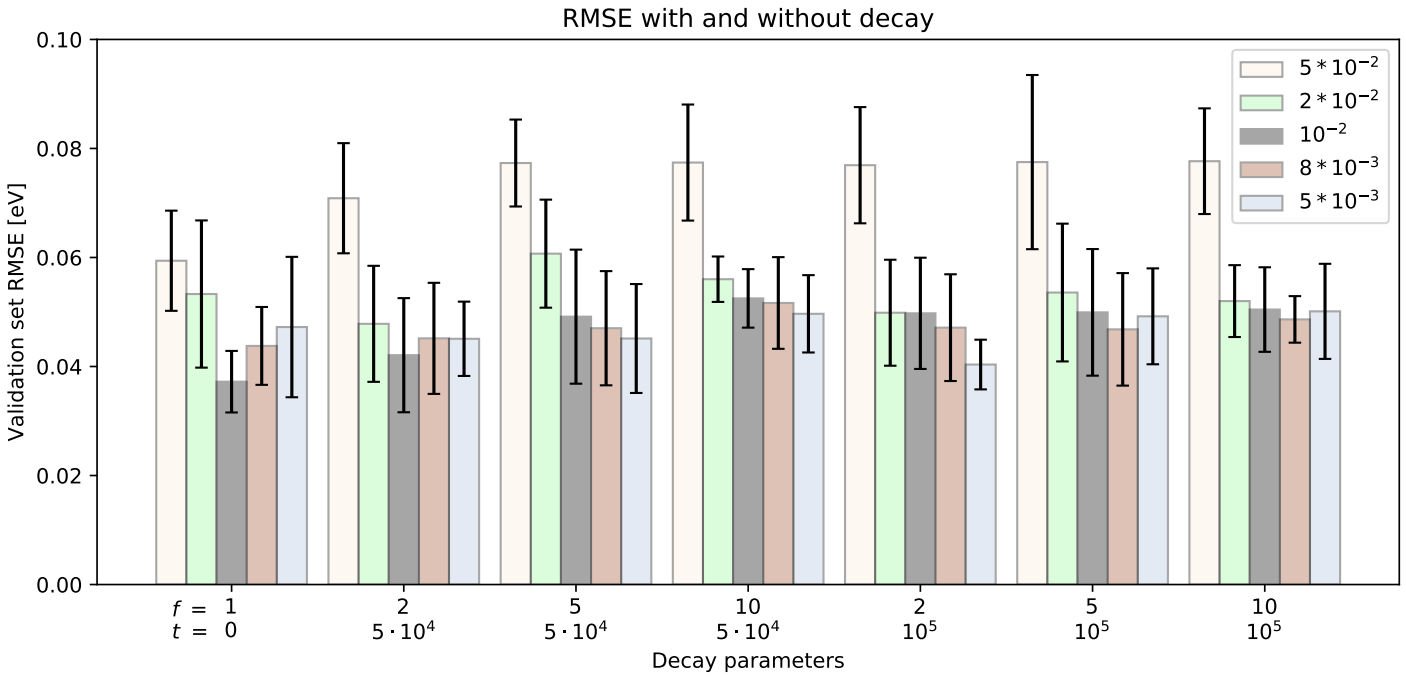


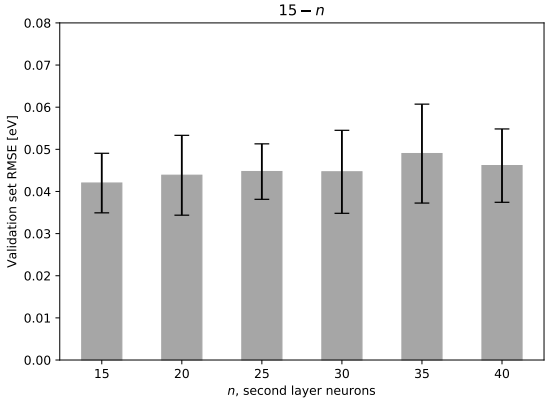
Figure 4.8: Mean validation set RMSE for a 25-25 network configuration trained with the GD+N optimizer using a batch size of 128 at different learning rates and decay parameters. Averages are based on 10 separate runs. Error bars represent the standard deviation.

each subfigure represents a set number of neurons in the first hidden layer.

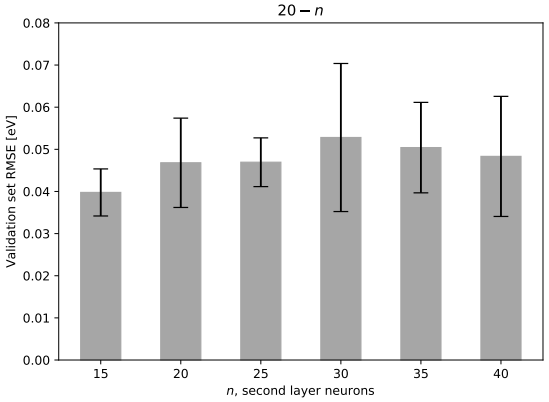
No systematic effect on the validation RMSE can be deduced here. The five network configuration that performed best are tabulated in Table 4.1, which all produced a mean RMSE of around 40 meV with standard deviations between about 5 and 10 meV.

Layout	RMSE$\pm\sigma$ [meV]
15-15	42.0 ± 7.1
20-15	39.8 ± 5.6
25-20	42.8 ± 10.3
25-25	37.2 ± 5.6
25-40	40.0 ± 7.7

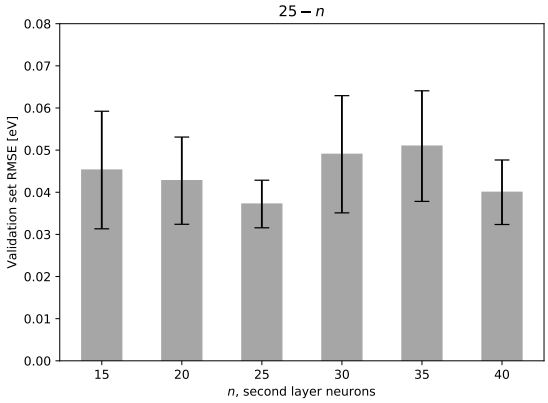
Table 4.1: Table of the five neural network layouts producing lowest mean validation RMSE in Figure 4.9. σ is the standard deviation.



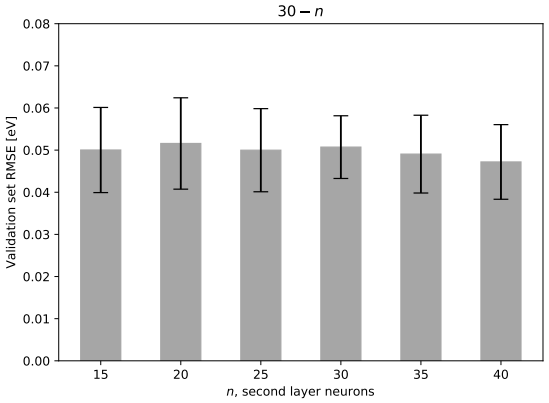
(a) 15 neurons.



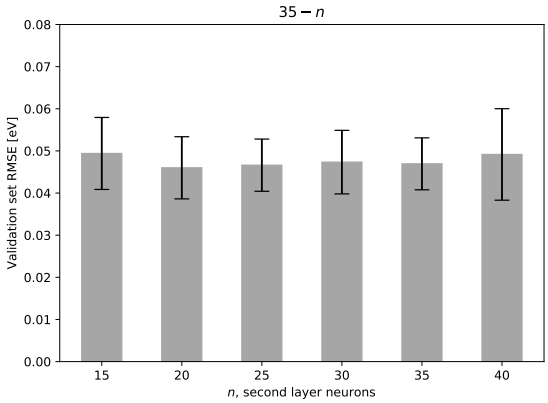
(b) 20 neurons.



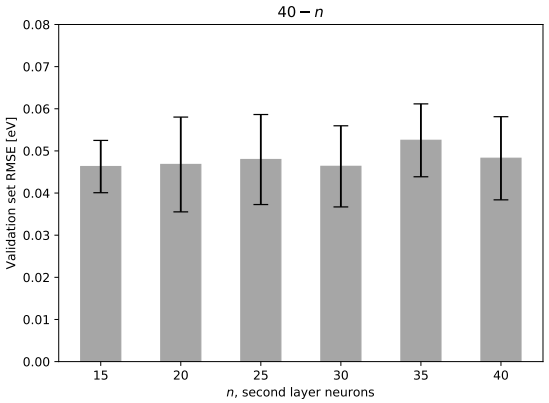
(c) 25 neurons.



(d) 30 neurons.



(e) 35 neurons.



(f) 40 neurons.

Figure 4.9: Mean validation set RMSE for different neural network configurations, averaged over 10 runs. Each figure represents a fixed number of neurons in the first hidden layer, with the x-axis representing the number in the second hidden layer. Error bars represent the standard deviation.

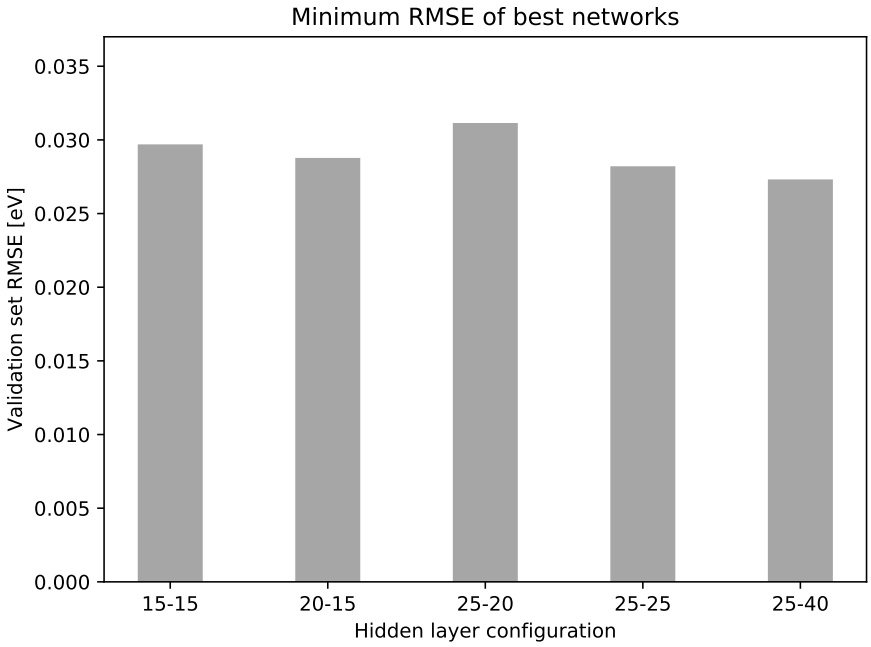


Figure 4.10: Best-of-50 validation set RMSE for the five models in Table 4.1.

4.2 Iterating upon the best models

Comparison between candidates

The five different neural network configurations in Table 4.1 were trained 50 times each with the GD+N optimizer using a batch size of 128 and a learning rate of 10^{-2}. The *lowest* validation set RMSE of each configuration is plotted in Figure 4.10.

The minimum RMSE observed is around 30 meV for all the network configurations. 25-40 performs best here, with a RMSE of about 27.0 meV.

To check that the DFT sampling density is sufficient in the regions relevant to the dissociation dynamics the 20-15 network whose RMSE is shown in Figure 4.10 was used to generate 100 classical trajectories using the velocity-Verlet algorithm; the predicted energies of each neural network for these trajectories were then compared. It was found that the energy curves behave differently enough in certain regions to suggest that the sampling is insufficient; an example of this is shown in Figure 4.11.

Comparison between candidates - Extended dataset

An additional 288 data points were collected by generating short MD trajectories with starting points in the configurations mentioned above. These were added to the dataset used previously, and this extended dataset was split into new training (60 %), validation (20 %) and test (20 %) sets randomly.

