AARHUS UNIVERSITY

Coversheet

This is the publisher's PDF (Version of Record) of the article.

This is the final published version of the article.

How to cite this publication:

J. Chem. Phys. **150**, 244113 (2019); <https://doi.org/10.1063/1.5100141>

Publication metadata

Title:	Machine learning for potential energy surfaces: An extensive database and assessment of methods
Author(s):	Gunnar Schmitz, Ian Heide Godtlielsen, and Ove Christiansen
Journal:	Journal of Chemical Physics
DOI/Link:	https://doi.org/10.1063/1.5100141
Document version:	Publisher's PDF (Version of Record)

This article may be downloaded for personal use only. Any other use requires prior permission of the author and AIP Publishing. This article appeared in J. Chem. Phys. **150**, 244113 (2019); <https://doi.org/10.1063/1.5100141> and may be found at <https://doi.org/10.1063/1.5100141>

General Rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognize and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

If the document is published under a Creative Commons license, this applies instead of the general rights.

Machine learning for potential energy surfaces: An extensive database and assessment of methods

Cite as: J. Chem. Phys. **150**, 244113 (2019); <https://doi.org/10.1063/1.5100141>

Submitted: 15 April 2019 . Accepted: 05 June 2019 . Published Online: 27 June 2019

Gunnar Schmitz , Ian Heide Godtliessen , and Ove Christiansen

View Online

Export Citation

CrossMark

ARTICLES YOU MAY BE INTERESTED IN

[Perspective: Machine learning potentials for atomistic simulations](#)

The Journal of Chemical Physics **145**, 170901 (2016); <https://doi.org/10.1063/1.4966192>

[Unsupervised machine learning in atomistic simulations, between predictions and understanding](#)

The Journal of Chemical Physics **150**, 150901 (2019); <https://doi.org/10.1063/1.5091842>

[Improve the performance of machine-learning potentials by optimizing descriptors](#)

The Journal of Chemical Physics **150**, 244110 (2019); <https://doi.org/10.1063/1.5097293>

Lock-in Amplifiers
up to 600 MHz

Watch

Machine learning for potential energy surfaces: An extensive database and assessment of methods

Cite as: J. Chem. Phys. 150, 244113 (2019); doi: 10.1063/1.5100141

Submitted: 15 April 2019 • Accepted: 5 June 2019 •

Published Online: 27 June 2019

Gunnar Schmitz,^{a)} Ian Heide Godtliebsen,^{b)} and Ove Christiansen^{c)}

AFFILIATIONS

Department of Chemistry, Aarhus Universitet, DK-8000 Aarhus, Denmark

^{a)} Author to whom correspondence should be addressed: gunnar.schmitz@chem.au.dk

^{b)} ian@chem.au.dk

^{c)} ove@chem.au.dk

ABSTRACT

On the basis of a new extensive database constructed for the purpose, we assess various Machine Learning (ML) algorithms to predict energies in the framework of potential energy surface (PES) construction and discuss black box character, robustness, and efficiency. The database for training ML algorithms in energy predictions based on the molecular structure contains SCF, RI-MP2, RI-MP2-F12, and CCSD(F12*)(T) data for around 10.5×10^6 configurations of 15 small molecules. The electronic energies as function of molecular structure are computed from both static and iteratively refined grids in the context of automated PES construction for anharmonic vibrational computations within the n -mode expansion. We explore the performance of a range of algorithms including Gaussian Process Regression (GPR), Kernel Ridge Regression, Support Vector Regression, and Neural Networks (NNs). We also explore methods related to GPR such as sparse Gaussian Process Regression, Gaussian process Markov Chains, and Sparse Gaussian Process Markov Chains. For NNs, we report some explorations of architecture, activation functions, and numerical settings. Different delta-learning strategies are considered, and the use of delta learning targeting CCSD(F12*)(T) predictions using, for example, RI-MP2 combined with machine learned CCSD(F12*)(T)-RI-MP2 differences is found to be an attractive option.

Published under license by AIP Publishing. <https://doi.org/10.1063/1.5100141>

I. INTRODUCTION

Algorithms originating in the field of machine learning (ML) have gained a lot of interest in the recent years and also found their way into quantum chemistry as shown by recent work from different groups.^{1–8} A recent review by Butler *et al.* about ML techniques in chemistry can be found in Ref. 9. Furthermore, we want to give attention to a recent special issue on this topic,¹⁰ which indicates the impact of this field for theoretical chemistry. In this work, we want to assess different ML algorithms in the context of constructing potential energy surfaces (PESs). We investigate various types of Gaussian Process Regression (GPR), Kernel Ridge Regression (KRR), Support Vector Regression (SVR), and Neural Networks (NNs).

Potential energy surfaces have pivotal importance in chemistry determining ultimately both spectroscopy and dynamics of

molecular systems. At the same time, it is very challenging to construct an accurate representation of a potential energy surface (PES), requiring multiple evaluation of the energy for different structures, here denoted single points (SPs). The required number of such SPs grows steeply with system size. To address this problem, different methods have been developed for obtaining the PES with a tolerable number of SPs. This includes the so-called n -mode expansion^{11–14} to avoid a full dimensional description of the PES [also denoted cut-HDMR¹⁵ (high dimensional model representation), many-body expansion, cluster expansion, or other^{16–19}]. It, however, is still necessary to decide on the points for which the energies are to be evaluated. These points can be defined in terms of a fixed human-defined static grid. Iterative methods have also been successfully used to find efficient ways of sampling the PES,^{20–22} including the adaptive density guided approach (ADGA),²⁰ which iteratively builds up a grid based on convergence of the vibrational density times

the PES. Other physical requirements that can be imposed on the PES and used during its construction are permutational invariance under exchange of equivalent atoms, for example, by using a permutational invariant polynomial basis as fitting basis.²³ In addition to single point computation (SPC), there is also the issue of how to represent or use the PES. In some works, the PES is fitted to an analytical form, while in others, the generated information is used on the fly. For both types, it is a very interesting perspective to use a ML representation for providing energies based on hopefully a much more limited data set as input to the ML algorithm. For the first category, the ML representation just gives another analytical form, and for the direct approach, one can think about an analytical ML representation learned on the fly from, for example, energy and gradients.

In our assessment, we use systematic “leave out” studies, where we construct large data sets and test the different ML algorithms’ ability to reproduce energies for specific SP given the remaining set of data. Our aim here is solely the performance of the ML protocols for energy predictions, and we do not want to entangle this study with the different setups in which they are to be used.

The use of the resulting approximate ML PES in various contexts, such as calculation of anharmonic vibrational frequencies, or other studies of molecular dynamics will be the study of future work to complement the already existing studies^{24–26} on this issue. An already published study²⁷ of two of us already demonstrated the fruitful combination of GPR in the PES construction invoking the n -mode expansion to extrapolate higher order mode couplings. Although this study is a success, the $\mathcal{O}(n^3)$ scaling of GPR may render the method computationally costly going beyond 10k–100k samples. We are therefore interested in comparing other approaches to GPR and to explore sparse variants with asymptotic linear scaling. Note that it is also interesting to include derivative information in the training set, which can significantly increase the training set size.

There are already some benchmark studies available. Reference 24 contains a comparison of KRR, SVR, and NNs for atomization energies, but we complement this work by the newly available sparse GPR (SGPR) variants and a different choice for the kernel than the usual Gaussian or Laplacian kernel in addition to the large data set of around 10.5×10^6 configurations for 15 molecules on the CCSD(F12*)(T) level of theory. A particular important point of our set is to give the possibility to assess different delta-learning schemes.

There are a number of previous works employing ML for describing PESs. Already in a recent work, GPR was successfully applied to construct PESs,^{5,28–31} to evaluate PES matrix elements in a convenient format,³² in modeling of collision dynamics,^{2,33} and to accelerate geometry optimizations.^{34,35} Behler and co-workers in different papers^{3,4,36,37} successfully constructed PESs using NNs for molecular dynamic simulations. For a different scope, Manzhos *et al.* also presented how to use NNs to represent PESs for vibrational structure theory.^{38–40} NNs for PESs and force fields have been reported in several places.^{4,37–43} Recently, Richings and Habershon showed how to accelerate MCTDH (Multiconfiguration Time Dependent Hartree) using KRR to set up a local PES on the fly.⁴⁴ They also picked up an interesting idea of Duvenaud *et al.* in terms of additive kernels,⁴⁵ built up from increasingly higher dimensional functions. Furthermore, kernel based approaches such as KRR were

successfully applied to predict different molecular properties.^{1,46–48} Evolutionary algorithms can be mentioned in this context as well since their principles also have stimulated interesting work. Genetic algorithms were, for example, successfully applied to different optimization purposes by Oganov and co-workers for crystal prediction^{7,49–51} and Hartke and co-workers^{6,52,53} as well as Sierka and co-workers for structure optimization.^{54,55} Hammer and co-workers showed how to combine evolutionary algorithms with ML techniques to find minimum energy structures, which can potentially be used for the targeted design of molecules.^{56–58} Thus, there are many new potentially helpful algorithms for the theoretical chemical community, and it is not clear yet which of these new approaches will become a standard in the future. The aim of this work is therefore a prescreening of suitable ML algorithms that can be useful in the context of describing PESs, and the production of a large database to be used in further benchmarks. We note that our special emphasis lies on PES construction for subsequent use in vibrational structure calculation.

The paper is organized as follows: In Sec. II, we briefly describe the used ML algorithms as they relate to this study. Subsequently, in Sec. III, we describe the necessary computational details. Thereafter, a discussion of the results and an assessment of the selected algorithms follow in Sec. IV, before the concluding remarks in Sec. V.

II. THEORY

A. Machine learning algorithms

In this subsection, we give a very brief description of the ML algorithms used in our work, and we refer to the Appendix for a more detailed description.

