Machine Learning in Chemistry

Jon Paul Janet and Heather J. Kulik

March 15, 2020

Contents

Chapter 1

Advancing Research through Machine Learning

1.1 Overview

Interest in merging machine learning with traditional scientific inquiry has surged in recent years among researchers in the physical sciences. In laboratories across the world, scientists are identifying ways to incorporate machine learning into their everyday research. In both experimental and theoretical chemistry, recent applications have included the development of reactive force fields trained on first-principles data[1,2]; rapid property prediction models for materials discovery[3–6]; and the prediction, design, and improvement of catalysts[7,8] or chemical reaction yields.[9] "Machine learning" is a broad term for statistical algorithms that build prediction or decision models based on inferences from available data without explicitly coded instructions or a predefined parametric form.

Machine learning in chemistry is not new (Figure 1.1).[10–15] A similar surge in interest in applying machine learning (e.g., artificial neural networks) occurred in the late 1980s to mid-1990s.[12] The earliest applications of data science techniques to chemical problems included the development of feedback-trained expert or decision-based systems[10] and dimensionality reduction and pattern recognition using chemical data sets.[16–18] By the 1990s, the computer science community had demonstrated that nonlinear models, such as neural networks, could model any input-output mapping[19] (e.g., with sufficient hidden layers); this had not previously been thought possible in the early stages of applying simple perceptron models that lacked hidden layers.[20]

Simultaneous developments in machine learning beyond fully connected artificial neural networks included advances in kernel methods, such as Gaussian process regression,[21] support vector machines,[22] regularized linear regression (i.e., LASSO),[23] and early forays into the developments of convolutional neural networks[24] (e.g., the neocognitron[25]). In addition, the advent of supercomputing resources meant that complex models such as fully connected artificial neural networks could be trained with the back-propagation algorithm,[26] even on large data sets (e.g., 32,000 mass spectra[27]), in a reasonable amount of time ranging from hours to weeks. During this model-training step, the model gets feedback to "learn" the mapping between the observed data and the inputs (e.g., mass spectral signals). Moreover, the software to carry out this training became increasingly

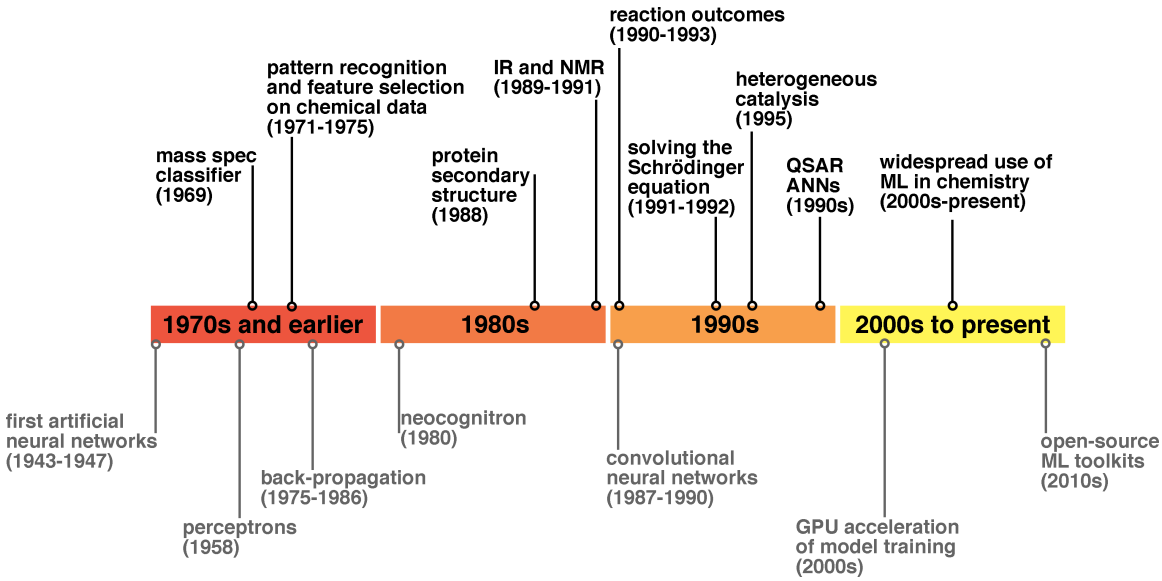

Figure 1.1: Timeline of representative advances in machine learning for the chemical sciences from the 1940s to the present day (black lines and text), along with parallel representative advances in computer science (gray lines and text). For brevity, emphasis is on developments pertaining to artificial neural networks.

available in commercial packages. Once trained, such models could be evaluated much faster than competing methods at the time, such as developing quantum chemical methods that were still quite costly to compute on most molecules and materials.[12,14]

While the parallels with modern efforts in applying machine learning to the chemical sciences and those of the 1980s-1990s may be surprising, chemists today share many of the broad interests and goals of earlier generations.

1.2 Machine Learning Terminology

We will first review some frequently used terms in machine learning. Machine learning algorithms can be most readily divided into two classes of algorithms: "unsupervised" and "supervised." Unsupervised learning algorithms are used to reveal structure in data sets. A typical unsupervised learning approach is "clustering," in which groupings within a data set are identified. Supervised learning involves the creation of a model that maps known inputs to known outputs. Linear regression is the simplest example of a supervised learning algorithm. In supervised learning, one obtains a series of training data that consists of the inputs and associated outputs. Most of the training data are used to train the model, with a portion "set aside" to test it.

A significant challenge in the development and application of machine learning algorithms is identifying when the model has "learned" a significant pattern in the data or when that pattern is spurious. In the case of regression, overfitting can be quite common, where a model passes through a series of points but introduces structure into the input-output mapping not present in the training data. For noisier data set sources and especially small data sets, this overfitting challenge can be particularly evident. A resampling procedure known as "cross-validation" is often essential to ensure that a machine learning model has not overfit the underlying data. In this case, the definitions of training and test data are shuffled periodically to ensure that a suitable error measure is achieved.

It can be useful to use statistical methods to identify the most predictive and informative features, which are the components of the input fed to the supervised model. As with any other chemical problem, a key challenge is lack of understanding of the essential dimensionality of the input-output mapping before the machine learning model is built; a good error metric is needed to ensure that the appropriate dimensionality is reflected in the model. "Feature selection" refers to a class of methods that can help find the most statistically significant features for mapping an input to an output in a machine learning model.

For newcomers interested in learning the fundamentals of machine learning, these basic concepts are detailed further in chapter 2. More sophisticated machine learning models also aim to learn the representation directly as part of their algorithmic structure. Advanced concepts of machine learning are detailed in chapter 6.

1.3 Machine Learning in the Chemical Sciences

As a reader of this primer, you likely already have an idea that machine learning could benefit your research in the chemical sciences. Growing computer power and data set size have caused a dramatic shift in accessibility for all key components of machine learning model development (i.e., data acquisition and model training). In experimental chemistry, robotics and automated, high-throughput experimentation[28,29] now enable generation of large, systematic data sets at surprisingly reasonable economic cost. Within computational and theoretical chemistry, the generation of large

data sets of small organic molecules (approximately 100,000 to 1 million) for machine learning is now increasingly routine,[30,31] thanks to dramatic advances in computing power. The challenge of storage of wave functions or structures and energies for these data sets, which would have been a concern 30 years ago, is now also straightforward.

The collection, curation, and sharing of such large data sets has also been an essential component of this new era of machine learning. An example is the extraction[32] of vast sets of experimental data by leveraging modern natural language processing tools. Where data sets have been carefully curated[30,31] or extracted,[32] it is now possible to train machine learning models to make predictions or extract patterns at scales that were not possible previously, simply because larger data set sizes are now accessible. The same advances in computing power have dramatically reduced the amount of time needed to train machine learning models. Open source toolkits[33,34] for training machine learning models and software tools that simplify architecture selection[35] have also made machine learning more broadly accessible.

At the same time, physics-based modeling methods have advanced significantly. This means that accurate reference data for organic molecules (e.g., from coupled cluster theory) can increasingly be generated for model training. It also means that, in many cases, training a machine learning model may not be necessary if good functional forms in physics-based models[36,37] provide an excellent balance of accuracy and computational cost. In addition, machine learning models have grown increasingly sophisticated and thus become amenable to broader uses. For example, sophisticated models called "autoencoders" have been increasingly applied to directly learn the most suitable molecular representations, and generative models have been trained to predict previously unseen chemical structures.[38]

In chemistry, machine learning can be most fruitfully applied to three broad areas: revealing patterns, overcoming limitations, and accelerating analysis.

1.3.1 Revealing patterns where few or none are known

Much of a modern scientist's or engineer's work involves the development of structure-property relationships. The researcher in the chemical sciences carries out experiments or computational modeling across a range of molecules or materials and then aims to understand what attributes of the chemical composition led to the observed outcome (e.g., reaction yield) or property (e.g., fluorescence). The scientist may then attempt to simplify the mapping between the set of molecules studied and the set of observed properties using a few quantities of the molecules, sometimes called "descriptors" or "features." Machine learning models and techniques can complement and assist this traditional form of inquiry. For instance, linear regression that avoids overfitting (e.g., through cross-validation) can be used to identify the most informative descriptors for property prediction. Collecting a large series of possible features or descriptors and evaluating their predictive performance, even in simple linear or kernel models on set-aside test data, can help to reveal previously unrecognized patterns in large data sets. Beyond set-aside test data from an original training data source, the generality of a relationship can also be tested by applying trained models to more distinct data sets. Furthermore, statistical analysis of the most important features, identified through feature selection or dimensionality-reduction techniques for multiple properties, can reveal the extent to which distinct properties can be independently tuned. For large data sets, statistical techniques can provide an excellent starting point for structure-property relationship if no patterns are known.

1.3.2 Overcoming the limits of simple models and human experience

As part of scientific inquiry, chemists frequently develop heuristics and theories to explain newly observed phenomena. However, people can be overconfident in heuristics and the extent of prior knowledge. Truly nonlinear or more complex relationships can be difficult to conceptualize but readily predicted by a neural network with precision beyond that of an expert. Even when using opaque models such as neural networks, their rapid evaluation can make it possible for the scientist to test predictions and reveal outcomes that were unknown previously. Human intuition and experience can be essential to guiding model development, but expert knowledge has a limit in transferability among researchers. Each new graduate student must consult and interpret the same literature or coursework to reach a specific level of expertise. Interaction between people and models can help to avoid excessive duplication of the time needed for knowledge acquisition. While the models need not replace traditional scientific inquiry, they can provide support to researchers.