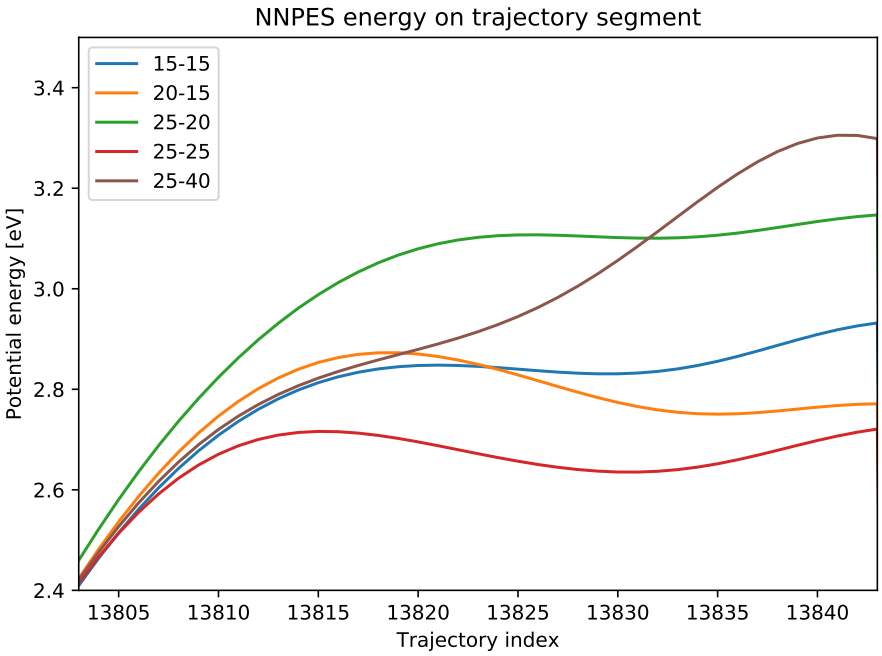


Figure 4.11: Potential energy as predicted by the five different networks from Figure 4.10 for a small part of a classical trajectory generated with the 20-15 network.

The network configurations from Table 4.1 were again trained 50 times each using the same parameters as those used for generating Figure 4.10, but with the extended dataset. The resulting minimum RMSEs for each configuration is plotted in Figure 4.12.

The addition of additional samples yielded a lower RMSE for the 15-15 and 25-20 configurations, but slightly higher for 25-25 and 20-15. The RMSE of the 25-40 network increased with over 6 meV.

The sampling density was checked once again by generating 100 new trajectories using the 20-15 network, on which the predicted energy between the network configurations was compared. The energy curves agree better than previously, but the 25-40 and 20-15 curves exhibit behaviour that differs from the others in certain regions; an example of this is shown in Figure 4.13.

Dropout

The effects of applying dropout to the hidden layers of a 25-20 network configuration is studied using the extended dataset from before. Training was done 20 separate times with differing dropout and learning rates, with a batch size of 128 using the GD+N optimizer.

The minimum validation RMSEs for each parameter configuration are plotted in Figure 4.14. It is clear that applying dropout in the first hidden layer significantly increases the error compared to not applying any at all. Applying dropout only

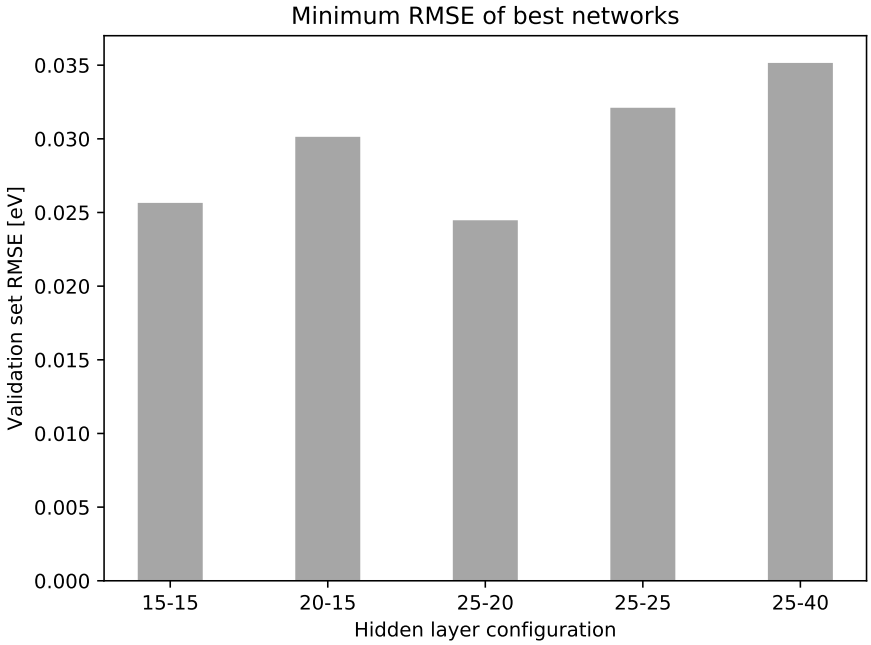


Figure 4.12: The same as Figure 4.10, but the networks were trained on the extended dataset.

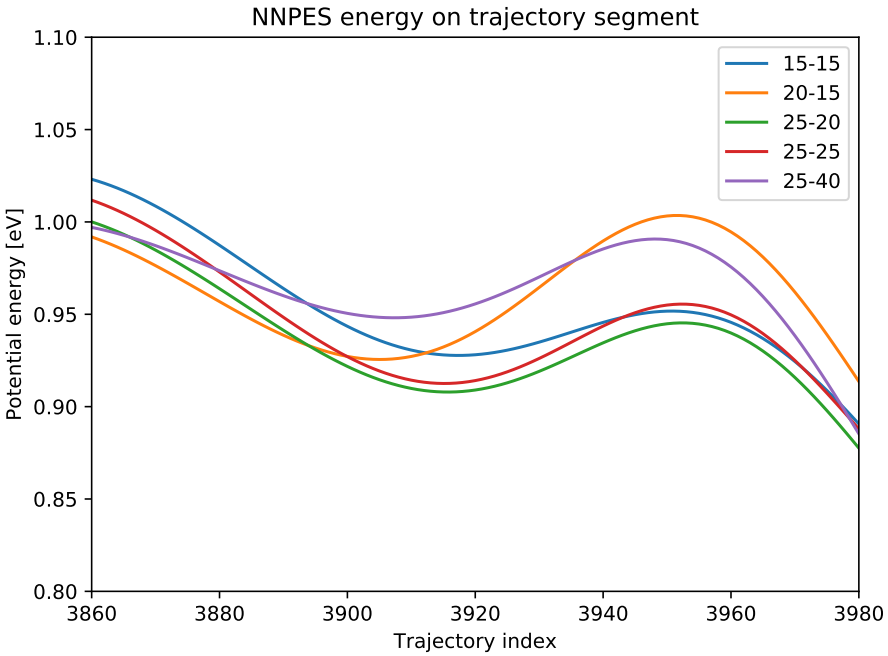


Figure 4.13: Potential energy as predicted by the five different networks from Figure 4.12 for a small part of a classical trajectory generated with the 20-15 network.

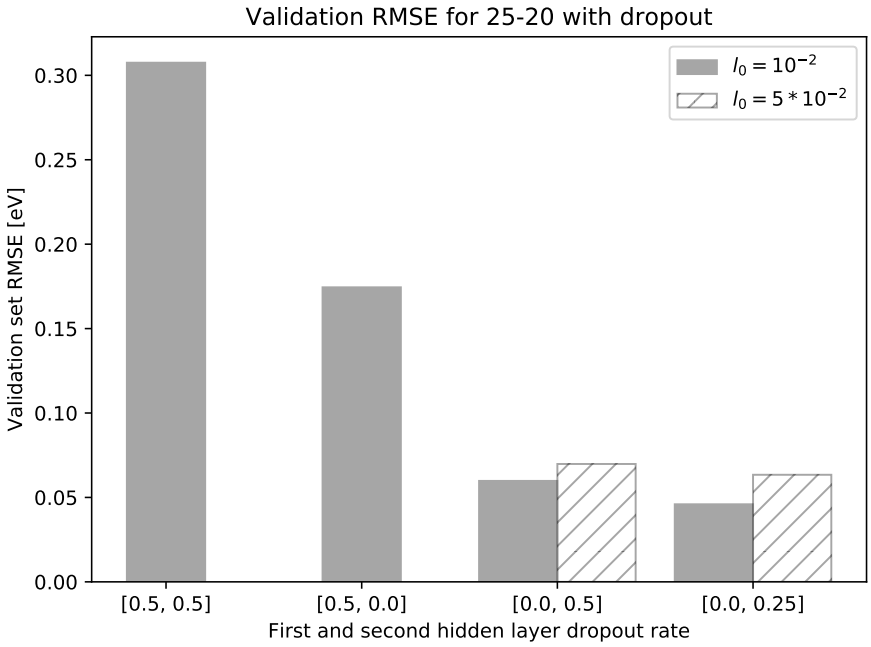


Figure 4.14: Best-of-20 validation set RMSE for a 25-20 network configuration with dropout applied.

to the second layer yields RMSEs closer to those observed in Figure 4.12, but still yields worse performance.

Srivasteva *et al* suggested that increased learning rates when dropout is applied may improve performance [25], but the opposite effect is observed here; using $5 * 10^{-2}$ gives higher RMSEs than 10^{-2} when dropout is applied to the second hidden layer.

Final model selection and evaluation

A committee machine consisting of the 25-25, 25-20 and 15-15 networks from Figure 4.12 is chosen to be the final model.

The accuracy of the model should be based on the test set, which has remained untouched until now. The test set RMSE of the committee machine is about 20.3 meV; the original dataset includes several outliers with energies too high to be relevant to the dissociation of hydrogen on Cu(111), however, as can be seen in the elbow plots in the next chapter. A more realistic error estimation is to take the RMSE based on points whose energies are below 3 eV, at which point the test RMSE is reduced to 19.5 meV.

5

Neural Network Potential Energy Surface

In this chapter the final NNPES, i.e. the comittee machine from before, is looked at in greater detail. It is also used to estimate the sticking probability of hydrogen dissociation on Cu(111) surfaces, which is compared to experimental and theoretical results.

Energy surface

The NNPES is visualized by fixing the hydrogen molecule to be perpendicular to the surface, with its center of mass located between two adsorption sites from Figure 2.7. The molecule's distance from the surface is the varied, as well as the separation between its two atoms. The PES is then plotted as contours, giving an idea of how the energy landscape looks around these locations.

Figure 5.1 shows the PES for dissociation with the center-of-mass above a bridge site, with the hydrogen atoms angled towards two adjacent hollow sites; in Figure 5.1a the underlying DFT data is shown, while Figure 5.1b shows the NNPES description. Figures 5.2a and 5.2b shows the PES for dissociation with the hydrogen center-of-mass located between two fcc sites. Figures 5.3a and 5.3b shows the PES for dissociation with the hydrogen center-of-mass located between two hcp sites. Figures 5.4a and 5.4b shows the PES for dissociation with the hydrogen center-of-mass atop an ontop site, with the hydrogen atoms angled towards hollow sites on its opposite sides.

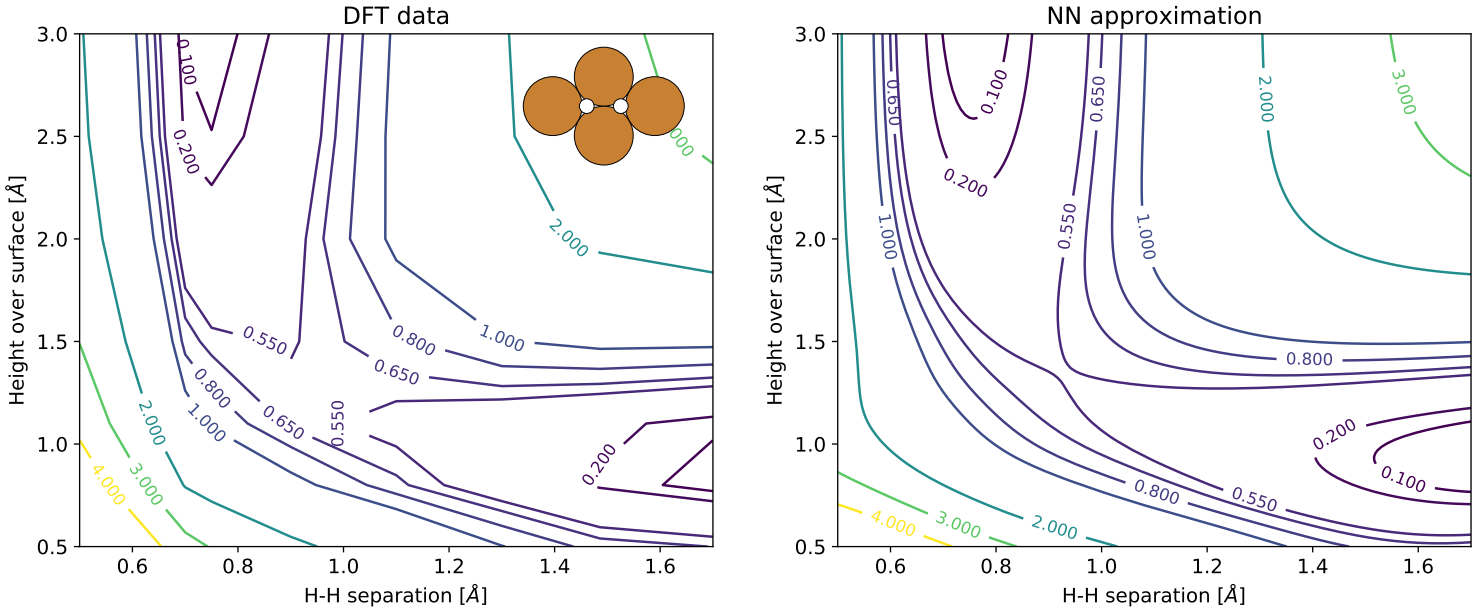
The magnitudes and approximate locations of the energy barriers in the NNPES are tabulated in Table 5.1. The zero-point energy, i.e. the lowest energy state possible for the hydrogen molecule, is 0.273 eV; the corresponding experimental value is 2179.3 cm^{-1} [38], which corresponds to around 0.270 eV.

