Automation of some macromolecular properties using a machine learning approach

IOP Publishing Mach. Learn.: Sci. Technol. 2 (2021) 035016 https://doi.org/10.1088/2632-2153/abe7b6

MACHINE LEARNING
Science and Technology

OPEN ACCESS

RECEIVED
8 October 2020

REVISED
24 January 2021

ACCEPTED FOR PUBLICATION
18 February 2021

PUBLISHED
14 June 2021

PAPER

Automation of some macromolecular properties using a machine learning approach

Merjem Hoxha and Hiqmet Kamberaj

Department of Computer Engineering, Faculty of Engineering, International Balkan University, Skopje, Macedonia

E-mail: h.kamberaj@gmail.com

Keywords: hydration free energy, heat of formation, artificial neural network, swarm optimization, bootstrapping, pKa, Gibbs free energy

Abstract

In this study, we employed a newly developed method to predict macromolecular properties using a swarm artificial neural network (ANN) method as a machine learning approach. In this method, the molecular structures are represented by the feature description vectors used as training input data for a neural network. This study aims to develop an efficient approach for training an ANN using either experimental or quantum mechanics data. We aim to introduce an error model controlling the reliability of the prediction confidence interval using a bootstrapping swarm approach. We created different datasets of selected experimental or quantum mechanics results. Using this optimized ANN, we hope to predict properties and their statistical errors for new molecules. There are four datasets used in this study. That includes the dataset of 642 small organic molecules with known experimental hydration free energies, the dataset of 1475 experimental pKa values of ionizable groups in 192 proteins, the dataset of 2693 mutants in 14 proteins with given experimental values of changes in the Gibbs free energy, and a dataset of 7101 quantum mechanics heat of formation calculations. All the data are prepared and optimized using the AMBER force field in the CHARMM macromolecular computer simulation program. The bootstrapping swarm ANN code for performing the optimization and prediction is written in Python computer programming language. The descriptor vectors of the small molecules are based on the Coulomb matrix and sum over bond properties. For the macromolecular systems, they consider the chemical-physical fingerprints of the region in the vicinity of each amino acid.

1. Introduction

Recently, the neural network method has seen a broad range of applications in molecular modeling [1, 2]. In [3], a hierarchical interacting particle neural network approach is introduced using quantum models to predict molecular properties. In this approach, different hierarchical regularization terms have been introduced to improve the optimized parameters' convergence. While in [4], the machine learning (ML) like-potentials are used to predict molecular properties, such as enthalpies or potential energies. The degree to which the general features are included in characterizing the chemical space of molecules to improve these models' predictions is also discussed in [5, 6]. Tuckerman and co-workers [7] used a stochastic neural network technique to fit high-dimensional free energy surfaces characterized by the reduced subspace of collective coordinates. In [8], an *ab initio* based neural network potential energy function is introduced to model the interactions of the zinc ion in water for use in molecular dynamics simulations. While very recently [9], a comparison study has been performed between the neural network approach and Gaussian process regression to fit the potential energy surfaces. One of the recognized problems in using ML approaches for predicting free energy surfaces is the inaccurate representation of the surface topology's available features by the training data. To improve on this, a combination of metadynamics molecular dynamics with neural network chemical models are also proposed [10]. It is worth noting that an accurate representation of the reduced subspace can be important in the prediction of free energy surfaces. For that,

Wehmeyer and Noé [11] have used the time-lagged auto-encoder to determine essential degrees of freedom of dynamical data.

ML approaches are also used in drug-design, for instance, in predicting drug-target interactions [12], and it is a promising approach. In particular, the method is used in combination with molecular dynamics to predict the ligand-binding mechanism to purine nucleoside phosphorylase [13], and it accurately identifies the mechanism of drug-target binding modes.

There exist a wide range of neural network algorithms used to predict molecular property based either on fixed molecular feature descriptors [6, 14–16] or graph convolution neural networks [17–20]. In [21] is presented a design for an evaluation setup of the learned molecular representations in the neural networks.

In this study, we developed a new method for predicting (macro)molecular properties using a bootstrapping swarm artificial neural network (BSANN) method as a ML approach. In this method, molecules are represented by the description vectors, which then are used as input in the BSANN for training the neural network. We aim to develop an efficient approach for performing an artificial neural network (ANN) training using either experimental or quantum mechanics data. For that, we created different databases of well-selected experimental (or quantum mechanics) results that can be used to train the network.

There also exist error models used to control the prediction confidence interval, which is important to increase the prediction reliability [22, 23]. In this work, we introduce the bootstrapping confidence interval as an error prediction model. For the sake of comparison, we also introduce in the text the state of the art of the error model in [23], and for the Bayesian training of the neural network [24, 25]. Furthermore, in this study, we introduce a new algorithm for the feature selection.

In this study, we used the database of 642 small organic molecules with known experimental hydration free energies [26], which well-studied in [23]; a database of 1475 experimental pKa values of ionizable groups in 192 proteins (including 153 wild-type proteins and 39 mutant proteins) [27–30] is also created. Furthermore, a database of 2693 mutants in 14 proteins with given experimental values of changes in the Gibbs free energy [31, 32] is created to predict the thermodynamic properties of proteins. Moreover, we used a database of 7101 quantum mechanics heat of formation calculations with the Perdew–Burke–Ernzerhof hybrid functional (PBE0) [6, 14].

It is interesting to mention that there exist several methods for calculation of the hydration free energies [33], pKa [29] (and the references therein), changes in the free energies [34], and quantum mechanics method calculations [6, 14, 35]. The methods of pKa calculations can generally be split into approaches using the Poisson–Boltzmann equation, [36] empirical approaches, [37] and molecular dynamics-based techniques [38, 39]. The free energy calculations can be either explicitly using the molecular dynamics method or implicitly, as reviewed in [40] (see also the references therein). The quantum mechanical calculations can be using the effective core potentials and basis sets for density functional as discussed elsewhere [6, 14, 35]. Besides, there exist many software packages and web-servers available for the calculations of protein pKa (such as H++ web-server [41] and PROKA program [37]). Furthermore, several molecular dynamics software packages, such as CHARMM [42], capable of performing pKa, hydration free energy, or changes on free energy calculations, either explicitly or implicitly (such as molecular mechanics Poisson–Boltzmann surface area approach). For performing quantum mechanics calculations using either static structures or ab-initio approaches, we can mention Car–Parrinello molecular dynamics computational chemistry software package [43]. In general, molecular dynamics-based methods are computationally much more expensive and not always more accurate in predicting the pKa or free energy values than the approaches using the Poisson–Boltzmann equation. On the other hand, continuum electrostatic methods suffer the conformation flexibility, for instance, in considering multiple amino acids side-chain rotamers. Also, more often used molecular mechanics force fields do not consider other effects, such as electronic polarizability, that could be important in predicting properties, such as protonation energies. Moreover, if these approaches have to be applied in predicting the properties mentioned earlier for a large dataset, they become practically inefficient. Therefore, the development of ML approaches capable of predicting any new molecule's properties' confidence interval should be important, in particular, *in silico* drug discovery and design [40].

All the structures are prepared and optimized with the AMBER force field for proteins and nucleic acids ff99SBnmr [44, 45] and generalized AMBER force field for small organic molecules [46] (as in [26, 47]) using CHARMM macromolecular computer simulation program [42]. The BSANN is a Python computer program for performing the optimizations and predictions. Besides, the descriptor vectors of the small molecules are based on the Coulomb matrix and the sum over bond properties. For the macromolecular systems, they consider the chemical-physical fingerprints of the region in the vicinity of each amino acid.

2. Materials and methods

ML approach provides a potential method to predict the properties of a system using decision-making algorithms, based on some predefined features characterizing these properties of the system [48]. The neural networks method considers a large training dataset. It tries to construct a system that is made up of rules for recognizing the patterns within the training data set by a learning process, which can be either supervised or unsupervised training. In particular, interest is shown in accelerating chemical discovery with ML [49], and other applications in molecular modeling [1, 2]. Here, we used an improved version of the standard supervised ANN, namely the BSANN [50]. We have also introduced in the following the BSANN approach.

2.1. Bootstrapping swarm ANN

In general, for a supervised ANN with K hidden layers (see also figure 1), the output Y_i is defined as

$$Y_i = f\Bigg(\underbrace{\sum_{l_K=1}^{L_K} f\Bigg(\underbrace{\sum_{l_{K-1}=1}^{L_{K-1}} f\Bigg(\cdots f\Bigg(\underbrace{\sum_{l_2=1}^{L_2} f\Bigg(\underbrace{\sum_{l_1=1}^{L_1} f\Bigg(\underbrace{\sum_{j=1}^{n} X_j W_{jl_1} + b_{l_1}}_{\text{input layer}}\Bigg)}_{\text{1st hidden layer}}\Bigg)}_{\text{2nd hidden layer}}\Bigg)\cdots\Bigg)}_{\cdots}\Bigg)}_{(K-1)\text{th hidden layer}} W_{l_K i} + b_i\Bigg)}_{K\text{th hidden layer}}. \tag{1}$$

Here, $\mathbf{W}$ and $\mathbf{b}$ are considered as free parameters, which need to be optimized for a given training data used as inputs and given outputs, which are known. To optimize these parameters the so-called loss function is minimized using gradient descent method [51]:

$$S(\mathbf{W},\mathbf{b}) = \sum_{i=1}^{m} \left(Y_i^0 - Y_i\right)^2 \tag{2}$$

where $\mathbf{Y}^0$ represent the true output vector. For that, the gradients of $S(\mathbf{W},\mathbf{b})$ with respect to $\mathbf{W}$ and $\mathbf{b}$ are calculated [51]:

$$\begin{aligned} \Delta\mathbf{W} &= -\left(\frac{\partial S(\mathbf{W},\mathbf{b})}{\partial \mathbf{W}}\right)_{\mathbf{b}} \\ \Delta\mathbf{b} &= -\left(\frac{\partial S(\mathbf{W},\mathbf{b})}{\partial \mathbf{b}}\right)_{\mathbf{W}}. \end{aligned} \tag{3}$$

To avoid over-fitting, which is one of the problems of the ML approaches [52], the following regularization terms have been introduced in literature:

$$\begin{aligned} \Delta'\mathbf{W} &= \gamma_w\left(\Delta\mathbf{W} + \gamma_1\mathbf{W}\right) \\ \Delta'\mathbf{b} &= \gamma_w\left(\Delta\mathbf{b} + \gamma_1\mathbf{b}\right) \end{aligned} \tag{4}$$

where γ_w is called learning rate for the gradient and γ_1 is called the regulation strength.

Usually, the gradient descent method often converges to a local minimum, and hence it provides a local optimization to the problem. Therefore, a new BSANN method is proposed in the literature [50].