1.3.3 Accelerating computations and analysis to enable rapid discovery in challenging materials spaces

Evaluation of a trained machine learning model is likely to be orders of magnitude faster than whichever method, experimental or computational, was used to generate the training data. Such a model can then be used, for example, to evaluate lead compounds if those compounds are similar enough to the training data for the model to be reliable. A key challenge is quantifying the uncertainty in model prediction to know when the limits of reliability have been reached. Alternatively, high-promise but low-certainty compounds can be exploited by turning this problem on its head in an approach known as "active learning," wherein new experiments or calculations can be carried out to acquire the most useful data to enrich models. In all cases, we should ask, What can we learn now that we did not know directly from the experiment or calculation? If the design objectives or observation from a dynamics trajectory are already present in training data, for example, there is no need for a machine learning model to be trained.[39] Conversely, if a large space must be screened to look for lead compounds that satisfy multiple objectives or if long time dynamics are needed to sample rare events, machine learning models become more useful. The machine learning model can be viewed as a focusing tool to enrich information about desired but rare molecules or events. Because the experiment or computation will still be carried out, knowledge of model reliability will be developed and physical or chemical principles can still be derived through more traditional avenues using the newly acquired data.

1.4 Summary

Machine learning can support the advancement of scientific research:

- Machine learning uses statistical algorithms to generate informational models based on inferences from the data, without explicit instructions or predefined forms.
- Machine learning can reveal structure in data sets (unsupervised) or map specific inputs to outputs (supervised).
- Key challenges include determining whether “learned” patterns are significant or spurious and selecting features to reflect appropriate dimensionality for input-output mapping.

- Benefits for the chemical sciences include revealing novel patterns in data, overcoming data and human limitations, and accelerating computation and analysis.

The rest of this primer will outline the tools needed to broadly understand and use machine learning approaches in chemical sciences research. The reader should understand the differences between machine learning algorithms and the best practices for developing machine learning models. The following chapters focus on working with data sets and general principles of statistical learning theory, including model selection (**Chapter 2**), simple interpretable (e.g., linear or kernel) models (**Chapter 3**), representation choice for molecular systems (**Chapter 4**), and more complex models (e.g., neural networks) (**Chapter 5**). Finally, practical guidance for training neural networks in chemistry is provided in **Chapter 6**. Each chapter is designed to be read independently so that readers can focus on the topics of greatest interest. Readers who are new to machine learning might find it useful to read Chapter 2 next for grounding in the concepts that follow.

Chapter 2

Supervised Machine Learning for the Chemical Sciences

2.1 Overview

Machine learning methods are applied to a variety of problems in chemistry, and novel applications will likely be found. Although each application requires specific adaptions and modeling choices, most fit into the basic framework of supervised learning. This chapter presents the basic structure and associated concepts for the majority of machine learning tasks in chemistry.

2.2 Overview of a Supervised Machine Learning Model

The fundamental objective of supervised learning is to develop a regression model or function, f, capable of making predictions in response to supplied inputs, x, denoted generally as $\hat{y}(x) = f(x)$. There are very few restrictions on what x and $\hat{y}$ may be; scalars, vectors, images, graphs; and more exotic types of data are possible. The inputs could be different arrangements of atoms in 3D space, and the outputs could be the corresponding energy of the system–this task is the main objective of computational chemistry (e.g., molecular mechanics or first-principles simulations). Many other choices are possible: the inputs could be potential drug-receptor protein docking molecules and the output a favorability score, or the inputs could be potential reactants and the outputs likely synthesis products. In the wider machine learning community, the inputs are often images (e.g., computer vision) and the outputs classifications or objects in the image, or the inputs may be text and the output sentiment or meaning (natural language processing).

The choice of model f is more standardized and is typically selected from a few major families, for example, kernel methods and random forests, or (deep) neural network models. In either case, the model depends on some parameters, denoted as W:

$$\hat{y}(x) := f(x, W) \tag{2.1}$$

These parameters may be coefficients in a linear model or weights in a neural network. The choice of parameters will uniquely determine the behavior of the model. The central task in training a machine learning model (i.e., "learning") is the selection of these parameters. Machine learning is

distinct from general regression in this way; little effort is made to adapt the structure of the machine learning model to the task at hand. Instead, a flexible model family with many parameters is preferred and effort is focused to make an intelligent choice of W (Figure 2.1). For example, neural network potentials (NNPs) are a family of neural network models that directly relate atomic coordinates to energy.[1] NNPs are similar in purpose to molecular mechanics force fields, and both are parameterized to agree with experimental observations or first-principles simulations. However, whereas force fields assume structured nonlinear equations based on polynomials (e.g., electrostatic repulsion and harmonic oscillators), NNPs make no explicit assumptions about the types of functions relating the atomic positions and energies. Instead, they start with a general form and learn to reproduce the structure-energy relationship directly from the data.

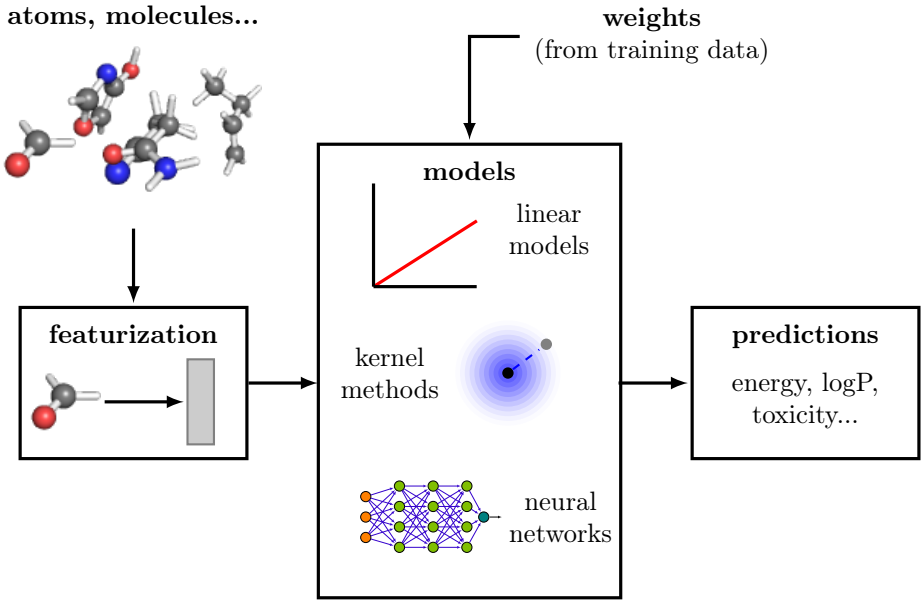

Figure 2.1: General flow of information for machine learning predictions on molecular systems. Molecules or atoms are converted into numerical vectors in the featurization process, and these features are passed to predictive models. Some example model types are shown in order of increasing complexity, from linear models (top) through kernel models (middle) to neural networks (bottom). All models depend on weights or parameters that must be learnt from many previous training examples. Some typical predicted quantities are listed, including the energy of the atomic configuration or physiochemical properties of input molecule.

The term "supervised learning" here refers to the use of training data, with n labeled pairs of previous examples, (x_i, y_i), that direct the relationship between the inputs and outputs, as controlled by the model parameters, W. In Equation 2.2, X and y are used without subscripts to refer to the collection of all data and with subscripts to refer to individual data points. Training data are used to evaluate different choices of W and to determine the best-performing W. This task immediately requires a way of determining how good a given value of W is for capturing the relationship between X and y with a "loss function." A loss function is a way to measure how well

the model predictions match the data. The obvious choice is to compute the (l^2) norm of the error between y and $\hat{y}$ for some input X:

$$\mathcal{L}(y, \hat{y}(X)) := \|y - \hat{y}(X)\|_2^2 = \|y - f(X, W)\|_2^2 = \sum_{i=1}^{n} (y_i - f(x_i, W))^2 \tag{2.2}$$

This equation is the well-known least-squares error function that is widely applied in, and nearly synonymous with, regression. Equation 2.2 has a number of useful properties worth discussing briefly. It is a strictly non-negative, even function of the errors $y_i - \hat{y}(x_i)$. This means that there is no bias toward under- or overestimating y_i, and we are assured that minimization of the total or average loss reduces our errors on the training data. It is differentiable, which is extremely important for optimization. Moreover, it is a convex function of ($\hat{y}$), which gives optimality guarantees to stationary points if f itself is convex, as in the case of linear least-squares regression. Finally, because it is a nonlinear function of the errors, it finds "balanced" solutions by penalizing large errors more severely: of all solutions with the same total absolute errors, the optimal ℓ^2 solution is the one for which all errors are equal to the absolute average error.

This simple metric is used in many applications of machine learning in chemical sciences. It is usually supplemented with additional terms that seek to penalize overly complicated models, a process called *regularization*. There are alternative loss functions as well, particularly for classification tasks where the output $\hat{y}$ is the probability associated with a certain label; however, these functions will not be discussed further because they share many of the properties of the l^2 loss function, and their differences tend to be technical as opposed to conceptual.

The learning task can be understood as a problem of minimizing the chosen loss function. This is often done iteratively by starting with some initial choice, perhaps normally distributed about zero, and then optimizing W using a traditional optimization routine, which utilizes the gradient of the loss function to choose how to update W. In some cases, this optimization can be done analytically, as is the case with linear and kernel methods, but generally it must be carried out numerically (e.g., by gradient descent and derivatives thereof) (Figure 2.2). The ease of this task will depend on the functional form of f and the exact relationship between $\hat{y}(x_i)$ and W. This explains why it is important to ensure that it is computationally simple to calculate derivatives of the model with respect to W.

Once the training procedure is complete and finalized model parameters are selected, the model is typically tested on some "out of sample (OOS)" or test data, composed of observations not drawn from the training data. This step is critical for the assessment of any fitted machine learning model, especially those with large numbers of parameters, because of the risk of overfitting for sufficiently complicated models.

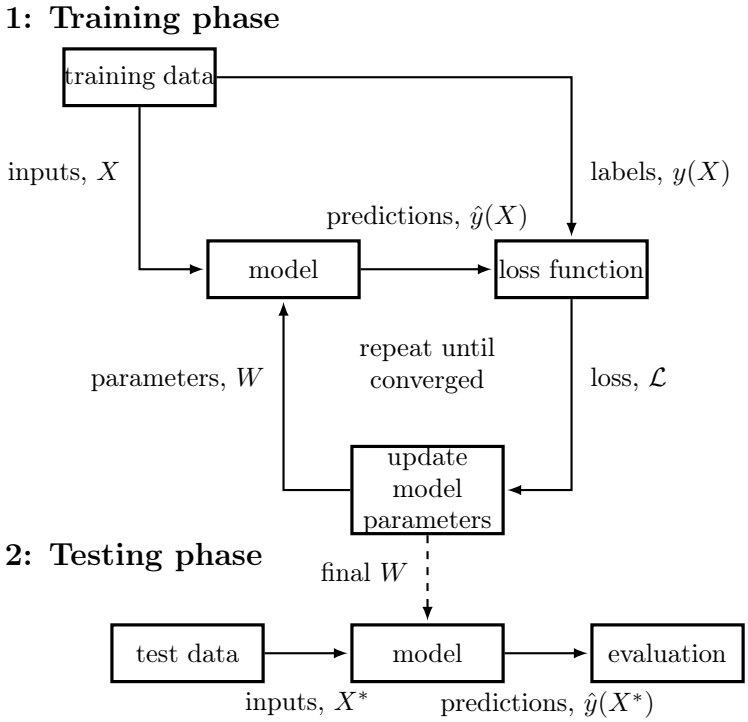

Figure 2.2: Overview of a typical iterative supervised learning training and testing process.

