

University of Groningen

HOAX

Thie, Albert; Menger, Maximilian F.S.J.; Faraji, Shirin

Published in:
Molecular Physics

DOI:
[10.1080/00268976.2023.2172732](https://doi.org/10.1080/00268976.2023.2172732)

IMPORTANT NOTE: You are advised to consult the publisher's version (publisher's PDF) if you wish to cite from it. Please check the document version below.

Document Version
Publisher's PDF, also known as Version of record

Publication date:
2023

[Link to publication in University of Groningen/UMCG research database](#)

Citation for published version (APA):
Thie, A., Menger, M. F. S. J., & Faraji, S. (2023). HOAX: A hyperparameter optimisation algorithm explorer for neural networks. *Molecular Physics*, 121(9-10), Article e2172732.
<https://doi.org/10.1080/00268976.2023.2172732>

Copyright

Other than for strictly personal use, it is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license (like Creative Commons).

The publication may also be distributed here under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license. More information can be found on the University of Groningen website: <https://www.rug.nl/library/open-access/self-archiving-pure/taverne-amendment>.

Take-down policy

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.

Downloaded from the University of Groningen/UMCG research database (Pure): <http://www.rug.nl/research/portal>. For technical reasons the number of authors shown on this cover page is limited to 10 maximum.



Molecular Physics

An International Journal at the Interface Between Chemistry and Physics

ISSN: (Print) (Online) Journal homepage: <https://www.tandfonline.com/loi/tmph20>


HOAX: a hyperparameter optimisation algorithm explorer for neural networks

Albert Thie, Maximilian F.S.J. Menger & Shirin Faraji

To cite this article: Albert Thie, Maximilian F.S.J. Menger & Shirin Faraji (2023) HOAX: a hyperparameter optimisation algorithm explorer for neural networks, Molecular Physics, 121:9-10, e2172732, DOI: [10.1080/00268976.2023.2172732](https://doi.org/10.1080/00268976.2023.2172732)

To link to this article: <https://doi.org/10.1080/00268976.2023.2172732>

 View supplementary material 

 Published online: 02 Feb 2023.

 Submit your article to this journal 

 Article views: 73

 View related articles 

 View Crossmark data 

HOAX: a hyperparameter optimisation algorithm explorer for neural networks

Albert Thie , Maximilian F.S.J. Menger  and Shirin Faraji 

Zernike Institute for Advanced Materials, Faculty of Science and Engineering, University of Groningen, Groningen, The Netherlands

ABSTRACT

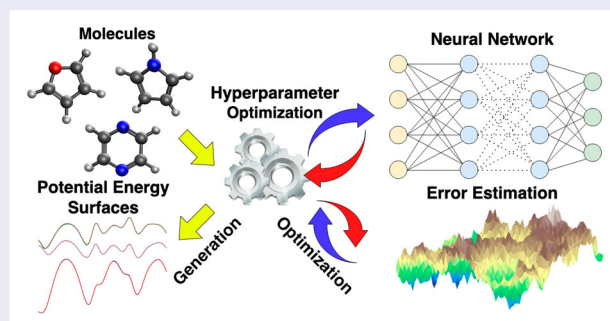
Computational chemistry has become an important tool to predict and understand molecular properties and reactions. Even though recent years have seen a significant growth in new algorithms and computational methods that speed up quantum chemical calculations, the bottleneck for trajectory-based methods to study photo-induced processes is still the huge number of electronic structure calculations. In this work, we present an innovative solution, in which the amount of electronic structure calculations is drastically reduced, by employing machine learning algorithms and methods borrowed from the realm of artificial intelligence. However, applying these algorithms effectively requires finding optimal hyperparameters, which remains a challenge itself. Here we present an automated user-friendly framework, HOAX, to perform the hyperparameter optimisation for neural networks, which bypasses the need for a lengthy manual process. The neural network-generated potential energy surfaces (PESs) reduce the computational costs compared to the ab initio-based PESs. We perform a comparative investigation on the performance of different hyperparameter optimization algorithms, namely grid search, simulated annealing, genetic algorithm, and Bayesian optimizer in finding the optimal hyperparameters necessary for constructing the well-performing neural network in order to fit the PESs of small organic molecules. Our results show that this automated toolkit not only facilitate a straightforward way to perform the hyperparameter optimisation but also the resulting neural networks-based generated PESs are in reasonable agreement with the ab initio-based PESs.

ARTICLE HISTORY

Received 20 June 2022
Accepted 19 January 2023

KEYWORDS

Quantum chemistry;
machine learning; neural
networks; hyperparameter
optimisation






1. Introduction

The application of machine learning (ML) algorithms has grown to encompass many new fields, such as data analysis of voting records [1], enhancing the motor capabilities in robotics [2] and even in creating fictional stories [3]. In the field of electronic structure theory, ML algorithms have also seen an increased range of applications. ML is applied to approximate density functionals [4], generating the ground-[5, 6] and excited-state potential energy surfaces (PESs) [7, 8]. A major challenge in applying

any ML model is choosing a suitable ML algorithm and the corresponding optimal hyperparameters [9, 10] (see Figure 1), for which various optimisation algorithms exist [11].

Currently, neural networks are among the most commonly applied ML algorithms in computational chemistry [10]. Most of the applications are centred around describing the ground-state molecular properties. For example, SchNet uses a deep learning architecture that is applied to the QM9 database [5], which consists of 134 k

CONTACT Shirin Faraji  s.s.faraji.rug.nl  Zernike Institute for Advanced Materials, Faculty of Science and Engineering, University of Groningen, Nijenborgh 4, Groningen 9747AG, The Netherlands

 Supplemental data for this article can be accessed here. <https://doi.org/10.1080/00268976.2023.2172732>

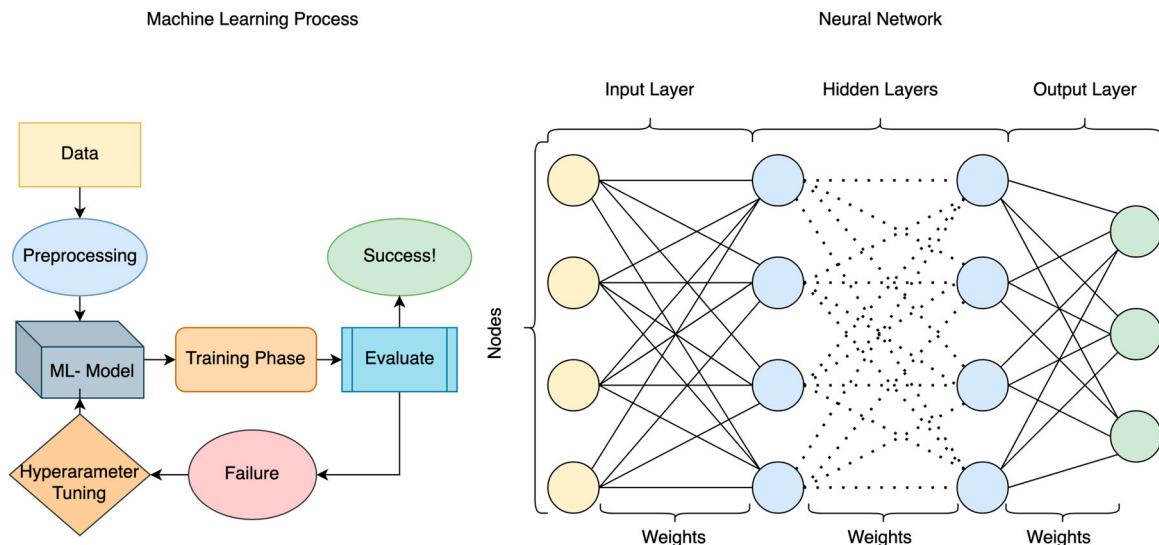


Figure 1. Systematic overview of the process of training a ML model (left) and a neural network architecture (right).

stable small organic molecules made of C, H, O, N, and F atoms [12], to predict a wide range of molecular properties across chemical space. PauliNet uses deep neural network quantum Monte Carlo-based approach that is capable to achieve nearly exact solutions of the electronic Schrödinger equation [13]. Other examples include systems that use locality to restrict the input space for the neural network [14], a hierarchically interacting particle neural network which learns to transform the input space into terms describing their interactions [15] or networks explicitly designed to be covariant [16]. Message-passing networks are another successful approach that uses graph neural networks to make accurate predictions of the quantum mechanical properties of small organic molecules [17–19]. Other studies have used kernel-based ML methods, such as kernel-based regression methods to predict various molecular properties, such as PES(s), normal modes, internal energy, and heat capacity [20–23].