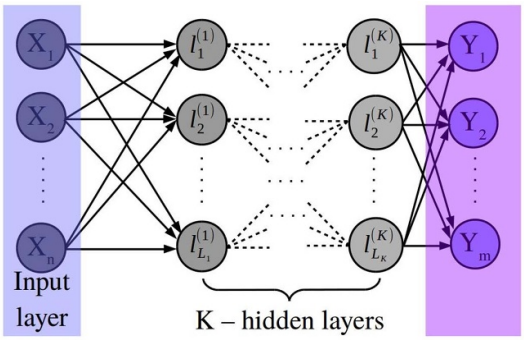

Figure 1. Illustration diagram of an artificial neural network (ANN). It is characterized by an input vector of dimension n, K hidden layers of $l_{L_1}^{(1)}, l_{L_2}^{(2)}, \ldots, l_{L_K}^{(K)}$ neurons each, and an output vector of dimension m [50]. Reprinted by permission from Springer Nature Customer Service Centre GmbH: [Nature] [Molecular Dynamics Simulations in Statistical Physics: Theory and Applications] [50] (2020).

The standard ANN method deals with random numbers, which are used to initialize the parameters **W** and **b**; therefore, the optimal solution of the problem will be different for each run. In particular, we can say that there exists an uncertainty in the calculation of the optimal solution (i.e. in determining **W** and **b**.) To calculate these uncertainties in the estimation of the optimal parameters, **W** and **b**, a new approach, bootstrapping ANN, was proposed in [53], or similar methods [54]. In this approach, M copies of the same neural network run independently using different input vectors. Here, we implement that at regular intervals to swap optimal parameters (i.e. **W** and **b**) between the two neighboring neural networks, which is equivalent to increasing the dimensionality of the problem by one; that is, if the dimensionality in each of the replicas is $d = K \times L$, then the dimensionality of the bootstrapping ANN method is $d + 1$. Figure 2 shows the layout of this configuration.

Furthermore, to achieve a good sampling of the phase space extended by the vectors **W** and **b**, we introduce two other regularization terms similar to the swarm-particle sampling approach. First, we define two vectors for each neural network, namely $\mathbf{W}_n^{\text{Lbest}}$ and $\mathbf{b}_n^{\text{Lbest}}$, which represent the best local optimal parameters for each neural network n. Also, we define $\mathbf{W}^{\text{Gbest}}$ and $\mathbf{b}^{\text{Gbest}}$, which represent the global best optimal parameters among all neural networks.

Then, the expressions in equation (4) are modified by introducing these two regularization terms as the following:

$$\begin{aligned}\Delta''\mathbf{W}_n &= \gamma_w\left(\Delta\mathbf{W}_n + \gamma_1\mathbf{W}_n\right.\\ &\quad\left. - \gamma_2 U(0,1)\left(\mathbf{W}_n - \mathbf{W}_n^{\text{Lbest}}\right) - \gamma_3 U(0,1)\left(\mathbf{W}_n - \mathbf{W}^{\text{Gbest}}\right)\right)\\ \Delta''\mathbf{b}_n &= \gamma_w\left(\Delta\mathbf{b}_n + \gamma_1\mathbf{b}_n\right.\\ &\quad\left. - \gamma_2 U(0,1)\left(\mathbf{b}_n - \mathbf{b}_n^{\text{Lbest}}\right) - \gamma_3 U(0,1)\left(\mathbf{b}_n - \mathbf{b}^{\text{Gbest}}\right)\right)\end{aligned} \tag{5}$$

for each neural network configuration n, $n = 1, 2, \ldots, M$. Here, $U(0,1)$ is a random number between zero and one, and γ_2 and γ_3 represent the strength of biases toward the local best optimal parameters and global best optimal parameters, respectively. The first term indicates the individual knowledge of each neural network and the second bias term the social knowledge among the neural networks. Then, the weights, $\mathbf{W}_n$, and biases, $\mathbf{b}_n$, for each neural network n are updated at each iteration step according to

$$\begin{aligned}\mathbf{W}_n^{\text{new}} &= \mathbf{W}_n^{\text{old}} + \Delta''\mathbf{W}_n\\ \mathbf{b}_n^{\text{new}} &= \mathbf{b}_n^{\text{old}} + \Delta''\mathbf{b}_n.\end{aligned} \tag{6}$$

A single hidden layer neural network with identically independent distributed initial parameters is equivalent to a Gaussian process described above in the case of the infinite network width, that is $L_1 \to \infty$; [55]. This, in turn, allows establishing a Bayesian inference framework for the infinite width neural network. Furthermore, it can generate kernel functions to describe the multi-layer ANNs, which can be used as covariance functions for Gaussian process regression, allowing full Bayesian prediction for an ANN [25].

In standard ANN, assuming that initial weights and bias parameters are taken as identically independent distributed random variables, that is

$$W_{ij}^0 \sim \mathcal{G}(0, \sigma_w^2/N_{\text{train}}), \quad b_j^0 \sim \mathcal{G}(0, \sigma_b^2) \tag{7}$$

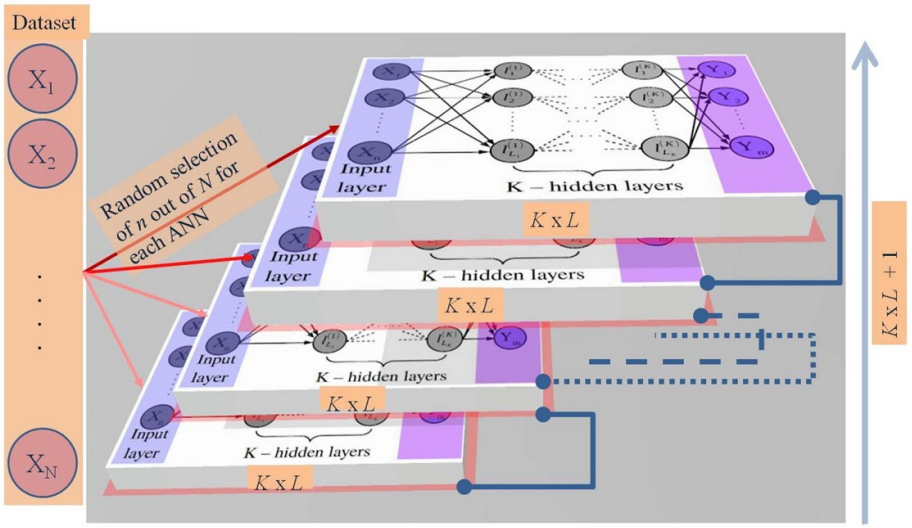

Figure 2. Layout of the BSANN as adopted by [50]. It is characterized by M different input vectors each of dimension n, K hidden layers of $l_{L_1}^{(1)}, l_{L_2}^{(2)}, \ldots, l_{L_K}^{(K)}$ neurons each, and M different output vectors each of dimension m. Every two neighboring neural networks communicate regularly with each other by swapping the optimized parameters.

then x_j^1 and $x_{j'}^1$ are independent for $j \neq j'$. In addition, $Z_i^1(x)$ (see also equation (13)) is sum of the identically independent distributed terms; therefore, based on the Center Limit Theorem in the limit of the infinite network width ($L_1 \to \infty$), it follows that $Z_i^1(x)$ is Gaussian distributed. Moreover, a finite process

$$Z_i^1(\mathbf{X}), \quad \mathbf{X} = (x_1, x_2, \ldots, x_n) \tag{8}$$

has a joint Gaussian distribution, that is, it forms a Gaussian process. Therefore, Z_i^1 is a Gaussian process with mean μ^1 and covariance K^1 [24]

$$Z_i^1 \sim \mathcal{G}(\mu^1, K^1) \tag{9}$$

where

$$\mu^1(x) = E[Z_i^1(x)] = 0 \tag{10}$$

and

$$K^1(x, x') = E[Z_i^1(x) Z_i^1(x')] = \sigma_b^2 + \sigma_w^2 C(x, x') \tag{11}$$

with C being the covariance:

$$C(x, x') = E[X_i^1(x) X_i^1(x')] = E\left[f(Z_i^0(x)) f(Z_i^0(x'))\right]. \tag{12}$$

For a K-hidden layer neural network, the output of the l layer is

$$Z_i^l(x) = \sum_{j=1}^{L_l} W_{ij}^l X_j^l(x) + b_i^l \tag{13}$$

$$X_j^l(x) = f\left(\sum_{k=1}^{L_{l-1}} W_{jk}^{l-1} X_k^{l-1} + b_j^{l-1}\right) \tag{14}$$

where $Z_i^l(x)$ are identically independent distributed random variables, and $Z_i^l(\mathbf{X})$ forms a Gaussian random process for $L_l \to \infty$: $Z_i^l \sim \mathcal{G}(0, K^l)$, with covariance K^l given as [55]

$$K^l(x, x') = E[Z_i^l(x) Z_i^l(x')] = \sigma_b^2 + \sigma_w^2 E\left[f(Z_i^{l-1}(x)) f(Z_i^{l-1}(x'))\right] \tag{15}$$

which can be re-written in the following recursive form [25]:

$$K^l(x,x') = \sigma_b^2 + \sigma_w^2 G_f\left(K^{l-1}(x,x'), K^{l-1}(x,x), K^{l-1}(x',x')\right) \tag{16}$$

where G_f is a deterministic function depending on the choice of the function f. This indicates that the ANN can be performed in a series of computations obtaining K^l as in the Gaussian process regression described above. Therefore, there exists an equivalence between the Gaussian process and the ANN in the limit of $L_k \to \infty$ and that initially the weights and bias parameters are drawn from identically independent distributed random variables by equation (7).

Therefore, we can use the Gaussian process to do Bayesian training of ANN [24]. Following [25], for that, assume a dataset $\mathcal{D}$ with elements (X_i, Y_i) for $i = 1, 2, \ldots, N_{\text{train}}$ representing the input-reference data-point pairs. The aim is to do a Bayesian prediction of some test point $X^\star$ using the distribution of the outputs $\mathbf{Z}(\mathbf{X})$ as obtained from a trained ANN with probability:

$$\begin{aligned} P(Y^\star|\mathcal{D}, X^\star) &= \int d\mathbf{Z} P(Y^\star|\mathbf{Z}, \mathbf{X}, X^\star) P(\mathbf{Z}|\mathcal{D}) \\ &= \frac{1}{P(\mathbf{Y})} \int d\mathbf{Z} P(Y^\star, \mathbf{Z}|X^\star, \mathbf{X}) P(\mathbf{Y}|\mathbf{Z}) \end{aligned} \tag{17}$$

where $Y^\star$ is the predicted output value of the input value $X^\star$. Note that the well-known relation $p(x,y) = p(x|y)p(y)$ is used twice to obtain the final expression. In equation (17), $P(\mathbf{Y}|\mathbf{Z})$ gives probability of obtaining the reference distribution $\mathbf{Y}$ from the ANN with an output of the distribution $\mathbf{Z}$ from the training dataset; therefore, it represents the error in the output of the ANN, and it can be modeled as noise centered at the output distribution $\mathbf{Z}$ and variance σ_ε^2 with an unbiased estimate as:

$$\sigma_\varepsilon^2 = \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} (Z_i - Y_i)^2 . \tag{18}$$