2.3 Data Sets and Scaling

Data sets used for the chemical sciences are as varied as the applications they serve: training cheap-to-evaluate NNPs might involve structure-energy pairs from millions of first-principles calculations[31], whereas directing the synthesis of new metallic glasses might involve hundreds of machine learning (ML)-selected high-throughput experiments[40]. The data consist of (hopefully many) observations x_i, which we will assume to be d-dimensional vectors in some space $\mathcal{X} \subset \mathbb{R}^d$. Data are often thought of as comprising vectors, although they might take a variety of forms (e.g., images, 3D coordinates, molecular graphs). Graphs in this context refer to the mathematical notion of graphs, which consist of a set of vertices or nodes connected by edges–naturally corresponding to the structure of atoms and bonds in molecules. All of these data types can be represented as vectors or matrices, and they encode everything that is known for each observation. This could include the size or composition of the chemical system, the relative positions of atoms (for NNPs especially), the identities of the species involved in the reaction, molecular properties such as frontier orbital energies (i.e., highest-occupied and lowest-unoccupied molecular orbitals), and many more. In the cheminformatics literature, popular representations are binary fingerprints that encode the presence or absence of certain chemical functional groups, such as FP2 or ECFP[41]. The choice of representation is paramount in obtaining good results from machine learning models. For now, we will assume that a consistently sized d-dimensional vector of descriptors is available for each data point.

We will adopt the convention of forming the training data into a *data matrix*, $X \in \mathbb{R}^{n \times d}$, that

has the n individual observations as rows and the descriptors for each observation as columns. For use in modeling, standard practice is to normalize the columns of the data to remove dependence on units of measurement and balance contributions from different descriptors. For example, if raw values were used and one descriptor was a temperature in Kelvin ($\sim \mathcal{O}(10^2)$) and the other was a Pauling electronegativity value ($\sim \mathcal{O}(1)$), the inherent mismatch of the scales of these values might skew models to weight temperature more highly. In addition, the model would build in sensitivity to arbitrary systems of units. Therefore, we normalize our data matrix, typically such that each column has zero mean and unit variance, which implies the *column sums* of X are zero. Such scaling is not universally necessary because some descriptors may be comparably scaled by design. Other rescalings are commonly used, such as scaling to run between -1 and 1, but achieve the same result.

The amount of data needed is highly application dependent, and the availability of data is related to the source and cost of acquiring the data. The construction of high-quality NNPs routinely uses thousands to hundreds of thousands of observations of molecular configurations evaluated using density functional theory (DFT)[1,42,43] because sufficient data are required to infer relationships between the bond angles and distances and the relative energy of the atomic configuration. Modern general-purpose NNPs for organic chemistry have been trained on more than 20 million DFT geometries.[2] However, specialized NNPs for limited element compositions and conditions—for example, those developed by Behler and coworkers for zinc oxides[44] or water clusters[45]—can achieve high accuracy with tens of thousands of DFT geometries or fewer. Data points for machine learning can be drawn from large databases (some of which are listed in Table 2.1), scraped from the literature,[32] or measured directly from experiments.[46] For the chemical sciences, the most common source is the output of computational chemistry calculations.

In some cases, databases will include "featurization," to translate chemical structures into mathematical form (discussed further in Chapter 4), but often provide only a description of the molecule (either a geometry or a SMILES string), and the featurization must be created by the practitioner. In this primer we will generally assume that $n > d$, that is, more data points are available than the dimension of the descriptors used. This assumption may not be true in contexts such as genomics[47] or where a very large set of features is generated[48], and these underspecified models pose additional challenges. "Regularization," introduced later in Section 2.5, can help fit underspecified models stably.

Table 2.1: Some common published data sets used for machine learning in chemical sciences, partly based on the MoleculeNet aggregate benchmark.[49]

Name	Year	Comments	Reference
ESOL	2004	2817 experimental aqueous solubility measurements for small organic molecules	[50]
PDBBind	2005–	13k protein ligand binding pairs	[51]
QM7	2012	7165 atomization energies and equilibrium DFT geometries for small organic molecules	[3]
AFLOW	2012–	500k first-principles properties for periodic systems, band structure is the most abundant property	[52]
USPTO Patent Database	2012	420k sets of reactant and product SMILES strings for patented organic reactions	[53]
The Materials Project	2013–	120k inorganic and 35k organic first-principles properties for periodic systems, band structure is the most abundant property	[54]
Tox21	2014	10k drug-like organic molecules and activity results against 12 different biological assays	[55]
QM7b	2013	extension of QM7 to 13 properties and levels of theory	[4]
QM9	2015	134k 13 properties at equilibrium DFT geometries for small organic molecules	[30]
QM8	2015	20k time-dependent density functional theory (RDDFT) excitation spectra small organic molecules,	[56]
ISO17	2017	645k energies and geometries for 129 unique small organic molecules from DFT *ab initio* dynamics, including forces	[42]
ANI-1	2017	20M energies for off-equilibrium geometries of 58k unique small organic molecules from DFT	[31]
SN2 reactions	2019	450k energies, forces, and dipole moments for reactive trajectories (including transition states) for methyl halide exchange reactions from DFT *ab initio* dynamics	[57]

2.4 Generalization and Statistical Learning Theory

The "training process" refers to choosing weights to make the model reproduce the input-output relationships contained in the data as faithfully as possible. We define the *empirical risk* as the average value of the loss function based on the set of n *training data* for a certain choice of model f (uniquely determined by the parameters W):

$$\mathcal{E}_{\text{emp}}(f) = \mathcal{E}_{\text{emp}}(W) = \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}(y_i, f(x_i, W)) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - f(x_i, W)\right)^2 \tag{2.3}$$

where the second equality applies only in the case of the least-squares loss function. For a linear model, the model weights would be the slope and intercept of the line. By changing these parameters, the values predicted by the linear model can be changed. Different choices will be better or worse at representing the relationship between the input vectors X and the predicted property y. The quantity in equation 2.3 is called the empirical risk because it is the amount of loss (or error) obtained on the training data; therefore, it is a fixed number for a given choice of the model weights. In a chemistry context, the empirical risk could be the average error made in predicting energies of different conformers, based on the set of conformers used to train the model. The basic process of supervised learning (Figure 2.2) is to update the model parameters, W, to make $\mathcal{E}_{\text{emp}}$ as small as possible. We denote function that obtains this minimum as

$$\hat{f} := \arg\min_{f \in \mathcal{T}} \mathcal{E}_{\text{emp}}(f) \tag{2.4}$$

This function is the best in the family of models, $\mathcal{T}$, that we consider. These models could range from a choice of linear functions to neural networks of a given size. For a linear model, this best function would correspond to the best-fit slope and intercept to match the relationship in the training data.

Our *objective* is to build a useful model of a physical process that is applicable to new values of x not in the training data. In drug design, for example, one might train a model to predict the activity of some known reference molecules and apply the model to screen unknown targets.[58]. Alternatively, one might want to use an NNP to assess the energy of unknown conformations[1] that are distinct from the training data. This process is called *generalization*, and it can be mathematically formulated as the expected risk for any fixed function, f:

$$\mathcal{E}(f) := \int_{\mathcal{X}\times\mathbb{R}} \mathcal{L}(y, f(x)) p(x, y)\, \mathrm{d}x\, \mathrm{d}y = \mathbb{E}\left[\mathcal{L}(Y, f(X))\right] \tag{2.5}$$

Here, the integral is taken over all possible inputs and outputs, and $p(x, y)$ represents the true, usually unknowable joint probability distribution between the random variables X and Y. Unlike empirical risk, this generalization error or true risk cannot be known or calculated because it refers to the performance of the model on all possible inputs (e.g., the energy prediction error on every possible conformer). Instead, this quantity must be approximated by applying the model to examples that are not in the training data. The laws of probability tell us that the function that minimizes $\mathcal{E}(f)$ is $f^*(x) := \mathbb{E}\left[\mathcal{L}\left(Y, f\left(X\right)\right) | X = x\right]$, that is, the conditional mean value of Y given x. This formulation allows for nondeterministic relationships between x and y, as in the case of processes with measurement noise; however, we often assume that the relationship is deterministic, and then f^* is a standard function of x. This assumption is reasonable if the training data come from a simulation such as DFT, in which the same result is always obtained from the same calculation, but experimental data may require that measurement noise be included.

The field of *statistical learning theory* is concerned with the analysis of equations 2.3–2.5 to determine when models found by minimizing empirical risk can be expected to generalize (i.e., provide low true risk). A central concept is that empirical risk will always overestimate true risk, and this *generalization error* can be decomposed into two contributions:

$$f^\dagger := \arg\min_{f \in \mathcal{T}} \mathcal{E}(f) \tag{2.6}$$

$$\mathcal{E}(\hat{f}) - \mathcal{E}(f^*) = \Big(\mathcal{E}(\hat{f}) - \mathcal{E}(f^\dagger)\Big) + \big(\mathcal{E}(f^\dagger) - \mathcal{E}(f^*)\big) \tag{2.7}$$

The first term is the *estimation error*, which arises when we make a suboptimal choice of function based on the training data, commonly called *overfitting*. This may occur when the training data cover only a small number of relevant possible inputs. For example, the training data for predicting the energy of conformers may include too few different dihedral angles, and the resulting model might give unreliable predictions based on unseen configurations. The second term is the *approximation error*, made by restricting the choice of model family. Many relationships in chemistry are inherently nonlinear, for example, the dissociation curve of a bond. No choice of parameters could allow a simple model class such as linear models to provide good predictions of these quantities; therefore, more complex models are necessary.

Although much of statistical learning theory is outside the scope of this primer, two critical ideas are worth noting,[59] using mild assumptions:

1. Assume that the space of possible models is complex enough to have near-zero approximation error (e.g., a very nonlinear model that can fit any data exactly). Then, if we draw training points from a fixed distribution, our empirical risk will converge to the risk for that distribution as the number of training points increases (i.e., our model will generalize).

2. The rate of convergence as we add more data is inversely related to the size of $\mathcal{T}$ (i.e., more complex model spaces require more data to generalize).

The extended, rigorous definition of "size" or complexity of the $\mathcal{T}$ model space is outside of the scope of this primer, but the reader is encouraged to interpret it in terms of how well models can adapt to fit distinct regions of points. For example, higher order functions are more complex than smooth, low-order functions. We can see an example of this in practice in the left panel of Figure 2.3, where we try to approximate the function $y = sin(\pi x)$ using some simple polynomial functions of increasing order, based on eight known points with some random measurement noise. The inset bar plots compare the empirical risk (evaluated as the average error of the model on the noisy measured points) and the true risk (evaluated based on the integral in eq. 2.5). A second-order polynomial cannot capture the multiple extrema in the data and performs poorly in both metrics. Moving to a more complicated fourth-order polynomial reduces both risks, but moving to an eighth-order model overfits the data, reducing the error on known points to zero while introducing spurious oscillations that increase the true risk.