For excited-state dynamics, the SchNarc [24] approach combines SchNet [5] and the non-adiabatic dynamics package SHARC [25]. This results in a deep learning approach that can run excited-state dynamics simulations based on the learned important properties, such as non-adiabatic couplings, gradients, Hessians, and spin-orbit couplings, to further simplify such simulations. Neural networks have also been used to generate surfaces for the treatment of the excited states of Formaldehyde [26], long-term simulation of excited-states dynamics replacing force field methods [27], and using kernel ridge regression to provide properties for decoherence-corrected fewest switches surface hopping [8]. A good overview of the recent works in this field can be found in Ref. [28].

Although these groundbreaking works have illustrated the potential of ML algorithms to accelerate and improve molecular simulations, a major challenge from the user perspective, besides accurate and comprehensive training data and the corresponding computational costs, is finding the suitable ML algorithm and the associated hyperparameters out of the wide range of existing ML methods. Kernel-based ML methods have the advantage that the hyperparameters can be determined a priori, using the descriptors without any reference to the target property [29]. Although this remains a challenge for neural network modelling, it has been shown in a comparative research that neural networks can produce better results, given the optimal hyperparameters are found [30]. The open-source PES-Learn package uses the hyperparameter optimizer (HyperOpt) to find the hyperparameters for neural networks and Gaussian processes [31, 32]. To the best of our knowledge, comparing hyperparameter optimisation methods to find optimal hyperparameters to construct a reliable neural network has not been investigated, which motivates the idea of introducing an automated procedure for hyperparameter optimisation that is the main focus of the present work.

In this work, we investigate the performance of neural networks in predicting the PESs of a set of small organic molecules, namely, SO_2 , Pyrazine, Pyrrole, and Furan. We investigate the performance of various derivative-free hyperparameter optimisation schemes [33], as we cannot directly compute a gradient for the hyperparameter optimisation. We, therefore, leave out gradient-dependent optimisation algorithms such as gradient descent [34]. We perform a cross-comparison of different optimisation methods, namely grid search (GS) [35], simulated annealing (SA) [36] genetic algorithms (GA) [37, 38] and

Bayesian optimisation (BO) [39], to explore the relationship between the performance of neural networks and the used optimisation technique. We also investigate the relationship between the molecule that is modelled and the corresponding hyperparameters, as well as the size of the training data.

This paper is organised as follows; we first discuss the structure of the neural networks and the hyperparameters that are employed. Secondly, we describe the hyperparameter optimisation algorithms that are used to find the optimal neural network models. Thirdly, we compare the results of hyperparameter optimizers and their respective neural network models in constructing the PESs of SO₂, Pyrazine, Pyrrole, and Furan. Finally, we discuss the effectiveness and possible extension of the package.

2. Theory

2.1. Neural networks

Neural networks are comprised of layers of independent discriminators, named nodes [40], containing an input layer, one or more hidden layers, and an output layer. Each node connects to other nodes and has an associated weight (see Figure 1). These weights are updated by training the neural network. The performance of a neural network is determined by the hyperparameters of the network, e.g. the number of nodes per layer, the number of hidden layers, the learning rate to only name a few, which are typically preselected by the user. In this work, we present the hyperparameter optimisation using a set of fixed and flexible hyperparameters. The fixed hyperparameters can be set by the user in the configuration file and are unchanged throughout the hyperparameter optimisation process. For the flexible hyperparameters, the user defines a range in which the hyperparameter optimisation algorithm looks for the optimal hyperparameters. The fixed and flexible hyperparameters that are used in this work can be found in Table 1 and are further described in the following.

An important hyperparameter is the activation function, which is applied to the activation a of every node. One can write the activation a of a node as the sum of

Table 2. Overview of the different activation functions available in HOAX.

Name	Formula
Sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$
ReLu	$ReLU(x) = \max(0, x)$
TanH	$TanH(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

Activation functions are applied to the output of each node in the neural network.

the inputs x_i for the node with their respective weights w_i , together with the bias β of the node, as seen in the following equation

$$a = \sum w_i x_i + \beta \quad (1)$$

The most commonly used activation functions are currently supported by the package (see Table 2).

Another important hyperparameter is the loss function, which determines how the error of a neural network is calculated. The error is the difference between the output of the neural network and the training (reference) output, for the given training input. In this research, the error is the difference between the energy in the database and the output of the network for each set of coordinates. Implemented in the package are the mean absolute error (MAE), $\frac{1}{N} \sum_i |E_i|$, the mean squared error (MSE), $\frac{1}{N} \sum_i E_i^2$, and the root mean squared error (RMSE), $\sqrt{\frac{1}{N} \sum_i E_i^2}$ as different loss functions. Previous works on predicting PESs using neural networks used the MAE or MSE as both the measure when reporting the results on the reference data and in training the neural network [5, 7, 15]. In this work, we use the MSE for training the neural network, while using the RMSE and MAE for the validation step.

The neural network internal optimizer is the algorithm that modifies the neural network weights after receiving the error from the loss function. The HOAX package currently supports two neural network optimizers; (i) the stochastic gradient descent (SGD) optimizer and (ii) the adaptive moment estimation (ADAM) optimizer [41]. The SGD uses the gradient of the error multiplied by the learning rate parameter to calculate the changes for the weights connected to the output layer of the network. It then iteratively adjusts the weights in each previous layer, calculating the error of each individual node using backpropagation [40]. The ADAM optimizer [41] on the other hand changes the weights in the network using an adaptive learning rate, depending on the amount of previous changes for individual weights. It also looks at the average of the gradients of a weight over time and uses this average to change the weights. It

Table 1. Overview of the fixed and flexible hyperparameters used in training the neural networks.

Fixed	Flexible
Activation function	Number of layers
Loss function	Number of nodes
Internal optimizer	Learning rate
Epoch number	Batch size

can therefore deal with more noisy and sparse data and provides better results and training speed than the SGD optimizer [41].

The epoch number is the maximum number of training cycles performed. An upper bound is required to constrain the training time when the network has not converged to a sufficient minimal error within the pre-defined training cycles.

The flexible parameters in our package are the number of layers, the number of nodes within each layer, the learning rate, and the batch size in the neural network. The learning rate controls the pace at which the optimizer updates the weights with respect to the loss function. It defines how quickly the neural network updates the weights. The batch size parameter refers to the amount of training examples that are used per training cycle. A batch number of 1 indicates the weights of the model which are updated after each training example. A higher number indicates the amount of training examples for generating an average error, which is used to update the weights.

2.2. Hyperparameter optimisation

In order to approach the problem of using the right hyperparameters, we can define the problem as optimising an objective function [42]. An objective function assumes that we can approach some function $f(x)$ by some approximate function $\tilde{f}(x)$. We can determine the $\tilde{f}(x)$ by looking at the difference between the output of the $\tilde{f}(x)$ and $f(x)$, for some $x \in X$. Here X is the range that is applied to the approximate function. We call the difference between $f(x)$ and $\tilde{f}(x)$ the error, E . The aim then is to minimise the E of the $\tilde{f}(x)$. In physics, we can think of this as attempting to minimise an energy function, while in ML models this is also called minimising the cost function [9]. While a loss function is for a single training input, a cost function, on the other hand, is the average loss over the entire training data set. It should be noted that any ML algorithm is an optimisation function in itself [43]. This means a hyperparameter optimisation scheme in this case can be described as applying the objective function $\tilde{f}(x)$ on top of the ML objective function $\tilde{g}(x)$, leading to $\tilde{f}(\tilde{g}(x))$.