Sticking

The dissociative sticking probability of hydrogen molecules on Cu(111) was evaluated using the NNPES by generating classical and quasiclassical trajectories. The hydrogen center-of-mass was placed 5 Å away from the surface and the molecule's orientation in space randomized. The classical trajectories started the molecule off in its equilibrium position, while the quasiclassical trajectories started it off in its

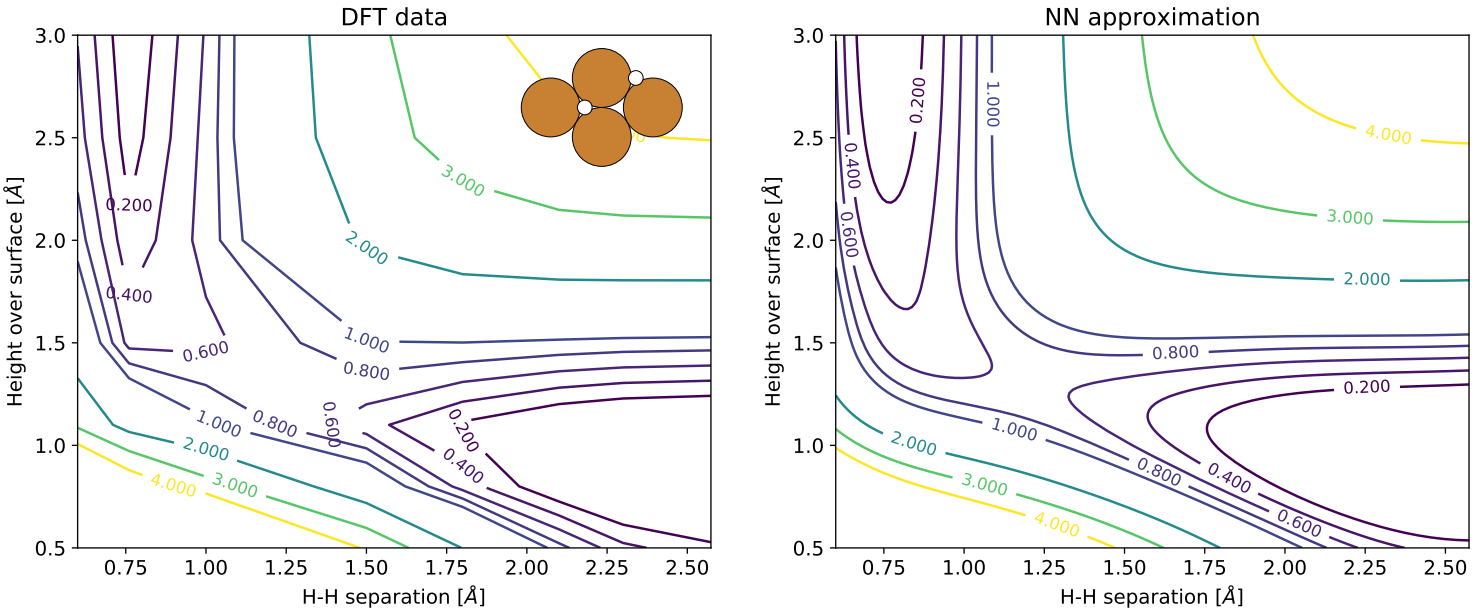
Figure	E_B [eV]	r_B [Å]	Z_B [Å]
5.1b	0.548	0.926	1.36
5.2b	0.633	1.20	1.31
5.3b	0.627	1.18	1.31
5.4b	0.788	1.34	1.43

Table 5.1: Barrier heights E_B as well as the location in the listed figures. r_B is the hydrogen-hydrogen separation at the barrier and Z_B the hydrogen distance from the surface.



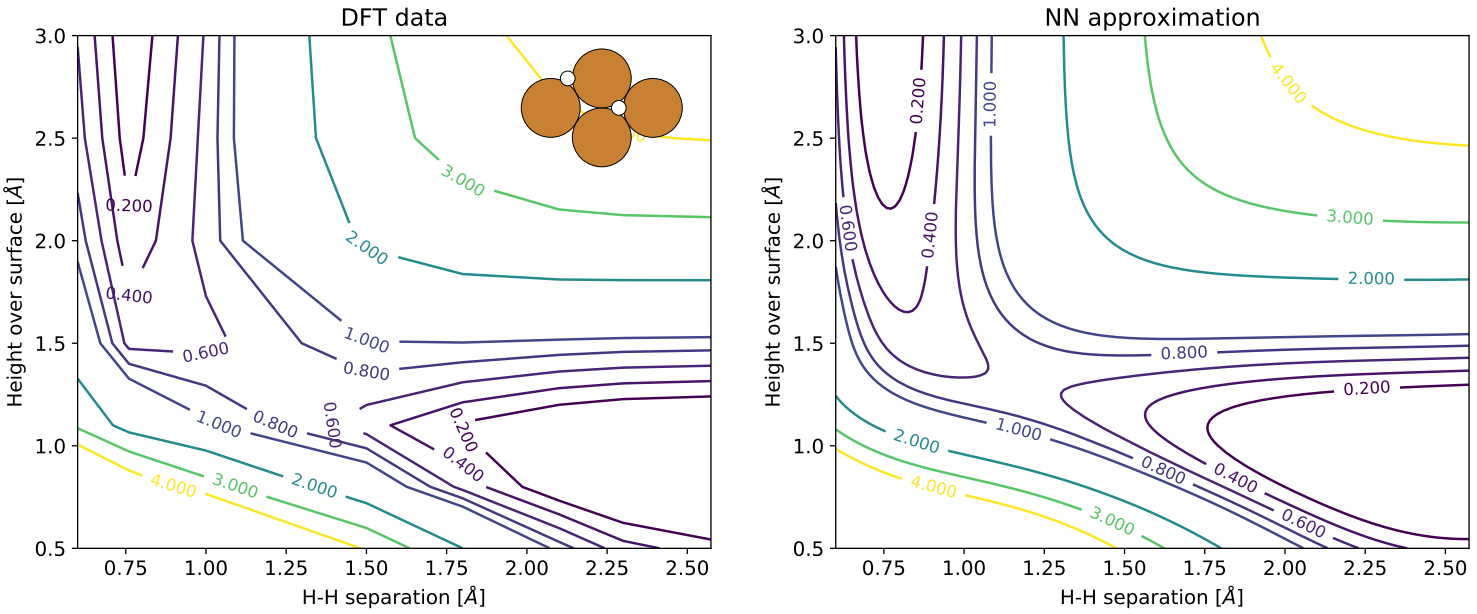
(a) **(b)**

Figure 5.1: 2D cut of the potential energy surface for hydrogen atoms on a line connecting two adjacent hollow sites. DFT data **(a)** and neural network approximation **(b)**. The inset is an illustration of the configuration on the surface.



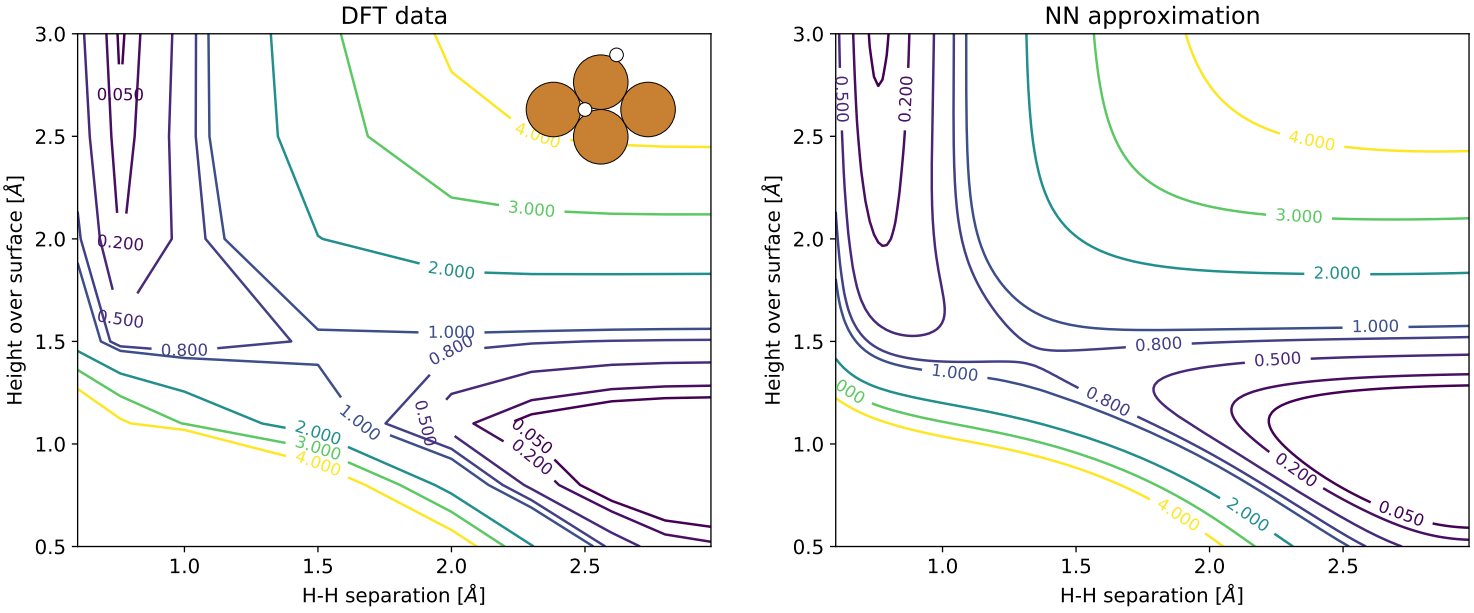
(a) **(b)**

Figure 5.2: 2D cut of the potential energy surface for hydrogen atoms on a line connecting two adjacent fcc sites. DFT data **(a)** and neural network approximation **(b)**. The inset is an illustration of the configuration on the surface.



(a) **(b)**

Figure 5.3: 2D cut of the potential energy surface for hydrogen atoms on a line connecting two adjacent hcp sites. DFT data **(a)** and neural network approximation **(b)**. The inset is an illustration of the configuration on the surface.



(a) **(b)**

Figure 5.4: 2D cut of the potential energy surface for hydrogen atoms on a line connecting two hollow sites on opposite sides of an ontop site. DFT data **(a)** and neural network approximation **(b)**. The inset is an illustration of the configuration on the surface.

ground-state vibration, as predicted by the NNPES, of 0.273 eV with randomized vibrational phase. In both cases the molecule was initially nonrotating.

The lowest vibrational energy (i.e. zero-point energy), equilibrium bond length, and vibrational phase randomization was evaluated by fixing the molecule at 5 Å distance from the surface, with its centre-of-mass in the computational cell's origin and bond axis oriented along the cell's x-axis. The final impact point on the surface slab was chosen randomly. 5000 trajectories were generated per value of translational kinetic energy, which was chosen to be between 0.2 and 2.0 eV in steps of 0.05 eV.