To obtain good generalization, our set of model functions $\mathcal{T}$ needs to be large or complex enough to have low approximation error but no larger. With limited data, we are often better off searching for our model in a simpler, smaller family of models that learn robustly and quickly, as opposed to complex models with many parameters. Conversely, a simple model will stop improving with more data past a certain point—where the approximation error dominates.

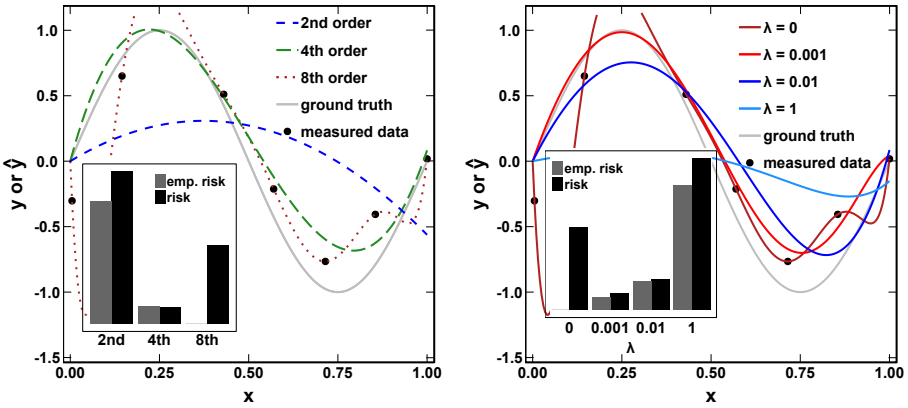

Figure 2.3: Comparison of fitting the function $y = \sin(\pi x)$ based on 8 points uniformly sampled in $[0, 1]$ with measurement noise $\mathcal{N}(0, 0.15^2)$, with with either polynomials of degrees 2, 4 and 8 (left) or an 8^{th} order polynomial with different levels of regularization (right). Sampled points are shown as black circles with the ground truth shown as a gray line and approximations shown with lines. Insets show the empirical and actual risk on $[0, 1]$ for each choice.

2.5 Regularization

Regularization techniques enable fine-grained control of model complexity, which allows us to search for the correct level of model complexity more smoothly. Rather than being forced to change the size or configuration of the models used in order to control complexity, we add a penalty, $R(f(X))$, to our loss function. This approach depends only on the complexity of the model and not on how well it fits the data.

$$\mathcal{L}'(y, f(X)) = \mathcal{L}(y, f(X)) + \lambda R(f(X)) \tag{2.8}$$

Here we use $\lambda \geq 0$ to balance making the model fit the data and forcing the model to be as simple as possible. As a concrete example, the most common type of regularization[59,60] is *Tikhonov* or ℓ_2 *regularization*, which is given by the ℓ_2 norm of the model parameters, $R(f(X, W)) := \|W\|_2^2$. Combining this example with the square loss function and writing out the explicit dependence on the model parameters gives:

$$\mathcal{L}'(y, f(X, W)) = \frac{1}{n} \|y - f(X, W)\|_2^2 + \lambda \|W\|_2^2 \tag{2.9}$$

Note that if we optimize $\mathcal{L}'$ in place of $\mathcal{L}$, we will trade some of the fit to training data to reduce the magnitude of the weights. We ask for the best model to fit the training data, given a complexity budget determined by λ. The minimizer $\mathcal{L}'$ will be a worse fit to the training data (with more ℓ_2 loss) compared with the model obtained by unrestrained minimization of $\mathcal{L}$, but it will have lower complexity. As a result, this model will often generalize better (i.e., have higher *approximation error* but lower *estimation error*). This effect is illustrated in the right panel of Figure 2.3, where increasing the value of λ moves the eighth-order polynomial all the way from overfit to underfit to the data, with the optimal value of the true risk obtained for an intermediate λ. It is useful to

note that, via Lagrange multipliers, minimizing equation 2.9 with respect to some p-dimensional parameters W is equivalent to minimizing:

$$\min_{W \in \mathbb{R}^p} \frac{1}{n} \|y - f(X, W)\|_2^2 \tag{2.10}$$

$$\text{s.t.} \|W\|_2^2 \leq r \tag{2.11}$$

Our value of λ establishes the budget for parameters (how large we allow r to become), such that large values of λ result in small coefficients W and vice versa. As opposed to parameters such as W, λ is called a *hyperparameter* because although both affect model performance, we would not choose λ based on the training data: it is set for the model before we begin the training process. Note that if we were to take the derivative of equation 2.9 with respect to λ to minimize our training loss, we would trivially set $\lambda = 0$, which aligns with our notion that more complicated models can always fit training data better. We will study how to select hyperparameters such as λ in Section 2.6.

2.6 Model Selection and Validation

Being able to perfectly reproduce finite training data does not guarantee that a model will generalize to new observations, particularly for complicated models. Fortunately, model complexity can be controlled in a nuanced way with regularization. Regularization, however, does not help to answer the important question, how complicated should my model be? Unfortunately, errors in training data cannot help determine how complicated a model the data set can support. Figure 2.4 shows a schematic illustration of this issue. Although the empirical risk continues to decrease as complexity is added, the real ability of the model to predict new results begins to degrade.

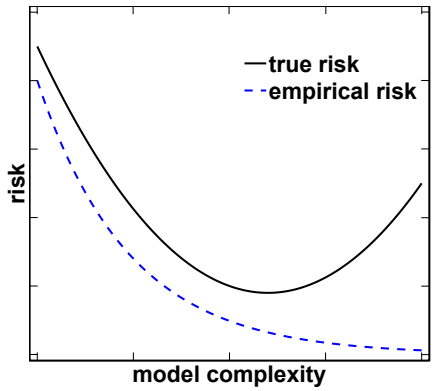

Figure 2.4: The 'classical' picture of over-fitting: illustration of the relationship between model complexity and true and empirical risk, showing that sufficiently flexible models will always fit training data arbitrarily well but will tend to generalize poorly.

Instead, one must use *validation* data to answer this question. The training data should be further divided into a set that will be used for training and a validation set to estimate how well

each model generalizes. The distinction between validation and test data might not be obvious at first, but the important principle is that validation data will help us construct our model (e.g., by selecting how much regularization to use), whereas test data are not used at any stage of model training. Effectively, the test data should be set aside and used only for model evaluation once all choices about model construction have been made.

To reduce dependence on the particular way we select our validation sets from the training data, it is standard practice to use a technique known as cross-validation (CV), which involves subdividing the training data into k equal folds. We remove the first fold from the training data and use the remaining $k-1$ folds to train the model, using a fixed choice of the hyperparameters and regularization strength. Once trained, the model is tested on the removed fold to generate an error estimate for OOS data. The way in which the error is evaluated on the removed fold can be selected freely. The least-squares error (eq. 2.2), for example, is a good choice. Note that this error is not a loss function because it is not used to train the model and should not include regularization terms. The ubiquitous mean-squared error (eq. 2.2) is a good choice for regression tasks. This error is recorded, and then the training process is repeated with the same hyperparameters on the next fold of the data. This process is repeated k times, such that each fold has been left out exactly once. The error over each fold is then averaged to produce the *CV error* (inner loop in Figure 2.6). This metric for error is an estimator for the generalization error made by the model because it is based only on data not used to train the model at each step. This approach increases the cost of training the model by a factor of k for each model that is to be evaluated.

Because the average CV error is an estimate of the generalizability of a given model and hyperparameter combination, it can be used to select among different models and hyperparameters (Figure 2.5). The choice of k is typically between 5 and the amount of data available, n. The special case where $k = n$ is called "leave one out" CV (LOOCV; used in Figure 2.5) because only one point is left out in each fold. In practice, 5 or 10 are common choices, with the smaller values of k being cheaper to evaluate because the training process is repeated fewer times. The larger the value of k, the less biased the CV error estimator is relative to the generalization potential of the model trained on all training data.[1] When using small data sets, small values of k may be undesirable because a model trained on only $\frac{k-1}{k}$ of the data may be highly unstable. When a large amount of data is available, the overhead of repeating the training process is impractical, and a large k may not be suitable. Such choices are also inherently dependent on the time required to train an individual model, which is generally connected to the model's complexity. If a large number of hyperparameters must be selected, as in neural networks, end-to-end repeats of the entire training process are less common.

[1] This bias is introduced precisely because a smaller fraction of data is used.

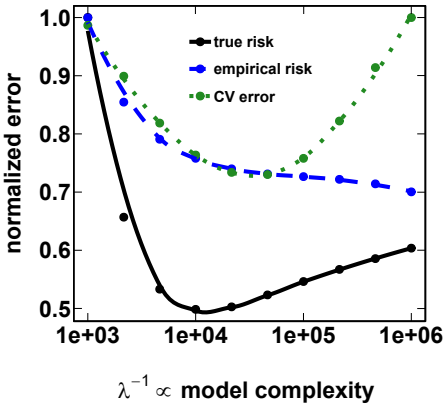

Figure 2.5: Demonstration of the CV error estimate as an indicator of true risk, showing the same polynomial regression problem as Figure 2.3, here estimated with a 15^{th} order polynomial and 30 training points and different levels of Tikhonov regularization λ, which is inversely correlated with model complexity. An LOOCV estimate of the error is computed for each value of λ (green), and compared to the empirical (blue) and true (black) risks. Points indicate calculations and smoothing lines are added to show trends only. It is apparent that the CV estimator can approximately indicate what value of λ to use, while the empirical risk strictly favors more complicated models. Compare also with the idealized case in Figure 2.4.