That is, under the condition of the initial choice of the parameters and assuming that network width is infinite, this implies that process

$$Z_1, Z_2, \ldots, Z_{N_{\text{train}}}, Y^\star \tag{19}$$

is a Gaussian process and hence $P(Y^\star, \mathbf{Z}|X^\star, \mathbf{X}) \sim \mathcal{G}(0, \mathbf{K})$ is a multivariate Gaussian with covariance [25]

$$\mathbf{K} = \begin{bmatrix} \mathbf{K}_{\mathcal{D},\mathcal{D}} & \mathbf{K}_{X^\star,\mathcal{D}}^T \\ \mathbf{K}_{X^\star,\mathcal{D}} & \mathbf{K}_{X^\star,X^\star} \end{bmatrix} \tag{20}$$

where $X^\star$ is the test point. In equation (20), $\mathbf{K}_{\mathcal{D},\mathcal{D}}$ is a $N_{\text{train}} \times N_{\text{train}}$ block matrix with elements $K_{ij} = K(X_i, X_j)$ where both X_i and X_j are drawn from $\mathcal{D}$. On the other hand, $\mathbf{K}_{X^\star,\mathcal{D}}$ is a block matrix whose elements are $K_{ij} = K(X^*, X_i)$ with only $X_i \in \mathcal{D}$. The integration in equation (17) can be performed exactly to get [25]

$$\begin{aligned} P(Y^\star|\mathcal{D}, X^\star) &\sim \mathcal{G}(\hat{\mu}, \hat{\mathbf{K}}) \\ \hat{\mu} &= \mathbf{K}_{X^\star,\mathcal{D}} \left(\mathbf{K}_{\mathcal{D},\mathcal{D}} + \sigma_\varepsilon^2 \mathbf{I}\right)^{-1} \mathbf{Y} \\ \hat{\mathbf{K}} &= \mathbf{K}_{X^\star,X^\star} - \mathbf{K}_{X^\star,\mathcal{D}} \left(\mathbf{K}_{\mathcal{D},\mathcal{D}} + \sigma_\varepsilon^2 \mathbf{I}\right)^{-1} \mathbf{K}_{X^\star,\mathcal{D}}^T . \end{aligned} \tag{21}$$

However, the prediction of $P(Y^\star|\mathcal{D}, X^\star)$ and the calculation of the mean value and variance of the predicted value $Y^\star$ are under the assumption of the infinite width of the networks to apply the Center Limit Theorem. That is not easy to implement. Therefore, the bootstrapping approach introduced here represents an alternative way of evaluating $P(Y^\star|\mathcal{D}, X^\star)$, the mean value and variance of the test data points.

2.2. Feature description vectors

To construct data-driven models, such as in the ML approach, we will need to specify a list of the input (macro) molecule's physical and chemical properties that contain necessary information about the system. Here, the input data will be presented by a vector of length N, called $\mathbf{X}$. That process is called *feature description*, and the input data are called *feature descriptors*.

Often, the simplified molecular input line entry system is used to represent a small molecule as a string of letters [56]. In such a case, the atoms could be encoded by a single integer number, such as H= 1, C= 2, N= 3, and so on, or by the nuclear charge Z, such as H= 1, C= 6, N= 7, and so on [57]. That creates an

(unnecessary) relationship between the input data, namely H<C<N, which could influence the performance of the network. Other encoding models are also suggested, for instance, representing each atom of the input molecule by the following fingerprint: H= [1 0 0 ⋯], C= [0 1 0 ⋯], N= [0 0 1 ⋯], and so on [57]. However, these fingerprints do also have drawbacks because the dimensions of the encoding vector depend on the number of atoms in the structure and may vary from molecule to molecule; moreover, based on this model, the atoms belonging to the same group in the periodic table of elements do not behave the same.

In this study, we used the Coulomb matrix, **C**, to encode the molecular features, which contains both the geometrical information of the three-dimensional structure and the type of atom [14]. For any two atoms i and j in a given input molecule, the matrix element C_{ij} is as follows:

$$C_{ij} = \begin{cases} \frac{Z_i^{2.4}}{2}, & i = j \\ \frac{Z_i Z_j}{r_{ij}}, & i \neq j \end{cases} \tag{22}$$

where Z_i is the atomic number of the ith atom and r_{ij} is the distance between the atoms i and j. The fingerprint represented by the Coulomb matrix, **C**, has some advantages. It considers the three-dimensional molecular structure, and it is invariant under rotation translation of the structure. To calculate **C** for a given molecular structure, we need the nuclear charges for each atom and the Cartesian coordinates of the atomic positions taken from the equilibrium geometry. However, note that **C** is not invariant under the permutations of the atom order in a molecule. Therefore, the spectrum of eigenvalues of matrix **C** can be used as a fingerprint of the molecule since they are invariant under both rotation/translation and permutations of the rows and columns. A second feature descriptor that we used in this study is the sum over bonds, which is a numerical descriptor representing the vector of bond types present in a molecule, similar to [21]. If N_b is the number of unique bonds in the dataset of the compounds studied, then a vector with dimensions N_b is constructed for each molecule with entry either zeros or the integers giving the frequency of appearance for each bond type in molecular structure. This fingerprint descriptor vector has a unique length within the dataset. Then, the vector descriptor of the sum over bonds concatenates at the end of the Coulomb matrix descriptor.

To construct the input descriptor vector for a macromolecule, we introduced the following model. For example, suppose we would like to calculate the change on the Gibbs free energy upon the mutations (either single or multiple mutations) or perform pKa calculations for a selected residue in a protein. We label each residue or nucleotide of the input sequence with an ID from 1 to 24. That is, we form a descriptor vector with length $N_1 = 24$, $\mathbf{X}_1$, which is a vector of zeros and ones defined as the following:

$$\begin{array}{rccccccl} & & \text{VAL} & & \cdots & \text{THR} & \cdots & \leftarrow \text{mutation point} \\ & & \downarrow & & \cdots & \downarrow & \cdots & \\ \mathbf{X}_1 = & 0 & 1 & 0 & \cdots & 1 & \cdots & \leftarrow \text{descriptor vector} \\ & \uparrow & \uparrow & \uparrow & \cdots & \uparrow & \cdots & \\ & \text{ALA} & \text{VAL} & \text{LEU} & \cdots & \text{THR} & \cdots & \leftarrow \text{A. A. dictionary.} \end{array} \tag{23}$$

Here, 'A. A. dictionary' represents a dictionary of names of all amino acids. In addition, to characterize the environment around any mutation point, we determine another descriptor vector, namely $\mathbf{X}_2$ with length $N_2 = 24$, which is defined as the following. For each mutation point amino acid i, we determine the nearest neighbor amino acids $k \in \{i_1, i_2, \ldots, i_{n.n.}\}$, based, for example, on the center of mass distance. Then, the jth element $X_j^{(2)}$ of the vector $\mathbf{X}_2$ is defined as a modified 'Coulombic matrix':

$$X_j^{(2)} = \sum_i \sum_{k=i_1}^{i_{n.n.}} \begin{cases} \frac{1}{r_{ik}}, & k = j \\ 0, & k \neq j \end{cases} \tag{24}$$

where the first sum runs over all point mutation amino acids, and the second sum runs over all nearest neighbors of amino acid i. In equation (24), r_{ik} denotes the center-to-center distance between the two amino acids. To take into account the polarity of the amino acids, we introduce a binary vector of dimension $N_p = 3$, such that

$$\mathbf{X}_p^{(1)} = [1\ 0\ 0], \quad \mathbf{X}_p^{(2)} = [0\ 1\ 0], \quad \mathbf{X}_p^{(3)} = [0\ 0\ 1] \tag{25}$$

where $\mathbf{X}^{(1)}$ represents a non-polar amino acid, $\mathbf{X}^{(2)}$ represents an uncharged polar amino acid, and $\mathbf{X}^{(3)}$ represents a charged polar amino acid. In addition, we also added another component to the net vector, which is a real value representing the percentage of the buried part of the amino acids ($\%SASA_{\text{buried}}$), which is defined as the ratio of the buried surface with the solvent accessible surface area of the amino acid in the

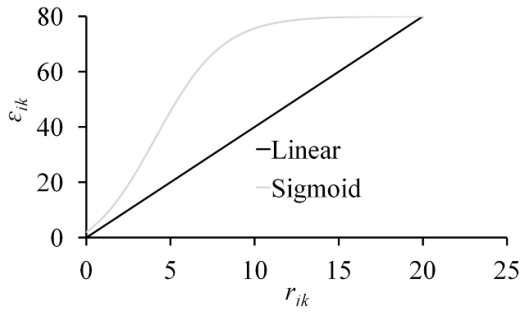

Figure 3. Dielectric constants as a function of the distance between the amino acids for the two cases: Linear, $\varepsilon_{ik} = Dr_{ik}$ with $D = 8$, and dielectric constant given by equation (28) for $\varepsilon_w = 80$ is the dielectric constant of water, $D_0 = 8$, $\kappa = 0.5/(\varepsilon_w + D_0)$, $k = (\varepsilon_w - \varepsilon_p)/(D_0 + \varepsilon_p)$ with $\varepsilon_p = 2$ being the dielectric constant of protein.

protein structure, and it is represented by the vector $\mathbf{X}_4$. Note that vector $\mathbf{X}_4$ can also include other properties, such as the temperature, concentration of the salt and pH value of the experiment; therefore, we can write

$$\mathbf{X}_4 = [\%SASA_{\text{buried}}\ T\ c\ pH\ \cdots] \tag{26}$$

where T, c, and pH are the temperature (in kelvin), concentration (in molar) and pH, respectively.

To determine the descriptor vector of the macromolecule, such as protein, we concatenate the vectors $\mathbf{X}_1$, $\mathbf{X}_2$, $\mathbf{X}_p^{(i)}$ and $\mathbf{X}_4$ into a net descriptor vector $\mathbf{X}$ with length $N = 55$. Note that in the expression given by equation (24), other properties can be encoded. For example, we can encode the dielectric properties in the vicinity of each amino acid in the structure by modifying equation (24) as follows:

$$X_j^{(2)} = \sum_i \sum_{k=i_1}^{i_{n.n.}} \begin{cases} \frac{1}{\varepsilon_{ik} r_{ik}}, & k = j \\ 0, & k \neq j \end{cases} \tag{27}$$

where ε_{ik} is the dielectric constant of the environment in the vicinity of the mutated amino acid i, which can be taken a simple distant dependent dielectric constant between the amino acid i and its nearest neighbor k: $\varepsilon_{ik} = Dr_{ik}$, where D is a constant, or even other complicated distance dependence functions [58, 59]. However, in this work, other more complicated distance dependent dielectric constant is considered, such as the sigmoidal function [58, 59]:

$$\varepsilon_{ik} = \frac{\varepsilon_w + D_0}{1 + k \exp\left(-\kappa\left(\varepsilon_w + D_0\right) r_{ik}\right)} - D_0. \tag{28}$$

Here, r_{ik} is the distance between two amino acids, respectively, i and k, $\varepsilon_w = 80$ is the dielectric constant of water, $D_0 = 8$, $\kappa = 0.5/(\varepsilon_w + D_0)$, $k = \left(\varepsilon_w - \varepsilon_p\right)/\left(D_0 + \varepsilon_p\right)$ with $\varepsilon_p = 2$ being the dielectric constant of protein. A plot of ε_{ik} versus the distance r_{ik} is presented in figure 3 for both simple function of the distance of dielectric constant and sigmoidal distance dependence function of the dielectric constant. Here, sigmoidal function gives a smooth variation of the dielectric constant screening the electrostatic interactions from 2 (which is the dielectric constant of the internal protein) to 80 (which is the dielectric constant of bulk water), as shown in figure 3.