Approaching the hyperparameter optimisation for a neural network in such a fashion has two clear advantages. For one, we can apply existing optimisation techniques from multiple domains, such as physics in the case of the SA [36] and artificial intelligence in the case of the GA [44], BO [39], and GS [35]. The second advantage is that we can split the hyperparameter optimisation process from the training process of the ML model. This allows us to use any hyperparameter optimisation with

any ML technique, as we only apply the hyperparameter optimisation to the result of the cost function. Combining these two advantages, we can create a package that can optimise any ML model given a clear cost function.

By implementing various hyperparameter optimisation schemes a user is able to generate better end results from the neural network. Currently, HOAX supports five hyperparameter optimizers, namely the RS(Random Search), GS, SA, GA, and BO.

In the RS algorithm, a random position in the hyperparameter space is selected for the neural network to be trained. After the training is completed, the algorithm moves to a new random position. There is no guiding principle behind these moves, other than the type of random function that is called. However, research has shown that the RS can perform equally well in comparison to other methods in some error landscapes and can do so in less computational time [45]. It is therefore a good inclusion as a baseline function.

In the GS algorithm [46], the set of n hyperparameters are put in a n dimensional grid. The algorithm defines the grid by adding a step value for each hyperparameter. This step value determines the rate for each hyperparameter change. The GS algorithm is an exhaustive search method. This leads to high computational time for a high number of hyperparameters. However, more advanced options for the GS exist, such as the constraining GS [46], to partially address this issue.

In the SA algorithm (see Figure 2), which is inspired by the annealing procedure of the metal working [36], the number of hyperparameters is again divided in a n dimensional grid. However, the algorithm will not visit the entire hyperparameter space in this grid. Therefore, one can choose a larger grid or additional hyperparameters. One starts the algorithm randomly in the hyperparameter space, within which the neural network is created and trained on the training set and is validated with the validation set that gives the minimum error. Then, it moves to a random neighbouring hyperparameter state. For each hyperparameter, it is randomly decided if the parameter is moved or not. If the parameter is moved, the direction is randomly chosen to be either a one-step increase or decrease. Using these new hyperparameters we train a new neural network on the training set that is again validated with the validation set that gives a new minimum error. This new state is accepted if the new error, E_{new} , is lower than the previous one, E_{old} . The temperature, T , is a parameter in the SA that affects the distance of a next hyperparameter space from the current state and also the probability of accepting the state with a higher objective function value. As the T decreases, the SA reduces the extent of its search to converge to a minimum. There is a probability, $P(A)$, to accept a higher error

state depending on the T , and the number of iterations m , given by the following equation.

$$P(A) = \exp\left(\frac{-E_{\text{new}} - E_{\text{old}}}{T/m}\right) \quad (2)$$

The number of iterations m determines the chance to accept higher error states, which decreases as m increases. The SA algorithm explores a large search space initially, while gradually reducing the search space to find a minimum. As it is possible that the predefined grid does not include the best hyperparameters, the algorithm also uses an ‘absorbing’ boundary [47]. Any time the algorithm reaches one of the hyperparameter boundaries of the grid, it has a certain chance (defined by the user) to remain at the boundary. This is done so that more time can be used to explore the other hyperparameters which have not reached the boundary.

In the GA algorithm (see Figure 2), which is inspired by the theory of evolution [48] one uses a ‘gene pool’ of six neural networks with random hyperparameters. After training, each of the neural networks is ranked by its error. The two networks with the highest error are removed from the pool. The two with the lowest error become the parent networks for the next generation and are used to generate two new offspring neural networks. To generate the new networks the value of each hyperparameter is converted to a bit-string. A point in this bit-string is selected, called the crossover point. Two new offspring bit-strings are created by swapping part of the parent bit-strings, once before and once after the crossover point for each hyperparameter. For each bit in the bit-string, there is a chance to mutate. The bit will then change from one to zero or vice versa. This rate depends on the bit length of the hyperparameter, which is shown to give improved results [49, 50]. The process

is repeated with the new ‘gene pool’ until a predefined minimum error or a maximum number of iterations is reached.

In the BO algorithm (see Figure 2), the Bayesian statistics [51, 52] is used to create a probabilistic model able to find the optimal set of hyperparameters. This model is created by sampling points in the hyperparameter space and updating the probabilistic model. The model then uses the statistic distribution to determine the next sampling point. The model consists of two components, the prior function and the acquisition function. The type of prior function determines how the probabilistic model treats uncertainty and the acquisition function determines how the BO decides which point should be sampled next. In this work, we use Gaussian priors and upper confidence bounds (UCB) for the prior function and acquisition functions, respectively [39].

3. Code infrastructure

HOAX (hyperparameter optimisation algorithm explorer) is an extendable open-source Python package that automates the hyperparameter optimisation search for the application of neural network models. HOAX is an extension of the PySurf [53] package, but can also be used as a standalone package. The package automates the process of finding hyperparameters which are typically done manually by the user. The user provides a database with training and validation data in the network common data form (NetCDF) or the hierarchical data format version 5 (HDF5) [54, 55], which can be generated by PySurf. The user also provides a configuration file in the JSON (javaScript object notation) format [56]. This file contains the configuration for processing the database, constructing the neural networks, initialising

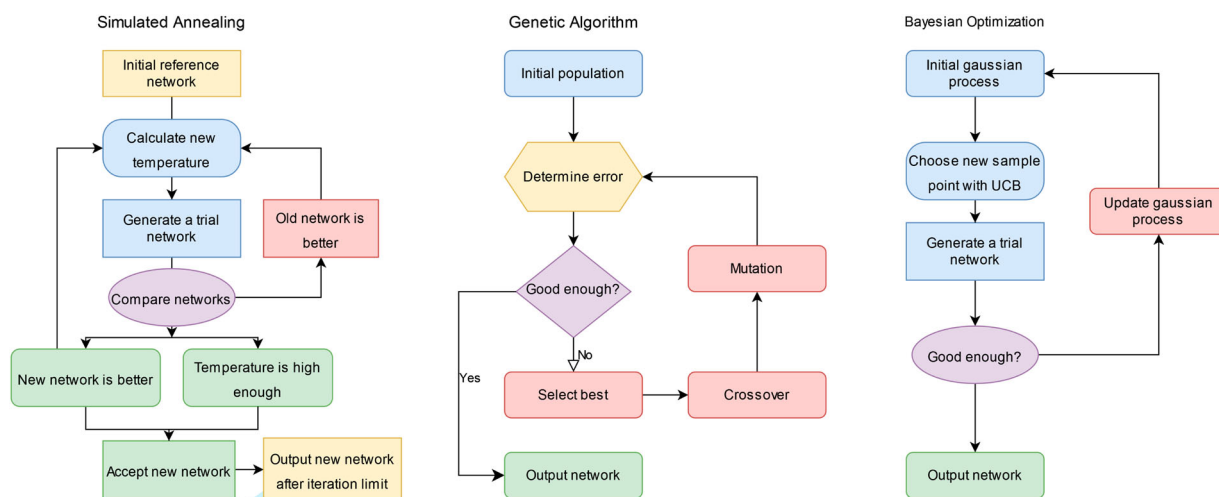


Figure 2. Schematic overview of the SA algorithm (left), the GA algorithm (middle), and the BO algorithm (right).

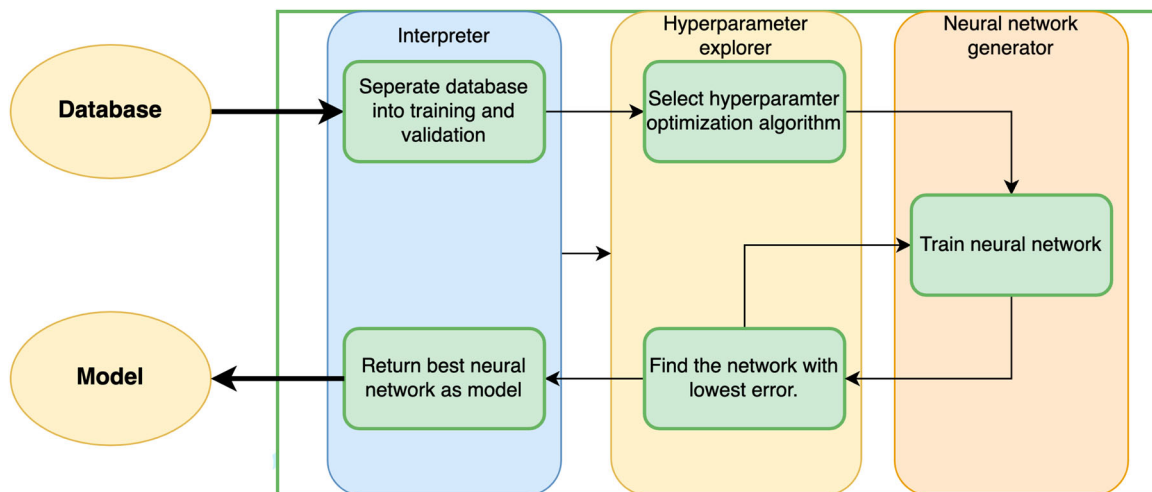


Figure 3. Schematic overview of the HOAX package. The package is divided into three parts, the interpreter, the neural network generator, and the hyperparameter explorer.

the fixed hyperparameters, and choosing the hyperparameter optimisation algorithm. The package is divided into three parts; (i) the interpreter, (ii) the neural network generator, and (iii) the hyperparameter explorer. A schematic overview of the HOAX can be seen in Figure 3.