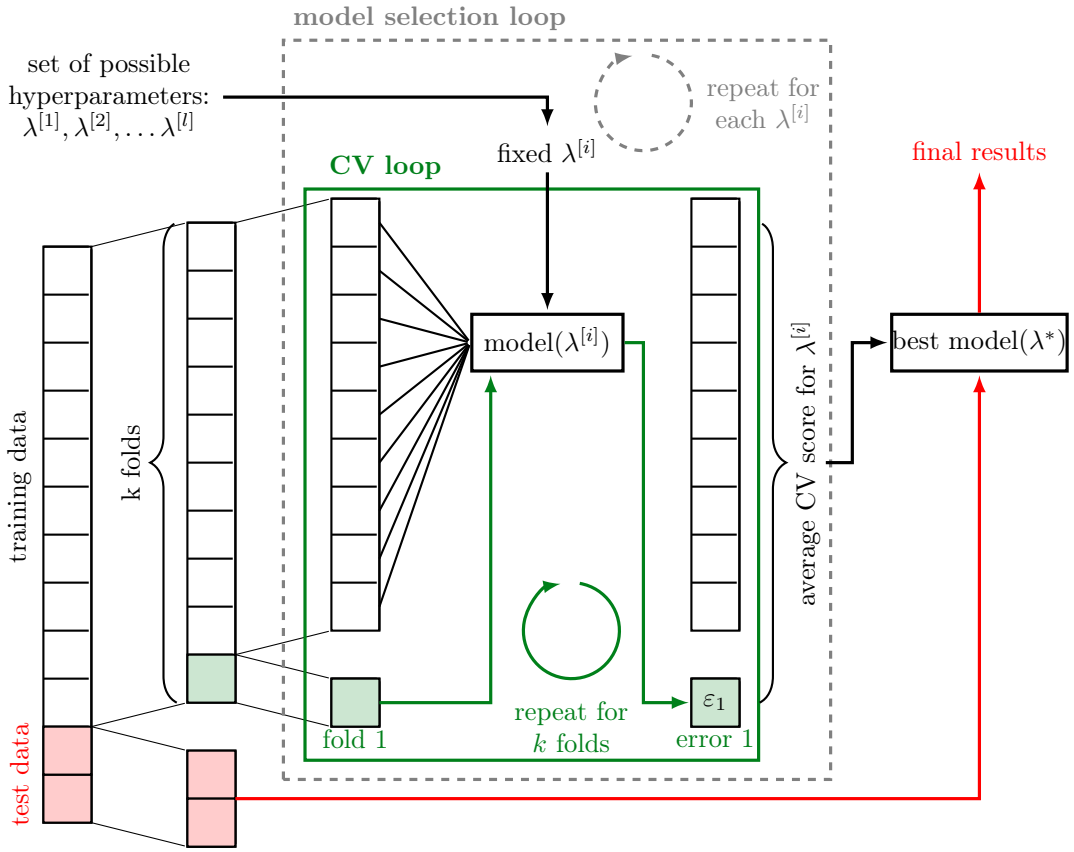

Figure 2.6: Illustration of the machinery for model selection via double loop grid search k-fold cross-validation (CV) for a single hyperparameter λ. The full dataset is split into test and train fractions. Then, for each choice of the hyperparameter λ (the gray outer loop), the training data is further split into k equal folds and then a series of k models are trained with one fold left out of the training process each time (green inner loop). Each model is evaluated on the left-out fold to generate k errors $\varepsilon_1, \varepsilon_2, \ldots \varepsilon_k$. The average error for each λ is compared and the best λ^* is chosen to train a model on the full training set. The test data is only used to evaluate the final model.

In practice, choosing good values of the hyperparameters is difficult, and hyperparameters are typically interrelated. A standard approach is to perform "grid search" CV (outer loop in Figure 2.6). A range of values is selected for each hyperparameter, and this range is discretized into sample values to try for each hyperparameter. Every combination of hyperparameters can then be scored using CV, and the best performing model is then selected and retrained on all of the training data (and evaluated against the held-out test data). This approach provides a robust method of model selection and can be used to decide between different types of models (e.g., linear vs. nonlinear or maximum degree of nonlinearity). This pipeline is essentially a double loop, requiring training the

model k times for each hyperparameter combination. For robustness in partitioning the training and test data, it is possible to repeat this entire process for different test/train splits, forming a triple-loop scheme, sometimes called "nested" CV. This approach can be important for small or highly heterogeneous data sets, and testing the robustness of test/train splits is always good practice.

Alternatives to CV exist. In bootstrapping[61], for example, random training sets of consistent size are assembled by drawing from the original data set with repetition - that is, each observation can occur multiple times in a given training set (unlike CV, where each observation is missing from only one fold and only occurs once in each of the other $k-1$ folds). These approaches face similar costs in terms of model training time. For more than a handful of hyperparameter values, an exhaustive grid search becomes infeasible. It is natural to seek efficient strategies, particularly for neural networks or other models where the training cost is substantial. One approach is to use Bayesian methods to optimally sample the hyperparameter space.[35,62] For complex models, practitioners may be forced to make do with a single validation split.

2.7 Summary

Most machine learning methods in chemistry use a supervised learning framework:

- The objective of supervised learning is to develop a model that is capable of making predictions based on supplied data. A training phase selects parameters to determine the model's behavior and a testing phase evaluates the output.
- Data sets for the chemical sciences are varied in application and size. All data types can be represented mathematically to encode everything known for each observation. The choice of representation is key to obtaining good machine learning results.
- A loss function measures how well the model predictions match the data, and the least-squares error function is the most common loss function.
- Generalization is the ability of a machine learning model to make accurate predictions on new inputs not in the training data. Complicated models are more prone to over-fitting, where the training data is well predicted but generalization performance is poor.
- Regularization techniques are used to control model complexity by penalizing overly complicated models. This helps improve generalization.
- Cross-validation is used to compare the generalization capacity of machine learning models and select the appropriate level of regularization. Performance on training data is never sufficient to assess generalization capacity.

Chapter 3

Linear Models, Kernels, and Trees

3.1 Overview

This chapter presents some of the simplest, and thus easiest to interpret, supervised machine learning models. Multiple linear regression (MLR) illustrates the process of data fitting through an easily understood model with similarities to more complex models. Nonlinear regression is introduced through a family of models called *kernel models* that enable the systematic introduction of nonlinear fitting terms. From this general kernel framework, the widely used kernel ridge regression (KRR) approach is introduced via the "kernel trick." Commonly encountered kernels and hyperparameter selection are discussed, along with other kernel-model formulations such as Gaussian processes. The chapter concludes with a brief overview of tree and random forests (RFs) models.

3.2 Multiple Linear Regression

Linear regression is the simplest type of functional relationship connecting input descriptors to output properties. Because of this simplicity and interpretability, MLR has a long history of application to problems in chemical science domains. Drug design in particular has seen widespread application of linear models[63,64]. MLR is still routinely used to model complex systems, such as catalysis,[65] solubility,[66] and reaction selectivity prediction.[67]. Low training cost also makes regularized MLR models a good toolbox[68,69] for testing different feature sets and extracting the most correlated descriptors.[1] However, because the model does not include any nonlinearity by definition, the target property must be well correlated with the descriptors used. This requirement may necessitate using difficult-to-compute descriptors (e.g., those from quantum mechanical calculations[70] or experiments, such as linear free energy relationships) or limiting the range of applicability to a small set of systems. This can be addressed somewhat by dividing the input space into regions and training only local models.[71] However, MLR models typically cannot match the quantitative accuracy of more flexible models. A simple model that is readily understood by nonspecialists is still extremely valuable. Therefore, it is generally good practice to start with an MLR model to establish baseline model performance before attempting nonlinear models.

An MLR model produces estimates, $\hat{y}$, for a quantity of interest, y, as a linear function of the

[1] See LASSO in Section 4.4.

data matrix of n, d-dimensional observations $X \in \mathbb{R}^{n \times d}$, by choosing $d+1$ parameters, w', which are commonly referred to as "weights" within machine learning. For example, the output could be catalytic yield or the solubility of a molecule, and the inputs could be numerical representations of the reaction conditions or chemical structure. There is one parameter per dimension and one optional constant or bias term, which means the number of degrees of freedom (i.e., terms to be learned during training) depends on the length of the representation used. It is convenient to augment the data matrix with an additional row of "ones" to account for the bias term, which will allow it to be treated seamlessly. Intuitively, a constant term is just a feature that is the same for all inputs:

$$\hat{y}_{\text{MLR}}(X) := Xw + b = \begin{bmatrix} X \\ 1, 1 \ldots, 1 \end{bmatrix} w = X'w' \tag{3.1}$$

$$w' \in \mathbb{R}^{d+1}, X' \in \mathbb{R}^{n \times (d+1)} \tag{3.2}$$

In these vector equations, the prediction at a new point $x^* \in \mathbb{R}^{1 \times d}$ is given by $\sum_{j=1}^{d} x_j^* w_j + w_{d+1}$. That is, there is one parameter for each dimension of the feature space and a bias term. For simplicity, we consider models of the form $\hat{y}_{\text{MLR}}(X) = Xw$ and set the number of dimensions to be d. However, when performing regularization, we do not wish to penalize the bias term because the size of the bias term does not affect the complexity of our model. Rather, the bias term raises or lowers all values by the same amount without affecting their relative positions. We will solve this equation using the least-squares loss function with l_2 regularization (eq. 2.9) by taking the derivative with respect to our parameters and setting it equal to zero:

$$\begin{aligned} \mathcal{L}(w) &= \|\hat{y}_{\text{MLR}}(X) - y\|_2^2 + \lambda \|w\|_2^2 = (Xw - y)^T (Xw - y) + \lambda w^T w \\ \frac{\partial \mathcal{L}(w)}{\partial w} &= 2\left(X^T X w - X^T y\right) + 2\lambda w = 0 \\ \implies w &= (X^T X + I_d \lambda)^{-1} X^T y \end{aligned} \tag{3.3}$$

We omitted the y argument to the loss function $\mathcal{L}(y, f(x))$ because we cannot change the data, only the model weights w. We can set $\lambda = 0$ to return to a fully standard linear least-squares regression problem, in which case the statements in equation (3.3) are called the "normal" equations because they represent a projection of the data y into the column space of X.[2] /indent Note that the best-fit weights w can be found algebraically, and there is no need to conduct any numerical optimization. One simply needs to invert the matrix $X^T X + I_d \lambda$, which is a $d \times d$ matrix. The computational cost of this operation classically scales with d^3, and the result is d best-fit weights. When using equation 3.3, it is best practice not to form the inverse $(X^T X + I_d \lambda)^{-1}$ directly for numerical reasons and instead to solve $(X^T X + I_d \lambda) w = X^T y$ though a factorization technique, for example, the lower-upper (LU)-decomposition (see Trefethen and Bau 1997 [72]). The linear solver in modern mathematical software packages typically handles this choice automatically.

[2] It is easily shown that $(y - \hat{y}) \perp Xw$, i.e., any remaining error is orthogonal to the model family.

3.2.1 Bias terms and data normalization

Now we return to the issue of bias terms, which we do not want to regularize. The optimization problem to solve, excluding regularization of the bias term, is:

$$\min_{w,b} \mathcal{L}(w,b) = \min_{w,b} \frac{1}{n} \sum_{i=1}^{n} (y^{(i)} - x^{(i)}w - b)^2 + \lambda \|w\|_2^2 \tag{3.4}$$

We adopted the convention of distinguishing different training examples with superscripts. For example, $x^{(1)} \in \mathbb{R}^{1\times d}$ is our first training observation (a row vector), and we use subscripts to refer to specific elements of a vector. In equation 3.4, we removed the bias term b from w so that it does not get penalized by regularization. Taking the derivative with respect to b and setting it equal to zero provides a definition of the bias term:

$$b = \bar{y} - \bar{x}w \tag{3.5}$$

$$\text{where} \quad w = \arg\min_{w} \frac{1}{n} \sum_{i=1}^{n} \left(\left[y^{(i)} - \bar{y} \right] - \left[x^{(i)} - \bar{x} \right] w \right)^2 + \lambda \|w\|_2^2 \tag{3.6}$$

Here, $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x^{(i)}$ and $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y^{(i)}$ are the mean values of the input and response, respectively. If we normalize the data by subtracting the mean values, we can solve the unbiased problem given by equation 3.6 without addressing the bias term by using equation 3.3. We can then compute the bias term later. This observation also suggests that normalizing data is good practice because it removes the need to account for constant shifts in the model family. In addition, reviewing equation 3.5 provides insight into the nature of bias terms: they correct for mismatch between the average model output ($\bar{x}w$) and the average output value through a rigid shift. This also helps explain why regularization of this bias term is not well justified.