1. Kernel functions

With the exception of NNs, we make use of kernel functions, which allows linear algorithms, which can be expressed in dot products, to adapt to nonlinear problems by replacing the dot products with a kernel function $k(\mathbf{x}, \mathbf{x}')$. The input vectors $\mathbf{x}$ and $\mathbf{x}'$ are then implicitly mapped to a higher dimensional feature space, in which the kernel is equivalent to a dot product in this feature space.^{59,60} For this work, we employ a modified Matérn 5/2 kernel

$$k(\mathbf{x}, \mathbf{x}') = \sum_i^d k^{M52}(x_i, x'_i; l_i), \quad (1)$$

$$k^{M52}(x_i, x'_i; l_i) = \sigma^2 \left(1 + \frac{\sqrt{5}|x_i - x'_i|}{l_i} + \frac{5|x_i - x'_i|^2}{3l_i^2} \right) \times \exp \left(-\frac{\sqrt{5}|x_i - x'_i|}{l_i} \right), \quad (2)$$

where σ^2 and l_i are usually referred to as hyperparameters and are optimized for the current regression or classification problem.

2. Gaussian process regression, kernel ridge regression, support vector regression, and variants thereof

GPR, in some context also referred to as kriging,^{61–63} belongs to the field of machine learning algorithms. It is assumed that our

observed function is distributed according to a normal distribution,

$$\mathbf{f} \sim \mathcal{N}(\mathbf{0}, \mathbf{K} + \sigma_n^2 \mathbf{I}), \quad (3)$$

of functions, where typically a zero mean function $\mathbf{0}$ is assumed and where $\mathbf{K}$ is the covariance matrix, in which elements i, j are obtained from a kernel function $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$, which encodes the correlation/closeness of the data points. The term $\sigma_n^2 \mathbf{I}$ is added as regularization term and imposes a noise on the observations and can increase numerical stability. From this prior distribution, a posterior distribution is obtained by conditioning the distribution to the observed inputs. The mean function of the posterior distribution is then used as predictor. The main cost for GPR is the inversion of the covariance matrix $\mathbf{K}$. This step scales as $\mathcal{O}(n^3)$, where n is the number of data points, and might prohibit large scale applications going beyond 100k samples. However, there are also sparse variants which select a suitable subset $m \ll n$ of the training set and allow training costs, which scale as $\mathcal{O}(m^2 n)$. The subset is often referred to as inducing points. These sparse variants all differ in the subset selection. In this work, we use the approach and implementation presented in Ref. 64, in which it was actually shown that the inducing points do not have to be part of the training points but can be artificially generated by the optimization procedure. For all sparse variants, we will prepend an S to the parent method, e.g., SGPR for GPR. An interesting variant of GPR, which we also pick up in this study, is Gaussian Process Markov Chains (GPMC). In GPMC, inference is done by a stochastic process, which allows us to also consider non-Gaussian likelihoods. However, to keep as close a comparison as possible to the other methods, we will limit ourselves to a Gaussian likelihood. A strength of GPR and its variants is that there exists an intrinsic route of optimizing the hyperparameters by maximizing the marginal likelihood without the need to partition the data set in training and validation set.

GPR is often interchangeably used with Kernel Ridge Regression (KRR). However, this may not be 100% precise in describing the theoretical context and usage. KRR is obtained by combining Ridge Regression^{65,66} with the kernel trick, and it turns out that the same predictive equations as for GPR are obtained. Nevertheless, GPR and KRR have a different theoretical background and the route of tuning the hyperparameters by maximizing the marginal likelihood is missing in KRR. Here, the parameters are usually obtained by grid search combined with cross-validation. Although the same recipe is sometimes also used for GPR, we make here a clear cut difference. If we refer to GPR, the hyperparameters are obtained by maximizing the marginal likelihood, and if we refer to KRR, they are determined using cross-validation.

Another interesting method for regression is Support Vector Regression (SVR).⁶⁷ Compared to KRR, the main idea is to use a different loss function, in which all errors inside a ε tube are set to zero and only fitting errors outside this tube are minimized. Usually, these conditions are loosened by introducing slack variables to achieve higher flexibility. The penalty imposed on observations that lie outside the ε tube is then adjusted by a box constraint constant C . By construction, SVR produces a sparse predictor consisting of n_{sv} support vectors. Depending on how n_{sv} is correlated with n , from linear scaling to cubic scaling, costs can be expected.

3. Neural networks

NNs⁶⁸ are vaguely inspired by biological neural networks utilizing the so-called *neurons* (sometimes also denoted *perceptrons*) organized in connected layers. The evaluation of each neuron consists of a linear transformation of the outputs from the previous layer with a set of weights. The architecture of a NN, i.e., number of layer and nodes/neurons per layer, is important for the accuracy and performance. The weights of the NN are optimized during the training period and enable the NN to learn the underlying relation of input and output signals. For this optimization, a loss function is defined and its gradient can be used for any gradient based optimization procedure. The gradient is calculated by the chain rule through the so-called back-propagation through the network.

B. The n -mode representation

Expressed in normal coordinates q with $M = 3N_{\text{atom}} - 6$ (5) degrees of freedom, a full dimensional description of the PES coupling all degrees of freedom quickly becomes unfeasible. This issue can be addressed by applying the n -mode expansion,^{11–14} which approximates the potential $V(\mathbf{q})$ as converging sequence of terms

$$V^{(1)}(\mathbf{q}), V^{(2)}(\mathbf{q}), V^{(3)}(\mathbf{q}), \dots, V^{(M)}(\mathbf{q}). \quad (4)$$

Here, $V^{(n)}$ includes at most n -mode couplings and is a linear combination of m -mode cuts ($m \leq n$; e.g., $V^{m_1 m_2}, V^{m_1}$). Thus, the potential can be written as

$$V = \sum_{\mathbf{m}_n \in \text{MCR}\{V\}} \tilde{V}^{\mathbf{m}_n}, \quad (5)$$

where MCR is the range of included mode combinations (MCs) $\mathbf{m}_n = (m_1, m_2, \dots, m_n)$. To avoid overcounting in this expansion, we use the definition

$$\tilde{V}^{\mathbf{m}_n} = \hat{S}^{\mathbf{m}_n} \sum_{n'=1}^n (-1)^{n-n'} \binom{n}{n'} V^{\mathbf{m}_{n'}}, \quad (6)$$

where $\hat{S}^{\mathbf{m}_n}$ is a symmetrization operator. For a 2-mode potential, this yields, for example,

$$\tilde{V}^{m_1 m_2} = V^{m_1 m_2} - V^{m_1} - V^{m_2}. \quad (7)$$

These potentials are usually obtained via fitting (by polynomials) to $\tilde{V}^{m_1 m_2}$ values obtained from the data of SPCs, but they could also be fitted to ML generated data. The number of the SPCs can become quite large as for an n -mode potential, it is given by

$$N_{\text{points}} = \sum_{k=1}^n \binom{M}{k} (g_k)^k, \quad (8)$$

where g_k is the number of SPs per direction in the generation of the direct product grid at the mode-coupling level k . These static grids enable an easy way for forming different systematic sets based on the spacing of the grid points. To classify the grids obtained in this way, we use the notation $g[m, n, l]$, which means that m SPs are carried out for the 1-mode part along each distortion, n for the 2-mode couplings, and l for the 3-mode couplings. Additionally, the reference point will be included as center. This gives for the 2-mode part $(n+1)^2$ and for the 3-mode part $(l+1)^3$ SPCs. A schematic

illustration can be found in Fig. 1. The 2-mode grid in this schematic drawing is obtained as the direct product of the 2-mode grids. We also use the short hand notation gm , which indicates a $g[m]$, a $g[m, m]$, or a $g[m, m, m]$ grid depending on the context.

It is possible to train the ML algorithm on the 1M level to be used for the 1M + 2M or 1M + 2M + 3M computations as it was recently shown,²⁷ but in the current work, we will use samples of the nM data to predict nM data. Thus, since the data set is built up from 1, 2, or 3-dimensional spaces, one can expect some correlation in the training set. What we observe is that some points can actually be quite close with respect to the closeness measure encoded by the kernel function. However, this does not necessarily help the ML algorithm. In particular, GPR and its variants can have numerical problems since rows and columns in the covariance matrix might become linearly dependent. But in our experience, imposing a noise of 1×10^{-7} circumvents this problem.

C. The adaptive density-guided approach

ADGA²⁰ is an iterative procedure to select points for fitting the PES by means of the average vibrational density,

$$\rho_{\text{iter}}^{\text{ave}}(q_m) = \frac{1}{N_{\text{modal}}} \sum_{s^m=1}^{N_{\text{modal}}} |\varphi_{s^m}^m(q_m)|^2, \quad (9)$$

which is constructed from the VSCF (vibrational self-consistent-field) one-mode mean field eigenfunctions, $\varphi_{s^m}^m(q_m)$. The convergence criteria of the ADGA determine whether the potential is stable in terms of changes in the energylike quantity,

$$\int_I \rho_{\text{iter}}^{\text{ave}}(q_m) V_{\text{iter}}^{\text{m}_n}(q_m) dq_m, \quad (10)$$

in different intervals I . The convergence is checked up to an absolute and a relative change from iteration to iteration. If necessary, ADGA also extends the grid boundaries. The static grids enable a more systematic way to partition the training sets, but the iteratively refined

grids obtained with the ADGA offer a more physical motivated sampling and might therefore be beneficial for the training.