The interpreter reads the database that consists of the atom coordinates as input and energies as output. The package contains a module to translate Cartesian coordinates to internal coordinates when this is specified in the configuration file. The interpreter then separates the data into training and validation data sets, according to the user-specified ratio. The training data set is used to train the neural networks, while the validation data is used to measure the performance of the neural network during training.

The neural network generator initialises the neural networks, which is implemented by providing an interface to the PyTorch library [57]. Through this interface, the generator creates neural networks based on the specifications provided by the user in the configuration file. The generator trains the neural network with the hyperparameters provided by the hyperparameter optimisation algorithm. When a cycle of training has finished, the generator provides a summary of the performance throughout the training, measured using the validation data. It also provides a fully trained neural network.

The hyperparameter explorer uses the results provided by the neural network generator to find new hyperparameters. To select a new set of hyperparameters, it uses an optimisation algorithm. Currently implemented algorithms are the RS, GS, SA, and BO algorithms. The BO algorithm is implemented using the Bayesian optimisation python package [58]. Each of these algorithms can be chosen to provide hyperparameters to the neural network generator. The neural network generator

then performs the training sequence again with the new hyperparameters and provides new performance data to the hyperparameter explorer. In this way the process of finding hyperparameters that provides a good fit on the provided data is automated. The package also includes a cross-validation module, which can be used to perform X-fold cross-validation on individual hyperparameter configurations found. It can also be used for cross-validation during the training phase while using any of the hyperparameter optimizers. The HOAX package can be found at <https://github.com/AlbertThie/HOAX>.

4. Computational details

To investigate the effect of the hyperparameter optimizers on neural networks, we used a set of trajectories of SO₂, Pyrazine (C₄H₄N₂) [59], Furan (C₄H₄O), and Pyrrole (C₄H₅N) [60] as a training data set. This training data set contains the atomic positions of the molecules as training input and the corresponding ground- and two excited-states energies as training output. Training data were created using the PySurf [53] implementation of the Landau–Zener surface hopping simulations [61] to propagate trajectories from initial starting conditions generated by a Wigner sampling [62] based on an initial optimised geometry.

For the ab initio calculations, different electronic structure methods were used for the different molecules, to maximise the number of useful data points and minimise the computational cost. All electronic structure calculations were done using the Q-chem 5.4 package [63]. For SO₂, time-dependent density functional theory (TDDFT) was used with the B3LYP functional [64] and the 6-31G* basis set [65]. For Pyrazine, we applied TDDFT at the PBE0/cc-pVDZ level of theory. For

Pyrrole and Furan, the spin-flip variant of TDDFT [66] was applied using the BHHLYP functional [67] and the cc-pVDZ basis set [68].

For each molecule, 100 trajectories were propagated for 100 fs with 0.5 fs timestep. For SO₂ and Pyrazine, the first three singlet states (S_0 , S_1 , and S_2) were calculated at each point of the trajectory. For Pyrrole and Furan, six states were calculated, and the first three singlet states (the ground and two singlet excited states) were selected based on their spin multiplicity (S^2). Each of these calculations yielded a database for the training of at most 20,000 data points. However, not all of the 100 trajectories converged, due to the SCF not converging, and therefore did not make the maximum of 200 data points per trajectory possible for all molecules. For SO₂, a database of 1541 data points was generated, while for Pyrazine the database contains 20,000 data points. For Pyrrole and Furan, 19,204 and 14,529 data points were generated, respectively. Therefore, not only the fitting and optimisation algorithms are tested on different molecular systems, but also for different levels of theories and the different sizes of the training data set.

All neural network calculations were performed with the ADAM optimizer [41], Tanh activation function (see Table 2), MSE as error function, and each training session was fixed to last at most 10,000 epochs. During these training sessions, 10% of training data was not used for training, but used as validation data. This validation data was presented to the network at intervals of 100 epochs. The best-performing iteration of the neural network during the 10,000 epochs was saved and exported by the package as the best-performing model. This was done to prevent overfitting and provide a realistic estimate of the performance of the network on the reference data

points (see Figure 3 of the Supporting Information). The reference data points correspond to the coordinates and energies obtained from an independent trajectory with exactly the same initial condition purely based on ab initio calculations without any support of the database and are neither included in the training nor in the validation set. All networks were trained on a data set containing internal coordinates as input and energies as output.

5. Results and discussion

To provide a working prototype, a neural network was trained on the data set of SO₂, using the SA hyperparameter optimizer. The hyperparameters were initially varied by hand and later explored by the SA optimizer. A network with 150 nodes per layer, nine hidden layers, a learning rate of 0.0001, and a batch size of 32 performed best on the data set. This network produced a RMSE of 0.017 eV on the validation set. The performance compared with the reference trajectory can be seen in Figure 4. The results on SO₂ show that the neural network implementation was successful in learning ground- and excited-state PESs. To test the automated cross-validation module, we have performed 10-fold cross-validation on the data set of SO₂ using the SA hyperparameter optimizer. The resulting best-performed neural networks, which are composed of 150 nodes per layer, eight hidden layers, a learning rate of 0.0001, and the batch size of 32, resulted in a RMSE of 0.013 eV on the validation sets. This new neural network architecture differs only in the number of hidden layers compared to the neural networks where no cross-validation was performed (8 vs. 9 hidden layers), and produces a slightly smaller RMSE (0.013 eV vs. 0.017).

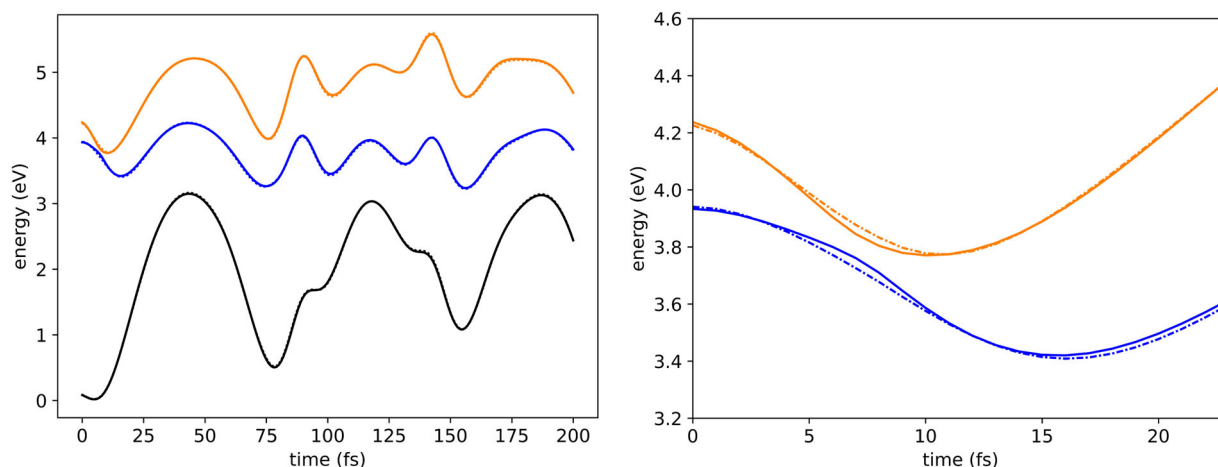


Figure 4. Comparison of neural network-generated PESs (dashed), for the best network found by the SA algorithm for SO₂ and reference ab initio-based PESs (solid). S_0 (black); S_1 (blue); S_2 (orange). A zoomed-in picture is provided on the right, showing the areas where the S_1 and S_2 surfaces are close.