Note that these fingerprints of the structures are rotation and translation invariant. Furthermore, as a sequence of the amino acids in a macromolecular structure, the protein data bank, RCSB PDB [60], can be used that is unique. Therefore, the descriptor vector $\mathbf{X}$ is an exclusive representation of a macromolecule in a dataset. Also, the descriptor vector $\mathbf{X}$ has the same length for any set of the macromolecules used as input.

It is important to note that if the chemical sample space of the input descriptor vector becomes quite large, then the principal components analysis [61] can be performed to reduce the degrees of freedom.

3. Datasets

In this study, we used four different databases. The first database contains 642 small organic molecules, for which we know the experimental hydration free energies [26]. Note that this database has also been subject to the previous studies [23, 47]. The second database contains 1475 experimental pKa values of ionizable groups in 192 proteins, both wild type (153 proteins) and mutated (39 proteins) [27–30]. The third database has 2693 experimental values of the Gibbs free energy changes in 14 mutant proteins [31, 32]. The last database has 7101 quantum mechanics heat of formation calculations [6] (and the references therein), the

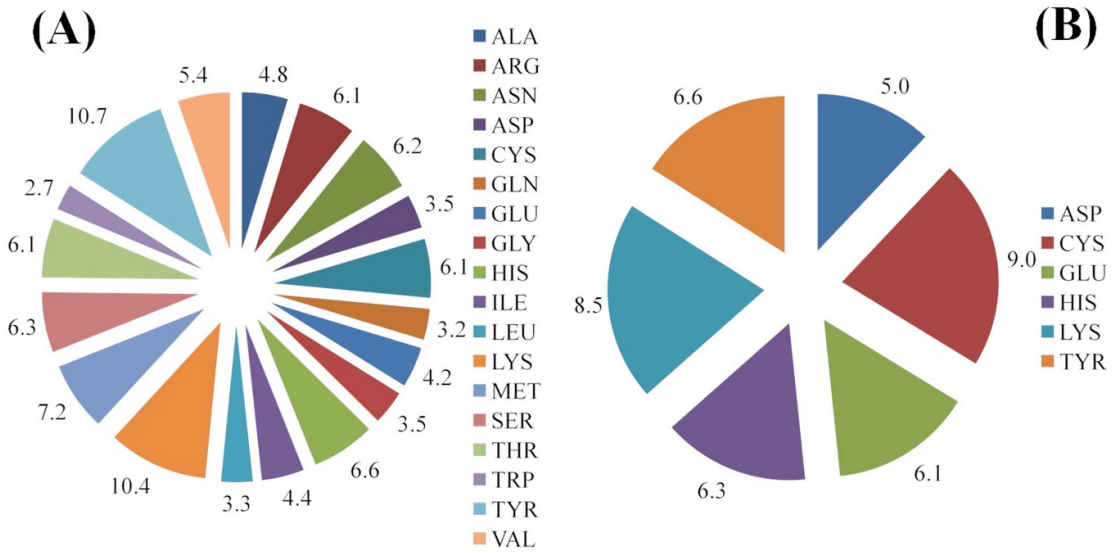

Figure 4. Average experimental pKa for each residue in the wild-type (A) and (B) mutants proteins of the database.

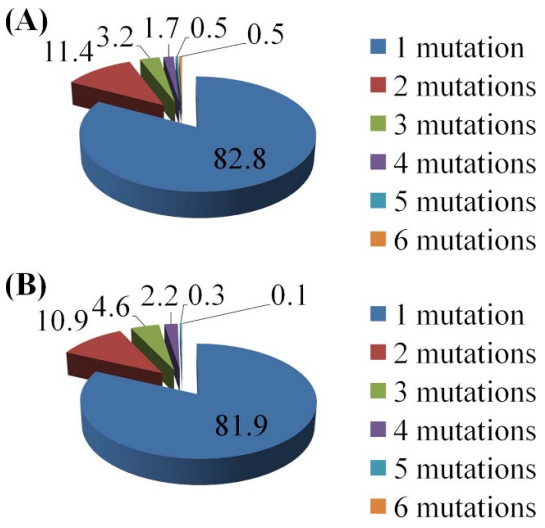

Figure 5. Percentage of each type of mutation in database for which we know either $\Delta\Delta G$ (A) or $\Delta\Delta G^{H_2O}$ (B). 1: single mutation; 2: double mutations, and so on, 6: six mutations.

QM7, which is a subset of the *GDB13* molecules, optimized at the quantum mechanics level with the Perdew–Burke–Ernzerhof hybrid functional (PBE0) [14].

Figure 4 shows the distribution of the average experimental pKa values for each residue in the wild-type and mutated proteins within the database.

In figure 5, we show the distribution of the percentage of each type of mutation in the database for which we know either $\Delta\Delta G$ or $\Delta\Delta G^{H_2O}$, namely 1: single mutation; 2: double mutations, and so on, up to 6: six mutations.

All the data are prepared and optimized in advance using the AMBER force field for proteins and nucleic acids ff99SBnmr [44, 45] and generalized AMBER force field for small organic molecules [46] (as in [26, 47]) using CHARMM macromolecular computer simulation program [42]. The BSANN code performing the optimization and prediction is written in Python computer programming language. The small molecules' descriptor vectors are based on the Coulomb matrix and the sum over bond properties. For the macromolecular systems, they consider the chemical-physical fingerprints of the region in the vicinity of each amino acid.

Software implementing the methods discussed in this study is free for download from the website http://hkamberajibu.wikidot.com/machine-learning. Also, one can access the databases of all molecular

structure and topology files used in our calculations as prepared with a general AMBER force field from the same site.

3.1. Data analysis

For that, consider a method to learn a function from a finite dataset $\mathcal{D}$ of input–output pairs, namely $(\mathbf{X}, \mathbf{Y})$, where $\mathbf{X}$ is the feature descriptor input vector for each atom and $\mathbf{Y}$ is the reference output vector for each atom, such as the hydration free energy, change on the Gibbs free energy, the heat of formation, amino acid pKa, and so on. The dataset is then split into a training dataset $\mathcal{D}_{\text{train}}$ used for learning (or gaining experience) and a validation dataset $\mathcal{D}_{\text{valid}}$ used for testing the knowledge, such that

$$\mathcal{D} = \mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{valid}}. \tag{29}$$

In this study, we will discuss the training dataset's ability to optimize the parameters of a supervised ANN as a function of the size of the training dataset. We aim to estimate the average bootstrapping confidence interval of error that can be used to predict any new test data.

It is important to note that an optimal average range is believed to provide the highest confidence level, within which most of the predicated values lie in. In the following discussion, we will use the term 'match' for such cases, that is, when the prediction interval of error coincides with that provided by the experimental value using statistical confidence of 95%, which is verified using the *Student t-Distribution*.

To determine the average confidence interval, we used the following statistical justification [62]. Suppose that there are N_{train} training data points, and n is the number of the neural networks defining the bootstrapping model given above. Then, the $100(1-\alpha)\%$ bootstrapping confidence interval of the average value for each data point prediction is given by

$$\left(\bar{Y}_i - c_{i,u}\frac{\sigma_i}{\sqrt{n}}, \bar{Y}_i - c_{i,l}\frac{\sigma_i}{\sqrt{n}}\right) \tag{30}$$

where σ_i is the unbiased standard deviation obtained from the bootstrapping data distribution. $c_{i,u}$ and $c_{i,l}$ are the upper and lower critical values, respectively, determined from the empirical distribution function F of the bootstrapping dataset as

$$\begin{aligned} F(t_i = c_{i,u}) &= 1 - \alpha/2 \\ F(t_i = c_{i,l}) &= \alpha/2 \end{aligned} \tag{31}$$

where t_i is the studentized bootstrapping random variable obtained from the data points of the ith prediction [62]. Then, the average statistical error from all prediction N_{train} data points is calculated using the chain rule as follows:

$$c_u\sigma = \sqrt{\sum_{i=1}^{N_{\text{train}}} \left(c_{i,u}\sigma_i\right)^2} \tag{32}$$

$$c_l\sigma = \sqrt{\sum_{i=1}^{N_{\text{train}}} \left(c_{i,l}\sigma_i\right)^2} \tag{33}$$

where it is assumed that the statistical errors obtained for each of the training data points are independent, which is indeed the case. Then, the average $100(1-\alpha)\%$ bootstrapping confidence interval of the average value for each data point prediction is defined as

$$\left(\bar{Y}_i - c_u\frac{\sigma}{\sqrt{n}}, \bar{Y}_i - c_l\frac{\sigma}{\sqrt{n}}\right). \tag{34}$$

Equation (34) is used to determine the upper and lower bound of the average values in all our predictions shown in the graphs of the following discussions. Note that in practice, the Student t-distribution can be used to approximate the distribution of the bootstrapping dataset, and hence $c_l = -c_u = t_{n-1,\alpha/2}$, which is the critical value for the t-distribution. In this study, the confidence level was $\alpha = 0.05$.

To compare the state of the art, we introduce in the following another error model [23]. According to this model, the deviation between the predicted values and experimental reference, for each measurement, is a sum of four independent terms: The first term is the uncertainty with known variance from the calculation; the second is the experimental error; the third is a general error due to simulation settings; the last term is an error representing the presence of different features in the system (such as the error due to the force field

parameters related to a particular feature.) Each of these terms is represented by a normal stochastic distribution, characterized by the mean (identifying the systematic error) and variance (identifying random error) for each new measurement that may or may not be in the training dataset. Thus, the total mean and variance are given as [23]

$$\mu = \mu_{\text{general}} + \sum_i N_i \mu_{\text{feature}}$$
$$\sigma^2 = \sigma^2_{FE} + \sigma^2_{\text{exp}} + \sigma^2_{\text{general}} + \sum_i N_i \sigma^2_{\text{feature}}. \tag{35}$$

Each term in equation (35) represents an average value calculated over all bootstrap iterations [23]. Comparing the state of arts of both methods, in our error model introduced here, we do not try to separate the systematic and random errors, as it is done on the error model in [23]. Secondly, we did not add the experimental error in our total uncertainty because we did not have the experimental error values for all datasets. It is interesting to note that the error model given in [23] is based on the uncertainty estimation by maximizing the likelihood function (given as the product of Gaussian probability densities) and the uncertainly on the most-likely average calculated using the bootstrapping technique. However, a rigorous mathematical formalism can also be derived for the calculation of the mean and variance of the predicted value using the Gaussian process to do Bayesian training of the ANN [24] under the assumption of the infinite width of the neural networks, as shown above.