D. Choice of coordinates

An important choice for the performance of the ML algorithm is the choice of coordinates. It is beneficial if they fulfill (some) physical properties by construction. Most importantly, they should obey physical invariances such as rotational and translational invariance. A simple choice which fulfills these requirements is a set of internal coordinates consisting of bond length, bond angles, and dihedrals. They allow for a straightforward physical interpretation in terms of energy changes, and $3N - 6$ coordinates are sufficient for the description of the system. An additional nice property would be invariance with respect to exchange of equivalent particles, which is missing in this choice. Note that it should be relatively straightforward to pick up ideas of Conte *et al.* on their work⁶⁹ on permutationally invariant fitting bases for PESs and that for the current purpose, this missing property in internal coordinates is not a big issue since the same order of atoms can easily be ensured. Furthermore, we refer to the work of Bartók *et al.*⁷⁰ who discussed several ways how to represent the molecular structure in the context of GPR. They also demonstrate atomistic descriptors, which probe their surroundings, and thus might also be interesting in the present context. However, internal coordinates are a more natural choice since they offer a direct physical interpretation and the main objective of the current study is the comparison of the different ML algorithms. Altogether, we employ the points obtained in the n -mode expansion PES construction in normal coordinates and transform these to a data set in $3N - 6$ nonredundant internal coordinates.

III. COMPUTATIONAL DETAILS

For our assessment, we made use of the GPflow library⁷¹ for GPR, SGPR, GPMC, and Sparse Gaussian Process Markov Chains (SGPMCs). Furthermore, we used the SciKit⁷² package for KRR and SVR. Tensorflow⁷³ was used for the NN based predictions. Note that especially for the efficiency comparison of KRR and GPR, the use of different implementations is not optimal. GPR and KRR should essentially have the same performance since only the route for hyperparameter optimization differs, but we will observe (*vide infra*) that GPR is significantly more efficient. This is, however, likely due to the more efficient implementation in GPflow. Thus, in this study, we are mainly interested in the accuracy comparison of GPR and KRR and how the different hyperparameters affect the accuracy of the predictor.

To generate the SPs required to train the ML algorithms, we used the TURBOMOLE program package V7.0.⁷⁴ For the Hartree-Fock calculations, we applied the DSCF module⁷⁵ and for the CCSD(F12*)(T) calculations the ccscdf12 module.⁷⁶ As basis, the cc-pVXZ-F12 basis set family of Peterson and co-workers was used, which will be abbreviated as XZ-F12.⁷⁷

Throughout this work, the frozen core approximation was used for calculating the correlation energy. All HF orbitals with an energy below -3.0 Hartree were excluded from the correlation treatment. In the F12 calculations, we applied the complementary auxiliary basis set (CABS) approach⁷⁸ with a CABS threshold of 1×10^{-8} .

FIG. 1. Schematic representation of a $g[4, 4]$ grid including the 1-mode part and 2-mode part.

The threshold for ADGA is chosen such that the algorithm converges when the relative change in the PES from one iteration to the other is less than 1×10^{-2} or the absolute change less than 1×10^{-6} . The static grids do not require any thresholds but instead a specification for the size and density of points in an equally spaced grid.

The optimization of the hyperparameters is crucial. Based on numerical testing, we propose a rather pragmatic choice in the framework of the n -mode expansion. Based on the 1-mode data, the Basin hopping algorithm⁷⁹ as implemented in the SciPy package⁸⁰ is applied, which is a global optimization algorithm followed by local reoptimization using the Adam optimizer as implemented in Tensorflow⁷³ based on the whole data. In the [supplementary material](#), a small discussion can be found, which initiated the outlined optimization strategy.

For SVR and KRR, where the hyperparameter cannot be selected by optimizing of the marginal likelihood, we tuned the parameters by a randomized search over parameters, where each set of hyperparameters is sampled from a distribution over possible parameter values. The errors were assessed using cross-validation and applying the negative of the squared deviation as the scoring function.

In the case of SVR, the regularization constant C was tuned using cross-validation, while for ϵ , we used the default value of 0.1 in SciKit.⁷² The weights in NNs were optimized via back-propagation using cross-validation as error measure. Thus, like in SVR and KRR, no statistical error measure like in GPR was optimized. We used at maximum 4096 training examples as batch size in one forward/backward pass. More details on the NN architecture will be given in Sec. IV C.

To train the ML algorithms, we used a nonredundant set of $3N - 6$ internal coordinates consisting of bond length, and angles and dihedrals as described in previous work.³⁴ For all sparse variant SGPR, SGPMC, etc., we used a maximum of 1000 inducing points. If the training data contained fewer points, we used as many inducing points as training points.

To test the ML algorithm, we performed a systematic “leave out” test from the g8 grid, where we leave out the interior part of the g4 grid and use that as the validation grid. This is illustrated for the 1-mode case in Fig. 2.

FIG. 2. Schematic representation a g[4] validation grid to be used together with a g[8] training grid. Points are systematically removed to only validate on interior points. The illustration is done for the 1-mode part. For the 2-mode part, a direct product grid of the 1-mode grids is used.

IV. RESULTS AND DISCUSSION

This section is structured as follows: First, we will give details on the test sets used in this study. Afterward, a benchmark of kernel based ML methods is given together with a closer analysis of delta-learning.⁴⁸ Thereafter, an assessment of NNs is provided to complement the study, and in the end, a comparison in terms of computational efficiency is carried out.

A. The test sets

We consider a set of 15 molecules, which is depicted in Table I for which we gathered typical data from the construction of a PES for the subsequent use in anharmonic vibrational wave function computations.

Data on the CCSD(F12*)(T)/DZ-F12 level from static and iteratively refined grids aim at constructing PESs using the n -mode expansion. We assess the various ML algorithms in their capability to predict molecular energies for the different contributions to the overall energy.

We assembled different test sets specified by the grid size. In the following, we discuss only a representative subset of these super-sets. All sets combined contain over 10.5×10^6 configurations. These data alone are valuable and publicly available at a link given in the [supplementary material](#) for further benchmark tests.

For the static grid, we use a systematic way to partition the training set as described in Sec. II B, and for the iteratively refined grid, we partition the data in training and validation set randomly.

The training set size is determined via the grid size and the degrees of freedom in a molecule. Taking the g8 grid as an example, we had 116 961 configurations in total in the database for ethene, 11 249 for H₂CO, and 729 for water. For the g16 grid, the analog numbers are 918 209, 85 857, and 4913. This gives a small impression on data set sizes, and in the current setup, molecules of comparable size will also have a comparable data set.

Besides the obtainable accuracy, we will assess the dependence of the prediction to the amount of training data as well as the computational costs with respect to the amount of training data and judge black box character and robustness of the tested approaches.

For the delta learning schemes, we introduce the following abbreviations:

- Δ_{ZERO} refers to the total CCSD(F12*)(T) energy and therefore to no delta learning.
- Δ_{SCF} refers to learning the difference of CCSD(F12*)(T) to SCF (HF).

TABLE I. List of the molecules in the test set (in no particular order).

Molecules		
NH ₃	<i>cis</i> -HNO ₂	Cl ₂ O
H ₂ S	H ₂ S ₂	NO ₂ H
ClNO	C ₂ H ₄	F ₂ O
NOF	SOCl ₂	<i>trans</i> -CHFCHF
H ₂ CO	H ₂ O ₂	H ₂ O

- Δ_{MP2} refers to learning the difference of CCSD(F12*)(T) to RI-MP2.
- $\Delta_{MP2-F12}$ refers to learning the difference of CCSD(F12*)(T) to RI-MP2-F12.

B. Kernel based methods

1. Systematic partitioned grid data

In the following, we discuss the results obtained with the static grids. As a validation set, we choose the g4 validation grid as described in Sec. III and Fig. 2. Figure 3 shows the relative RMSD (root mean square deviation) in this test for using 1M, 1M + 2M, and 1M + 2M + 3M data. The relative errors for Δ_{ZERO} are lowest with an increase in the order $\Delta_{SCF}/\Delta_{MP2}$ and $\Delta_{MP2-F12}$. This pattern is quite consistent except for KRR. On first sight, these results are a bit counterintuitive. It should be harder to get a good predictor for Δ_{ZERO} since the total energy is stronger varying with respect to conformational changes. However, good relative errors might still not be enough since the absolute error is more important in practice. We will discuss this issue in more detail in Sec. IV B 2.

The RMSDs for the tested GPR models are very similar. However, for SGPR, the problems of predicting $\Delta_{MP2-F12}$ with large accuracy are more pronounced. In most cases, GPMC is more accurate than GPR or SGPR. However, the behavior of SGPMC is not as well-defined. While SGPR is for our test set a close approximation to GPR, the deviations found between GPMC and SGPMC are larger. Nevertheless, the SGPMC results are comparable to the GPR and SGPR results in accuracy. KRR and SVR often perform quite well although their way to optimize hyperparameters is more limited in the sense that only cross-validation can be used. In many cases, they are even better than GPR. The dependence on the training set size in terms of using 1M, 1M + 2M, or 1M + 2M + 3M data is slightly larger for KRR. Compared to the other tested models, the RMSDs are lowest for the 3M case. For the tested GPR models, the picture approximately stays the same for the 1M, 1M + 2M, and 1M + 2M + 3M cases. Here, it has to be noted that from a pragmatical view,

the only difference between GPR and KRR is the selection of the hyperparameters. Using cross-validation, an actual error measure is minimized, while in GPR, the marginal likelihood is maximized. The different extrema in these different optimization problems need not to be 100% related. It seems that maximizing the marginal likelihood gives more stable results, while the minima obtained via cross-validation can give occasionally better results. In the future, it might be interesting to study the possibility of a hybrid approach, e.g., a simultaneous weighted optimization on both surfaces. Concerning SVR, it has to be noted that it has more problems to predict $\Delta_{MP2-F12}$ correctly. Here, in most cases, the other algorithms are superior.

We also investigated how the relative RMSD changes if the grid set size is increased in terms of more grid points. For schematic representation for training grids ranging from g4 to g16, we refer to the [supplementary material](#). Here, we just summarize the key findings when examining the larger test sets: It turns out that the effect of increasing the training set size for the various GPR models and SVR is not very strongly pronounced. Already using just a few data points is sufficient for an accurate description. For KRR, a stronger dependence on the training set size is observed. Again, a possible explanation is that the more sophisticated optimization of hyperparameters in the GPR case gives a more robust outcome.