To explore the relationship between the performance of the neural network and the hyperparameter optimisation techniques, we performed a cross-comparison of the GS, SA, GA, and BO algorithms on Pyrazine, Pyrrole, and Furan. We also investigate the relationship between the molecule that is modelled, the corresponding hyperparameters, as well as the size of the training data.

To study the performance of the GS, the flexible parameters were limited to only two, i.e. the number of nodes and the number of layers. This was done to prevent an exhaustive search of all hyperparameters, which is computationally not feasible. Data was accumulated using the GS with 10 to 90 nodes and 5 to 9 hidden layers. The training was performed with a batch size of 256 and a learning rate of 0.0001. 10% of the data points were randomly chosen as validation data points and therefore not used in training. The results of the GS algorithm are shown in Figure 5. In each point, the lowest error on the validation set is plotted. The slopes of the error descend towards the RMSE of < 0.025 eV. The exception is the neural networks trained for Furan on half of

the data set. Here the minimum error is 0.034 eV. This might be attributed to the low amount of data points available for Furan, leading to a training data set consisting of 6538 data points where the 10% validation data set is not included. However, the model performs well for the Furan full data set. From these results, we can see that a higher number of layers and nodes has a positive effect on the error, while the effect of the former seems to be more pronounced. This can be clearly seen for the Pyrrole full data set results, where a network with 10 nodes and 9 layers can produce an error of < 0.025 eV. It can also be seen that the minimum size of the network differs between molecules, where a more complex network is necessary for Pyrazine and Pyrrole compared to Furan. A possible explanation could be that Furan possesses less atoms and therefore it requires less complex modelling. Another explanation could be that the initial parameters selected for the GS algorithm are closer to the optimal parameters for Furan.

Extending the GS optimizer beyond node and layer parameters is computationally costly, as the cost increases

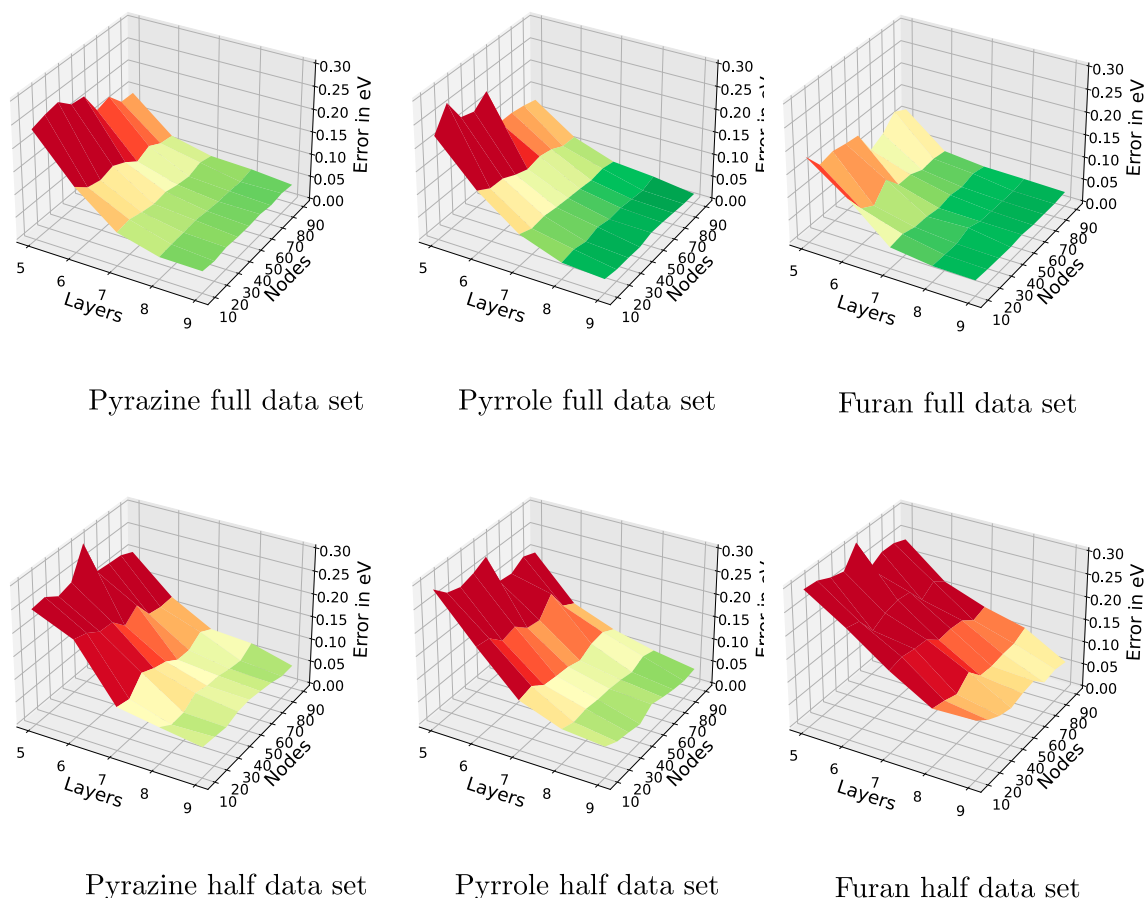


Figure 5. Results of performing the GS between the bounds of 5–9 layers and 10–90 nodes on each molecule. The upper figures show the result for a full data set. The bottom shows the results of using only half the data set. Red coloured areas indicate a high RMSE while green areas indicate a low RMSE.

exponentially with respect to the number of hyperparameters. To this end, and to avoid a full scan of the hyperparameter surface, the metaheuristic algorithms SA, GA, and BO are used [36, 39, 44]. Using these hyperparameter optimizers, the parameter search space is increased without increasing the computational costs. This allowed us to include four hyperparameters, i.e. the number of layers and nodes, learning rate, and batch size, as presented in Tables 3–5.

The SA and BO results in Tables 3 and 5 were reached after 20–30 iterations, while the results for the GA in Table 4 were taken after 10–15 iterations. The slightly lower iterations for the GA are attributed to the fact that the GA will generate two new networks at each iteration, while SA and BO will generate one. On average the required computational time between the SA, GA, and BO is comparable, as the computational time depends on the training of the neural network. The computational time also depends on the hyperparameters of the neural network and the number of data points. A larger network increases computational time, while using a halved data set reduces the computational time. For all optimizers, 144 hours of training on 4 Intel(R) Xeon(R) CPU E5-2695(v4) with 2.10 GHz generated the

results. All optimizers outperformed the GS in reaching a lower RMSE. All optimizers need a different configuration of hyperparameters to reach the lowest RMSE. This could indicate that the error landscape contains many local minima. It is also the case that the hyperparameters themselves are correlated. For example, the learning rate found for the neural networks for Pyrrole by the GA in Table 4 is higher than the one found by SA in Table 3. However, the batch size is much higher, meaning the learning updates are performed much less frequently than in the SA algorithm. In a similar fashion, the Furan model found by the SA has a higher number of layers but a smaller number of nodes than the one found by the GA. We can therefore see that for different molecules, totally different hyperparameters are necessary to reach a well-performing network. This is in fact related to the dependency of the hyperparameters to the target property for the neural network methods and acts as a limiting factor in the transferability of the optimal hyperparameters to new molecules. However, finding similarities for the bounds of the hyperparameter search area for different molecules helps in narrowing these search bounds and thus saves time in finding the best hyperparameters for new molecules with comparable size and complexity. We

Table 3. Hyperparameter configurations for the simulated annealing optimizer.

Simulated annealing results						
	Pyrazine	Pyrazine Half	Furan	Furan Half	Pyrrole	Pyrrole Half
Number of nodes	120	60	140	150	60	40
Number of layers	4	5	19	13	6	11
Learning rate	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001
Batch size	64	32	128	128	32	32
RMSE in eV	0.023	0.029	0.020	0.015	0.016	0.029
MAE in eV	0.013	0.015	0.012	0.010	0.010	0.015

Each column shows the lowest error found on the data set and the hyperparameters used in the model.