In all our calculations, predicted values using the bootstrapping technique depend on the average of the some quantity (μ) over a finite number (M) of the ML setups. Following [50], we take this average as an integral of μ weighted with probability distribution of μ, $P(\mu)$:

$$A_{\text{pred}} = \int \mu P(\mu) d\mu. \tag{36}$$

Assuming that μ are functions of identically distributed random variables, then μ have a Gaussian distribution:

$$P(\mu) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(\mu - \langle\mu\rangle)^2}{2\sigma^2}\right). \tag{37}$$

If we require that σ to be comparable with $k_B T$, we write

$$\sigma^2 = \frac{\langle\mu^2\rangle - \langle\mu\rangle^2}{(k_B T)^2} = \frac{Var(\mu)}{(k_B T)^2} \tag{38}$$

where Var(μ) denotes the variance of μ. Note that $P(\mu)$, in practice, may also be slightly different from a Gaussian distribution, but close to a Gaussian-like shape. Combining equations (36) and (37), we get

$$A_{\text{pred}} = \langle\mu\rangle - \frac{1}{2(k_B T)^2} \text{Var}(\mu) \tag{39}$$

where the first term is the average of μ measured using bootstrapping technique and the second term depends on the fluctuations of μ. While the first term can be positive or negative, the second one is always negative. Therefore, the accuracy in measuring A_{pred} depends on the balance between these two terms. We can also write equation (39) as

$$A_{\text{pred}} = \langle\mu\rangle - \gamma\sigma \tag{40}$$

where $\gamma = \sqrt{\text{Var}(\mu)}/(2k_B T)$. Thus, for $\sqrt{\text{Var}(\mu)} = nk_B T$, where n integer number, we get

$$A_{\text{pred}} = \langle\mu\rangle - \frac{n}{2}\sigma. \tag{41}$$

For $n = 1$ or $\sqrt{\text{Var}(\mu)} = k_B T$), 95% of the values of μ fall in the region $\langle\mu\rangle \pm 2\sigma$, and from equation (41) we can see that $A_{\text{pred}} = \langle\mu\rangle - \sigma/2$, which falls inside the region where most of the μ are sampled. That is, the bootstrapping approach in ML will result in accurate measure of A_{pred}. On the other hand, for $n > 4$ or $\sqrt{\text{Var}(\mu)} > 4k_B T$), more than 97% of the μ values fall in the region $\langle\mu\rangle \pm 2\sigma$, and from equation (41) we can see that $A_{\text{pred}} < \langle\mu\rangle - 2\sigma$, which falls outside the region where most of the μ are sampled; hence, in this case inaccurate measure of A_{pred} will be produced due to the sampling inefficiency. However, there is no physical reason to assume that the fluctuations of μ given by equation (38) should be comparable to $k_B T$. That

requirement is often satisfied by the computer simulations (such as molecular dynamics and Monte Carlo), resulting in small statistical errors, as will be illustrated here when comparing the predicted values using the computer simulations and those predicted by ML. In fact, this is because of the assumption that $P(\mu)$ in equation (37) obeys to Maxwell–Boltzmann probability distribution.

3.2. Feature selection algorithm

When the training data size is substantially larger than the number of features (dimension of the description feature vector), then the distribution of the properties over the range of the features is reasonably accurate. However, this is not always the case; for instance, in the hydration free energy database, the number of used data points is 415, and the number of features is over 1000 features. That is, in a training dataset, there might often be a tiny fraction of the data with non-zero values of some features. Therefore, the computation of the joint probability distribution over all features will be inaccurate.

Estimating the marginal distribution of each term in a classified dataset (training and validation dataset) instead of the joint distribution of all parts is suggested to improve the accuracy significantly [63]. We identify for a limited size dataset whether a given feature appears more frequently in one class of datasets than another. For that, we split the data into two categories, namely the training and validation datasets.

In the following, we will describe the χ^2-test algorithm [63] used to select the features from a collection of observed data in a database. Suppose we have N training data samples from each class, and f is a fixed feature. Here, we use standard statistics to test if two quantities are significantly correlated. For each feature f, we label 0 and 1 the two classes, namely the training and validation dataset, respectively. We denote $n_{i,0}$ the number of data in the class i not containing the feature f, and $n_{i,1}$ the number of data in the class i containing the feature f. In this way, we calculate a 2×2 matrix $\mathbf{K}$ with elements n_{CI_f} such that

$$\mathbf{K} = \begin{bmatrix} n_{00} & n_{01} \\ n_{10} & n_{11} \end{bmatrix} \tag{42}$$

where C and I_f are two indicator random variables, defined as

$$C = \begin{cases} 1 & \text{randomly chosen molecule belongs to validation dataset} \\ 0 & \text{randomly chosen molecule belongs to training dataset} \end{cases} \tag{43}$$

and

$$I_f = \begin{cases} 1 & \text{randomly chosen molecule contains the feature } f \\ 0 & \text{randomly chosen molecule does not contain the feature } f. \end{cases} \tag{44}$$

Therefore, n_{ij} is a random variable denoting the number of observations with $C = i$ and $I_f = j$. The algorithm verifies if the random variables are independent or not. Note that

$$N = \sum_{i=0,1} \sum_{j=0,1} n_{ij}. \tag{45}$$

The marginal probability distributions are given as [63]

$$P(C=0) = \frac{n_{00} + n_{01}}{N} \tag{46}$$

$$P(C=1) = \frac{n_{10} + n_{11}}{N} \tag{47}$$

$$P(I_f=0) = \frac{n_{00} + n_{10}}{N} \tag{48}$$

$$P(I_f=1) = \frac{n_{01} + n_{11}}{N}. \tag{49}$$

The joint probability distribution of the random variables C and I_f is

$$P(C=i, I_f=j) = \frac{n_{ij}}{N} \tag{50}$$

which implies that $n_{ij} = NP(C=i, I_f=j)$. Furthermore, the condition of the independence requires that

$$P(C=i, I_f=j) = P(C=i)P(I_f=j). \tag{51}$$

Table 1. Pearson coefficient, MAE (in kcal mol^{-1}), RMSE (in kcal mol^{-1}) and Matches (in %) for different numbers of training data. Predictions are based on the neural network parameters optimized using only the training data set.

Pearson	MAE (kcal mol^{-1})	RMSE (kcal mol^{-1})	Matches (%)	Set	Size
0.964	0.539	1.135	97.3	All dataset	415
0.998	0.168	0.231	100.0	Training dataset	300
0.883	1.507	2.124	90.4	Validation dataset	115
0.970	0.442	1.040	93.7	All dataset	415
0.999	0.170	0.228	97.3	Training dataset	330
0.852	1.495	2.254	80.0	Validation dataset	85
0.984	0.353	0.742	94.9	All dataset	415
0.998	0.197	0.266	97.4	Training dataset	350
0.886	1.192	1.770	81.5	Validation dataset	65
0.988	0.272	0.628	92.8	All dataset	415
0.998	0.176	0.258	93.4	Training dataset	380
0.878	1.313	1.988	85.7	Validation dataset	35

In other words, the condition of the independence indicates that a feature is observed regardless of the classified dataset, either training or validation dataset. The χ^2-test can be used to measure the deviations between the two distributions, namely $P(C=i, I_f=j)$ and the product $P(C=i)P(I_f=j)$ [63]:

$$\chi^2 = N \sum_{i=0,1} \sum_{j=0,1} \frac{\left(\frac{n_{ij}}{N} - P(C=i)P(I_f=j)\right)^2}{P(C=i)P(I_f=j)}. \tag{52}$$

The values of χ^2 is a measure of the confidence of the independence condition; that is, smaller the values of χ^2 higher the confidence. Next step, we sorted the values of χ^2_f, obtained for each feature f, in decreasing order of their values, and considered for training of the dataset the features with the highest values of χ^2.

Also, the mathematical model presented in [64, 65] is employed to keep high diversity of the observed properties (such as hydration free energies, pKa, heats of formation, and changes on Gibbs free energies) in the classified datasets.

4. Results

In this section, we show some results of the predictions using different datasets.

4.1. The hydration free energy database

Table 1 summarizes the Pearson coefficient, mean average error (MAE) (in kcal mol^{-1}), root mean square error (RMSE) (in kcal mol^{-1}) and Matches (in %) for different lengths of training dataset. Predictions are based on the neural network parameters optimized using only the training dataset. Our results show that an MAE value as small as 1.192 kcal mol^{-1} is obtained in the validation dataset, corresponding to a Pearson coefficient of 0.886 and an RMSE of 1.770 kcal mol^{-1}, for a size of the training dataset of 350 molecules (or equivalently, 84% of the overall dataset). For that case, the percentage of matches between the predicted and experimental values is 81.5% with a 95% statistical confidence.

Figure 6 presents the scatter plots of the experimental hydration free energies and predicted ones for two different training data sizes, $N=330$ and $N=380$ molecules. Errors are calculated using the bootstrapping method, and the straight line represents the function $f(x)=x$. Besides, we have indicated the 95% bootstrapping confidence interval of error with parallel lines. Results show that when a training dataset of size 380 molecules is used to train the network, more matches are found between the predicted and the experimental values compared to the network trained using a dataset of size 330 molecules. That is because the distribution of the data points of the training dataset influences the input data's topology. Thus, a large input dataset can reveal more insights into the structure of the data distribution.

We also used a larger dataset of 630 molecules to train and test the neural network, as shown in figure 7. For this case, we used 560 (equivalent to 89% of the total size of the dataset) data points to train the neural network, and the rest about 70 data points for validation (or equivalently, about 11% of the entire dataset). Our results show an improvement of the predictions when compared with the smaller dataset, as used above, that can be shown by our results presented in figure 7, indicating that all the validation data points lie inside the 95% bootstrapping confidence interval. For this dataset, the following values of MAE, RMSE, and Pearson correlation coefficient R were obtained. For the training data, MAE = 0.208 kcal mol^{-1}, RMSE = 0.286 kcal mol^{-1} and $R=0.997$; for the validation data, MAE = 0.732 kcal mol^{-1}, RMSE = 1.050 kcal mol^{-1} and $R=0.945$.

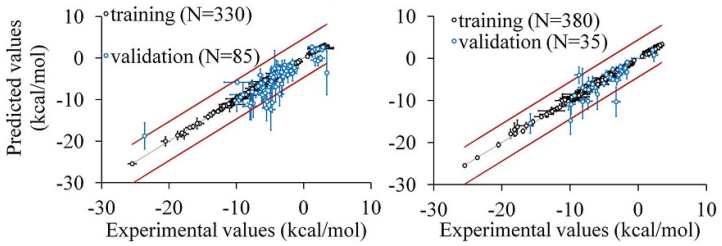

Figure 6. Parity plots between the experimental hydration free energy and predicted free energies. Errors are calculated using the bootstrapping method and the straight line represents the function $f(x) = x$. The dataset size was 415 molecules. The straight red lines represent the boundary of the 95% bootstrapping confidence interval of error.