In Fig. 3, we averaged over all molecules in our test set. It is also interesting and important to validate that the molecules behave approximately the same and that no extreme outliers are present. For this purpose, we plot in Fig. 4 the relative error for the tested delta-learning schemes using GPR, where we used the 1M + 2M data for training. For Δ_{ZERO} , the relative error spread is large, but all relative errors are quite small regardless. It is not possible to identify any particularly “bad” molecules. For Δ_{SCF} and Δ_{MP2} , the spread of errors is much more narrow and they behave approximately the same. For $\Delta_{MP2-F12}$, we can see a larger discrepancy between the molecules. Here, larger errors can be found for *cis*-HNO₂, ClNO, F₂O, H₂O₂, NO₂H, NOF, and H₂O. In most of

FIG. 3. Relative RMSD (unitless) for different ML algorithms using 1M, 1M + 2M, and 1M + 2M + 3M data from a g8 grid. Validation is done on the g4 validation grid.

FIG. 4. Scatter plot of the relative errors for the different molecules and tested delta-learning schemes for GPR using the 1M + 2M data from a g8 grid. Validation is done on the g4 validation grid.

these cases, the predicted correction is also relatively small compared to the other examples. Predicting a small number with large relative accuracy is problematic (*vide infra*) and here the main source for the clear separation. In conclusion, there are no particular problematic cases for all delta-learning schemes. The major difficulty lies in the absolute magnitude of the prediction value and not the molecular system.

A fine grained look at the molecular level is also interesting for comparing the different kernel methods. In Fig. 5, we present for Δ_{MP2} the errors for the different tested methods as a function of molecules. It turns out again that GPMC is overall the most accurate approach, followed by KRR and GPR. The largest errors can be found for SVR. SGPR is observed as a better approximation to GPR than SGPMC to GPMC. In particular, interesting is the comparison of GPR and KRR. We already observed in Fig. 3 for Δ_{MP2} better accuracy for KRR than for GPR, and naturally this is also observed in

FIG. 5. Scatter plot of the relative errors for the different molecules and Δ_{MP2} for the different kernel based methods using the 1M + 2M data from a g8 grid. Validation is done on the g4 validation grid.

the more fine grained view. It is not surprising that the direct minimization of an error leads to a smaller error than optimizing the marginal likelihood. However, in the latter case, we do not need to partition the dataset in training and validation sets. Again, we point out that it might be interesting to explore hybrid schemes, where both error measures are simultaneously optimized. Eventually, this could result in being able to use smaller validation sets. Again, there are some variations, but it is not directly possible to identify outliers.

2. Closer analysis concerning delta learning

So far, we mainly discussed relative errors, but a small relative error for large energies can still render a method less useful. Therefore, we complement our findings with absolute errors for GPR to answer the question which delta learning strategy offers the largest accuracy or is the most economical one. Figure 6 depicts the absolute errors relative to CCSD(F12*)(T) for delta learning schemes with decreasing Δ. If the GPR is trained on the full energy, the errors show a large spread. The spread decreases in the order Δ_{SCF}, Δ_{MP2}, and Δ_{MP2-F12}. The last one shows also the smallest error. On first sight, this might be counterintuitive since we observed most numerical problems for this approach, but we need to keep in mind that the correction to the target method is also quite small and therefore larger errors in the corrections are acceptable. Nevertheless, the numerical problems remain and the data reveal that correcting RI-MP2 is not much worse than correcting RI-MP2-F12. If one also keeps in mind that RI-MP2 is typically 7–10 times faster than RI-MP2-F12, it makes this strategy the most economical.

Still, it is important to investigate and understand the large relative errors for the Δ_{MP2-F12} correction further. This problem is manifold. The correction is rather small, and we are now in a regime where the numerical noise originating from the approximations introduced in CCSD(F12*)(T) will play a role. These approximations are of course valid in standard CCSD(F12*)(T) but can cause difficulties if one is interested in calculating small corrections. On the other side, there are also numerical issues for the ML algorithm in dealing with small numbers. The energy differences can be rescaled to lift this restriction to some extent, but the problem remains that not every digit will carry physical significance but may be dominated by numerical noise. To investigate this further, we looked at a few examples in more detail. For H₂CO, H₂O, F₂, FH, and H₂, we also created molecular data sets, where we tightened many numerical thresholds and increased the auxiliary basis sets (CABS and density fitting) to those corresponding to cc-pVQZ-F12 while still employing cc-pVDZ-F12 as orbital basis. Figure 7 shows the comparison of the predictive accuracy for the different test sets using the same setup for the GPR. It is clear that the accuracy can be dramatically increased. Due to the reduced numerical noise, the underlying physical pattern for changes in the energy due to geometrical changes can now more easily be detected by the GPR. But reducing the numerical noise comes with an increase in computational costs for training, which is not economical. It is thus important to find the sweet spot between large and small Δ using delta-learning due to economical reasons and numerical stability. In the specific case, the use of RI-MP2 to target CCSD(F12*)(T) seems more favorable than using RI-MP2-F12, which on paper is the better method.

FIG. 6. Overview of the absolute errors for a GPR prediction of the CCSD(F12*)(T) focusing on delta learning. The assessment is based on 2M data. (a) shows the errors for all delta-learning schemes, while in (b), the plot is repeated for just Δ_{MP2} and $\Delta_{\text{MP2-F12}}$ where the y axis has been adjusted accordingly. On the x axis, the index of the datapoint is shown.

3. Randomized partitioned grid data

For the iteratively refined grids, we also explore the extreme case of a randomized partition of training and validation set. Figure 8 shows the analog RMSD plots. It is evident that in some cases, SGPR, SGPMC, and SVR perform very poorly when predicting $\Delta_{\text{MP2-F12}}$ and are in this regard less robust than the other methods. The sparse methods seem to be more sensitive if the training data are contaminated with numerical noise or less systematically selected with possible holes in the training data.

But excluding this case, the overall impression is that reasonable accuracy is obtained. While it is a more difficult challenge,

most algorithms show a similar accuracy compared to the previous assessment. However, the previous results with the systematic partition are slightly better, presumably due to that randomly taking point out is more challenging than based on a systematic, physical criterion.

C. Neural networks

So far, we only discussed the results for the GPR based approaches, KRR and SVR, and did not look at the accuracy of NNs. The reason is that NNs have many more adjustable parameters, and therefore, we discuss them here on their own. In a broad screening, we tested several activation functions including sigmoid, relu, elu, selu, and softplus. We found that elu and selu give the best performance for our purposes. In order to reduce the search space, we therefore limit the study to these two. Besides the activation function, the architecture of the NN is determined by the number of hidden layers L and the number of nodes m_n per hidden layer. To reduce the number of adjustable parameters, we use the same number of nodes for each layer. The number of nodes are chosen as an integer multiple N_m of the input dimension (length of the input vector, e.g., number of coordinates) in order to reflect the different information content for different molecules. Water as an example has 3 internal coordinates. If the factor N_m is 5, this means that 15 nodes per layer are used (see Fig. 13). The initial testing suggested choosing N_m at least to be in the range of 5–10. Using smaller numbers resulted in a low accuracy. The so far mentioned parameters obviously influence the predictive accuracy of the NN, but also the initial guess for the weights and the used optimization algorithm, including its internal parameters, can lead to quite different accuracy. In this study, we used the Adam optimizer, where we also have studied the effect of applying different learning rates (LRs). In most cases, the deviations are in a reasonable range, but

FIG. 7. Relative RMSD (unitless) for GPR on a g16 grid using 1M and 1M + 2M data for predicting the correction from RI-MP2-F12 to CCSD(F12*)(T) correlation energies. The bars in red indicate default settings in the electronic structure program, and the bars in blue indicate tighter settings with respect to auxiliary basis sets and convergence thresholds.

FIG. 8. Relative RMSD (unitless) for different ML algorithms using 1M, 1M + 2M, and 1M + 2M + 3M data from an ADGA grid. Validation is done on randomly selected points.

larger differences might be an indication that more training data are required. This is one way to check the sanity of a NN, but other methods exist. For example, in MD simulations, two different NNs are fitted and compared: One on which the actual MD is run and the other as validation. If the two NNs start to differ too much, they are retrained.³⁶

We did many tests but limit the presentation to the results obtained using NNs with $N_m = 10$ and $L = 5$ with the g16 grid for 1M and 1M + 2M data for training since for these data, the results are sufficiently accurate and different start guesses have no huge impact on the accuracy. Figure 9 shows results for these NNs with different learning rates and epsilon values in the Adam optimizer.

In the [supplementary material](#), we supply results for other tested architectures. Although the RMSDs are mostly in the same range as for the kernel based methods, Fig. 9 demonstrates that the different settings can lead to a quite different accuracy for the different delta-learning schemes. The learning rate has still a large influence on Δ_{ZERO} . For the other Δ schemes, a more well-behaved trend is observed. As a comparison, also the analog results for GPR are given. For Δ_{ZERO} and $\Delta_{\text{MP2-F12}}$, GPR is more accurate, but in the other cases, the results are comparable or the NN results are slightly more accurate.

The setup with an elu activation function and a learning rate of 1×10^{-2} and an epsilon of 1×10^{-4} seem to work for the range

FIG. 9. Overview of the relative RMSDs (unitless) for a NN with $N_m = 10$ and $L = 5$ using different learning rates (LRs) and epsilon (ϵ) values in the Adam optimizer. The results for two different random starting guesses are depicted, which are referred to as 1. run and 2. run. The NNs were trained for 5000 epochs (epoch = forward + backward pass in NN). As a comparison, the analog results for GPR are given.

of tested delta-learning schemes. The shown setup is also stable in terms of restarts with different initial guesses for the weights.