The classical as well as one of the quasiclassical curves were generated by assuming that dissociation occurs once the hydrogen molecule gets over the energy barriers in Figures 5.1b, 5.2b, 5.3b, and 5.4b, or more precisely if any of the following occurs simultaneously:

$$r_{12} > 1.0 \quad \text{Å}, E_{\text{pot}} < 0.55 \text{ eV}, Z < 2.5 \text{ Å} \tag{5.1}$$

$$r_{12} > 1.2 \quad \text{Å}, E_{\text{pot}} < 0.7 \text{ eV}, Z < 2.5 \text{ Å} \tag{5.2}$$

$$r_{12} > 1.35 \text{ Å}, E_{\text{pot}} < 0.8 \text{ eV}, Z < 2.5 \text{ Å} \tag{5.3}$$

The second criterion captures dissociation in both Figures 5.2b and 5.3b, as they are very similar. The sticking probabilities under these conditions are plotted in Figure 5.5, along with experimental and theoretical results. One additional quasiclassical curve was generated under the assumption that dissociation occurs if the

hydrogen-hydrogen distance exceeds 2 Å while $Z < 2.5$ Å.

The experimental results are from work by Rettner *et al* [7], who performed desorption experiments and from this deduced relative adsorption probabilities. They then fit this data to the function

$$s_0 = A(\nu, J)\frac{1 + \mathrm{erf}\left(\frac{E_n - E_0(\nu,J)}{W(\nu,J)}\right)}{2}, \tag{5.4}$$

where s_0 is the sticking probability, $A(\nu, J)$ a scaling factor equal to the value s_0 eventually settles to, E_n the normal energy, i.e. the translational kinetic energy of the molecule in the direction of the surface normal, $E_0(\nu, J)$ the kinetic energy required for s_0 to reach half of $A(\nu, J)$, and $W(\nu, J)$ a width parameter. The values they found for these parameters are $E_0(0, 0) = 0.592$ and $W(0, 0) = 0.172$. As the desorption experiment gives no information on $A(\nu, J)$, it has been set to 1.

The quantum dynamical curve has been adapted from an article by Liu *et al* [39], who took a potential energy surface constructed by Dai and Zhang [40] partly based on DFT calculations by Hammer *et al* [41] and performed quantum dynamics simulations using the time-dependent wave packet method to deduce sticking probabilities at different kinetic energies.

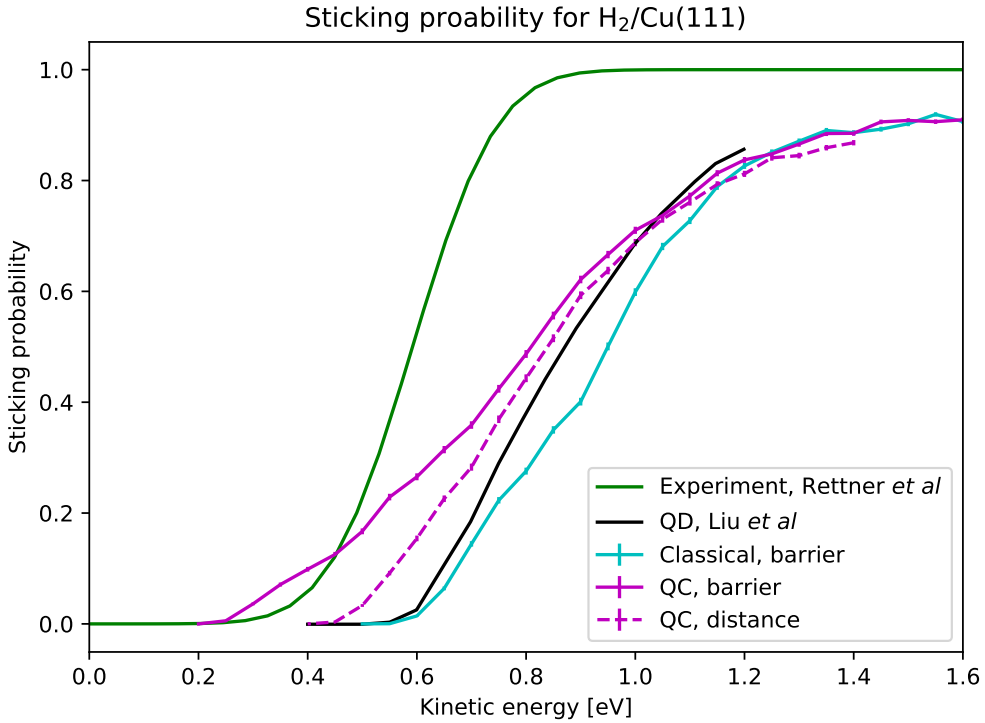


Figure 5.5: Sticking probabilities at different incident kinetic energies. Experimental curve is adapted from work by Rettner *et al* [7], and the quantum dynamical (QD) results are adapted from work by Liu *et al* [39]. Quasiclassical (QC) and classical results using the criterion based on energy barriers, as well as quasiclassical results using a criterion based only on distance, are also shown. The error bars on the NNPES curves represent the standard errors.

6
Discussion and conclusion

We have gone through a rather arduous optimization procedure to end up with the final NNPES. This does not mean that the end result is optimal, however: the inherent stochasticity in the training process means that repeating the process using the same parameters likely results in different results each time. The point was to simply find parameters that appear to work well, and then use these many times to find something that works exceptionally.

The mismatch in the dataset, i.e. the fact that only around one third of the points in the entire dataset were calculated using a relaxed surface slab while the rest were calculated using an unrelaxed surface slab, introduces some errors into the PES. The NNPES nevertheless accurately reproduces the underlying DFT data in slices taken between different hollow sites. The barrier height for dissociation over a bridge to two adjacent hollow sites is also within 10 % of the corresponding barrier height for unstrained and compressively strained surface slabs calculated using the PW91 functional in a study by Sakong and Groß [42], which indicates that the NNPES description is reasonable. It should be noted that the RMSE of the final model is an order of magnitude larger than what Jiang and Guo presented using the same method for a similar system [34]; the reason for this discrepancy is likely the dataset mismatch.

Figure 6.1a shows a 2D slice of a PES for hydrogen dissociation on a Cu(111) surface over the bridge site into neighboring hollow sites. The figure is taken from work by Hammer *et al* [41], who used the PW91 functional to investigate this reaction. Figure 6.1b shows the NNPES approximation over the same area. The figures are rather similar; the only obvious differences are the energy barrier heights, which is about 0.18 eV higher for Hammer *et al*'s PES, and the hydrogen-hydrogen distance at which the barrier is located, which is roughly 0.1 Å lower for the NNPES. The authors perform a convergence test on the barrier heights, however, and find that the barrier height should in fact be somewhere around 0.48-0.54 eV, which is in line with what was found in this work. The article by Sakong and Groß mentioned above gives both barrier heights and hydrogen separations that differ from the ones in this work by no more than 10 % and 6 % respectively, for both unstrained and compressively strained slabs.

The NNPES obtained with the PBE functional in this work is clearly similar to those generated using PW91. This is reasonable, since PBE was designed to essentially be a simplified version of PW91 [16] and the two have been shown to give

very similar predictions of some properties of atomic systems [43], but neither is tailored specifically towards describing dissociative reactions over surfaces. Other XC functional choices that could have been considered are RPBE [44], which adapts the κ parameter of the PBE exchange functional to better predict chemisorption energies for certain molecule-surface interactions, or SRP48 [45], which is tailored specifically to this particular reaction.

The computational parameters used in the DFT calculations are not necessarily entirely converged with respect to the bulk energy of the system; a decision was made to keep them at the chosen values to reduce computational costs as a relatively large set of DFT points were needed. Allowing it to converge further would likely have the effect of slightly adjusting the barrier heights and locations; Hammer *et al*'s work [41] and the work of Sakong and Groß reflects this [42] as the PES in the latter work, where the DFT calculations were more rigorous, differs from the former in barrier height and location.

The experimental curve of Figure 5.5, taken from work by Rettner *et al* [7], contains no information about what the sticking probability eventually settles to as it was taken from a desorption experiment in which a Cu(111) surface held at around 925 K from which hydrogen molecules desorb after which their time-of-flight was measured quantum state-specifically. Of note is that this is essentially the *opposite* reaction to the one of interest to us, but the principle of detailed balance allows the two to be related [27, p. 194]. The important measure here is the kinetic energy at which half of the maximum adsorption probability occurs; about 0.59 eV as predicted by their experiment, and about $0.8 - 0.9$ eV in the quasidynamical simulations performed in this work. We thus fail to capture the behaviour of the experiments quantitatively.

In the same article, Rettner *et al* published the results of molecular beam experiments done to measure the adsorption probabilities of H_2 on a Cu(111) surface whose temperature was held at 120 K. They found a sticking probability of just under 0.1 at normal[1] kinetic energies of 0.5 eV; the quasiclassical simulations with a dissociation criterion based on energy barriers in this work predicts a sticking probability of around 0.167 at that kinetic energy. These values are not necessarily directly comparable, however, as those experimental results contain contributions from higher vibrational and rotational states. They also fitted the adsorption data to the form of Equation 5.4 and find that values of $A(\nu, J)$ between 0.15 and 0.5 produce equally good fits; it is thus unclear to what value the sticking probability should converge towards.

Smeets *et al* [8] compared the sticking probabilities for D_2 on Cu(111), a system very similar to the one under study in this work, predicted via quasi-classical trajectory simulations using several different exchange-correlation functionals with those from molecular beam experiments; they showed that PBE severely overestimates the sticking probability in collision energy ranges between 45 and 80 kJ/mol (≈0.47-0.83 eV). The authors also performed quasidynamical trajectory simulations to test an

[1] Corresponding to movement normal to the surface.

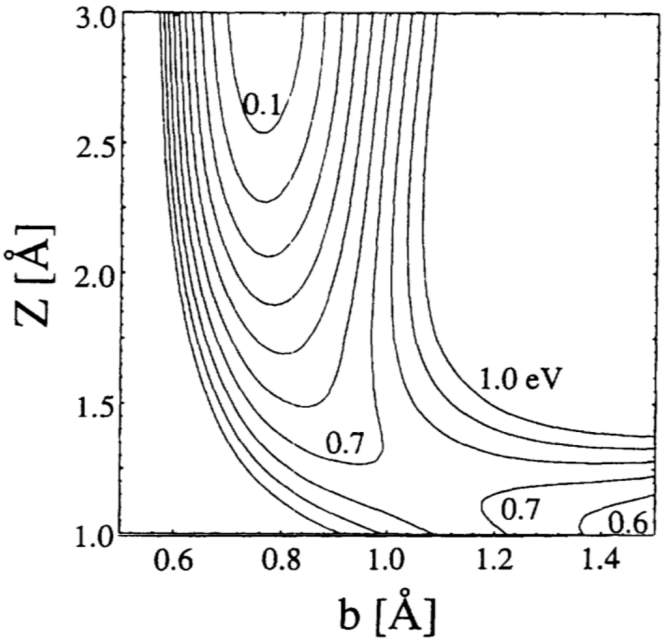
approach to exchange-correlation functional construction and found that their results for H_2 dissociation are in excellent agreement with adsorption experiments of Rettner *et al* [7]; they worked within the frozen surface approximation, but applied a different sticking criterion than what was used in this work. It is nevertheless clearly *possible* to recreate experimental results computationally without explicitly taking the motion of the lattice into account at lower incident kinetic energies, although the PBE functional appears to be insufficient for that purpose.

Of note is that the experimental results Smeets *et al* were able to model accurately were performed at a temperature where neglecting the motion of the lattice is reasonable; at the temperatures the experimental curve of Figure 5.5 was generated at this does not hold true, and the frozen surface approximation fails [46].

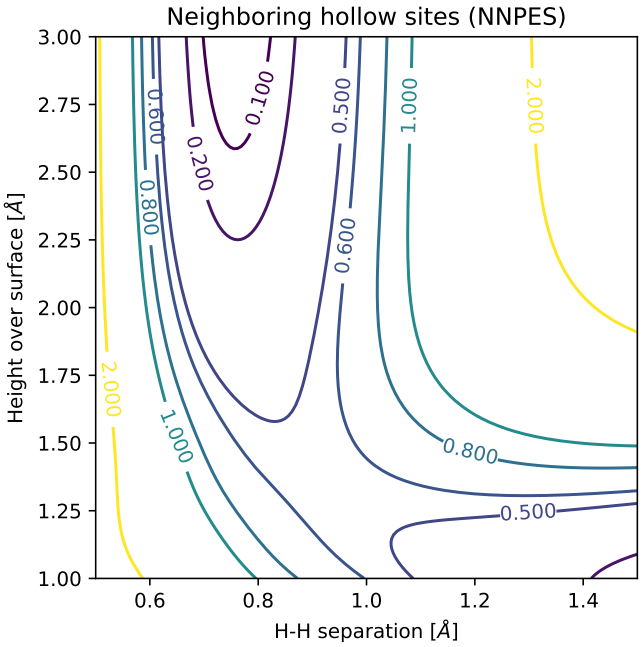
The sticking probability curve in Figure 5.5, taken from Liu *et al* [39], corresponds to a quantum dynamical simulation done with the time-dependent wave packet method. The PES used by them was constructed by Dai and Light [40], who used the DFT data from the work by Hammer *et al* [41]. This means that the dissociation barrier heights are slightly higher than the NNPES used in this work, which affects the sticking probabilities. While the quantum dynamical method used in their simulations explicitly takes the quantum states of the particles into account, it is nevertheless possible to achieve very similar results using quasiclassical trajectory methods [47]; comparing the results of Liu *et al* to what is found in this work is thus not unreasonable.