Table 4. Hyperparameter configurations for genetic algorithm optimizer.

Genetic algorithm results						
	Pyrazine	Pyrazine Half	Furan	Furan Half	Pyrrole	Pyrrole Half
Number of nodes	80	100	180	180	70	160
Number of layers	12	18	17	11	14	16
Learning rate	0.0001	0.0001	0.0001	0.0001	0.001	0.0001
Batch size	256	128	256	256	256	32
RMSE in eV	0.027	0.016	0.019	0.016	0.018	0.013
MAE in eV	0.015	0.010	0.012	0.011	0.012	0.010

Each column shows the lowest error found on the data set and the hyperparameters used in the model.

Table 5. Hyperparameter configurations for the Bayesian optimizer.

Bayesian optimisation results						
	Pyrazine	Pyrazine Half	Furan	Furan Half	Pyrrole	Pyrrole Half
Number of nodes	90	190	60	190	80	130
Number of layers	14	6	13	5	7	5
Learning rate	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001
Batch size	128	128	32	32	32	64
RMSE in eV	0.021	0.018	0.021	0.029	0.017	0.016
MAE in eV	0.014	0.013	0.011	0.014	0.010	0.011

Each column shows the lowest error found on the data set and the hyperparameters used in the model.

can, however, see a trend where networks with more layers combined with a large batch size are more likely to produce more favourable results, whereas smaller networks require a smaller batch size to be successful. These relations reflect the complexity of finding an optimal set of hyperparameters.

In addition, the hyperparameters are in most cases dependent on each other, and therefore the resulting high dimensional error surface that the optimizers have to navigate through shows many local minima. It is true that for fully converged optimisations it is expected that all three optimizers (GA, SA, and BO) lead to similar results; however, finding these global minima is a challenge itself and is beyond the scope of this work. In this work, we focus on finding the optimised hyperparameters leading to an error lower than a given threshold. The configurational hyperparameter space that the SA, GA, and BO optimizer visit during the optimisation is given by their respective algorithm and can be significantly different from each other. This leads in our opinion to the discrepancy between the optimal found hyperparameters given the different optimizers.

We can also see that halving the number of data points results in different hyperparameters, but still workable neural networks. This shows that the optimizer is capable of producing workable hyperparameters for different

amounts of data points. It should be noted that the poor performance of the GS that used two hyperparameters for the Furan half data set is no longer the case when the SA, GA, and BO algorithms are used with four hyperparameters.

It is important to check the validity and reliability of the generated neural network-based PESs that are prerequisites for sound non-adiabatic excited-state dynamics such as surface hopping simulations. To verify this, in Figure 6, we show the PESs as functions of time for the reference *ab initio* based trajectory (solid) together with the neural network-generated PESs constructed with the best hyperparameters (dashed). It is hard to see any difference as the curves lie on top of each other, reflecting their excellent agreement. The neural network-generated PESs are smooth and confirm the shape of the reference trajectories. The model PESs are able to capture the different characteristics for each molecule, e.g. the higher energy gap of Furan and the steeper energy landscape of Pyrrole.

A good model PESs relies on appropriate training data set that naturally samples the important parts of the conformational space including the areas of interest to surface hopping, namely the areas of high hopping probabilities that is when the PESs are getting close to each other. Figure 7 shows a close view of the crucial

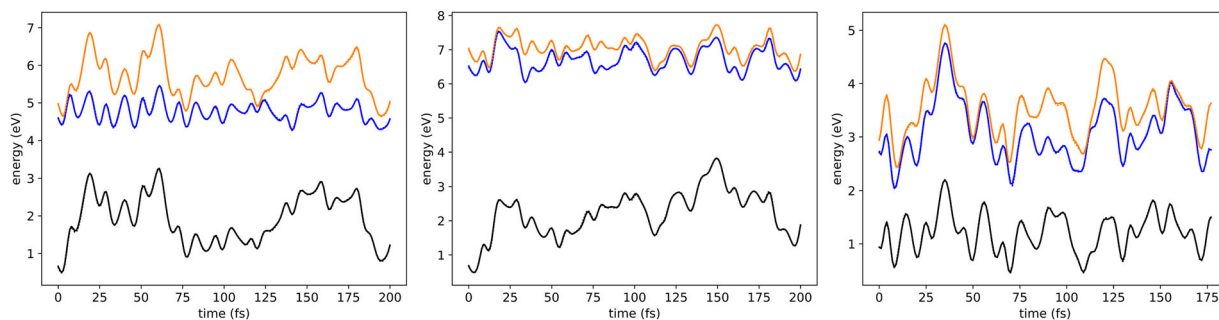


Figure 6. Comparison of neural network-generated PESs (dashed), for the best network found by the SA algorithm for Pyrazine (left), the GA algorithm for Furan (middle), and the BO algorithm for Pyrrole (right) and reference *ab initio* based PESs (solid). S_0 (black); S_1 (blue); S_2 (orange).

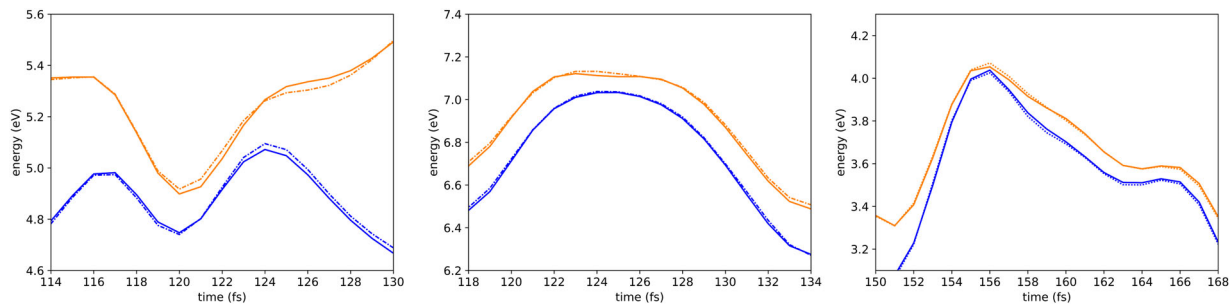


Figure 7. Comparison of neural network-generated PESs (dashed), for the best network found by the SA algorithm for Pyrazine (left), the GA algorithm for Furan (middle), and the BO algorithm for Pyrrole (right) and reference *ab initio*-based PESs (solid), zoomed in on the areas where the S_1 and S_2 surfaces are close. S_1 (blue); S_2 (orange).

areas, i.e. where the S_1 and S_2 surfaces are close to each other. It is clear that these important areas are also modelled well. The neural network-generated PESs (dashed) show a slightly larger energy gap than the reference PESs (solid). The difference in the energy gap is about 11 meV, which is significantly small considering the fact that neural network-generated PESs require less human effort and reduce much of the computational cost associated with constructing the reference PESs. Analogous Figures for the best networks found by the GS for Pyrazine, Pyrrole, and Furan can be found in the Supporting Information (Figures 1 and 2).

6. Conclusion

The HOAX package is an extendable open-source package written fully in Python that facilitates the hyperparameter optimisation search for the application of neural network models for constructing PESs in an automated fashion, which bypasses the need for a lengthy manual process and reduces computational costs compared to the ab initio based PESs. It uses the PyTorch interface to generate fast and customised implementations of neural networks. Additionally, it is currently an extension of the PySurf package as a new Plugin engine. Thus, it has direct access to the PySurf database for training and validation data in the NetCDF and the HDF5 formats. HOAX can be easily adapted to other packages that provide molecular data in HDF5 compatible format. The hyperparameter optimizer could also be extended to perform automated cross-validation on the best networks found, even during the neural network training process.