We note that in the construction of Behler type NN potentials, usually fewer hidden layers and nodes per layers are used.³⁷ The key difference to their approach is that we use internal coordinates as descriptors, which for our purpose is sufficient, but using more advanced descriptors such as in the mentioned NN potentials reduces these requirements. However, this also requires more labor to set up a suitable descriptor, which may require some trial and error. Note that although internal coordinates might be a bit unusual and not optimal in many chemical ML related projects, they are well-established and good for representing the PES since they offer an easy interpretation and offer the basic requirements of a descriptor of being invariant under rotations and translations. An additional nice property would be permutational invariance under exchange of equivalent particles; however, for the current benchmark, this is not important since the same order of atoms can easily be ensured. We note in passing that the permutational invariance could be achieved by adopting the ideas of Bowman and co-workers on permutationally invariant polynomial fitting bases;⁶⁹ see also the work of Jiang *et al.*⁸¹

D. Efficiency

Besides the accuracy, it is also important that the underlying ML algorithm is sufficiently fast. Therefore, we focus in this section on wall time costs for training. The time for prediction is comparatively small for all tested algorithms. Thus, depending on the intended use, the time required for training can be crucial. If the ML algorithm is used to construct a specialized force field, to be used afterward many times, the time for training, which has to be done only once, might be not the most important factor in choice of model but rather accuracy and a fast evaluation for the predictor. If the training is frequently repeated, such as during an iterative PES construction, it is important to consider alternatives with low costs for training.

It is difficult to compare the real life performance since the number of iterations for the hyperparameter optimization might be quite different and also very dependent on the molecule as well on the training data, and therefore, we use a simplified model. We use only a squared exponential kernel, focus on one molecule (ethene) and one training set which we gradually increase, and only do a local optimization of the hyperparameters. This is a very constructed example, but it enables an easier comparison of the different algorithms and the observed trend qualitatively carries over to the other examples.

Figure 10 shows on a double logarithmic plot the wall times required for the training for the different models. More or less for all models, it can be observed that the asymptotic scaling kicks in around 1000 data points, and before that, low scaling parts with a larger overhead dominates. For GPR, we can then observe the cubic scaling, while SGPR shows a linear scaling. GPR and GPMC are quite close, while GPMC for large data sets is a bit more favorable. On the other hand, SGPMC is an order of magnitude more costly than SGPR. The comparison of GPR and KRR is a bit difficult since two different implementations are used. GPR and KRR use the same working equations and thus their efficiency should be very close in terms of Wall clock time. Only the offset might be different

FIG. 10. Wall time required for training including hyperparameter optimization. As an example, the g16 including 2M couplings for ethene was used and gradually increased. For the NN, an elu activation function was used and it consisted of 5 hidden layers with 100 nodes and was trained for 1000 epochs.

depending on the number of iterations used in the hyper parameter optimization. Here, however, KRR is significantly less efficient. To a large part, this is probably related to the different linear algebra backends used in SciKit and GPFlow. The latter uses the highly efficient Tensorflow library.

Looking at the data, SVR comes out as the fastest method for large data sets followed by SGPR. The curve for SVR is in the beginning quite flat and later starts to behave more linearly. The number of determined support vectors is 9, 17, 52, 107, 125, 342, 745, 1445, 6212, and 8052 for the given data points. In the beginning, the ratio of n_{sv}/n is close to 1.0 and decreases while increasing the data set size n . For the last data point, n_{sv}/n is observed to be 0.47. Thus, less than half of the training data are required to set up the predictor. Figure 10 also includes an example for a NN curve

FIG. 11. Wall time required for training the NN with different architectures and number of epochs. As an example, the g16 including 2M couplings for ethene was used and gradually increased. The NN used an elu activation function.

as a comparison. However, a direct comparison is not straightforward. While the dependence of the computational costs with respect to the training set size is small compared to the other tested ML algorithms, the costs depend heavily on the architecture [number of hidden layers, nodes per layer, and epochs (epoch = forward + backward pass in NN)], which can be quite different for different setups. To complement the findings, Fig. 11 shows the computational costs for NNs, where the mentioned parameters are varied and where the GPR timings are repeated for comparison. In this comparison, NNs carry some overhead, which makes them computationally less favorable for small training set sizes compared to GPR, but around 1000 data points, NNs become computationally more efficient. It is important to note again that it is a model system but hopefully somewhat representative for the typical molecular system size and training set size regime, where these algorithms are applied.

V. OUTLOOK AND CONCLUSION

We assessed various ML algorithms in the context of constructing potential energy surfaces in their ability to predict different molecular energies. GPMC is accuracywise a good alternative to GPR, and exploration of non-Gaussian likelihoods can lead to further improvements. SGPR is often a close approximation to GPR. SGPMC is, however, less accurate, and the computational costs are larger than for SGPR. KRR and SVR can also be attractive alternatives, and both can in some cases actually produce better results than GPR, but their performance was found to be more dependent on the specific case, e.g., training set size, while the tested GP models provide more stable results. SVR seems to be a particular interesting candidate for further investigations with large amounts of data. The computational costs for training are relatively low in our cases, and the accuracy is often comparable to GPR. It has the nice feature that it provides an intrinsic way to determine the important subset of the training data for the particular case. The drawback is that an intrinsic way for hyperparameter optimization is not available in standard SVR, and in a few cases, it was found to be less robust than the other tested methods. Note, however, that for SVR, there has been a recent work⁸² including a Bayesian statistics component, which may be an interesting future perspective.

For NNs, we obtained good results with 5 hidden layers and where the number of nodes is a multiple of 10 of the input dimension using the elu activation function.

Concerning the black box characteristic, the kernel based methods are superior, while NNs need more experimentation, but if one finds a working architecture, one can obtain good results, and further adjustments to improve the accuracy are straightforward. With respect to robustness, GPR and GPMC are in our study the methods of choice but computationally less attractive than the other methods for large data sets. Here, SVR, SGPR, and NNs are superior. If it is ensured that the noise-level in the training data is comparably small to the output values, one can advocate to use the tested sparse kernel based methods and preferably SVR or SGPR.

It was found that it is easier to predict the correlation energy with a sufficiently large accuracy than the total energy since the total energy is larger in magnitude and varying. Additionally, it is also

numerically challenging predicting corrections, which are small in magnitude. Therefore, one has to find the optimal choice for the size of the Δ to learn in a delta-learning scheme. For our intended use, we conclude that it is best and most economical to compute a RI-MP2 surface and correct it with a ML algorithm to the CCSD(F12*)(T) surface. The spread of the errors is slightly less if corrected from RI-MP2-F12, but then, numerical issues emerge in the ML algorithm and RI-MP2-F12 is significantly more costly than plain RI-MP2.

SUPPLEMENTARY MATERIAL

The [supplementary material](#) contains an overview about the generated datasets and instructions how to download them. Furthermore, additional benchmark data are given for other NN architectures.

ACKNOWLEDGMENTS

We thank the Villum foundation for financial support. In addition, O.C. acknowledges support from the Lundbeck Foundation, the Danish e-infrastructure Cooperation (DeiC), and the Danish Council for Independent Research through a Sapere Aude III (Grant No. DFF—4002-00015).

APPENDIX: Mathematical background of the tested Machine Learning algorithms

Before discussing the used ML algorithm, we introduce the kernel trick, which is a rather powerful tool to adapt algorithms for linear problems to be able to solve nonlinear problems. Afterwards, we focus on Gaussian Process Regression (GPR), variants of it, Kernel Ridge Regression (KRR), Support Vector Regression (SVR), and Neural Networks (NNs).

1. The kernel trick

An algorithm solely defined in terms of dot products in the input space can be lifted implicitly into a higher dimensional feature space by replacing the dot products by a kernel $k(\mathbf{x}, \mathbf{y})$. In this context, the term kernel refers to a generalization of a positive function or matrix. Let $\mathcal{X}$ be a nonempty index set. A symmetric function $\mathbf{K} : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ is called a positive definite kernel on $\mathcal{X}$ if

$$\sum_{i=1}^n \sum_{j=1}^n c_i c_j k(\mathbf{x}_i, \mathbf{x}_j) \geq 0 \quad (\text{A1})$$

holds for any $n \in \mathbb{N}$, $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathcal{X}$, $c_1, \dots, c_n \in \mathbb{R}$. The kernel trick is essentially based on Mercer's theorem,^{59,60} which states that if to a kernel k a linear operator on functions defined as

$$[T_k f](\mathbf{t}) = \int_a^b k(\mathbf{t}, \mathbf{s}) f(\mathbf{s}) d\mathbf{s} \quad (\text{A2})$$

can be associated, then the kernel can be represented as

$$k(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^{\infty} \lambda_j e_j(\mathbf{x}) e_j(\mathbf{y}), \quad (\text{A3})$$

using the eigenfunction e_j and eigenvalues λ_j of T_k . This expression is equivalent to a dot product in a different feature space using the

mapping $\phi(\mathbf{x})$,

$$k(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle. \quad (\text{A4})$$

Thus, an algorithm defined for linear problems expressed in dot products can be adapted to a nonlinear problem by substituting the dot products with a kernel. One can argue that the selection of a kernel is an empirical choice since one generally has no knowledge about how the kernel maps to the feature space, but even if a mapping would be known, one still does not know an exact relation between features and the learned function. Therefore, the selection of a suitable kernel for a certain problem will always be partly based on empirical knowledge. In this numerical study, we make use of a Matérn 5/2 kernel for the GP models,

$$k^{\text{M52}}(\mathbf{x}, \mathbf{x}'; l) = \sigma^2 \left(1 + \frac{\sqrt{5}|\mathbf{x} - \mathbf{x}'|}{l} + \frac{5|\mathbf{x} - \mathbf{x}'|^2}{3l^2} \right) \times \exp \left(-\frac{\sqrt{5}|\mathbf{x} - \mathbf{x}'|}{l} \right), \quad (\text{A5})$$

since in a previous study,³⁴ we found it superior to other choices such as the squared exponential kernel, which is often the standard choice. The parameters σ^2 and l are usually referred to as hyperparameters and are optimized for the current regression or classification problem. For our specific case, we use for each dimension of the input vector a different hyperparameter l to increase the flexibility and to account for a different strong dependence of the function value changes with respect to changes of specific orthogonal directions in the input space. To achieve this, the kernel is constructed as a sum of one dimensional Matérn 5/2 kernel,

$$k(\mathbf{x}, \mathbf{x}') = \sum_i^d k^{\text{M52}}(x_i, x_i'; l_i). \quad (\text{A6})$$