3.2.2 Conditioning and regularization

It is worth thinking about whether the inverse of equation 3.3 exists and when it might not. X^TX is a positive semidefinite matrix and has strictly non-negative eigenvalues, which is not enough to ensure that it is invertible. Recall that a matrix is not invertible if it has any eigenvalues equal to zero. This is the same as saying that there are vectors in the (right-)nullspace of X, that is, if there are vectors v such that $Xv = 0$, this v will be an eigenvector of X^TX with an eigenvalue of zero.

Consider the case in which the number of observations is greater than the number of features, $n \geq d$. This null vector v exists[72] only if the row-rank of X is less than the number of rows n; this can occur if two exact rows or columns are copies of each other (or linear combinations). It is somewhat unlikely in applications where each observation corresponds to a different molecule or configuration. A more likely occurrence in practice is that the rows of X will be nearly colinear. In this case, the the eigenvalues of the matrix X^TX will be close to zero, and the resulting weights will be unstable and change drastically in response to small perturbations in the training data. This can be understood though the condition number of the matrix (see any text on numerical linear algebra for more details, e.g., ref.[72]).

This problem occurs frequently when studying molecular systems because many molecules are fundamentally similar. For example, a data set might include two molecules differing by only a single hydrogen atom, and in many representations (see Chapter 4), the fingerprints associated with these two molecules will be nearly identical, perhaps differing in only a few of the hundreds

of different columns. In cases where the number of features d is larger than the number of observations n, the inverse is nonunique, and this degeneracy must be broken, either by downselecting important features through feature selection (Section 4.4) or by adding additional constraints to the optimization problem through the use of regularization.[73]

The regularization term λ can address the challenges associated with building this inverse, even when the eigenvalues are near zero. The eigenvalues of $X^TX + I_d\lambda$ are simply $\omega_i + \lambda$, where ω_i represents the eigenvalues of X^TX (note that $w \neq \omega$).[3] This means that the regularization term shifts all the eigenvalues by a fixed amount away from zero and thus always improves the conditioning of the least squares matrix. Even a small positive value of λ will typically be enough to "fix" any nearly colinear matrix elements because it will shift the smallest eigenvalues away from zero. In addition, λ reduces the magnitude of coefficients and makes the model more stable. In the context of linear models, equation 2.9 implies that coefficients (here, slope) of the linear model will be closer to zero—that is, the model changes more slowly, as when the data changes when using $\lambda > 0$. In the limit of large λ, a highly regularized model will be invariant to the data and zero slope will be obtained, giving the same prediction for all inputs: a flat line.

3.2.3 The linear kernel

The best-fit weights can be formulated another way that will help interpretation of MLR and its extension to nonlinear regression (i.e., general kernel methods). Because $n > d$ and rows of X or columns of X^T are full rank, equation 3.3 can be rewritten to express $w = X^Ta$ for $a \in \mathbb{R}^n$. This corresponds to a shift of basis for w in the row space of X, instead of the column space. Such a transform is, by necessity, not unique because there are more coefficients a than parameters w. However, the use of regularization with $\lambda > 0$ included in equation 3.3 causes us to favor the smallest solution (as measured by the norm; e.g., the ℓ_2 norm for Tikhonov regularization).
Because $w = X^Ta$, the model is not changed in any way, and predictions from the model parameterized by a are exactly the same as the model parameterized by w. Now we can write out the MLR equations in terms of this transform at the new (test) point $x^* \in \mathbb{R}^{1\times d}$. Here, each observation is a row vector:

$$\hat{y}_{\text{MLR}}(x^*) = x^*w = \sum_{j=1}^{d} x_j^* w_j = x^*X^Ta = \sum_{i=1}^{n} x^*\left(x^{(i)}\right)^T a_i = \sum_{i=1}^{n} k\left(x^*, x^{(i)}\right) a_i \tag{3.7}$$

where $k(x', x) = \langle x', x\rangle : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is the *linear kernel function*. Equation 3.7 provides two equivalent formulations of the linear model. The first is as a sum of d weight terms, one per feature space dimension. The other is as a sum of n points, one for each training point. These dual interpretations provide insight into how linear models work. If the vector corresponding to a new point x^* is orthogonal to a given training point $x^{(i)}$ (i.e., mathematically, $k(x^*, x^{(i)}) = x^*(x^{(i)})^T = 0$), then point i has no influence on the predicted value at x^*. This observation holds regardless of the coefficients a. Conversely, holding the norm of the points fixed, the maximum value of $k(x^*, x^{(i)})$ is obtained if the points are colinear. The linear kernel defines the similarity of inputs by the euclidean inner product $\langle x', x\rangle = (x')^Tx$. This means that orthogonal points such as $[0, 1]$ and $[1, 0]$ are maximally dissimilar, resulting in a zero kernel term, whereas points that are colinear are the 'most similar. This concept of data similarity and how it influences model prediction is useful when comparing higher order models. For completeness, we also write the vector form of the

[3] $X^TX + I_d\lambda = V^{-1}\Omega V + V^{-1}V\lambda = V^{-1}(\Omega + I_d\lambda)V$, a property for all sums of commutative matrices.

kernel equation using the kernel matrix K, with $K_{i,j} = k(x^{(i)}, x^{(j)})$, and find the best-fit values for a using the same approach as equation 3.3:

$$\hat{y}(X) = Ka \tag{3.8}$$

$$a = (K + I_n\lambda)^{-1}y \tag{3.9}$$

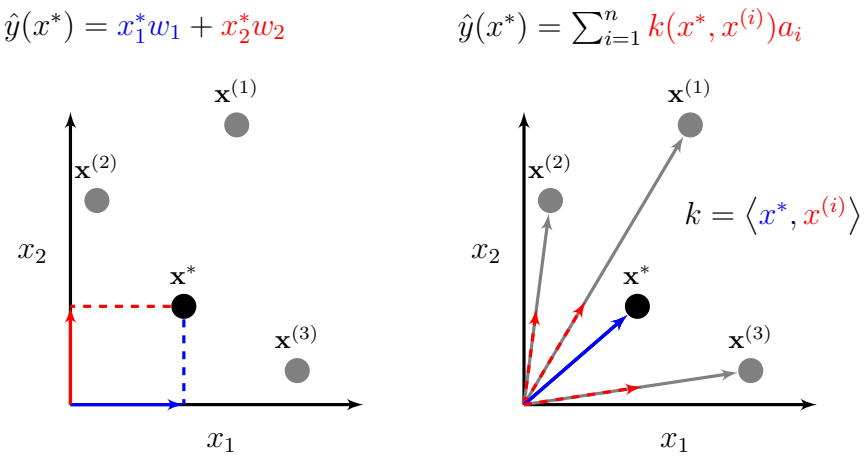

Figure 3.1: Two ways of parameterizing linear regression for a linear function of two input dimensions (x_1, x_2) evaluated at a new point x^*: the traditional interpretation (left) gives the value of $\hat{y}(x^*)$ as a sum of the position in the coordinate space with weights (w_1, w_2) while the linear kernel (right) gives the same prediction expressed as a summation over pairwise similarities between x^* and the training points $x^{(1)}$–$x^{(3)}$.

The linear kernel is an equivalent explanation of regularized MLR. The difference in this explanation is that we interpret predictions in terms of the pairwise similarity between points with coefficients a (Figure 3.1). This interpretation is formally equivalent to the more conventional interpretation of the properties of the input data in one coordinate space, (e.g., x_1 and x_2) and how that transforms to the output data via coefficients w (Figure 3.1). This equivalence is due to the fact that the coefficients w are inferred directly from the data. In addition, recall that the ℓ_2 regularization expression applies to w. This means that the regularization term can be expressed as $\|w\|_2^2 = \left\|X^T a\right\|_2^2 = a^T K a = \|a\|_K^2$; that is, the normal constraint on a is weighted by the kernel matrix K.

3.3 Nonlinear Regression and the Kernel Trick

Not all functions are well expressed as linear combinations of their properties. For example, we do expect the energy of a chemical bond to be a linear function of the distance between the atoms. In general, nonlinear effects have to be modeled from perhaps arbitrary functions of features (e.g., electrostatic interactions decaying as $\frac{1}{r^2}$ in Coulomb's Law) or interactions between the different

features (the strength of the bond depends on the properties of both participating atoms). To keep the notation simple, consider fitting a quadratic polynomial to a case with two features per input (two properties for each observation $x^{(j)}$: $x_1^{(j)}$ and $x_2^{(j)}$). We propose a nonlinear model of the following form:

$$y_{\text{QUAD}}(x) := w_1 + w_2\sqrt{2}x_1 + w_3\sqrt{2}x_2 + w_4\sqrt{2}x_1x_2 + w_5x_1^2 + w_6x_2^2 \tag{3.10}$$

that is, a general quadratic function with a cross term, involving both features, controlled by w_4. The presence of the $\sqrt{2}$ terms will simplify the expressions later, but they could be combined with the coefficients w. Now we can find these coefficients based on our data matrix of n, two-dimensional observations, $X = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} \\ \vdots & \vdots \\ x_1^{(n)} & x_2^{(n)} \end{bmatrix}$. To avoid confusion, we included parentheses in the superscripts to be clear that we are referring to different observation indexes, not powers. Notice that although the features are transformed nonlinearly, the model is linear in its parameters. We start by defining a nonlinear *feature transform* φ that maps our original observations in $\mathbb{R}^{1\times 2}$ to their expanded features in $\mathbb{R}^{1\times 6}$ as:

$$\varphi(\begin{bmatrix} x_1 & x_2 \end{bmatrix}) = \begin{bmatrix} 1 & \sqrt{2}x_1 & \sqrt{2}x_2 & \sqrt{2}x_1x_2 & x_1^2 & x_2^2 \end{bmatrix} \tag{3.11}$$

This enlarged feature space is sometimes called the "lifted feature space"[74] because it is usually of a higher dimension than the original feature space. We define the matrix that is obtained by applying φ to each row of X as $\varphi(X) \in \mathbb{R}^{n\times 6}$. Now our model can be written out as:

$$y_{\text{QUAD}}(x) := \varphi(X)w = \begin{bmatrix} 1 & \sqrt{2}x_1^{(1)} & \sqrt{2}x_2^{(1)} & \sqrt{2}x_1^{(1)}x_2^{(1)} & (x_1^{(1)})^2 & (x_2^{(1)})^2 \\ \vdots & & & \vdots & & \\ 1 & \sqrt{2}x_1^{(n)} & \sqrt{2}x_2^{(n)} & \sqrt{2}x_1^{(n)}x_2^{(n)} & (x_1^{(n)})^2 & (x_2^{(n)})^2 \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_6 \end{bmatrix} \tag{3.12}$$