In this work we show that the package can produce neural network model PESs for SO_2 , Pyrazine, Pyrrole, and Furan, using the GS, SA, GA, and BO algorithms. The GS algorithm results show that a higher number of layers and nodes has a positive effect on the error. The BO, SA, and GA models show similar performance at the same computational time. All three hyperparameter optimizers are able to search a four-dimensional hyperparameter space and consistently find the optimal hyperparameters configuration. The 50% reduction of the data set still produces good model PESs, showing the robustness of the model. The neural network-generated PESs are smooth and confirm the shape of the reference ab initio based trajectories. The model PESs are able to capture the different characteristics for each molecule. The areas of interest to surface hopping, namely the areas of high hopping probabilities that is when the PESs are getting close to each other, are also modelled well.

The HOAX package currently supports the optimisation of neural network hyperparameters, but it can be extended to other ML models such as, kernel-based

methods [22], Gaussian processes [69], and random forest regression [70]. Also, different neural network architectures could be introduced, such as graph neural networks [18] or convolutional neural networks [71]. The hyperparameter optimizers can also be extended with additional non-gradient methods, such as a greedy randomised adaptive search procedure [72]. Different molecular representations for the training data, such as Coulomb-matrices or graph representations could be considered. The modular setup of the package provides flexibility to add custom functionalities. Work in this direction is currently in progress in our research team. The model can also be extended for other types of data, such as charges, dipole moments, and orbital energies, as long as the data is provided in the HDF5 or NetCDF format.

Acknowledgments

Prof. Peter Gill is one of the main contributors to the electronic structure program Q-Chem and drove the development of theoretical chemistry in his successful research carrier. We are happy to honour him with this debut of the HOAX software package. The authors thank Prof. Peter Gill, Dr. Andrew Gilbert, and E. Salazar for many inspiring and intellectually stimulating conversations and for giving an insight into the process of new method development. Additionally, the authors are thankful to Dr. Kiana Moghaddam for her help in proofreading the manuscript.

Disclosure statement

No potential conflict of interest was reported by the author(s).

Funding

This work is part of the Innovational Research Incentives Scheme Vidi 2017 with project number 016.Vidi.189.044, which is financed by the Dutch Research Council (NWO).

ORCID

Albert Thie  <http://orcid.org/0000-0003-0978-4588>

Maximilian F.S.J. Menger  <http://orcid.org/0000-0003-1442-9601>

Shirin Faraji  <http://orcid.org/0000-0002-6421-4599>

References

- [1] J. Baldwin-Philippi, *Political Commun.* **34** (4), 627–633 (2017). doi:10.1080/10584609.2017.1372999
- [2] E. Gardner, *J. Phys. A Math. Theor.* **19**, L1047–L1056 (1986). doi:10.1088/0305-4470/19/16/017
- [3] P. Gervás, B. Lönneker-Rodman, J.C. Meister and F. Peinado, in *International Conference on Language Resources and Evaluation. Satellite Workshop: Toward Computational Models of Literary Analysis*, pp. 44–51.

- [4] J.C. Snyder, M. Rupp, K. Hansen, K.R. Müller and K. Burke, *Phys. Rev.* **108** (25), 253002 (2012). doi:10.1103/PhysRevLett.108.253002
- [5] K.T. Schütt, H.E. Sauceda, P.J. Kindermans, A. Tkatchenko and K.R. Müller, *J. Chem. Phys.* **148** (24), 241722 (2018). doi:10.1063/1.5019779
- [6] J. Behler and M. Parrinello, *Phys. Rev.* **98**, 146401 (2007). doi:10.1103/PhysRevLett.98.146401
- [7] J. Westermayr and P. Marquetand, *Chem. Rev.* **121** (16), 9873–9926 (2021). doi:10.1021/acs.chemrev.0c00749
- [8] P.O. Dral, M. Barbatti and W. Thiel, *J. Phys. Chem. Lett.* **9** (19), 5660–5663 (2018). doi:10.1021/acs.jpcllett.8b02469
- [9] J. Berner, P. Grohs, G. Kutyniok and P. Petersen, arXiv preprint arXiv:2105.04026, 2021.
- [10] H.J. Kulik, T. Hammerschmidt, J. Schmidt, S. Botti, M.A.L. Marques, M. Boley, M. Scheffler, M. Todorović, P. Rinke, C. Oses, A. Smolyanyuk, S. Curtarolo, A. Tkatchenko, A.P. Bartók, S. Manzhos, M. Ihara, T. Carrington, J. Behler, O. Isayev, M. Veit, A. Grisafi, J. Nigam, M. Ceriotti, K.T. Schött, J. Westermayr, M. Gastegger, R.J. Maurer, B. Kalita, K. Burke, R. Nagai, R. Akashi, O. Sugino, J. Hermann, F. Noé, S. Pilati, C. Draxl, M. Kuban, S. Rigamonti, M. Scheidgen, M. Esters, D. Hicks, C. Toher, P.V. Balachandran, I. Tamblyn, S. Whitelam, C. Bellinger and L.M. Ghiringhelli, *Electron. Struct.* **4** (2), 023004 (2022). doi:10.1088/2516-1075/ac572f
- [11] J. Bergstra, R. Bardenet, Y. Bengio and B. Kégl, *Adv. Neural Inf. Process. Syst.* **24**, 2546–2554 (2011). doi:10.1103/PhysRevLett.108.253002.
- [12] R. Ramakrishnan, P.O. Dral, M. Rupp and O.A. von Lilienfeld, *Sci. Data* **1**, (2014). doi:10.1038/sdata.2014.22
- [13] J. Hermann, Z. Schatzle and F. Noé, *Nat. Chem.* **12** (10), 891–897 (2020). doi:10.1038/s41557-020-0544-y
- [14] K. Gubaev, E.V. Podryabinkin and A.V. Shapeev, *J. Chem. Phys.* **148** (24), 241727 (2018). doi:10.1063/1.5005095
- [15] N. Lubbers, J.S. Smith and K. Barros, *J. Chem. Phys.* **148** (24), 241715 (2018). doi:10.1063/1.5011181
- [16] B. Anderson, T.S. Hy and R. Kondor, *Adv. Neural Inf. Process. Syst.* **32**, (2019). doi:10.48550/arXiv.1906.04015
- [17] D.K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik and R.P. Adams, *Adv. Neural Inf. Process. Syst.* **28**, (2015). doi:10.5555/2969442.2969488
- [18] J. Gilmer, S.S. Schoenholz, P.F. Riley, O. Vinyals and G.E. Dahl, in *Proceedings of the 34th International Conference on Machine Learning*, Vol. 70, pp. 1263–1272, 2017. Sydney
- [19] K.T. Schütt, F. Arbabzadah, S. Chmiela, K.R. Müller and A. Tkatchenko, *Nat. Commun.* **8** (1), (2017). doi:10.1038/ncomms13890
- [20] A.S. Christensen, F.A. Faber and O.A. von Lilienfeld, *J. Chem. Phys.* **150** (6), 064105 (2019). doi:10.1063/1.5053562
- [21] F.A. Faber, A.S. Christensen, B. Huang and O.A. von Lilienfeld, *J. Chem. Phys.* **148** (24), 241717 (2018). doi:10.1063/1.5020710
- [22] R. Ramakrishnan and O.A. von Lilienfeld, *CHIMIA* **69** (4), 182 (2015). doi:10.2533/chimia.2015.182
- [23] M. Rupp, R. Ramakrishnan and O.A. Von Lilienfeld, *J. Phys. Chem. Lett.* **6** (16), 3309–3313 (2015). doi:10.1021/acs.jpcllett.5b01456
- [24] J. Westermayr, M. Gastegger and P. Marquetand, *J. Phys. Chem. Lett.* **11** (10), 3828–3834 (2020). doi:10.1021/acs.jpcllett.0c00527
- [25] S. Mai, M. Richter, M. Heindl, M.F.S.J. Menger, A. Atkins, M. Ruckebauer, F. Plasser, L.M. Ibele, S. Kropf, M. Opper, P. Marquetand and L. González, SHARC2.1: Surface Hopping Including Arbitrary Couplings – Program Package for Non-Adiabatic Dynamics sharc-md.org 2019.
- [26] W. Koch, M. Bonfanti, P. Eisenbrandt, A. Nandi, B. Fu, J. Bowman, D. Tannor and I. Burghardt, *J. Chem. Phys.* **151** (6), 064121 (2019). doi:10.1063/1.5113579
- [27] Y. Guan and D.R. Yarkony, *J. Phys. Chem. Lett.* **11** (5), 1848–1858 (2020). doi:10.1021/acs.jpcllett.0c00074
- [28] J. Westermayr, M. Gastegger, M.F. Menger, S. Mai, L. González and P. Marquetand, *Chem. Sci.* **10** (35), 8100–8107 (2019). doi:10.1039/C9SC01742A
- [29] B. Schölkopf, A.J. Smola and F. Bach, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond* (MIT Press, United States, 2002).
- [30] G. Schmitz, I.H. Godtlielsen and O. Christiansen, *J. Chem. Phys.* **150** (24), 244113 (2019). doi:10.1063/1.5100141
- [31] A.S. Abbott, J.M. Turney, B. Zhang, D.G.A. Smith, D. Altarawy and H.F. Schaefer, *J. Chem. Theory Comput.* **15** (8), 4386–4398 (2019). doi:10.1021/acs.jctc.9b00312
- [32] J. Bergstra, B. Komer, C. Eliasmith, D. Yamins and D.D. Cox, *Comput. Sci. Discov.* **8** (1), 014008 (2015). doi:10.1088/1749-4699/8/1/014008
- [33] L.M. Rios and N.V. Sahinidis, *J. Glob. Optim.* **56** (3), 1247–1293 (2013). doi:10.1007/s10898-012-9951-y
- [34] K. Chandra, E. Meijer, S. Andow, E. Arroyo-Fang, I. Dea, J. George, M. Grueter, B. Hosmer, S. Stumpos, A. Tempest and S. Yang, arXiv preprint arXiv:1909.13371, 2019.
- [35] Q. Huang, J. Mao and Y. Liu, in *2012 IEEE 14th International Conference on Communication Technology*, pp. 1022–1026.
- [36] S.R. Wilson, W. Cui, J.W. Moskowitz and K.E. Schmidt, *J. Comput. Chem.* **12** (3), 342–349 (1991). doi:10.1002/(ISSN)1096-987X
- [37] F. Hoof, A. Perez de Albaz Ortiz and B. Ensing, *J. Chem. Theory Comput.*, 2294–2306 (2021). doi:10.1021/acs.jctc.0c00981
- [38] S. Ding, C. Su and J. Yu, *Artif. Intell. Rev.* **36**, 153–162 (2011). doi:10.1007/s10462-011-9208-z
- [39] J. Snoek, H. Larochelle and R.P. Adams, *Adv. Neural Inf. Process. Syst.* **25**, 2951–2959 (2012). doi:10.5555/2999325.2999464
- [40] K. Gurney, *An Introduction to Neural Networks* (CRC Press, London, 2014).
- [41] D.P. Kingma and J. Ba, arXiv preprint arXiv:1412.6980, 2017.
- [42] A.M. Geoffrion, *Math. Program* **13** (1), 23–37 (1977). doi:10.1007/BF01584321
- [43] G. Harman and S. Kulkarni, *Reliable Reasoning: Induction and Statistical Learning Theory* (MIT Press, Cambridge, MA, 2012).
- [44] L. Davis, *Genetic Algorithms and Simulated Annealing* (Morgan Kaufman Publishers, Inc., Los Altos, CA, United States, 1987).
- [45] J. Bergstra and Y. Bengio, *J. Mach. Learn. Res.* **13** (2), 281–305 (2012).