2. Gaussian process regression

Gaussian process regression (GPR), in some context also referred to as kriging,^{61–63} belongs to the field of machine learning algorithms. The procedure is based on Bayesian inference.⁸³ A Gaussian Process (GP) is a statistical distribution of functions, for which every finite linear combination of samples has a joint Gaussian distribution. Considering the representation of an arbitrary function $f(\mathbf{x})$, we introduce the vector $\mathbf{f} = (f(\mathbf{x}_1), f(\mathbf{x}_2), \dots, f(\mathbf{x}_n))$ containing the function values for data points and where $\mathbf{x} = (x_1, x_2, \dots, x_d)$ is also a vector of dimensionality d containing the coordinates of a given data point. It is assumed that the elements of $\mathbf{f}$ have a multivariate Gaussian distribution and the joint probability density is therefore given as

$$p(\mathbf{f}|\mathbf{m}, \mathbf{K}) = \frac{1}{(2\pi)^{n/2} |\mathbf{K}|^{1/2}} \exp \left(-\frac{1}{2} (\mathbf{f} - \mathbf{m})^T \mathbf{K}^{-1} (\mathbf{f} - \mathbf{m}) \right), \quad (\text{A7})$$

for which the short hand notation

$$\mathbf{f} \sim \mathcal{N}(\mathbf{m}, \mathbf{K}) \quad (\text{A8})$$

can be used. $\mathbf{K}$ is the covariance matrix, in which elements i, j are obtained from a kernel function $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. Following the widespread practice, we assume a zero mean function and introduce

a small constant noise term σ_n^2 , which can be seen as a regularization term,

$$\mathbf{f} \sim \mathcal{N}(\mathbf{0}, \mathbf{K} + \sigma_n^2 \mathbf{I}). \quad (\text{A9})$$

In order to predict unknown points for the function, it is exploited that the observed/training data $\mathbf{f}$ and the unknown data $\mathbf{f}_* = (f(\mathbf{x}_1^*), f(\mathbf{x}_2^*), \dots, f(\mathbf{x}_n^*))$ are distributed according to

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N} \left(\begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \begin{bmatrix} K(\mathbf{X}, \mathbf{X}) & K(\mathbf{X}, \mathbf{X}_*) \\ K(\mathbf{X}_*, \mathbf{X}) & K(\mathbf{X}_*, \mathbf{X}_*) \end{bmatrix} \right), \quad (\text{A10})$$

where $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n)$ is a vector of the n data points. This distribution is referred to as prior distribution since it only contains the prior assumption on how the data points are related. By conditioning the joint Gaussian prior distribution on the observations, one obtains the Bayesian conditional probability density (or posterior distribution),

$$\mathbf{f}_* | \mathbf{f} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}), \quad (\text{A11})$$

with posterior mean and covariance functions,

$$\boldsymbol{\mu} = K(\mathbf{X}_*, \mathbf{X}) (K(\mathbf{X}, \mathbf{X}) + \sigma_n^2 \mathbf{I})^{-1} \mathbf{f}, \quad (\text{A12})$$

$$\boldsymbol{\Sigma} = K(\mathbf{X}_*, \mathbf{X}_*) - K(\mathbf{X}_*, \mathbf{X}) K(\mathbf{X}, \mathbf{X})^{-1} K(\mathbf{X}, \mathbf{X}_*). \quad (\text{A13})$$

The posterior mean function is used for predictions of the function for unknown points, and the posterior covariance matrix provides a statistical error estimate for the prediction. The time dominating costs for GPR is the inversion of the covariance matrix $\mathbf{K}$ or equivalently solving the linear systems of equations,

$$(\mathbf{K} + \sigma_n^2 \mathbf{I}) \boldsymbol{\omega} = \mathbf{f}, \quad (\text{A14})$$

which can for positive definite matrices in general be done most efficiently with the aid of a Cholesky decomposition. In both cases, the costs scale as $\mathcal{O}(n^3)$. After obtaining the weights, the function can be expressed as

$$f(\mathbf{x}) \approx \sum_k \omega_k k(\mathbf{x}, \mathbf{x}_k). \quad (\text{A15})$$

One strength of GPR is that it has an intrinsic route to optimize the hyperparameters by maximizing the logarithm of the marginal likelihood,⁸³

$$\log p(\mathbf{f}|\mathbf{X}, \theta) = -\frac{1}{2} (\mathbf{f} - \mathbf{m})^T \mathbf{K}^{-1} (\mathbf{f} - \mathbf{m}) - \frac{1}{2} \log |\mathbf{K}| - \frac{n}{2} \log 2\pi. \quad (\text{A16})$$

An alternative is to use cross-validation, which we do not use for GPR in this study, but for KRR, which will be discussed in Subsection 3 of the [Appendix](#).

a. Sparse Gaussian process regression

The $\mathcal{O}(n^3)$ scaling prohibits direct application of GPR for cases with many ($>10^4$) data points, but there are numerous works on how to arrive at lower scaling formulations. One of these approaches is Sparse Gaussian Process Regression (SGPR) as described in Ref. 64. The main idea, such as in many approaches, is to use a smaller subset of the training set m with $m < n$. The computational costs

then scale with $\mathcal{O}(m^2 n)$ and therefore linearly with the training set size. The chosen m points are often referred to as inducing points. In contrast to other approximations based on subset selection, these points are together with the hyperparameters variationally selected by maximizing a lower bound to the exact marginal likelihood. Still, the number of inducing points has to be specified by the user and depending on the case, it might not be obvious how many inducing points are sufficient. Because the inducing points are variationally optimized in SGPR, often a random initial guess is sufficient. One can, however, also use clustering such as k-means clustering⁸⁴ to preselect m representative inducing points as the center of m clusters. For the current work, the second approach is used.

b. Gaussian process Markov chains

In the models discussed so far, inference has been done deterministically (solving the Bayesian inference problem by integration), which in most cases can only be exactly done using a Gaussian likelihood. If the inference is done using a stochastic approach (simulate several candidate solutions and evaluate the resulting integral while conditioning the probability density onto the observations using Monte-Carlo integration), other likelihoods can be also considered. One approach is to use Markov chain sampling giving rise to Gaussian process Markov Chains (GPMC).⁸⁵ GPMC has the strength that non-Gaussian likelihoods, where exact inference is not possible, could be used. Nevertheless, we restrict ourselves to Gaussian likelihoods for the current study to have a close as possible comparison with the other GPR variants. Furthermore, the exploration of other likelihoods would be a work on its own.

GPMC can be combined with the same approximations as SGPR to arrive at a sparse model (SGPMC) in which the inducing points are variationally optimized as described in Ref. 86.

3. Kernel ridge regression

Kernel ridge regression has a different background but is in its working equations very similar to GPR and gives the same predictive equations. The idea is to combine ridge regression,^{65,66} which is a linear least squares fit including a regularization term with the kernel trick. As in GPR, the function is approximated as

$$f(\mathbf{x}_i) = \sum_{j=1}^{N_{\text{train}}} \omega_j k(\mathbf{x}_i, \mathbf{x}_j), \quad (\text{A17})$$

where the weights are determined by minimizing

$$\min_{\omega} \left(\sum_j^{N_{\text{train}}} [f(\mathbf{x}_j) - f^{\text{ref}}(\mathbf{x}_j)]^2 + \sigma_n^2 \omega^T \mathbf{K} \omega \right), \quad (\text{A18})$$

which has the analytical solution

$$\omega = (\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1} \mathbf{f}^{\text{ref}}. \quad (\text{A19})$$

The predictive equations are thus identical to that of GPR, and if the same kernel and the same hyperparameters are chosen, both models give the same prediction. However, this is in general not the case. KRR has no intrinsic way to optimize the hyperparameters; therefore, they are usually selected by a grid search using cross-validation.

Since KRR and GPR are very similar, the terms are often used interchangeably. In some implementations, the hyperparameters for GPR are also optimized using cross-validation, but in this work, we make here a clear cut difference. If we refer to GPR, the hyperparameters were optimized by maximizing the marginal likelihood and if we use the term KRR, they are obtained via cross-validation.