Equation 3.12 shows that the proposed model has an interesting property: although it is nonlinear in terms of the original variables, it is linear in the lifted feature space. Therefore, we can revisit the discussion of linear regression and find the best-fit coefficients by taking the derivative of equation 2.9 with respect to w, finding:

$$w = (\varphi(X)^T\varphi(X) + I_{d'}\lambda)^{-1}\varphi(X)^T y \tag{3.13}$$

This approach gives a theoretical framework for building models from linear combinations of nonlinear functions. Note that the dimension of the inverse in equation 3.13 is now $d' = 6$, larger than the $d = 2$ linear case, and thus the computational cost of solving the equation is, in principle, $\sim 30\times$ larger. The same comments about regularization apply to nonlinear models, as they did to linear regression. In general, the dimension of the lifted feature space will grow rapidly with the dimension of the underlying features. For example, extending general second-order polynomials to a four-variable case (including all cross-terms) leads to 16 coefficients. Fortunately, the kernel formulation proposed in equations 3.8–3.9 will also apply. One can operate equivalently on the

$n \times n$ kernel matrix, which is defined for this case as:

$$K = \varphi(X)\varphi(X)^T \in \mathbb{R}^{n \times n}; \tag{3.14}$$

$$K_{i,j} = \left\langle \varphi(x^{(i)}), \varphi(x^{(j)}) \right\rangle = \begin{bmatrix} 1 & \sqrt{2}x_1^{(i)} & \dots & (x_1^{(i)})^2 & (x_2^{(i)})^2 \end{bmatrix} \begin{bmatrix} 1 \\ \sqrt{2}x_1^{(j)} \\ \vdots \\ (x_1^{(j)})^2 \\ (x_2^{(j)})^2 \end{bmatrix} \tag{3.15}$$

Again, we see that the kernel matrix elements are inner products but in the transformed space instead of the underlying features. As equations 3.8–3.9 apply, the number of parameters in the kernel version of the regression problem and the dimension of the matrix that needs to be inverted is only n, the number of training points, regardless of the dimension of the feature transform. Even if one uses an arbitrarily high-dimensional feature transform to make a prediction at a new point x^*, only one coefficient is needed for each of the training points and the influence of each training point $x^{(j)}$ scales linearly with $\left\langle \varphi(x^{(i)}), \varphi(x^{(j)}) \right\rangle$. This is useful only if the computation of these inner products is straightforward and of low computational cost. Fortunately, this is often the case—for a quadratic model, we have the following expression for the kernel:

$$K_{i,j} = \left((x^{(1)})^T x^{(2)} + 1 \right)^2 \tag{3.16}$$

Readers may find it a useful exercise to verify the equivalence of equations 3.15–3.16. By using this expression to form the kernel, we can calculate all terms that we need for the regression problem using only vector products in the dimension of the original features, in this case, 2. This means that calculating and inverting the kernel matrix scales with the number training points as n^3 but does not scale with the dimension of our transformed feature space as long as we can compute the kernel efficiently. We can use our computed kernel in equations 3.8–3.9 to solve the regression problem. Before moving on, it should be clear that not all possible nonlinear models can be formulated as linear mdoels in a transformed space. For example, we could not fit models that were nonlinear in their parameters and in the data such as $w_1 \sin(w_2 x_1)$. General nonlinear minimization must be used in such cases, and the loss function must be optimized numerically. Neural networks are a good example of this problem.

3.4 Kernel Ridge Regression

We have seen how the concept of kernels allows us to pose general regression problems that are linear in their parameters as linear regression in a larger, lifted space. We have also discussed how inner products in these spaces can be used to obtain the coefficients at fixed cost by relating new points to training data only. This general approach is kernel ridge regression (KRR). These models have widespread application for prediction of energy and various electronic and orbital properties,[4,43,75–77] electron densities,[78] and catalytic properties for heterogeneous[79] and homogeneous[80] systems based on structural molecular descriptors. Many of these models use relatively small numbers of training examples ($\leq 10^4$ observations), especially compared with artificial neural network (ANN) models trained for similar purposes. This suggests that KRR methods are a good choice for applications with more limited data. KRR models are also a good means of comparing

different feature sets and identifying which representations best describe the appropriate chemical similarity[81].

The theory that underlies KRR models is called reproducing kernel Hilbert spaces (RKHSs)[59,82,83] and is one of the best-developed and most understood branches of machine learning. Hilbert spaces are spaces of functions, and the theory underlying kernel methods is very general, applying to problems posed in the space of both functions and finite collections of transforms[84] such as the quadratic polynomial that we studied in Section 3.3.

Notice that the variable transform introduced in Section 3.3 is characterized completely by its kernel function, equation 3.15. This function defines data similarity as the inner product in the transformed space. We can reverse this observation by assuming there is a manner through which we can already define similarity. Thus we may bypass the explicit transformation by selecting a kernel $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R} = \langle \varphi, \varphi \rangle$ and not explicitly considering the transformation function $x \to \varphi(x)$. A question is whether any function can be the kernel. Broadly, no, but a wide range of functions are suitable. The minimum requirements for kernels are given in Mercer's theorem,[83] but we generally want to formulate kernel functions that satisfy the following criteria[84]:

1. The function must be non-negative and symmetric: $k(x', x) = k(x, x') \geq 0$ for all inputs.
2. The kernel must be positive definite, that is, $\sum_{i=1}^{n} \sum_{j=1}^{n} k(x^{(i)}, x^{(j)}) c_i c_j \geq 0$ for all possible sets of n training data $x^{(i)}, x^{(j)}$ and real coefficients c_i, c_j; this is equivalent to saying that eigenvalues of the kernel matrix are ≥ 0, which we need to compute equation 3.9 anyway.

These requirements are a direct consequence of the kernel needing to correspond to an inner product in the transformed space. All kernel functions discussed thus far were explicitly derived as inner products and satisfy the requirements automatically. In general, we can then select a function to be the definition of similarity and then confirm that it is a suitable kernel, even if the corresponding feature transform, φ, is not known. This simplifies the machine learning task because determining the most suitable transform in high dimensions is challenging. Quantifying similarity between points can be more straightforward.

To be concrete, KRR models are solutions to an optimization problem, which we illustrate with the least squares loss function:

$$\min_{w} \sum_{i=1}^{n} \left(y^{(i)} - y_{\mathrm{KRR}}(x^{(i)}) \right)^2 + \lambda \left\| w \right\|^2 \tag{3.17}$$

$$\text{where } y_{\mathrm{KRR}}(x^{(i)}) = \sum_{j=1}^{d'} w_j \varphi_j(x^{(i)}) = \sum_{k=1}^{n} a_k k(x^{(k)}, x^{(i)}) \tag{3.18}$$

The optimized KRR models consist of linear functions of a transformed basis set φ_i that minimize the regularized loss function. Instead of working in this transformed set, we will work in the kernel view. There is an important consideration that we have not emphasized in the discussion of the kernel problem thus far. One can always express the optimal solution to the regression problem as a sum over n coefficients a (i.e., one per training point) instead of d' coefficients w (i.e., one per transformed dimension). This statement holds regardless of the dimension of w. Even in the case of infinitely many basis functions, one needs only as many degrees of freedom as there are training points. This result is known as the representer theorem.[82] It is a direct consequence of the properties of Hilbert spaces.[4] We will not walk through this detail here, but the key idea is

[4] for a recent, general proof see Ref. [82].

that the minimizer of the training error can be expressed as a function of the training points only. The presence of the regularization term in equation 3.17 ensures that any additional terms increase the regularization penalty without decreasing the loss function value. This is how, even in the case of infinitely many basis functions φ, we can still apply KRR and obtain the optimal function and parameterize it with exactly n variables.

3.4.1 Selecting kernel functions

Having outlined the theory of KRR, we turn to the practical question of selecting kernel functions. The kernel function determines how similar our inputs are; therefore, the most obvious idea is to base kernels on the geometric distance between points in their feature space, $\left\|x^{(i)} - x^{(j)}\right\|_2^2$. This type of translation-invariant kernel is called "stationary." It has inherent advantages because it is invariant to constant shifts in the feature space. This means that only the relative similarity between inputs matters. In a chemistry context, using atomic coordinates as input features, it could be that the energy of a bond would be modeled only as a function of the pairwise distance (and possible angles) between atoms and not their absolute position, as is the case in force fields. However, just using the raw distance would not make a reasonable kernel because it would assign no similarity to nearly equal points ($\left\|x^{(i)} - x^{(j)}\right\|_2^2 \approx 0$), whereas distant points would have large kernel terms. A family of kernels can be obtained by taking the negative exponent of the pairwise distance, giving the Gaussian or radial basis function (RBF) kernel:

$$K_{i,j} = \exp\left(-\gamma\left\|x^{(i)} - x^{(j)}\right\|_2^2\right) = \exp\left(-\gamma\sum_{k=1}^{d}\left(x_k^{(i)} - x_k^{(j)}\right)^2\right) \tag{3.19}$$

In comparison to the linear kernel function discussed earlier, pairwise similarity in this case is an exponentially decaying function of the distance between each point. The prediction of new points is taken as a linear function of these pairwise distances (Figure 3.2). Note that this kernel depends on one hyperparameter, $\gamma > 0$, sometimes referred to as the "inverse correlation length." The inverse correlation length controls how quickly the similarity function decays as the distance between points increases. Recall that the magnitude of $K_{i,j}$ governs how large values of γ correspond to rapid decay and very "local" influence in feature space, whereas small values of γ correspond to long-range interactions. In some formulations, the hyperparameter in equation 3.19 is placed in the denominator in closer analogy to a Gaussian distribution.

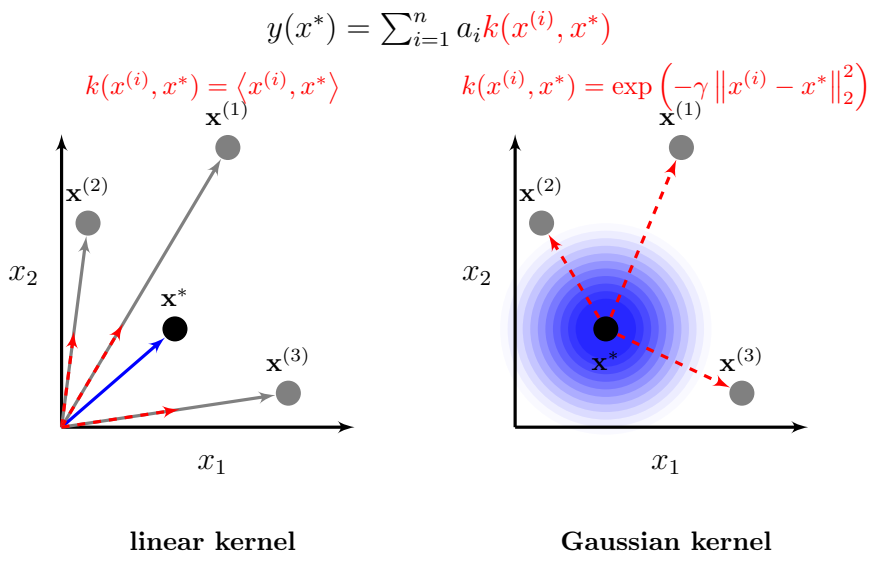

Figure 3.2: Comparison of linear and Gaussian kernel similarity in a function of two input dimensions (x_1, x_2) evaluated at a new point x^*: the linear kernel (left) considered the inner product between x^* and the training points $x^{(1-3)}$ while the Gaussian kernel (right) considers a decaying exponential of the Euclidean distance between x^* and the training points $x^{(1-3)}$, illustrated as a fading blue region of influence. Kernel expressions and the overall regression equation are given above.