For the quasiclassical sticking simulations with barrier-based criteria sticking starts to occur at around incident kinetic energies of 0.3 eV; this is in line with what one would intuitively expect, as this combined with the vibrational energy of 0.273 eV should be just enough to get over the energy barrier in Figure 5.1b. In the classical simulations sticking starts occuring at around 0.55 eV, which is reasonable by the same argument. The quantum dynamical curve predicts that sticking starts at around 0.55 eV, which agrees very well with our classical curve but disagrees with the barrier-based quasiclassical one. The quasiclassical curve with distance-based criterion is in better agreement with the quantum dynamical one, and the remaining discrepancies are likely explained by differences in the underlying PES.

The dissociation criteria based on the energy barriers is not necessarily realistic; they completely ignore the possibility of recombination once the hydrogen molecule has passed the barrier, and do not necessarily capture all possible entry channels for dissociation. Figure 6.2 shows a two-dimensional cut of the PES for dissociation over an ontop site into two bridge sites on opposite ends; it is rather similar to the energy landscape in 5.4b, but with an energy barrier at slightly larger H-H separation. This energy landscape is not properly taken into account by the chosen barrier-based criteria of Equation 5.3. Basing the criterion solely on the hydrogen-hydrogen distance may be a better choice, and is in fact what Smeets *et al* did, though even that is a bit arbitrary as the specific cutoff value will influence the shape of the curve.



(a) Two-dimensional slice of PES for H_2 dissociation over the bridge site of a Cu(111) surface; Z is the distance from the copper surface, and b the hydrogen-hydrogen separation. Adapted with permission from [41]. Copyright 1994 by The American Physical Society.



(b) Two-dimensional slice over the same site as in Figure 6.1a, generated with the NNPES.

Figure 6.1: Comparison between two-dimensional slices for dissociation over a bridge site. a) from work by Hammer *et al* [41], b) using the NNPES.

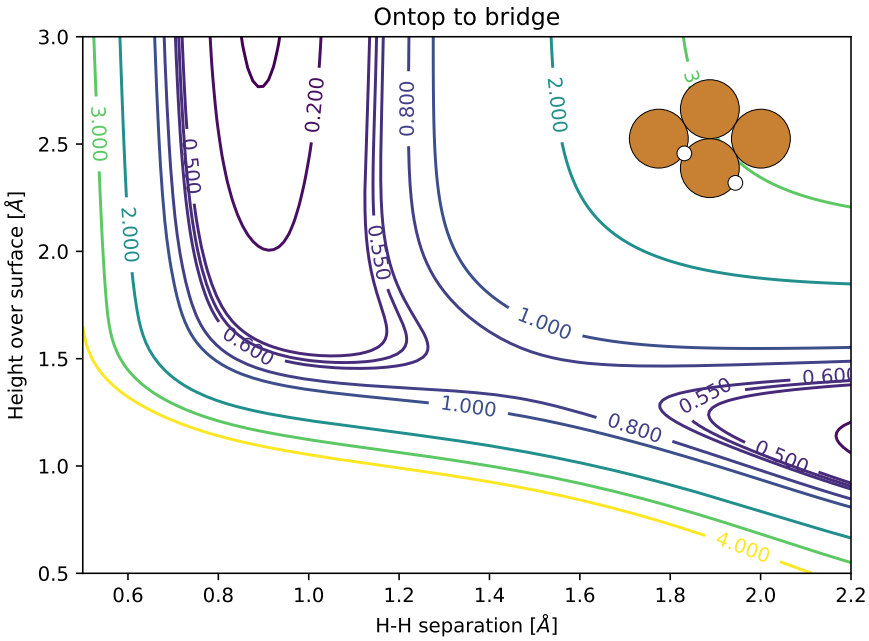


Figure 6.2: PES for dissociation with hydrogen center-of-mass above an ontop site, with hydrogen atoms angled towards bridge sites on its opposite sides.

Conclusion

The final NNPES representation accurately reproduces the features of the underlying PES between adsorption sites on the Cu(111) surface, despite the fact that one third of the dataset was geometrically mismatched with the rest. The geometry of the PES agrees well with previous results, although the PES used to compare with overestimates the barrier to dissociation over the bridge site. The corresponding NNPES barrier height is nevertheless within 10 % of the predicted barrier height for that site.

While quasidynamical trajectory methods are able to predict experimental results rather accurately, we have failed to do so in this work. The likely explanation, as discussed above, is a mix of the failure of the frozen surface approximation at the surface temperature of the experiment, the exchange-correlation functional not being entirely sufficient for the context and the simplistic way dissociation was determined. We have managed to capture the behaviour of the sticking probability as kinetic energy increases qualitatively, however.

Bibliography

[1] J. R. Kitchin. Machine learning in catalysis. *Nature Catalysis*, 1(4):230–232, 2018.

[2] P. Covington, J. Adams, and E. Sargin. Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, New York, NY, USA, 15–19 Sep 2016.

[3] M. Bojarski, et al. End to End Learning for Self-Driving Cars. *arXiv preprint arXiv:1604.07316*, 2016.

[4] F. Brockherde, et al. Bypassing the Kohn-Sham equations with machine learning. *Nature Communications*, 8(1):872, 2017.

[5] S. Haykin. *Neural Networks and Learning Machines*. Pearson, London, third edition, 2008.

[6] G. R. Darling and S. Holloway. H_2 dissociation dynamics on metals: where do we stand? In *Surface Dynamics*, volume 11 of *The Chemical Physics of Solid Surfaces*, pages 27 – 49. Elsevier, 2003.

[7] C. T. Rettner, H. A. Michelsen, and D. J. Auerbach. Quantum-state-specific dynamics of the dissociative adsorption and associative desorption of H_2 at a Cu(111) surface. *The Journal of Chemical Physics*, 102(11):4625–4641, 1995.

[8] E. W. F. Smeets, J. Voss, and G.J. Kroes. Specific Reaction Parameter Density Functional Based on the Meta-Generalized Gradient Approximation: Application to H_2 + Cu(111) and H_2 + Ag(111). *The Journal of Physical Chemistry A*, 123(25):5395–5406, 2019.

[9] F. Giustano. *Materials Modelling using Density Functional Theory*. Oxford University Press, Oxford, first edition, 2014.

[10] W. Koch and M. C. Holthausen. *A Chemist's Guide to Density Functional Theory*. Wiley-VCH, Weinheim, second edition, 2001.

[11] P. Hohenberg and W. Kohn. Inhomogeneous Electron Gas. *Phys. Rev.*, 136:B864–B871, 1964.

[12] D. Scholl and J. Steckel. *Density Functional Theory: A Practical Introduction*. Wiley, Hoboken, New Jersey, first edition, 2009.

[13] W. Kohn and L. J. Sham. Self-Consistent Equations Including Exchange and Correlation Effects. *Phys. Rev.*, 140:A1133–A1138, 1965.

[14] J. P. Perdew, et al. Prescription for the design and selection of density functional approximations: More constraint satisfaction with fewer fits. *The Journal of Chemical Physics*, 123(6):062201, 2005.

[15] R. Martin. *Electronic Structure: Basic Theory and Practical Methods*. Cambridge University Press, Cambridge, first edition, 2004.

[16] J. P. Perdew, K. Burke, and M. Ernzerhof. Generalized Gradient Approximation Made Simple. *Phys. Rev. Lett.*, 77:3865–3868, 1996.

[17] J. P. Perdew and S. Kurth. Density functionals for non-relativistic coulomb systems in the new century. In C. Fiolhais, F. Nogueira, and M. A. L. Marques, editors, *A Primer in Density Functional Theory*, pages 1–55. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[18] P. E. Blöchl. Projector augmented-wave method. *Phys. Rev. B*, 50:17953–17979, 1994.

[19] P. E. Blöchl, C. J. Först, and J. Schimpl. Projector augmented wave method: ab initio molecular dynamics with full wave functions. *Bulletin of Materials Science*, 26(1):33–41, 2003.

[20] J. Thijssen. *Computational Physics*. Cambridge University Press, Cambridge, second edition, 2007.

[21] C. Díaz and F. Martín. Using Molecular Reflectivity to Explore Reaction Dynamics at Metal Surfaces. In R. Díez Muiño and H. F. Busnengo, editors, *Dynamics of Gas-Surface Interactions: Atomic-level Understanding of Scattering Processes at Surfaces*, pages 75–100. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[22] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747v2*, 2017.

[23] J. Behler. Constructing high-dimensional neural network potentials: A tutorial review. *International Journal of Quantum Chemistry*, 115(16):1032–1050, 2015.

[24] G. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[25] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[26] I. Chorkendorff and J. W. Niemantsverdriet. *Concepts of Modern Catalysis and Kinetics*. Wiley-VCH, Weinheim, third edition, 2017.

[27] H. A. Michelsen, C. T. Rettner, and D. J. Auerbach. The Adsorption of Hydrogen at Copper Surfaces: A Model System for the Study of Activated Adsorption. In R. J. Madix, editor, *Surface Reactions*, pages 185–237. Springer Berlin Heidelberg, Berlin, Heidelberg, 1994.

[28] J. J. Mortensen, L. B. Hansen, and K. W. Jacobsen. Real-space grid implementation of the projector augmented wave method. *Phys. Rev. B*, 71:035109, 2005.

[29] J. Enkovaara, et al. Electronic structure calculations with GPAW: A real-space implementation of the projector augmented-wave method. *Journal of Physics: Condensed matter*, 22(25):253202, 2010.

[30] A. Larsen, et al. The Atomic Simulation Environment — A Python library for working with atoms. *Journal of Physics: Condensed Matter*, 29(27):273002, 2017.

[31] M. Abadi, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[32] F. Chollet et al. Keras. `https://keras.io`, 2015.

[33] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh and M. Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[34] B. Jiang and H. Guo. Permutation invariant polynomial neural network approach to fitting potential energy surfaces. III. Molecule-surface interactions. *The Journal of Chemical Physics*, 141(3):034109, 2014.

[35] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice-Hall, Upper Saddle River, New Jersey, second edition, 1999.

[36] N. E. Henriksen and F. Y. Hansen. *Theories of Molecular Reaction Dynamics*. Oxford University Press, Oxford, first edition, 2008.

[37] D. P. Kroese, T. Taimre, and Z. I. Botev. *Handbook of Monte Carlo Methods*. Wiley, Hoboken, New Jersey, first edition, 2011.

[38] K. K. Irikura. Experimental Vibrational Zero-Point Energies: Diatomic Molecules. *Journal of Physical and Chemical Reference Data*, 36(2):389–397, 2007.

[39] T. Liu, B. Fu, and D. H. Zhang. Validity of the site-averaging approximation for modeling the dissociative chemisorption of H_2 on Cu(111) surface: A quantum dynamics study on two potential energy surfaces. *The Journal of Chemical Physics*, 141(19):194302, 2014.

[40] J. Dai and J. Z. H. Zhang. Quantum adsorption dynamics of a diatomic molecule on surface: Four-dimensional fixed-site model for H_2 on Cu(111). *The Journal of Chemical Physics*, 102(15):6280–6289, 1995.

[41] B. Hammer, M. Scheffler, K. W. Jacobsen, and J. K. Nørskov. Multidimensional Potential Energy Surface for H_2 dissociation over Cu(111). *Phys. Rev. Lett.*, 73:1400–1403, 1994.

[42] S. Sakong and A. Groß. Dissociative adsorption of hydrogen on strained Cu surfaces. *Surface Science*, 525(1):107 – 118, 2003.

[43] B.T. Teng, X.D. Wen, M. Fan, F.M. Wu, and Y. Zhang. Choosing a proper exchange–correlation functional for the computational catalysis on surface. *Phys. Chem. Chem. Phys.*, 16:18563–18569, 2014.

[44] B. Hammer, L. B. Hansen, and J. K. Nørskov. Improved adsorption energetics within density-functional theory using revised Perdew-Burke-Ernzerhof functionals. *Phys. Rev. B*, 59:7413–7421, 1999.