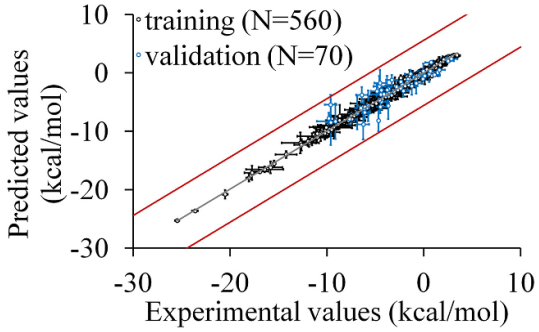

Figure 7. Parity plots between the experimental hydration free energy and predicted free energies. Errors are calculated using the bootstrapping method and the straight line represents the function $f(x) = x$. The dataset size was 630 molecules. The straight red lines represent the boundary of the 95% bootstrapping confidence interval of error. For the training dataset: MAE = 0.208 kcal mol^{-1}, RMSE = 0.286 kcal mol^{-1} and $R = 0.997$; for the validation dataset: MAE = 0.732 kcal mol^{-1}, RMSE = 1.050 kcal mol^{-1} and $R = 0.945$.

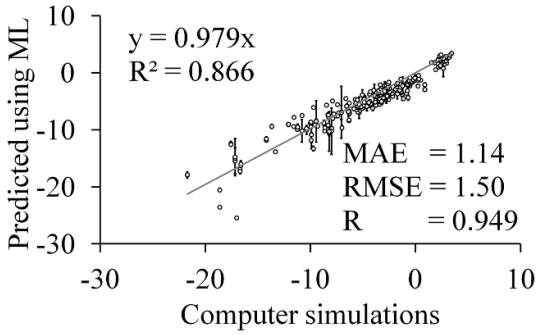

Figure 8. Parity plots between the predicted hydration free energies using computer simulation techniques [23, 26, 66] (such as molecular dynamics and Monte Carlo) and predicted hydration free energies using ML approach. Errors are calculated using the bootstrapping method and the straight line represents the fitting line. The dataset size was 415 molecules. For this dataset: MAE= 1.140 kcal mol^{-1}, RMSE= 1.499 kcal mol^{-1} and $R = 0.949$.

We also compared hydration free energies predicted using computer simulation techniques (such as molecular dynamics and Monte Carlo) and those predicted using the ML approach. Predicted computer simulation hydration free energies are obtained from [23, 26, 66]. The results are shown graphically in figure 8 for a dataset of 415 molecules, where the training dataset size was 380 molecules, and the validation dataset size was 35 molecules. For this dataset: MAE = 1.140 kcal mol^{-1}, RMSE = 1.499 kcal mol^{-1} and $R = 0.949$. Our results indicate an excellent correlation between the predicted values by both methods. One can see some discrepancies for large values of the hydration free energies; however, we do not have many experimental measurements. We also compared the computer simulation values of hydration free energies against the experimental values and found the following: MAE = 1.188 kcal mol^{-1}, RMSE = 1.588 kcal mol^{-1}, and $R = 0.941$. On the other hand, the comparison between the predicted hydration free energies using the method presented in this study and experimental values for the same dataset of molecules gives MAE = 0.272 kcal mol^{-1}, RMSE = 0.628 kcal mol^{-1}, and Pearson correlation coefficient of $R = 0.988$ (see also table 1).

Table 2. Pearson coefficient, MAE (in kcal mol^{-1}), RMSE (in kcal mol^{-1}) and Matches (in %) for different numbers of training data. Predictions are based on the neural network parameters optimized using only the training dataset. The sizes of the datasets are $N = 953$ and $N = 1240$ pKa calculations.

		Training data				Validation data			
N_{train}	% of data	Matches %	MAE kcal mol^{-1}	RMSE kcal mol^{-1}	Pearson	Matches %	MAE kcal mol^{-1}	RMSE kcal mol^{-1}	Pearson
				$N = 953$					
750	79	92	0.104	0.164	0.998	74	0.419	0.742	0.951
800	84	91	0.144	0.238	0.996	78	0.310	0.565	0.961
850	89	89	0.118	0.176	0.997	82	0.269	0.416	0.992
				$N = 1240$					
800	65	99	0.034	0.058	1.000	81	0.451	0.843	0.940
1000	81	99	0.037	0.075	1.000	83	0.413	0.738	0.962
1100	89	98	0.038	0.073	1.000	84	0.295	0.485	0.983

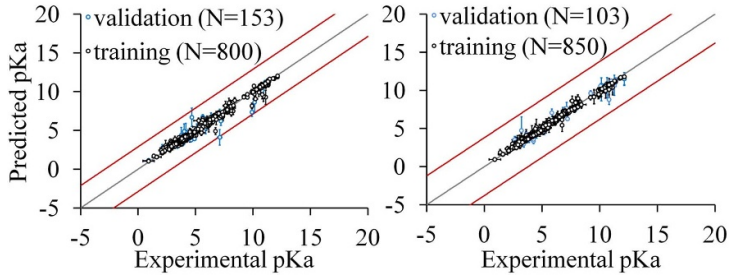

Figure 9. Parity plots between the experimental pKa values and predicted pKa values. Errors are calculated using the bootstrapping method and the straight line represents the function $f(x) = x$. The size of the dataset was $N = 953$. The straight red lines represent the boundary of the 95% bootstrapping confidence interval of error.

4.2. The pKa of amino acids in proteins

Table 2 presents the results of the predictions on both the training and validation datasets. The size of the entire dataset is $N = 953$ pKa calculations. Our results indicate that the Pearson correlation coefficient is above 0.95 in both training and validation datasets. The smallest MAE and RMSE were 0.104 kcal mol^{-1} on the training dataset and 0.164 kcal mol^{-1}, respectively. For the validation dataset, the smallest MAE and RMSE values were 0.269 and 0.416 kcal mol^{-1}, respectively, obtained for the size of the training dataset about 89% of the entire dataset. Besides, the matches between the experimental and predicted values of the pKa on the validation dataset are 82% with a statistical confidence of 95%.

Figure 9 presents the predicted and experimental pKa values graphically as a scatter plot. Also, we show the average 95% bootstrapping confidence interval of error of the predicted values within the training dataset. The scenarios are created for two training data sizes, respectively, 84% and 89% of the entire dataset. Interestingly, our results indicate that almost all the validation data prediction of pKa values lies inside the average bootstrapping confidence interval of error when 89% of the dataset is used in training the neural network.

To check the influence of the dataset size on the learning efficiency from the data, we optimized the neural network for a larger dataset of 1240 pKa calculations. We implemented three different training datasets for optimizing the neural network to check the influence of the size of the training data set and the length of the entire dataset, which determine the topology of the input data. We notice that the dataset with $N_{train} = 1100$ (which is about 89% of the entire dataset) data points for training provided the best optimization. The results are summarized in table 2, and plotted in figure 10. Our results show that the MAE decreases about twice for the same percentage of data in the training set taken from a smaller dataset, namely MAE= 0.038 kcal mol^{-1}; a smaller RMSE is also obtained in this dataset of about 0.073 kcal mol^{-1}. Furthermore, it can be seen that the percentage of matches on the training dataset increases to 95% with a perfect Pearson correlation between the experimental and the predicted of exactly $R = 1.000$. Moreover, our results show (figure 9) that the average 95% bootstrapping confidence interval of error is larger, and all the predicted values of pKa of the validation set lie within the bootstrapping confidence interval.

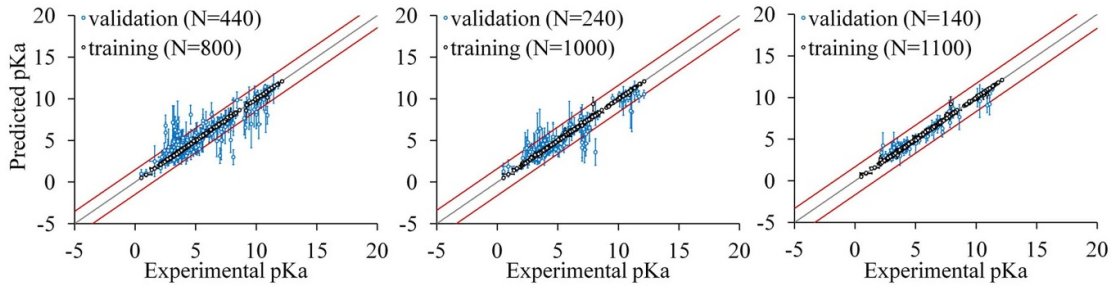

Figure 10. Parity plots between the experimental pKa values and predicted pKa values. Errors are calculated using the bootstrapping method and the straight line represents the function $f(x) = x$. The size of the dataset was $N = 1240$. The straight red lines represent the boundary of the 95% bootstrapping confidence interval of error.

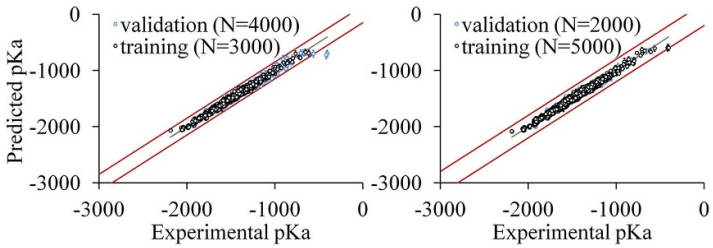

Figure 11. Parity plots between the quantum mechanics heat of formation values and predicted values. Errors are calculated using the bootstrapping method and the straight lines represent the boundary of the 95% bootstrapping confidence interval of error. Quantum mechanics heat of formation is calculated using the PBE0 method.

4.3. Quantum mechanics database

The results of the predicted values of the heat of formation from the quantum mechanics calculations using the PBE0 method are shown in figure 11. The size of the dataset is 7000 molecules. We used two different sets of the training data with lengths 3000 (or equivalently 43% of the size of the dataset) and 5000 (or equivalently, approximately 71% of the entire dataset). Our results indicate an excellent performance of the predictions using the optimized neural network on the validation data; using just 43% of the entire dataset for the training of the neural network, and the test on the validation data show only a few data are outside the predicted average 95% bootstrapping confidence interval of error, and when 71% of the total data are used for training, then all the tested calculations from the validation dataset lie inside the 95% bootstrapping confidence interval.

As intuitively expected, our results indicate that the data's length influences the optimization of the neural network and the knowledge in the dataset. That explains that the topology of the input dataset plays an essential role in the experience gained from the training of the neural network. In the following discussion, we argue that this should be related to the topology of the input data.

4.4. Thermodynamics of proteins

Here, we show the results of the predicted changes in the Gibbs free energy due to mutations for 1063 mutations in different mutant proteins. The results of predictions using ML are shown in figure 12. Our results support the findings that increasing the training dataset's size improves the bootstrapping confidence interval of error, hence the prediction probability. For example, for a training dataset of length 1000 points (or, equivalently 94% of the total dataset), all the predicted values of the Gibbs free energy values fall inside the 95% bootstrapping confidence interval of error (see also figure 12).