4. Support vector regression

One problem with KRR is that the solution for the weights depends on all training inputs. Support Vector Regression (SVR) offers a sparse alternative, which was introduced by Vapnik and co-workers in 1996.⁶⁷ Due to pedagogical reasons, we will briefly summarize linear SVR and note that nonlinear SVR can be obtained using the kernel trick. A linear function fitting the data is expressed as

$$f(\mathbf{x}) = \langle \mathbf{x}, \boldsymbol{\omega} \rangle + b. \quad (\text{A20})$$

The idea of SVR is sketched in Fig. 12, where the special loss function

$$L_{\epsilon} = \begin{cases} 0, & \text{if } |y - f(\mathbf{x})| \leq \epsilon, \\ |y - f(\mathbf{x})| - \epsilon, & \text{otherwise,} \end{cases} \quad (\text{A21})$$

is introduced. Only fitting errors outside a ϵ tube contribute, and the weights $\boldsymbol{\omega}$ are optimized subject to this loss function. This constraint is usually loosened by introducing the so-called slack variables $\xi_i^{(+)}$ and $\xi_i^{(-)}$ to increase the flexibility. Regression errors for specific points up to the value of $\xi_i^{(+)}$ and $\xi_i^{(-)}$ can exist while still satisfying the required conditions. This can be expressed with the convex optimization problem,

$$\text{minimize} \quad \frac{1}{2} \|\boldsymbol{\omega}\|^2 + C \sum_{i=1}^N (\xi_i^{(+)} + \xi_i^{(-)}), \quad (\text{A22})$$

$$\text{subject to} \quad \begin{cases} y_i - \langle \boldsymbol{\omega}, \mathbf{x}_i \rangle - b \leq \epsilon + \xi_i^{(+)} \\ \langle \boldsymbol{\omega}, \mathbf{x}_i \rangle + b - y_i \leq \epsilon + \xi_i^{(+)} \\ \xi_i^{(+)}, \xi_i^{(-)} \geq 0, i = 1, \dots, n \end{cases}, \quad (\text{A23})$$

where C is the box constraint, a positive number which controls the penalty imposed on observations that lie outside a margin of ϵ . This constant is meant to adjust the regularization to prevent overfitting. Usually, the problem above is not solved directly, but

FIG. 12. Schematic illustration of a linear SVR using a soft margin.

instead a Lagrangian problem is formulated, which includes the constraints. The Lagrangian formulation allows the usage of the kernel trick as the optimization can be formulated in terms of dot products in that case. It can be shown^{67,87} that most Lagrange multipliers can be eliminated leaving only two Lagrange multipliers $\alpha_i^{(+)}$ and $\alpha_i^{(-)}$. One arrives at the following optimization problem of finding the Lagrangian:

$$L(\alpha^{(\pm)}) = \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n (\alpha_i^{(+)} - \alpha_i^{(-)}) (\alpha_j^{(+)} - \alpha_j^{(-)}) \langle \mathbf{x}_i, \mathbf{x}_j \rangle + \epsilon \sum_{i=1}^n (\alpha_i^{(+)} + \alpha_i^{(-)}) + \sum_{i=1}^n y_i (\alpha_i^{(-)} - \alpha_i^{(+)}), \quad (\text{A24})$$

where the Lagrange multipliers $\alpha_i^{(+)}$ and $\alpha_i^{(-)}$ need to fulfill

$$0 \leq \alpha_i^{(+)} \leq C, \quad (\text{A25})$$

$$0 \leq \alpha_i^{(-)} \leq C. \quad (\text{A26})$$

It turns out that the weights ω can be completely described as a linear combination of the observations in the training set as⁸⁷

$$\omega = \sum_{i=1}^n (\alpha_i^{(+)} - \alpha_i^{(-)}) \mathbf{x}_i. \quad (\text{A27})$$

Furthermore, a closer analysis shows that many of the terms $(\alpha_i^{(+)} - \alpha_i^{(-)})$ vanish, and thus, we only need to consider the input data in cases for which $(\alpha_i^{(+)} - \alpha_i^{(-)}) > 0$. This subset of size n_{sv} defines the support vectors, and we can express the function as

$$f(\mathbf{x}) = \sum_{i=1}^{n_{sv}} (\alpha_i^{(+)} - \alpha_i^{(-)}) \langle \mathbf{x}_i, \mathbf{x} \rangle. \quad (\text{A28})$$

Thus, by construction, a sparse predictor is obtained. The computational costs for SVR scale as $\mathcal{O}(n_{sv}^3)$. Since it is hard to predict in advance how many support vectors are needed, it is difficult to make a general statement on the computational costs. In the worst case scenario $\mathcal{O}(n^3)$ scaling of the computational cost is obtained,

which is comparable to that for GPR, but in the best case, we have linear scaling costs $\mathcal{O}(n)$ with respect to the number of data points in addition to the mentioned third-order scaling with n_{sv} . Therefore, SVR has the potential to be low scaling with respect to data points compared to conventional GPR and KRR. For a more detailed overview about SVR, we refer to Ref. 87. Analogous to GPR, we combine SVR with the kernel trick to use it as a nonlinear predictor.

5. Neural networks

We employ standard feed-forward neural network⁶⁸ consisting of $L + 2$ layers of the so-called *neurons* (sometimes also denoted *perceptrons*), where the result of evaluating the neurons in one layer is fed- or propagated-forward as input to the next layer, as depicted in Fig. 13. Each neuron evaluation consists of a linear transformation of the outputs from the previous layer with a set of weights, followed by a nonlinear activation through an activation function $f_k^{(l)}$. We can thus write the evaluation of neuron k in layer l as

$$h_k^{(l)} = f_k^{(l)}(\mathbf{w}_k^{(l)T} \mathbf{h}^{(l-1)}). \quad (\text{A29})$$

Assuming that each neuron in layer l is activated by the same activation function $f^{(l)}$, the evaluation of layer l can be compacted to a matrix expression,

$$\mathbf{h}^{(l)} = f^{(l)}(\mathbf{W}^{(l)T} \mathbf{h}^{(l-1)}). \quad (\text{A30})$$

Defining a loss function, the gradient corresponding to the weights can be calculated by the so-called back-propagation through the network. Using the loss function gradient, any gradient optimization algorithm can be used for optimizing the weights of the neural network, on a prespecified training set. The calculation of inputs to output is denoted as forward pass NN, while the calculation of the gradient is denoted as a backward pass. Doing a forward and a backward pass through the full training set is usually referred to as an epoch.

FIG. 13. Feed-forward neural network.

FIG. 14. Graphical comparison of selected activation functions. The selu activation function in (b) differs only from the elu in (a) by an input specific scaling factor.

There are numerous choices of activation functions. Some are inspired by biological neural networks to model the action potential firing in the cell. In its simplest form, it is a binary activation, but one can also use continuous functions. Figure 14 shows the representation of two such possible choices, namely, the exponential linear unit (elu) activation function,

$$f(x) = \begin{cases} \alpha(e^x - 1), & x < 0 \\ x, & x \geq 0, \end{cases} \quad (\text{A31})$$

which is what we apply in our NNs, and as a comparison, the sigmoid activation function

$$f(x) = \frac{e^x}{e^x + 1}, \quad (\text{A32})$$

which is of historical significance. Besides the elu function, we also make use of the scaled exponential unit (selu) function, which is a scaled elu function, where the scaling factor is determined from the input vector, and their shapes are thus similar.

REFERENCES

- ¹M. Rupp, *Int. J. Quantum Chem.* **115**, 1058 (2015).
- ²J. Cui and R. V. Krems, *Phys. Rev. Lett.* **115**, 073202 (2015).
- ³J. Behler and M. Parrinello, *Phys. Rev. Lett.* **98**, 146401 (2007).
- ⁴J. Behler, *Phys. Chem. Chem. Phys.* **13**, 17930 (2011).
- ⁵L. Mones, N. Bernstein, and G. Csányi, *J. Chem. Theory Comput.* **12**, 5100 (2016).
- ⁶B. Hartke, *J. Phys. Chem.* **97**, 9973 (1993).
- ⁷A. R. Oganov and C. W. Glass, *J. Chem. Phys.* **124**, 244704 (2006).
- ⁸T. C. Le and D. A. Winkler, *Chem. Rev.* **116**, 6107 (2016).
- ⁹K. T. Butler, D. W. Davies, H. Cartwright, O. Isayev, and A. Walsh, *Nature* **559**, 547 (2018).
- ¹⁰M. Rupp, O. A. von Lilienfeld, and K. Burke, *J. Chem. Phys.* **148**, 241401 (2018).
- ¹¹S. Carter, S. J. Culik, and J. M. Bowman, *J. Chem. Phys.* **107**, 10458 (1997).
- ¹²S. Carter, J. M. Bowman, and L. B. Harding, *Spectrochim. Acta, Part A* **53**, 1179 (1997), Special issue: *Ab initio* and *ab initio* derived force fields: State of the science.
- ¹³J. Kongsted and O. Christiansen, *J. Chem. Phys.* **125**, 124108 (2006).
- ¹⁴J. M. Bowman, S. Carter, and X. Huang, *Int. Rev. Phys. Chem.* **22**, 533 (2003).
- ¹⁵H. Rabitz and Ö. F. Aliş, *J. Math. Chem.* **25**, 197 (1999).
- ¹⁶M. Griebel, in *Foundations of Computational Mathematics (FoCM05)*, Santander, edited by L. Pardo, A. Pinkus, E. Suli, and M. Todd (Cambridge University Press, 2006), pp. 106–161.
- ¹⁷J. O. Jung and R. B. Gerber, *J. Chem. Phys.* **105**, 10332 (1996).
- ¹⁸K. Yagi, S. Hirata, and K. Hirao, *Theor. Chem. Acc.* **118**, 681 (2007).
- ¹⁹D. Toffoli, J. Kongsted, and O. Christiansen, *J. Chem. Phys.* **127**, 204106 (2007).
- ²⁰M. Sparta, D. Toffoli, and O. Christiansen, *Theor. Chem. Acc.* **123**, 413 (2009).
- ²¹G. Rauhut, *J. Chem. Phys.* **121**, 9313 (2004).
- ²²F. Richter, P. Carbonniere, A. Dargelos, and C. Pouchan, *J. Chem. Phys.* **136**, 224105 (2012).
- ²³Z. Xie and J. M. Bowman, *J. Chem. Theory Comput.* **6**, 26 (2010).
- ²⁴K. Hansen, G. Montavon, F. Biegler, S. Fazli, M. Rupp, M. Scheffler, O. A. von Lilienfeld, A. Tkatchenko, and K.-R. Müller, *J. Chem. Theory Comput.* **9**, 3404 (2013).
- ²⁵F. A. Faber, L. Hutchison, B. Huang, J. Gilmer, S. S. Schoenholz, G. E. Dahl, O. Vinyals, S. Kearnes, P. F. Riley, and O. A. von Lilienfeld, *J. Chem. Theory Comput.* **13**, 5255 (2017).
- ²⁶T. T. Nguyen, E. Székely, G. Imbalzano, J. Behler, G. Csányi, M. Ceriotti, A. W. Götz, and F. Paesani, *J. Chem. Phys.* **148**, 241725 (2018).
- ²⁷G. Schmitz, D. G. Artiukhin, and O. Christiansen, *J. Chem. Phys.* **150**, 131102 (2019).
- ²⁸A. P. Bartók, M. C. Payne, R. Kondor, and G. Csányi, *Phys. Rev. Lett.* **104**, 136403 (2010).
- ²⁹A. P. Bartók and G. Csányi, *Int. J. Quantum Chem.* **115**, 1051 (2015).
- ³⁰J. Cui and R. V. Krems, *J. Phys. B: At., Mol. Opt. Phys.* **49**, 224001 (2016).
- ³¹B. Kolb, P. Marshall, B. Zhao, B. Jiang, and H. Guo, *J. Phys. Chem. A* **121**, 2552 (2017).
- ³²J. P. Alborzpour, D. P. Tew, and S. Habershon, *J. Chem. Phys.* **145**, 174112 (2016).