[45] F. Nattino, C. Díaz, B. Jackson, and G.J. Kroes. Effect of Surface Motion on the Rotational Quadrupole Alignment Parameter of D_2 Reacting on Cu(111). *Phys. Rev. Lett.*, 108:236104, 2012.

[46] M. Bonfanti, C. Díaz, M. F. Somers, and G.J. Kroes. Hydrogen dissociation on Cu(111): the influence of lattice motion. part I. *Phys. Chem. Chem. Phys.*, 13:4552–4561, 2011.

[47] A. D. Kinnersley, G. R. Darling, S. Holloway, and B. Hammer. A comparison of quantum and classical dynamics of H_2 dissociation on Cu(111). *Surface Science*, 364(3):219–234, 1996.

[48] J. P. Perdew and Y. Wang. Accurate and simple analytic representation of the electron-gas correlation energy. *Phys. Rev. B*, 45:13244–13249, 1992.

A

Perdew-Burke-Ernzerhof enhancement factor

The Perdew-Burke-Ernzerhof (PBE) functional treats exchange and correlation energies by modifying the corresponding LSDA energies [16] from which the enhancement factor over local exchange, F_{XC}, may be derived after generalizing to arbitrary spin polarization.

The PBE exchange energy is

$$E_{\mathrm{X}}^{\mathrm{PBE}}(s) = \int d\mathbf{r} n \varepsilon_{\mathrm{X}}^{\mathrm{hom}}(r_s) F_{\mathrm{X}}(s), \tag{A.1}$$

where $\varepsilon_{\mathrm{X}}^{\mathrm{hom}}(r_s) = -\frac{3}{4\pi}\frac{(9\pi/4)^{1/3}}{r_s}$ is the exchange energy per electron of a spin-unpolarized homogeneous electron gas, $r_s = \left(\frac{3}{4\pi n}\right)^{1/3}$ the local Seitz radius, and $F_{\mathrm{X}}(s)$ the exchange enhancement factor defined as

$$F_{\mathrm{X}} = 1 + \kappa - \frac{\kappa}{1 + \mu s^2/\kappa}, \tag{A.2}$$

where $s = \frac{|\nabla n|}{2k_F n}$ is a reduced density gradient describing how the density varies on the scale of the local Fermi wavelength $2\pi/k_F = 2\pi/(3\pi^2 n)^{1/3}$ [17, p. 36]; $\kappa = 0.804$ and $\mu = 0.21951$ are constants.

The PBE correlation energy is[1]

$$E_{\mathrm{C}}(\zeta, t) = \int d\mathbf{r} n \left[\varepsilon_{\mathrm{C}}^{\mathrm{hom}}(r_s, \zeta) + H(r_s, \zeta, t)\right], \tag{A.3}$$

where $\varepsilon_{\mathrm{C}}^{\mathrm{hom}}$ is the correlation energy per electron of a uniform electron gas, $\zeta = (n_\uparrow - n_\downarrow)/n$ relative spin-polarization, and H a function defined by

$$H(r_s, \zeta, t) = \gamma\phi^3 \ln\left(1 + \frac{\beta}{\gamma} t^2 \left[\frac{1 + At^2}{1 + At^2 + A^2 t^4}\right]\right), \tag{A.4}$$

with

$$A = \frac{\beta}{\gamma}\left(\exp\left(-\varepsilon_{\mathrm{C}}^{\mathrm{hom}}(r_s, \zeta)/\gamma\phi^3\right) - 1\right). \tag{A.5}$$

[1] In the original paper by Perdew, Burke and Ernzerhof on the functional H has an additional multiplicative factor, which reflects a different scaling of the energies. This has been excluded here.

$t = \frac{|\nabla n|}{2k_s n}$ is another reduced density gradient similar to s, but describing the gradient on the scale of the Thomas-Fermi screening length $1/k_s = \left(\frac{\pi}{4}\right)^{1/2}\left(\frac{4}{9\pi}\right)^{1/6} r_s^{1/2}$, and $\phi = \frac{1}{2}\left((1+\zeta)^{2/3} + (1-\zeta)^{2/3}\right)$; $\beta = 0.066725$ is a constant, while γ is a weak function of ζ for which the $\zeta = 0$ value $\gamma = 0.031091$ is used.

Both exchange and correlation effects may be included in a single expression by defining the *enhancement factor over local exchange* F_{XC},

$$E_{\mathrm{XC}} = \int d\mathbf{r}\varepsilon_{\mathrm{X}}^{\mathrm{hom}}(r_s)F_{\mathrm{XC}}(r_s,\zeta,s), \tag{A.6}$$

where $F_{\mathrm{XC}} = F_{\mathrm{X}} + F_{\mathrm{C}}$. $F_{\mathrm{X}}(\zeta, s)$ is simply the PBE exchange enhancement factor generalized to arbitrary spin polarization [17, p. 46],

$$F_{\mathrm{X}}(\zeta,s) = \frac{1}{2}(1+\zeta)^{4/3}F_{\mathrm{X}}\left(\frac{s}{(1+\zeta)^{1/3}}\right) + \frac{1}{2}(1-\zeta)^{4/3}F_{\mathrm{X}}\left(\frac{s}{(1-\zeta)^{1/3}}\right). \tag{A.7}$$

F_C may be derived by writing the correlation energy on the form of Equation A.6,

$$E_{\mathrm{C}}(\zeta,t) = \int d\mathbf{r}n\varepsilon_{\mathrm{X}}^{\mathrm{hom}}(r_s)\left(\frac{\varepsilon_{\mathrm{C}}^{\mathrm{hom}}(r_s,\zeta) + H(r_s,\zeta,t)}{\varepsilon_{\mathrm{X}}^{\mathrm{hom}}(r_s)}\right), \tag{A.8}$$

from which it follows that

$$F_C(r_s,\zeta,s) = \frac{\varepsilon_{\mathrm{C}}^{\mathrm{hom}}(r_s,\zeta) + H(r_s,\zeta,t(s))}{\varepsilon_{\mathrm{X}}^{\mathrm{hom}}(r_s)}, \tag{A.9}$$

where one dimensionless gradient has been expressed in terms of the other via $t(s) = \left(\frac{\pi}{4}\right)^{1/2}\left(\frac{4}{9\pi}\right)^{1/6}\frac{s}{r_s^{1/2}}$.

The correlation energy per electron, $\varepsilon_{\mathrm{C}}^{\mathrm{hom}}(r_s,\zeta)$, can be accurately approximated for $\zeta = 0$ and $\zeta = 1$ using the expression [48]

$$\chi(r_s) = -2c_0(1+\alpha r_s)\ln\left(\frac{1}{1+2c_0(\beta_1 r_s^{1/2} + \beta_2 r_s + \beta_3 r_s^{3/2} + \beta_4 r_s^2}\right), \tag{A.10}$$

using different constants c_0, α, β_1, β_2, β_3, and β_4. To find $\varepsilon_{\mathrm{C}}^{\mathrm{hom}}$ for arbitrary spin polarization an interpolation procedure is used,

$$\varepsilon_{\mathrm{C}}^{\mathrm{hom}}(r_s,\zeta) = \varepsilon_{\mathrm{C}}^{\mathrm{hom}}(r_s,0) + \alpha_{\mathrm{C}}(r_s)\frac{f(\zeta)}{f''(0)}(1-\zeta^4) + \left(\varepsilon_{\mathrm{C}}^{\mathrm{hom}}(r_s,1) - \varepsilon_{\mathrm{C}}^{\mathrm{hom}}(r_s,0)\right)f(\zeta)\zeta^4. \tag{A.11}$$

α_{C} is the correlation contribution to spin stiffness; $-\alpha_{\mathrm{C}}$ may be approximated using Equation A.10 with yet another set of constants [48]. $f(\zeta)$ is defined by

$$f(\zeta) = \frac{(1+\zeta)^{4/3} + (1-\zeta)^{4/3} - 2}{2^{4/3} - 2}, \tag{A.12}$$

and $f''(0) = 1.709921$.

B

Derivation of forces from neural network PES

From Equation 2.17 it is obvious that the force acting upon hydrogen atom number n in the direction of coordinate α is

$$F_{\alpha_n} = -\frac{\partial E}{\partial \alpha_n}, \tag{B.1}$$

where E is the system's potential energy surface and α_n the α-coordinate of the nth hydrogen atom. Since the neural networks in this work are approximations of E it holds that

$$F_{\alpha_n} \approx -\frac{\partial \mathcal{O}}{\partial \alpha_n}, \tag{B.2}$$

where $\mathcal{O}$ is the output of such a neural network.

The derivation below assumes two hidden layers, denoted by indices H_1 and H_2. Input and output neurons are denoted $\mathcal{I}$ and $\mathcal{O}$ respectively, while the corresponding layers are denoted by indices I and O. Connections between two layers are denoted by W_{mn}^{AB}, where W the weight of the connection between neuron m of layer A and neuron n of layer B. Recall that the output layer consists only of one neuron and uses the identity function as its activation function, while the hidden layers use the hyperbolic tangent function.

Using the chain rule the right-hand side of Equation B.2 may be evaluated as

$$\begin{aligned}
\frac{\partial \mathcal{O}}{\partial \alpha_n} &= \frac{\partial}{\partial \alpha_n}\left(\sum_i W_i^{H_2 O} \chi_i^{H_2} + b^O\right) \\
&= \frac{\partial}{\alpha_n}\left(\sum_i W_i^{H_2 O} \frac{\partial}{\partial \alpha_n} \chi_i^{H_2}\right) \\
&= \sum_i W_i^{H_2 O} \frac{\partial}{\partial \alpha_n} \chi_i^{H_2}.
\end{aligned} \tag{B.3}$$

The process is repeated for $\chi_i^{H_2}$,

$$\begin{aligned}\frac{\partial}{\partial\alpha_n}\chi_i^{H_2} &= \frac{\partial}{\partial\alpha_n}\tanh\left(\sum_j W_{ji}^{H_1H_2}\chi_j^{H_1} + b_i^{H_2}\right)\\ &= \left(1-\tanh^2\left(\sum_j W_{ji}^{H_1H_2}\chi_j^{H_1} + b_i^{H_2}\right)\right)\frac{\partial}{\partial\alpha_n}\left(\sum_j W_{ji}^{H_1H_2}\chi_j^{H_1}\right)\\ &= \left(1-\left(\chi_i^{H_2}\right)^2\right)\left(\sum_j W_{ji}^{H_1H_2}\frac{\partial}{\partial\alpha_n}\chi_j^{H_1}\right),\end{aligned} \tag{B.4}$$

and $\chi_j^{H_1}$,

$$\begin{aligned}\frac{\partial}{\partial\alpha_n}\chi_j^{H_1} &= \frac{\partial}{\partial\alpha_n}\tanh\left(\sum_k W_{kj}^{IH_1}I_k + b_j^{H_1}\right)\\ &= \left(1-\tanh^2\left(\sum_k W_{kj}^{IH_1}I_k + b_j^{H_1}\right)\right)\frac{\partial}{\partial\alpha_n}\left(\sum_k W_{kj}^{IH_1}\mathcal{I}_k\right)\\ &= \left(1-\left(\chi_j^{H_1}\right)^2\right)\left(\sum_k W_{kj}^{IH_1}\frac{\partial}{\partial\alpha_n}\mathcal{I}_k\right).\end{aligned} \tag{B.5}$$

Assembling these expressions yields the neural network approximation:

$$F_{\alpha_n} \approx -\sum_i W_i^{H_2O}\left(1-\left(\chi_i^{H_2}\right)\right)\cdot\left(\sum_j W_{ji}^{H_1H_2}\left(\left(1-\left(\chi_j^{H_1}\right)^2\right)\left(\sum_k W_{kj}^{IH_1}\frac{\partial}{\partial\alpha_n}\mathcal{I}_k\right)\right)\right). \tag{B.6}$$

Only the derivatives with respect to the thirteen input functions $\mathcal{I} = G_1, \ldots, G_{13}$ remain unevaluated. They are repeated here for clarity:

$$G_1 = p_1(x_1, y_1) + p_1(x_2, y_2), \tag{B.7a}$$
$$G_2 = p_1(x_1, y_1)p_1(x_2, y_2), \tag{B.7b}$$
$$G_3 = p_2(x_1, y_1) + p_2(x_2, y_2), \tag{B.7c}$$
$$G_4 = p_2(x_1, y_1)p_2(x_2, y_2), \tag{B.7d}$$
$$G_5 = p_z(z_1) + p_z(z_2), \tag{B.7e}$$
$$G_6 = p_z(z_1)p_z(z_2), \tag{B.7f}$$
$$G_7 = p_r(r), \tag{B.7g}$$
$$G_8 = p_1(X, Y), \tag{B.7h}$$
$$G_9 = p_2(X, Y), \tag{B.7i}$$
$$G_{10} = p_z(Z), \tag{B.7j}$$
$$G_{11} = p_1(x_1, y_1)p_2(x_1, y_1) + p_1(x_2, y_2)p_2(x_2, y_2), \tag{B.7k}$$
$$G_{12} = p_1(x_1, y_1)p_z(z_1) + p_1(x_2, y_2)p_z(z_2), \tag{B.7l}$$
$$G_{13} = p_2(x_1, y_1)p_z(z_1) + p_2(x_2, y_2)p_z(z_2). \tag{B.7m}$$