- [46] J.Y. Hesterman, L. Caucci, M.A. Kupinski, H.H. Barrett and L.R. Furenlid, *IEEE Trans. Nucl. Sci.* **57** (3), 1077–1084 (2010). doi:10.1109/TNS.2010.2045898
- [47] S. Xu and Y. Rahmat-Samii, *IEEE Trans. Antennas Propag.* **55** (3), 760–765 (2007). doi:10.1109/TAP.2007.891562
- [48] D. Charles, *On the Origin of Species by Means of Natural Selection, Or the Preservation of Favoured Races in the Struggle for Life* (John Murray, London, 1859).
- [49] G. Moghadampour, in *Proceedings of the 13th International Conference on Enterprise Information Systems*, pp. 54–62. Beijing
- [50] A. Hassanat, K. Almohammadi, E. Alkafaween, E. Abunawas, A. Hammouri and V.B.S. Prasath, *Information* **10** (12), (390.2019). doi:10.3390/info10120390
- [51] S. Russel and P. Norvig, *Artificial Intelligence: A Modern Approach* (Pearson Education, London, UK, 2003).
- [52] K.R. Koch, *Introduction to Bayesian Statistics* (Springer Science & Business Media, Berlin, 2007).
- [53] M.F. Menger, J. Ehrmaier and S. Faraji, *J. Chem. Theory Comput.* **16** (12), 7681–7689 (2020). doi:10.1021/acs.jctc.0c00825
- [54] R. Rew and G. Davis, *IEEE Comput. Graph. Appl.* **10** (4), 76–82 (1990). doi:10.1109/38.56302
- [55] S. Brown, M. Folk, G. Goucher, R. Rew and P. Dubois, *Comput. Phys. Commun.* **7**, 304 (1993). doi:10.1063/1.4823180
- [56] F. Pezoa, J.L. Reutter, F. Suarez, M. Ugarte and D. Vrgoč, in *Proceedings of the 25th International Conference on World Wide Web*, pp. 263–273. Montreal
- [57] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein and L. Antiga, *Adv. Neural Inf. Process. Syst.* **32**, 8026–8037 (2019). doi:10.5555/3454287.3455008
- [58] F. Nogueira Bayesian Optimization: Open source constrained global optimization tool for Python <https://github.com/fmfn/BayesianOptimization>, 2014.
- [59] A. Raab, G.A. Worth, H.D. Meyer and L. Cederbaum, *J. Chem. Phys.* **110** (2), 936–946 (1999). doi:10.1063/1.478061
- [60] R. Burcl, R.D. Amos and N.C. Handy, *Chem. Phys. Lett.* **355** (1), 8–18 (2002). doi:10.1016/S0009-2614(02)00122-7
- [61] A.K. Belyaev and O.V. Lebedev, *Phys. Rev. A* **84**, 014701 (2011). doi:10.1103/PhysRevA.84.014701
- [62] L. Sun and W.L. Hase, *J. Chem. Phys.* **133** (4), 044313 (2010). doi:10.1063/1.3463717
- [63] E. Epifanovsky, A.T.B. Gilbert, X. Feng, J. Lee, Y. Mao, N. Mardirossian, P. Pokhilko, A.F. White, M.P. Coons, A.L. Dempwolff and Z. Gan, *J. Chem. Phys.* **155** (8), 084801 (2021). doi:10.1063/5.0055522
- [64] C. Lee, W. Yang and R.G. Parr, *Phys. Rev. B* **37**, 785–789 (1988). doi:10.1103/PhysRevB.37.785
- [65] W.J. Hehre, R. Ditchfield and J.A. Pople, *J. Chem. Phys.* **56** (5), 2257–2261 (1972). doi:10.1063/1.1677527
- [66] Y. Shao, M. Head-Gordon and A.I. Krylov, *J. Chem. Phys.* **118** (11), 4807–4818 (2003). doi:10.1063/1.1545679
- [67] A.D. Becke, *J. Chem. Phys.* **98** (2), 1372–1377 (1993). doi:10.1063/1.464304
- [68] T.H. Dunning, *Chem. Phys.* **90** (2), 1007–1023 (1989). doi:10.1063/1.456153
- [69] N. Srinivas, A. Krause, S.M. Kakade and M.W. Seeger, *IEEE Trans. Inf. Theory* **58** (5), 3250–3265 (2012). doi:10.1109/TIT.2011.2182033
- [70] V. Svetnik, A. Liaw, C. Tong, J.C. Culberson, R.P. Sheridan and B.P. Feuston, *J. Chem. Inf. Comput. Sci.* **43** (6), 1947–1958 (2003). doi:10.1021/ci034160g
- [71] S. Albawi, T.A. Mohammed and S. Al-Zawi, in *2017 International Conference on Engineering and Technology (ICET)*, pp. 1–6. Antalya
- [72] T.A. Feo and M.G. Resende, *J. Glob. Optim.* **6** (2), 109–133 (1995). doi:10.1007/BF01096763