In table 3, we have summarized the calculation results of the Pearson coefficient, MAE (in kcal mol^{-1}), RMSE (in kcal mol^{-1}), and Matches (in %) for different numbers of training data. Predictions are based on the neural network parameters optimized using only the training dataset. As expected, for the training dataset of size 94% of the entire dataset, significant improvement in predicting the changes in the Gibbs free energy is obtained. For example, the Pearson correlation coefficient between the experimental and predicted values for the testing dataset is 0.925. The MAE is 0.488 kcal mol^{-1}, RMSE is 0.665 kcal mol^{-1}, and the number of matches is 71%.

We have also compared our algorithm's performance (BSANN) with other ML algorithms, such as the so-called DeepDDG [67], used for predicting the change in Gibbs free energy due to the point mutations in

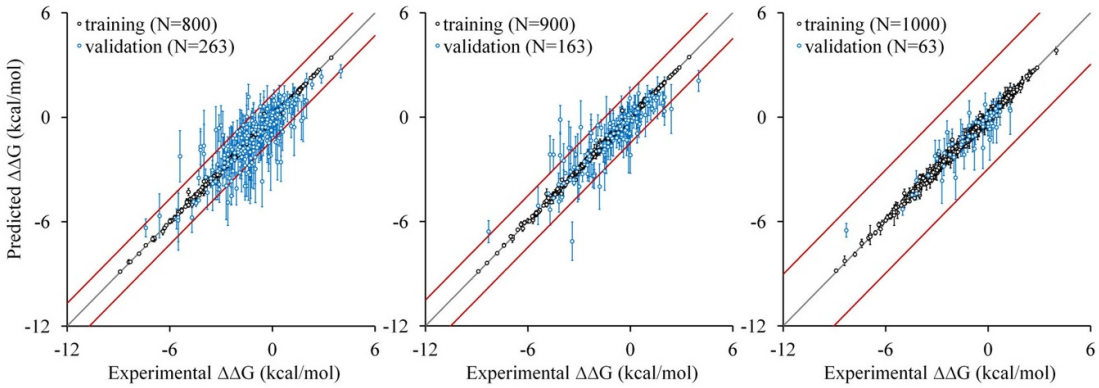

Figure 12. Parity plots between the experimental changes on the Gibbs free energy values and predicted values. Errors are calculated using the bootstrapping method, and the straight lines represent the boundary of the 95% bootstrapping confidence interval of error.

Table 3. Pearson coefficient, MAE (in kcal mol^{-1}), RMSE (in kcal mol^{-1}) and Matches (in %) for different numbers of training data. Predictions are based on the neural network parameters optimized using only the training dataset. The sizes of the dataset is $N = 1063$ Gibbs free energy calculations.

		Training data				Validation data			
N_{train}	% of data	Matches %	MAE kcal mol^{-1}	RMSE kcal mol^{-1}	Pearson	Matches %	MAE kcal mol^{-1}	RMSE kcal mol^{-1}	Pearson
800	75	86	0.050	0.081	0.999	65	0.725	1.029	0.834
900	85	86	0.049	0.078	0.999	60	0.610	0.898	0.875
1000	94	89	0.090	0.125	0.998	71	0.488	0.665	0.925

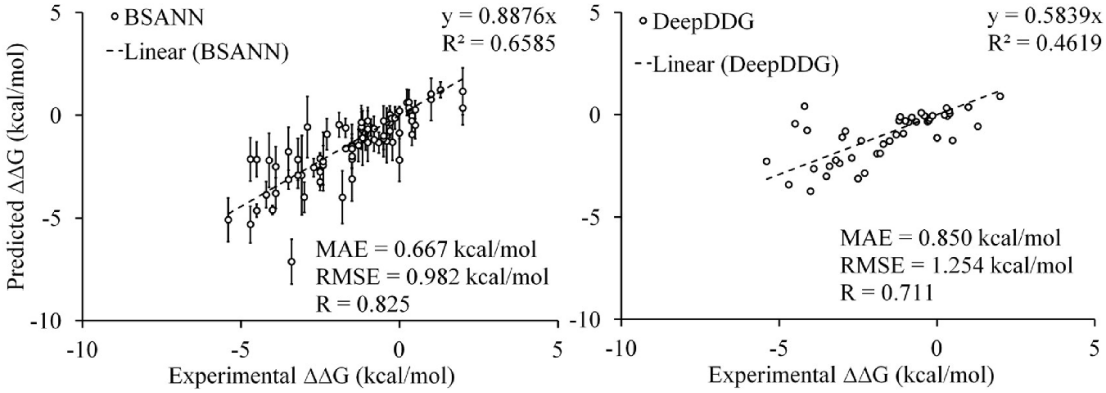

Figure 13. Blind prediction of $\Delta\Delta G$ using the trained BSANN algorithm, presented in this study, and DeepDDG algorithm [67].

proteins. The computations using the DeepDDG algorithm are performed using online web-server [67]. Figure 13 shows the results graphically. It is interesting to note that none of the test dataset points is used to train the BSANN algorithm, which is a blind prediction. In this work, we considered mutations in Barnase wildtype structure protein [68] (PDB Id: 1BNI) and Bacteriophage T4 Lysozyme [69] (PDB Id: 2LZM). In total, we used about 80 amino acid single mutations of the wild-type structures for the prediction test comparisons. Here, we used the BSANN algorithm trained with 900 mutations dataset. We have also calculated the MAE, RMSE, and Pearson correlation coefficient, as shown in figure 13. Our results indicate that BSANN performs better than DeepDDG in all metrics. It is interesting to note that the fitting straight line of the predicted $\Delta\Delta G$ using BSANN algorithm is close to the baseline ($y = x$), namely $y = 0.8876x$, compared to the DeepDDG algorithm, $y = 0.5839x$. At this point, it is difficult to say which algorithm is generally more predictive because we have not used the same training dataset to optimize the algorithms and the input description vectors are not the same for both algorithms. However, our results of the comparison indicate that BSANN is a promising ML algorithm for predicting changes in proteins' stability due to the point mutations.

5. Discussion

In this work, we intend to establish a methodology for an automated machine-like supervised learning approach for predicting different (macro)molecular properties. In particular, for a training dataset of molecules, $\mathcal{D}$ created of N_{train} pairs (X_i, Y_i) for $i = 1, 2, \ldots, N_{\text{train}}$, where the vector $\mathbf{X}$ denote the feature descriptor vector of dimension $N_{\text{features}} \times N_{\text{train}}$ and $\mathbf{Y}$ of dimensions $N_{\text{properties}} \times N_{\text{train}}$ the reference values. That aims to obtain an estimate of the probability $P(\mathbf{Y}^\star|\mathcal{D}, \mathbf{X}^\star)$ to predict the output $\mathbf{Y}^\star = \left(Y_1^\star, Y_2^\star, \ldots, Y_{\text{properties}}^\star\right)$ of an optimized neural network for any input test data-point $\mathbf{X}^\star = (X_1^\star, X_2^\star, \ldots, X_{\text{features}}^\star)$. This calculation is now an automated process since the black box is trained to predict the output value described by the probability $P(\mathbf{Y}^\star|\mathcal{D}, \mathbf{X}^\star)$, which makes the predictions of the physical properties an efficient automation process.

However, the accuracy in estimation of $P(\mathbf{Y}^\star|\mathcal{D}, \mathbf{X}^\star)$ is a data-driven process, and the prediction of $\mathbf{Y}^\star$ depends on the used training dataset. In particular, it depends on the diversity of the feature descriptors for the dataset of molecules, that is, the amount of N_{features}. Besides, it depends on the size of the dataset, N_{train}. Both the diversity of the feature descriptors of the compounds and the size of the dataset are interconnected; however, a large size dataset is practically difficult to be established due to the lack of experimental data, and quantum mechanics data may be expensive to obtain. Besides, increasing the dimensions of the input feature descriptor vector, $N_{\text{features}} \times N_{\text{train}}$, is equivalent to increasing the amount of information processed by the computer. Based on a mass-energy-information equivalence principle [70–72], it can be expected to increase the amount of the irreversible heat generated during the data processing. That is related to another computer term, namely 'big-data' processing. In [72] has been introduced a formalism trying to quantify the *weight* of the big-data information by using a physical interpretation of information and the principle of the equivalence mass-energy-information. That allows establishing an equivalence between the (necessary) amount of the input training data for accurate prediction of $P(\mathbf{Y}^\star|\mathcal{D}, \mathbf{X}^\star)$ and the limit of the amount of the information that can be processed by a computer considering the heat generated during the computer processing, and so the amount of the external work necessary to process that big-data of information by a computer.

Furthermore, to characterize the stability of the input training data, a rigorous mathematical model can be employed, introduced in [50]. It is important to note that the feature descriptor vectors for each compound are considered time-invariant in our discussion above. Thus they represent only two- and three-dimensional feature descriptors of the (macro)molecules. However, higher feature descriptor vectors can also be constructed, such as four-dimensional feature descriptor vectors, where the time is the fourth component. In that case, to build the three-dimension part of the feature descriptor vectors, different conformations of the compounds can be considered, for example, as generated from the molecular dynamics simulations. In that case, one can map the three-dimensional configurations of the compounds produced from the simulations into a three-dimensional grid, where the centers of the grid points will represent the average positions of each atom obtained from its fluctuations after the configurations are aligned to remove the overall translation and rotation motion of the compounds. Therefore, the feature descriptor vectors derived from these average structures mapped into a three-dimension grid are translation and rotation invariant. A review of such higher-dimensional descriptor vectors is discussed in [73].

6. Conclusions

This study presented a methodology for the automation of (macro)molecular properties predictions using a new algorithm integrated into a ML approach. We gave the results of predictions for four different databases of both molecular and macromolecular systems properties. Each dataset contains the results of the experimental values, including the error when provided in the literature.

Furthermore, we showed how to create an input descriptor vector for a supervised ANN for small organic molecules and macromolecular systems. The descriptor features included both the two-dimensional (macro)molecular fingerprints and the three-dimensional structure of the systems. Moreover, we presented a statistical approach of how to estimate the bootstrapping confidence interval of the error.

The application of that new algorithm in our data indicated that the topological chemical spaces extended by the molecular description vectors on the relevance or irrelevance of perturbations in the data analysis are crucial. Furthermore, we envision that the persistence homology can be considered necessary as the renormalization group theory in statistical physics when applied to equilibrium phenomena to understand the relevant or irrelevant interactions. In this analogy, the resolution scaling factor on the topological data analysis can be considered similar to the characteristic correlation length scale that determines the judgment of the strong interactions and correlations renormalization group theory. Besides,

we introduced a mathematical framework for representing the input dataset to characterize the stability of the data and a new algorithm for feature selection.

Besides, the results of the comparison between the DeepDDG and BSANN algorithms indicated that BSANN could be a good ML algorithm for predicting changes in proteins' stability due to the point mutations.

Associated content

Software

The software (written in Python programming language) is published on the following website: http://hkamberajibu.wikidot.com/machine-learning.