The derivative with respect to each coordinate of both hydrogen atoms is required. Some derivatives are obvious through inspection; $\frac{\partial}{\partial z_i}G_1 = 0$, for instance. The rest

may be constructed from the derivatives of the primitive functions p_1, p_2, p_z and p_r, where index i denotes the atomic coordinates with respect to which derivation takes place, index j the coordinates of the other atom, and η the x or y coordinate:

$$\frac{\partial}{\partial \eta_i} G_1 = \frac{\partial}{\partial \eta_i} p_1(x_i, y_i); \;\; \frac{\partial}{\partial z_i} G_1 = 0, \tag{B.8a}$$

$$\frac{\partial}{\partial \eta_i} G_2 = p_1(x_j, y_j) \frac{\partial}{\partial \eta_i} p_1(x_i, y_i); \;\; \frac{\partial}{\partial z_i} G_2 = 0, \tag{B.8b}$$

$$\frac{\partial}{\partial \eta_i} G_3 = \frac{\partial}{\partial \eta_i} p_2(x_i, y_i); \;\; \frac{\partial}{\partial z_i} G_3 = 0, \tag{B.8c}$$

$$\frac{\partial}{\partial \eta_i} G_4 = p_2(x_j, y_j) \frac{\partial}{\partial \eta_i} p_2(x_i, y_i); \;\; \frac{\partial}{\partial z_i} G_4 = 0, \tag{B.8d}$$

$$\frac{\partial}{\partial \eta_i} G_5 = 0; \;\; \frac{\partial}{\partial z_i} G_5 = \frac{\partial}{\partial z_i} p_z(z_i), \tag{B.8e}$$

$$\frac{\partial}{\partial \eta_i} G_6 = 0; \;\; \frac{\partial}{\partial z_i} G_6 = p_z(z_j) \frac{\partial}{\partial z_i} p_z(z_i), \tag{B.8f}$$

$$\frac{\partial}{\partial \eta_i} G_7 = \frac{\partial}{\partial \eta_i} p_r(r); \;\; \frac{\partial}{\partial z_i} G_7 = \frac{\partial}{\partial z_i} p_r(r), \tag{B.8g}$$

$$\frac{\partial}{\partial \eta_i} G_8 = \frac{\partial}{\partial \eta_i} p_1\left((x_i + x_j)/2, (y_i + y_j)/2\right); \;\; \frac{\partial}{\partial z_i} G_8 = 0, \tag{B.8h}$$

$$\frac{\partial}{\partial \eta_i} G_9 = \frac{\partial}{\partial \eta_i} p_1\left((x_i + x_j)/2, (y_i + y_j)/2\right); \;\; \frac{\partial}{\partial z_i} G_9 = 0, \tag{B.8i}$$

$$\frac{\partial}{\partial \eta_i} G_{10} = 0; \;\; \frac{\partial}{\partial z_i} G_{10} = \frac{\partial}{\partial z_i} p_z\left((z_i + z_j)/2\right), \tag{B.8j}$$

$$\frac{\partial}{\partial \eta_i} G_{11} = \frac{\partial}{\partial \eta_i} p_1(x_i, y_i) \frac{\partial}{\partial \eta_i} p_2(x_i, y_i) + p_2(x_i, y_i) \frac{\partial}{\partial \eta_i} p_1(x_i, y_i); \;\; \frac{\partial}{\partial z_i} G_{11} = 0, \tag{B.8k}$$

$$\frac{\partial}{\partial \eta_i} G_{12} = p_z(z_i) \frac{\partial}{\partial \eta_i} p_1(x_i, y_i); \;\; \frac{\partial}{\partial z_i} G_{12} = p_1(x_i, y_i) \frac{\partial}{\partial z_i} p_z(z_i), \tag{B.8l}$$

$$\frac{\partial}{\partial \eta_i} G_{13} = p_z(z_i) \frac{\partial}{\partial \eta_i} p_2(x_i, y_i); \;\; \frac{\partial}{\partial z_i} G_{13} = p_2(x_i, y_i), \frac{\partial}{\partial z_i} p_z(z_i). \tag{B.8m}$$

All that remains is to differentiate the primitive functions, which are repeated here for clarity, with λ set to 1:

$$p_1(x, y) = \cos\left(\frac{4\pi y}{a\sqrt{3}}\right) + 2\cos\left(\frac{2\pi x}{a}\right)\cos\left(\frac{2\pi y}{a\sqrt{3}}\right), \tag{B.9a}$$

$$p_2(x, y) = \sin\left(\frac{4\pi y}{a\sqrt{3}}\right) - 2\cos\left(\frac{2\pi x}{a}\right)\sin\left(\frac{2\pi y}{a\sqrt{3}}\right), \tag{B.9b}$$

$$p_z(z) = \exp\left(-z\right), \tag{B.9c}$$

$$p_r(r) = \exp\left(-r\right). \tag{B.9d}$$

Here, a is the distance between two copper atoms on a Cu(111) surface and $r = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$ the distance between the two hydrogen atoms.

The nonzero derivatives are

$$\frac{\partial}{\partial x_i} p_1(x_i, y_i) = -\frac{4\pi}{a} \sin\left(\frac{2\pi x_i}{a}\right) \cos\left(\frac{2\pi y_i}{a\sqrt{3}}\right), \tag{B.10a}$$

$$\frac{\partial}{\partial y_i} p_1(x_i, y_i) = -\frac{4\pi}{a\sqrt{3}} \left(\sin\left(\frac{4\pi y_i}{a\sqrt{3}}\right) + \cos\left(\frac{2\pi x_i}{a}\right) \sin\left(\frac{2\pi y_i}{a\sqrt{3}}\right)\right), \tag{B.10b}$$

$$\frac{\partial}{\partial x_i} p_2(x_i, y_i) = -\frac{4\pi}{a} \sin\left(\frac{2\pi x}{a}\right) \sin\left(\frac{2\pi y}{a\sqrt{3}}\right), \tag{B.10c}$$

$$\frac{\partial}{\partial y_i} p_2(x_i, y_i) = \frac{4\pi}{a\sqrt{3}} \left(\cos\left(\frac{4\pi y_i}{a\sqrt{3}}\right) - \cos\left(\frac{2\pi x_i}{a}\right) \cos\left(\frac{2\pi y_i}{a\sqrt{3}}\right)\right), \tag{B.10d}$$

$$\frac{\partial}{\partial z_i} p_z(z_i) = -\exp\left(-z_i\right), \tag{B.10e}$$

$$\frac{\partial}{\partial \alpha_i} p_r(r) = -\left(\alpha_i - \alpha_j\right) \frac{\exp\left(-r\right)}{r}, \tag{B.10f}$$

where α again denotes any of the hydrogen coordinates.

Any normalization of the dataset needs to be taken into account here as well; for normalization to zero mean and unit variance this essentially redefines each input function to be $(G_i - M_i)/N_i$, where M_i is the mean value of the nonnormalized input G_i and N_i its normalization constant. As such, each derivative of G_i needs to be divided by N_i.