- ³³J. Cui, Z. Li, and R. V. Krems, *J. Chem. Phys.* **143**, 154101 (2015).
- ³⁴G. Schmitz and O. Christiansen, *J. Chem. Phys.* **148**, 241704 (2018).
- ³⁵A. Denzel and J. Kästner, *J. Chem. Phys.* **148**, 094114 (2018).
- ³⁶J. Behler, *Angew. Chem., Int. Ed.* **56**, 12828 (2017).
- ³⁷J. Behler, *J. Chem. Phys.* **145**, 170901 (2016).
- ³⁸S. Manzhos and T. Carrington, *J. Chem. Phys.* **125**, 194105 (2006).
- ³⁹S. Manzhos and T. Carrington, *J. Chem. Phys.* **129**, 224104 (2008).
- ⁴⁰S. Manzhos, R. Dawes, and T. Carrington, *Int. J. Quantum Chem.* **115**, 1012 (2015).
- ⁴¹H. Wang and W. Yang, *J. Phys. Chem. Lett.* **9**, 3232 (2018).
- ⁴²V. Botu, R. Batra, J. Chapman, and R. Ramprasad, *J. Phys. Chem. C* **121**, 511 (2017).
- ⁴³S. Chmiela, H. E. Sauceda, K.-R. Müller, and A. Tkatchenko, *Nat. Commun.* **9**, 3887 (2018).
- ⁴⁴G. W. Richings and S. Habershon, *J. Chem. Phys.* **148**, 134116 (2018).
- ⁴⁵D. K. Duvenaud, H. Nickisch, and C. E. Rasmussen, in *Advances in Neural Information Processing Systems 24*, edited by J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger (Curran Associates, Inc., 2011), pp. 226–234.
- ⁴⁶M. Rupp, A. Tkatchenko, K.-R. Müller, and O. A. von Lilienfeld, *Phys. Rev. Lett.* **108**, 058301 (2012).
- ⁴⁷G. Montavon, M. Rupp, V. Gobre, A. Vazquez-Mayagoitia, K. Hansen, A. Tkatchenko, K.-R. Müller, and O. A. von Lilienfeld, *New J. Phys.* **15**, 095003 (2013).
- ⁴⁸R. Ramakrishnan, P. O. Dral, M. Rupp, and O. A. von Lilienfeld, *J. Chem. Theory Comput.* **11**, 2087 (2015).
- ⁴⁹C. W. Glass, A. R. Oganov, and N. Hansen, *Comput. Phys. Commun.* **175**, 713 (2006).
- ⁵⁰A. O. Lyakhov, A. R. Oganov, and M. Valle, *Comput. Phys. Commun.* **181**, 1623 (2010).
- ⁵¹A. O. Lyakhov, A. R. Oganov, H. T. Stokes, and Q. Zhu, *Comput. Phys. Commun.* **184**, 1172 (2013).
- ⁵²F. Koskowsky and B. Hartke, *J. Comput. Chem.* **26**, 1169 (2005).
- ⁵³R. Palanguntikul, R. Polly, and B. Hartke, *Phys. Chem. Chem. Phys.* **6**, 5456 (2004).
- ⁵⁴B. Helmich, M. Sierka, J. Dobler, and J. Sauer, *Phys. Chem. Chem. Phys.* **16**, 8441 (2014).
- ⁵⁵M. Sierka, *Prog. Surf. Sci.* **85**, 398 (2010).
- ⁵⁶X. Chen, M. S. Jorgensen, J. Li, and B. Hammer, *J. Chem. Theory Comput.* **14**, 3933 (2018).
- ⁵⁷S. A. Meldgaard, E. L. Kolsbjerg, and B. Hammer, *J. Chem. Phys.* **149**, 134104 (2018).
- ⁵⁸E. L. Kolsbjerg, A. A. Peterson, and B. Hammer, *Phys. Rev. B* **97**, 195424 (2018).
- ⁵⁹J. Mercer, *Philos. Trans. R. Soc., A* **209**, 415 (1909).
- ⁶⁰B. Schölkopf and A. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*, Adaptive Computation and Machine Learning (MIT Press, Cambridge, MA, USA, 2002), p. 644.
- ⁶¹T. L. Fletcher and P. L. A. Popelier, *J. Chem. Theory Comput.* **12**, 2742 (2016).
- ⁶²S. M. Kandathil, T. L. Fletcher, Y. Yuan, J. Knowles, and P. L. A. Popelier, *J. Comput. Chem.* **34**, 1850 (2013).
- ⁶³M. J. L. Mills and P. L. A. Popelier, *Theor. Chem. Acc.* **131**, 1137 (2012).
- ⁶⁴M. Titsias, in *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics*, Proceedings of Machine Learning Research Vol. 5, edited by D. van Dyk and M. Welling (PMLR, Hilton Clearwater Beach Resort, Clearwater Beach, FL, USA, 2009), pp. 567–574.
- ⁶⁵T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, Springer Series in Statistics (Springer New York, Inc., New York, NY, USA, 2001).
- ⁶⁶A. E. Hoerl and R. W. Kennard, *Technometrics* **12**, 55 (1970).
- ⁶⁷H. Drucker, C. J. C. Burges, L. Kaufman, A. Smola, and V. Vapnik, in *Proceedings of the 9th International Conference on Neural Information Processing Systems, NIPS'96* (MIT Press, Cambridge, MA, USA, 1996), pp. 155–161.
- ⁶⁸C. Bishop, *Neural Networks for Pattern Recognition* (Oxford University Press, USA, 1995).
- ⁶⁹R. Conte, C. Qu, and J. M. Bowman, *J. Chem. Theory Comput.* **11**, 1631 (2015).
- ⁷⁰A. P. Bartók, R. Kondor, and G. Csányi, *Phys. Rev. B* **87**, 184115 (2013).
- ⁷¹A. G. d. G. Matthews, M. van der Wilk, T. Nickson, K. Fujii, A. Boukouvalas, P. León-Villagrà, Z. Ghahramani, and J. Hensman, *J. Mach. Learn. Res.* **18**, 1 (2017).
- ⁷²F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, *J. Mach. Learn. Res.* **12**, 2825 (2011).
- ⁷³M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, 2015, software available from <http://www.tensorflow.org>.
- ⁷⁴See <http://www.turbomole.com> for TURBOMOLE V7.0 2015, a development of University of Karlsruhe and Forschungszentrum Karlsruhe GmbH, 1989–2007, TURBOMOLE GmbH, since 2007.
- ⁷⁵R. Ahlrichs, M. Bär, H. Horn, and C. Kölmel, *Chem. Phys. Lett.* **162**, 165 (1989).
- ⁷⁶C. Hättig and F. Weigend, *J. Chem. Phys.* **113**, 5154 (2000).
- ⁷⁷K. A. Peterson, T. B. Adler, and H.-J. Werner, *J. Chem. Phys.* **128**, 084102 (2008).
- ⁷⁸E. F. Valeev, *Chem. Phys. Lett.* **395**, 190 (2004).
- ⁷⁹D. J. Wales and J. P. K. Doye, *J. Phys. Chem. A* **101**, 5111 (1997).
- ⁸⁰See <http://www.scipy.org/> for SciPy: Open source scientific tools for Python.
- ⁸¹B. Jiang, J. Li, and H. Guo, *Int. Rev. Phys. Chem.* **35**, 479 (2016).
- ⁸²F. Wenzel, T. Galy-Fajou, M. Deutsch, and M. Kloft, in *Machine Learning and Knowledge Discovery in Databases*, edited by M. Ceci, J. Hollmén, L. Todorovski, C. Vens, and S. Džeroski (Springer International Publishing, Cham, 2017), pp. 307–322.
- ⁸³C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)* (The MIT Press, 2005).
- ⁸⁴S. Lloyd, *IEEE Trans. Inf. Theory* **28**, 129 (1982).
- ⁸⁵M. K. Titsias, M. Rattray, and N. D. Lawrence, “Markov chain Monte Carlo algorithms for Gaussian processes,” in *Bayesian Time Series Models*, edited by D. Barber, A. T. Cemgil, and S. Chiappa (Cambridge University Press, 2011), pp. 295–316.
- ⁸⁶J. Hensman, A. G. d. G. Matthews, and Z. Ghahramani, in Proceedings of AISTATS, 2015.
- ⁸⁷M. Awad and R. Khanna, “Support vector regression,” in *Efficient Learning Machines: Theories, Concepts, and Applications for Engineers and System Designers* (Apress, Berkeley, CA, 2015), pp. 67–80.