Datasets

The three-dimensional structures are included as optimized with the AMBER force field using the CHARMM program: http://hkamberajibu.wikidot.com/machine-learning.

Databases

The databases containing the information about the systems studied here and the experimental information are published on the following website: http://hkamberajibu.wikidot.com/machine-learning.

Data availability statement

The data that support the findings of this study are available from the corresponding author upon reasonable request.

Acknowledgments

The author (H K) is grateful for the support of Eco/Logical Learning and Simulation Environments in Higher Education (ELSE) under the grant IT02-KA203-048006. The authors are also thankful for the help of the International Balkan University. The authors like to thank the CPU time provided by the DeepDDG Web-server (http://protein.org.cn/ddg.html).

ORCID iD

Hiqmet Kamberaj https://orcid.org/0000-0001-7357-7490

References

[1] Mayr A, Klambauer G, Unterthiner T, Steijaert M, Wegner J K, Ceulemans H, Clevert D A and Hochreiter S 2018 *Chem. Sci.* **9** 5441
[2] Mater A C and Coote M L 2019 *J. Chem. Inf. Model.* **59** 2545–59
[3] Lubbers N, Smith J S and Barros K 2018 *J. Chem. Phys.* **148** 241715–8
[4] Gastegger M, Schwiedrzik L, Bittermann M, Berzsenyi F and Marquetand P 2018 *J. Chem. Phys.* **148** 241709–11
[5] Goh G B, Siegel C, Vishnu A, Hodas N and Baker N 2018 How much chemistry does a deep neural network need to know to make accurate predictions? *2018 IEEE Conf. on Applications of Computer Vision (WACV)* pp 1340–9
[6] Collins C R, Gordon G J, von Lilienfeld O A and Yaron D J 2018 *J. Chem. Phys.* **148** 241718–11
[7] Schneider E, Dai L, Topper R Q, Drechsel-Grau C and Tuckerman M E 2017 *Phys. Rev. Lett.* **119** 150601
[8] Xu M, Zhu T and Zhang J Z H 2019 *J. Phys. Chem.* A **123** 6587–95
[9] Kamath A, Vargas-Hernández R A, Krems R V, Carrington T J and Manzhos S 2018 *J. Chem. Phys.* **148** 241702–7
[10] Herr J E, Yao K, McIntyre R, Toth D W and Parkhill J 2018 *J. Chem. Phys.* **148** 241710–9
[11] Wehmeyer C and Noé F 2018 *J. Chem. Phys.* **148** 241703–9
[12] Chen R, Liu X, Jin S, Lin J and Liu J 2018 *Molecules* **23** 2208–15
[13] Decherchi S, Berteotti A, Bottegoni G, Rocchia W and Cavalli A 2015 *Nat. Commun.* **6** 6155
[14] Rupp M, Tkatchenko A, Müller K R and von Lilienfeld O A 2012 *Phys. Rev. Lett.* **108** 058301
[15] Bereau T, DiStasio R A, Tkatchenko J A and von Lilienfeld O A 2018 *J. Chem. Phys.* **148** 241706
[16] Faber F A, Christensen A S, Huang B and von Lilienfeld O A 2018 *J. Chem. Phys.* **148** 241717
[17] Duvenaud D K, Maclaurin D, Iparraguirre J, Bombarell R, Hirzel T, Aspuru-Guzik A and Adams R P 2015 Convolutional Networks on Graphs for Learning Molecular Fingerprints *Advances in Neural Information Processing Systems* eds Cortes C, Lawrence N, Lee D, Sugiyama M and Garnett R (Red Hook, NY: Curran Associates) 28
[18] Battaglia P, Pascanu R, Lai M, Rezende D J and Kavukcuoglu K 2016 Interaction Networks for Learning about Objects, Relations and Physics *Proc. 30th Int. Conf. on Neural Information Processing Systems Barcelona* (Red Hook, NY: Curran Associates) pp 4509–17
[19] Schütt K T, Arbabzadah F, Chmiela S, Müller K R and Tkatchenko A 2017 *Nat. Commun.* **8** 13890
[20] Coley C W, Barzilay R, Green W H, Jaakkola T S and Jensen K F 2017 *J. Chem. Inf. Model.* **57** 1757–72
[21] Yang K *et al* 2019 *J. Chem. Inf. Model.* **59** 3370–88
[22] C-Ciriano I and Bender A 2018 *J. Chem. Inf. Model.* **59** 1269–81

[23] Riquelme M, Lara A, Mobley D L, Verstraelen T, Matamala A R and V-Martinez E 2018 *J. Chem. Inf. Model.* **58** 1779–97
[24] Rasmussen C E and Williams C K 2006 *Gaussian Processes for Machine Learning* vol 1 (Cambridge: MIT Press)
[25] Lee J, Bahri Y, Novak R, Schoenholz S S, Pennington J and Sohl-Dickstein J 2018 Deep neural networks as gaussian processes *Conf. Proc. in ICLR* pp 1–8
[26] Mobley D L 2015 UC Irvine: Department of Pharmaceutical Sciences, UCI (www.escholarship.org/uc/item/6sd403pz) (accessed July 19)
[27] Thurlkill R L, Grimsley G R, Scholtz J M and Pace C N 2006 *Protein Sci.* **15** 1214–8
[28] Pace C N, Grimsley G R and Scholtz J M 2009 *J. Biol. Chem.* **284** 13285–9
[29] Click T H and Kaminski G A 2009 *J. Phys. Chem.* B **113** 7844–50
[30] Pahari S, Sun L and Alexov E 2019 *Database* 1–7
[31] Gromiha M M, An J, Kono H, Oobatake M, Uedaira H, Kitajima K and Sarai A 1999 *Nucleic Acids Res.* **27** 286–8
[32] Bava K A, Gromiha M M, Uedaira H, Kitajima K and Sara A 2004 *Nucleic Acids Res.* **32** D120–1
[33] Ooi T, Oobatake M, Némethy G and Scheraga H A 1987 *Proc. Natl Acad. Sci. USA* **84** 3086–90
[34] Wereszczynski J and McCammon J A 2012 *Q. Rev. Biophys.* **45** 1–25
[35] Xu X and Truhlar D G 2011 *J. Chem. Theory Comput.* **7** 2766–79
[36] Bashford D 2004 *Front Biosci.* **9** 1082–99
[37] Sondergaard R C, Olsson M H M, Rostkowski M and Jensen J H 2011 *J. Chem. Theory Comput.* **7** 2284–95
[38] Wallace J A and Shen J K 2011 *J. Chem. Theory Comput.* **7** 2617–29
[39] Wu X and Brooks B R 2015 *Plos Computat. Biol.* **11** e1004480
[40] Homeyer N and Gohlke H 2013 Advances in molecular dynamics simulations and free-energy calculations relevant for drug design *In Silico Drug Discovery and Design* Lill M A (London: Future Science) pp 50–63
[41] Gordon J C, Myers J B, Folta T, Shoja V, Heath L S and Onufriev A 2005 *Nucleic Acids Res.* **33** W368–71
[42] Brooks B R *et al* 2009 *J. Comput. Chem.* **30** 1545–614
[43] Car R and Parrinello M 1985 *Phys. Rev. Lett.* **55** 2471–4
[44] Wang J, Wolf R M, Caldwell J W, Kollman P A and Case D A 2004 *J. Comput. Chem.* **25** 1157–74
[45] Li D W and Bruschweiler R 2010 *Angew. Chem. Int. Ed.* **49** 6778–80
[46] Wang J, Wolf R M, Caldwell J W, Kollman P A and Case D A 2004 *J. Comput. Chem.* **25** 1157–74
[47] Izairi R and Kamberaj H 2017 *J. Chem. Inf. Model.* **57** 2539–53
[48] Bergomi M G, Frosini P, Giorgi D and Quercioli N 2019 *Nat. Machine Intell.* **1** 423–33
[49] Janet J P, Chan L and Kulik H J 2018 *J. Phys. Chem. Lett.* **9** 1064–71
[50] Kamberaj H 2020 *Molecular Dynamics Simulations in Statistical Physics. Theory and Applications* Scientific Computation (Switzerland AG: Springer Nature) (https://doi.org/10.1007/978-3-030-35702-3)
[51] Qian N 1999 *Neural Netw.* **12** 145–51
[52] Srivastava N, Hinton G E, Krizhevsky A, Sutskever I and Salakhutdinov R 2014 *J. Mach. Learn. Res.* **15** 1929
[53] Singh G and Panda R K 2015 *Int. J. Hydrol. Sci. Technol.* **5** 333
[54] Zhou Z, Kearnes S, Li L, Zare R N and Riley P 2019 *Sci. Rep.* **9** 10752
[55] Anderson J A 1995 *An Introduction to Neural Networks* (Cambridge, MA: MIT Press)
[56] Weininger D 1988 *J. Chem. Inf. Comput. Sci.* **28** 31–6
[57] Unke O T and Meuwly M 2018 *J. Chem. Phys.* **148** 241708–15
[58] Mehler E L and Solmajer T 1991 *Protein Eng.* **8** 903–10
[59] Mehler E L and Guarnieri F 1999 *Biophys. J.* **77** 3–22
[60] Berman H M, Westbrook J, Feng Z, Gilliland G, Bhat T N, Weissig H, Shindyalov I N and Bourne P E 2000 *Protein Data Bank Nucleic Acids Res.* **28** 235–42
[61] Karhunen K 1947 *Ann. Acad. Sci. Fenn. A1* **37** 1–79
[62] Dekking E M, Kraaikamp C, Lopuhaä H P and Meester L E 2005 *A Modern Introduction to Probability and Statistics. Understanding Why and How* (London: Springer)
[63] Chakrabarti S 2003 *Mining the Web. Discovering Knowledge From Hypertext Data* (San Francisco: Morgan Kaufmann Publishers)
[64] Hoxha M, Rahmani B and Kamberaj H 2019 *Bull. Nat. Sci.* **28** 62–80
[65] Rahmani B and Kamberaj H 2019 (*bioRxiv*)
[66] Mobley D L, Dill K A and Chodera J D 2008 *J. Phys. Chem.* B **112** 938–46
[67] Cao H, Wang J, He L, Qi Y and Zhang J Z 2019 *J. Chem. Inf. Model.* **59** 1508–14
[68] Buckle A M, Henrick K and Fersht A R 1993 *J. Mol. Biol.* **234** 847
[69] Weaver L H and Matthews B W 1987 *J. Mol. Biol.* **193** 189
[70] Landauer R 1961 *IBM J. Res. Dev.* **5** 183–91
[71] Landauer R 1996 *Phys. Rev.* A **217** 188–93
[72] Vopson M M 2019 *AIP Adv.* **9** 095206–4
[73] Peter S C, Dhanjal J K, Malik V, Radhakrishnan N, Jayakanthan M and Sundar D 2019 Quantitative structure-activity relationship (QSAR): modeling approaches to biological applications *Encyclopedia of Bioinformatics and Computational Biology*, ed S Ranganathan, M Gribskov, K Nakai and C Schönbach (Oxford: Academic) pp 661–76