C

NNPES script

```
'''
    Script for calculation of forces and potential energies
    for the H_2/Cu(111) system, implemented as an ASE calculator.

    The script assumes that the atomic system consists of a 2x2
    three-layer slab, with the hydrogen atoms being the 13th and
    14th entries. Generalizing beyond this should be trivial.

    It further assumes that weights and biases have been exported
    into .npy files in the format [w1, b1, w2, b2, ...], where wn
    and bn are the weight and bias arrays for the nth layer of the
    neural network. It is also assumed that all networks to be
    included in the calculator have two hidden layers, and that the
    file names are n_m_weights.npy, where n is the number of neurons
    in the first hidden layer and m the number in the second;
    changing this is, again, trivial.
'''

import numpy as np

from ase.calculators.calculator import Calculator, all_changes

'''
    Define file names for weight arrays; e.g. ['25_25']
    for a single network with a 25-25 layout, or
    ['25_25', '25_20'] for a committee machine
    consisting of two networks with 25-25 and 25-20
    layouts.
'''

# If data set has been normalized, load the normalization factors
# Otherwise, uncomment the following two lines:
# mean = np.zeros(13)
# normalizer = np.ones(13)
mean = np.load('meaner.npy')
normalizer = np.load('normalizer.npy')
```

```
weight_networks = ['25_25', '15_15', '25_20']
num_networks = len(weight_networks)

weights_array = [0]*num_networks
biases_array = [0]*num_networks

for i in range(num_networks):
    weight_matrix = np.load(
        weight_networks[i]+'_weights.npy', allow_pickle=True)

    weights_array[i] = np.array(
        [weight_matrix[0], weight_matrix[2], weight_matrix[4]])
    biases_array[i] = np.array(
        [weight_matrix[1], weight_matrix[3], weight_matrix[5]])

num_layers = len(weights_array[0])

# z coordinate of atoms in the topmost layer of slab
top_height = 14.17499184683426
# Distance between two atoms in the topmost layer of slab;
# 3.639641 is the bulk lattice parameter
surf_latpar = 3.639641/2**0.5


class nn_calc(Calculator):
    implemented_properties = ['energy', 'forces']

    def __init__(self, **kwargs):
        Calculator.__init__(self, **kwargs)

    def calculate(self, atoms=None, properties=['energy'],
                  system_changes=all_changes):
        Calculator.calculate(self, atoms, properties, system_changes)

        self.positions = self.atoms.positions

        temp_energy = 0.0
        temp_forces = np.array([
                [0., 0., 0.],
                [0., 0., 0.],
                [0., 0., 0.],
                [0., 0., 0.],
                [0., 0., 0.],
                [0., 0., 0.],
                [0., 0., 0.],
```

```
                [0., 0., 0.],
                [0., 0., 0.],
                [0., 0., 0.],
                [0., 0., 0.],
                [0., 0., 0.],
                [0., 0., 0.],
                [0., 0., 0.]])
        for i in range(num_networks):
            temp_energy += self.energy_calc(
                weights_array[i], biases_array[i])
            temp_forces += self.force_calc(
                weights_array[i], biases_array[i])

        self.results['energy'] = temp_energy/num_networks
        self.results['free_energy'] = temp_energy/num_networks
        self.results['forces'] = temp_forces/num_networks


    def energy_calc(self, weights, biases):
        x1_ = self.positions[12, 0]
        y1_ = self.positions[12, 1]
        z1_ = self.positions[12, 2] - top_height
        x2_ = self.positions[13, 0]
        y2_ = self.positions[13, 1]
        z2_ = self.positions[13, 2] - top_height
        current_layer = self.transformer(
            x1_, y1_, z1_, x2_, y2_, z2_)

        # Propagates inputs forward layer-by-layer
        for layer in range(num_layers-1):
            current_layer = np.tanh(np.matmul(
                current_layer, weights[layer])+biases[layer])
        current_layer =
            (np.matmul(current_layer, weights[num_layers-1])+
            biases[num_layers-1])

        return current_layer[0]


    def force_calc(self, weights, biases):
        x1_ = self.positions[12, 0]
        y1_ = self.positions[12, 1]
        z1_ = self.positions[12, 2] - top_height
        x2_ = self.positions[13, 0]
        y2_ = self.positions[13, 1]
        z2_ = self.positions[13, 2] - top_height
```

```
current_layer = self.transformer(
    x1_, y1_, z1_, x2_, y2_, z2_)
derivative_x1 = self.dx1(
    x1_, y1_, z1_, x2_, y2_, z2_)/normalizer
derivative_y1 = self.dy1(
    x1_, y1_, z1_, x2_, y2_, z2_)/normalizer
derivative_z1 = self.dz1(
    x1_, y1_, z1_, x2_, y2_, z2_)/normalizer
derivative_x2 = self.dx2(
    x1_, y1_, z1_, x2_, y2_, z2_)/normalizer
derivative_y2 = self.dy2(
    x1_, y1_, z1_, x2_, y2_, z2_)/normalizer
derivative_z2 = self.dz2(
    x1_, y1_, z1_, x2_, y2_, z2_)/normalizer

# Calculates derivatives layer-by-layer
for layer in range(num_layers-1):
    current_layer = np.tanh(np.matmul(
        current_layer, weights[layer])+biases[layer])
    premult = 1-current_layer**2
    derivative_x1 = np.multiply(premult,
        np.matmul(derivative_x1, weights[layer]))
    derivative_y1 = np.multiply(premult,
        np.matmul(derivative_y1, weights[layer]))
    derivative_z1 = np.multiply(premult,
        np.matmul(derivative_z1, weights[layer]))
    derivative_x2 = np.multiply(premult,
        np.matmul(derivative_x2, weights[layer]))
    derivative_y2 = np.multiply(premult,
        np.matmul(derivative_y2, weights[layer]))
    derivative_z2 = np.multiply(premult,
        np.matmul(derivative_z2, weights[layer]))

derivative_x1 =
    -np.matmul(derivative_x1, weights[num_layers-1])
derivative_y1 =
    -np.matmul(derivative_y1, weights[num_layers-1])
derivative_z1 =
    -np.matmul(derivative_z1, weights[num_layers-1])
derivative_x2 =
    -np.matmul(derivative_x2, weights[num_layers-1])
derivative_y2 =
    -np.matmul(derivative_y2, weights[num_layers-1])
derivative_z2 =
    -np.matmul(derivative_z2, weights[num_layers-1])
```

```
        return np.array([
                [0., 0., 0.],
                [0., 0., 0.],
                [0., 0., 0.],
                [0., 0., 0.],
                [0., 0., 0.],
                [0., 0., 0.],
                [0., 0., 0.],
                [0., 0., 0.],
                [0., 0., 0.],
                [0., 0., 0.],
                [0., 0., 0.],
                [0., 0., 0.],
                [derivative_x1,
                 derivative_y1,
                 derivative_z1],
                [derivative_x2,
                 derivative_y2,
                 derivative_z2]])

    '''
        Transforms Cartesian coordinates into input space functions
    '''
    def transformer(self, x1, y1, z1, x2, y2, z2):
        return (np.array([
            self.p12(x1,y1,z1,x2,y2,z2),
            self.G1(x1,y1,x2,y2),
            self.G2(x1,y1,x2,y2),
            self.G3(z1,z2),
            self.G4(x1,y1,x2,y2),
            self.G5(x1,y1,x2,y2),
            self.G6(z1,z2),
            self.G7(x1,y1,x2,y2),
            self.G8(x1,y1,z1,x2,y2,z2),
            self.G9(x1,y1,z1,x2,y2,z2),
            self.pX(x1,y1,x2,y2),
            self.pY(x1,y1,x2,y2),
            self.pZ(z1,z2)])-mean)/normalizer

    '''
        Input space functions
    '''
    def p12(self, x1,y1,z1,x2,y2,z2):
        r = np.sqrt((x1-x2)**2+(y1-y2)**2+(z1-z2)**2)
        return np.exp(-r)
```

```
    def px(self, x,y):
        return (np.cos(4*np.pi*y/(surf_latpar*3**0.5))+
            2*np.cos(2*np.pi*x/(surf_latpar))*
            np.cos(2*np.pi*y/(surf_latpar*3**0.5)))

    def py(self, x,y):
        return (np.sin(4*np.pi*y/(surf_latpar*3**0.5))-
            2*np.cos(2*np.pi*x/(surf_latpar))*
            np.sin(2*np.pi*y/(surf_latpar*3**0.5)))

    def pz(self, z):
        return np.exp(-z)

    def G1(self, x1,y1,x2,y2):
        return self.px(x1, y1) + self.px(x2, y2)

    def G2(self, x1,y1,x2,y2):
        return self.py(x1, y1) + self.py(x2, y2)

    def G3(self, z1,z2):
        return self.pz(z1) + self.pz(z2)

    def G4(self, x1,y1,x2,y2):
        return self.px(x1, y1) * self.px(x2, y2)

    def G5(self, x1,y1,x2,y2):
        return self.py(x1, y1) * self.py(x2, y2)

    def G6(self, z1,z2):
        return self.pz(z1) * self.pz(z2)

    def G7(self, x1,y1,x2,y2):
        return (self.px(x1, y1)*self.py(x1, y1)+
            self.px(x2, y2)*self.py(x2, y2))

    def G8(self, x1,y1,z1,x2,y2,z2):
        return (self.px(x1, y1)*self.pz(z1)+
            self.px(x2,y2)*self.pz(z2))

    def G9(self, x1,y1,z1,x2,y2,z2):
        return (self.py(x1, y1)*self.pz(z1)+
            self.py(x2,y2) * self.pz(z2))

    def pX(self, x1,y1,x2,y2):
        x_mean = (x1+x2)/2.
        y_mean = (y1+y2)/2.
```

```
        return self.px(x_mean, y_mean)

    def pY(self, x1,y1,x2,y2):
        x_mean = (x1+x2)/2.
        y_mean = (y1+y2)/2.
        return self.py(x_mean, y_mean)

    def pZ(self, z1, z2):
        z_mean = (z1+z2)/2.
        return self.pz(z_mean)


    '''
        Derivatives of input space functions
    '''
    def dx1(self, x1, y1, z1, x2, y2, z2):
        X = (x1+x2)/2
        Y = (y1+y2)/2

        dp12 = self.dp12(x1, y1, z1, x2, y2, z2, x1, x2)
        dG1 = self.dpxdx(x1, y1)
        dG2 = self.dpydx(x1, y1)
        dG3 = 0.0
        dG4 = self.dpxdx(x1, y1) * self.px(x2, y2)
        dG5 = self.dpydx(x1, y1) * self.py(x2, y2)
        dG6 = 0.0
        dG7 = (self.dpxdx(x1, y1) * self.py(x1, y1) +
            self.px(x1, y1) * self.dpydx(x1, y1))
        dG8 = self.dpxdx(x1, y1) * self.pz(z1)
        dG9 = self.dpydx(x1, y1) * self.pz(z1)
        dpX = self.dpxdx(X,Y)/2.
        dpY = self.dpydx(X,Y)/2.
        dpZ = 0.0
        return np.array([dp12, dG1, dG2, dG3, dG4, dG5,
            dG6, dG7, dG8, dG9, dpX, dpY, dpZ])

    def dy1(self, x1, y1, z1, x2, y2, z2):
        X = (x1+x2)/2
        Y = (y1+y2)/2

        dp12 = self.dp12(x1, y1, z1, x2, y2, z2, y1, y2)
        dG1 = self.dpxdy(x1, y1)
        dG2 = self.dpydy(x1, y1)
        dG3 = 0.0
        dG4 = self.dpxdy(x1, y1) * self.px(x2, y2)
        dG5 = self.dpydy(x1, y1) * self.py(x2, y2)
```

```
        dG6 = 0.0
        dG7 = (self.dpxdy(x1, y1) * self.py(x1, y1) +
            self.px(x1, y1) * self.dpydy(x1, y1))
        dG8 = self.dpxdy(x1, y1) * self.pz(z1)
        dG9 = self.dpydy(x1, y1) * self.pz(z1)
        dpX = self.dpxdy(X,Y)/2.
        dpY = self.dpydy(X,Y)/2.
        dpZ = 0.0
        return np.array([dp12, dG1, dG2, dG3, dG4, dG5,
            dG6, dG7, dG8, dG9, dpX, dpY, dpZ])

    def dz1(self, x1, y1, z1, x2, y2, z2):
        Z = (z1+z2)/2.

        dp12 = self.dp12(x1, y1, z1, x2, y2, z2, z1, z2)
        dG1 = 0.0
        dG2 = 0.0
        dG3 = self.dpz(z1)
        dG4 = 0.0
        dG5 = 0.0
        dG6 = self.dpz(z1) * self.pz(z2)
        dG7 = 0.0
        dG8 = self.px(x1, y1) * self.dpz(z1)
        dG9 = self.py(x1, y1) * self.dpz(z1)
        dpX = 0.0
        dpY = 0.0
        dpZ = self.dpz(Z)/2.
        return np.array([dp12, dG1, dG2, dG3, dG4, dG5,
            dG6, dG7, dG8, dG9, dpX, dpY, dpZ])

    def dx2(self, x1, y1, z1, x2, y2, z2):
        X = (x1+x2)/2
        Y = (y1+y2)/2

        dp12 = self.dp12(x1, y1, z1, x2, y2, z2, x2, x1)
        dG1 = self.dpxdx(x2, y2)
        dG2 = self.dpydx(x2, y2)
        dG3 = 0.0
        dG4 = self.dpxdx(x2, y2) * self.px(x1, y1)
        dG5 = self.dpydx(x2, y2) * self.py(x1, y1)
        dG6 = 0.0
        dG7 = (self.dpxdx(x2, y2) * self.py(x2, y2)+
            self.px(x2, y2) * self.dpydx(x2, y2))
        dG8 = self.dpxdx(x2, y2) * self.pz(z2)
        dG9 = self.dpydx(x2, y2) * self.pz(z2)
        dpX = self.dpxdx(X,Y)/2.
```

```
        dpY = self.dpydx(X,Y)/2.
        dpZ = 0.0
        return np.array([dp12, dG1, dG2, dG3, dG4, dG5,
            dG6, dG7, dG8, dG9, dpX, dpY, dpZ])

    def dy2(self, x1, y1, z1, x2, y2, z2):
        X = (x1+x2)/2
        Y = (y1+y2)/2

        dp12 = self.dp12(x1, y1, z1, x2, y2, z2, y2, y1)
        dG1 = self.dpxdy(x2, y2)
        dG2 = self.dpydy(x2, y2)
        dG3 = 0.0
        dG4 = self.dpxdy(x2, y2) * self.px(x1, y1)
        dG5 = self.dpydy(x2, y2) * self.py(x1, y1)
        dG6 = 0.0
        dG7 = (self.dpxdy(x2, y2) * self.py(x2, y2) +
            self.px(x2, y2) * self.dpydy(x2, y2))
        dG8 = self.dpxdy(x2, y2) * self.pz(z2)
        dG9 = self.dpydy(x2, y2) * self.pz(z2)
        dpX = self.dpxdy(X,Y)/2.
        dpY = self.dpydy(X,Y)/2.
        dpZ = 0.0
        return np.array([dp12, dG1, dG2, dG3, dG4, dG5,
            dG6, dG7, dG8, dG9, dpX, dpY, dpZ])

    def dz2(self, x1, y1, z1, x2, y2, z2):
        Z = (z1+z2)/2.

        dp12 = self.dp12(x1, y1, z1, x2, y2, z2, z2, z1)
        dG1 = 0.0
        dG2 = 0.0
        dG3 = self.dpz(z2)
        dG4 = 0.0
        dG5 = 0.0
        dG6 = self.dpz(z1) * self.pz(z2)
        dG7 = 0.0
        dG8 = self.px(x2, y2) * self.dpz(z2)
        dG9 = self.py(x2, y2) * self.dpz(z2)
        dpX = 0.0
        dpY = 0.0
        dpZ = self.dpz(Z)/2.
        return np.array([dp12, dG1, dG2, dG3, dG4, dG5,
            dG6, dG7, dG8, dG9, dpX, dpY, dpZ])

    def dpxdx(self, x,y):
```

```
        return ((-4*np.pi/surf_latpar)*
            np.sin(2*np.pi*x/surf_latpar)*
            np.cos(2*np.pi*y/(surf_latpar*3**0.5)))

    def dpxdy(self, x,y):
        return ((-4*np.pi/(surf_latpar*3**0.5))*(
            np.sin(4*np.pi*y/(surf_latpar*3**0.5))+
            np.cos(2*np.pi*x/surf_latpar)*
            np.sin(2*np.pi*y/(surf_latpar*3**0.5))))

    def dpydx(self, x,y):
        return (-(4*np.pi/surf_latpar)*
            np.sin(2*np.pi*x/surf_latpar)*
            np.sin(2*np.pi*y/(surf_latpar*3**0.5)))

    def dpydy(self, x,y):
        return ((4*np.pi/(surf_latpar*3**0.5))*(
            np.cos(4*np.pi*y/(surf_latpar*3**0.5))-
            np.cos(2*np.pi*x/surf_latpar)*
            np.cos(2*np.pi*y/(surf_latpar*3**0.5))))

    def dp12(self, x1,y1,z1,x2,y2,z2,der1,der2):
        r12 = np.sqrt((x1-x2)**2+(y1-y2)**2+(z1-z2)**2)
        return -(der1-der2)*np.exp(-r12)/r12

    def dpz(self, z):
        return -np.exp(-z)
```