It is useful to examine what nonlinear feature map, $\varphi(X)$, underlies the kernel in equation 3.19. In this case, the underlying feature map is not finite dimensional. It is an expansion into infinite series from the Taylor series expansion of the exponential function, shown for $x \in \mathbb{R}$ for simplicity:

$$\varphi(x) = e^{-\gamma x^2}\left[1 \quad \sqrt{\tfrac{2\gamma}{1!}}x \quad \sqrt{\tfrac{2^2\gamma^2}{2!}}x^2 \quad \sqrt{\tfrac{2^3\gamma^3}{3!}}x^3 \quad \cdots\right]$$

Therefore, performing KRR with this kernel is equivalent to linear regression in this functional space. Many other choices for one-hyperparameter kernel function are possible (Table 3.1). In general, the different kernels can be distinguished by how quickly they decay with respect to the distance between points (e.g., Figure 3.3), and it is difficult to know which kernel is best to use in a given problem a priori. Although most applications use the Gaussian kernel, Lapacian[75] and Matern[85] kernels (as in Table 3.1) have been reported to offer superior performance in some applications. It is worth trying a number of different kernels and comparing the results.

A drawback to KRR methods is that they do not scale well as the number of training examples becomes large (although they have been routinely applied to $\sim 10^4$ examples). Recent work[86] has attempted to address this issue by fragmenting organic molecules into local environments that generalize with fewer (100s–1000s) training examples. In practice, it is best to test a number of kernels and select the best one through cross-validation. It is also possible to introduce additional hyperparameters to make the kernel more flexible. For example, a nonspherical Gaussian kernel

has the form:

$$K_{i,j} = \exp\left(-\sum_{k=1}^{d} \gamma_k \left(x_k^{(i)} - x_k^{(j)}\right)^2\right) \tag{3.20}$$

with one correlation scale (γ_k) per dimension of the feature space, for a total of d parameters. This form can allow different dimensions of the feature space to vary independently, correlating points based on the directions in which they are different instead of by distance alone. The drawback of this type of approach is the increase in difficulty estimating these parameters robustly. In practical chemical applications,[75,87] equation 3.19 is often preferred.

Table 3.1: Some commonly used kernel terms and their hyperparameters.

Name	Expression
Gaussian/RBF	$k(x^{(i)}, x^{(j)}) = \exp\left(-\gamma \left\|x^{(i)} - x^{(j)}\right\|_2^2\right)$
Exponential	$k(x^{(i)}, x^{(j)}) = \exp\left(-\gamma \left\|x^{(i)} - x^{(j)}\right\|_2\right)$
Laplacian	$k(x^{(i)}, x^{(j)}) = \exp\left(-\gamma \left\|x^{(i)} - x^{(j)}\right\|_1\right)$
Matérn 3/2	$k(x^{(i)}, x^{(j)}) = \left(1 + \gamma \left\|x^{(i)} - x^{(j)}\right\|_2\right) \exp\left(-\gamma \left\|x^{(i)} - x^{(j)}\right\|_2\right)$

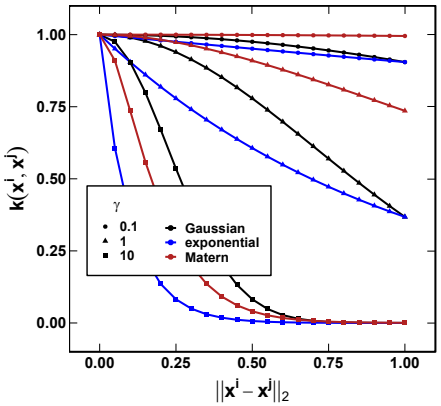

Figure 3.3: Comparison of some common kernel functions as a function of Euclidean distance between points, illustrated for a series of γ values.

3.4.2 Selecting hyperparameters

Similar to MLR, once hyperparameters are selected, the kernel coefficients a can be solved for directly using equations 3.8–3.9 without any iterative solvers or other approximations. The primary difficulty and computational expense incurred when training KRR models is rigorous hyperparameter selection. Selecting the wrong hyperparameters can have a large impact on model performance, as judged through errors in the set-aside test data. A typical task using a one-parameter kernel from Table 3.1 involves selecting two hyperparameters: the kernel length scale, γ, and the regularization strength, λ. Both of these hyperparameters can be thought of in terms of tuning the

smoothness of the function. As γ becomes large, the support of the kernel function around each data point becomes narrower, and the fitting function becomes sharper or higher frequency. Taking large values of λ biases toward smooth functions.

We can examine these effects by considering what happens to equation 3.19 as γ becomes very large. All off-diagonal entries of the kernel matrix will decay to zero, giving $K \approx I$. Applying this to equations 3.8–3.9 with limiting of large γ gives:

$$y(X) = Ka = I_n(I_n + I_n\lambda)^{-1}y = \begin{bmatrix} \frac{y_1}{1+\lambda} \\ \vdots \\ \frac{y_n}{1+\lambda} \end{bmatrix}$$

This process will return the training data y exactly for small λ and provides a transparent example of why training errors cannot be used for hyperparameter selection. It is also clear that including the regularization parameter will worsen the fit to the training data. Regarding predictions on a new point, consider that the vector $K(X, x^*) \approx 0$ for large enough γ if x^* is not in the training set. We see that the prediction for out-of-sample points will be zero:

$$y(x^*) = K(X, x^*)a = \begin{bmatrix} K(x^{(1)}, x^*) \\ \vdots \\ K(x^{(n)}, x^*) \end{bmatrix} a = 0$$

It is typically recommended that the out-of-sample data be scaled by the training data average. The result will be that we will make the average training prediction, $\bar{y}$, for all out-of-sample points because the $\bar{y}$ value is added back when converting into original units. For KRR models, this observation also highlights the importance of noting when out-of-sample data are so far from available training data that the KRR model test error observed is simply the deviation of these points from the training data mean.

For the large λ limit, one can neglect the kernel term in equations 3.8–3.9, leading to:

$$y(X) = Ka = K(I_n\lambda)^{-1}y = K \begin{bmatrix} \frac{y_1}{\lambda} \\ \vdots \\ \frac{y_n}{\lambda} \end{bmatrix} = 0$$

In this limit, the training predictions are zero (or the average, if the data have been scaled). In this case, coefficients a are also zero, so the predictions based on any out-of-sample point will also be zero. Consequently, both hyperparameter limits give the same out-of-sample predictions but for distinct reasons (Figure 3.4). The high γ model is a corrugated surface, rising up to meet each training point and falling away just as quickly to zero in between them. The high λ model is flat everywhere and very smooth. Neither of these limits has practical use, and realistic models will lie somewhere in between these extremes. However, verifying this asymptotic behavior is a good debugging exercise. This analysis can reveal, for example, two training points that are similar in the feature space but distinct in the property space, likely due to erroneously labeled or prepared data. Having two points with very different output values close together requires a shorter correlation

length, because the model needs to traverse that range over a short distance in the feature space. The resulting model will be shifted toward the red extreme in Figure 3.4.

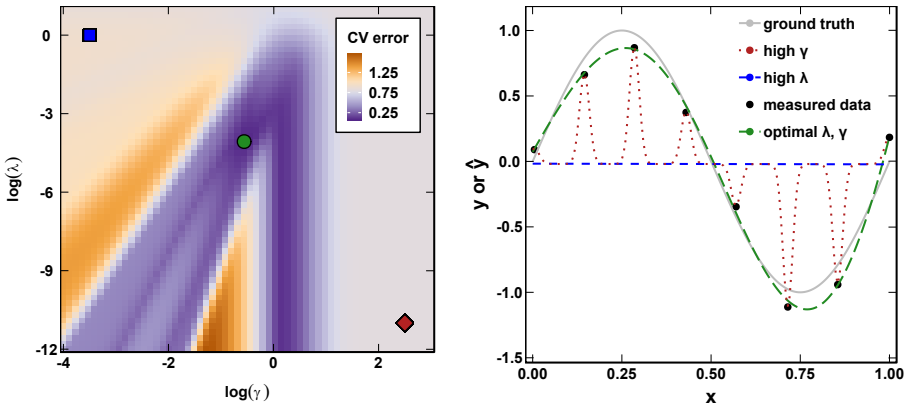

Figure 3.4: Comparison of fitting the function $y = \sin(\pi x)$ based on 8 points uniformly sampled in $[0, 1]$ with measurement noise $\mathcal{N}(0, 0.15^2)$ using KRR with a Gaussian kernel. Hyperparameter estimation by 2D grid search (left) shows complicated dependence of the leave-one-out cross validation (CV) error as a function of γ and λ. The optimal values, $\gamma = 0.28$ and $\lambda = 8.6 \times 10^{-5}$ are shown as a green circle, while an example of a too-high value of $\gamma = 10^{2.5}$ is shown as a red diamond while a too-high value of $\lambda = 10$ is shown as blue square. The result on the predicted function for these hyperparameters is shown with lines in corresponding colors (right). Sampled points are shown in black, with the ground truth shown with a gray line.

Because the two hyperparameters are coupled, they must be chosen simultaneously. A standard approach would be to conduct a two-dimensional, grid-search, cross-validation scheme evaluating all combinations for a range of λ and γ values. It is possible to obtain multiple local minima (Figure 3.4), representing locally optimal models with different levels of smoothness. Although an exhaustive grid search is computationally demanding, it is easily automated. Solving for the coefficients is direct for each combination, so the grid search can be completed reasonably quickly. The search is limited by the time needed to invert the kernel matrix, which is fractions of a second on commodity hardware for even a few thousand points. This observation is true even when the kernel type is also varied in an outer loop. Because it is challenging to know which ranges of hyperparameter values are appropriate for a new problem, it is often useful to start with a coarse grid of values (anything from 10^{-6} to 10^{2} or larger) and then carry out a finer grid search in promising regions of the parameter space.

3.5 Gaussian process regression

Gaussian process regression (GPR) provides an alternative theoretical basis for the type of regression discussed in Section 3.4. In general, the GPR framework is invoked when it is desirable to use the machine learning model to recommend data points for acquisition or to track model uncertainty. Kernel models in the guise of Gaussian processs (GPs) have been applied widely in chemistry. So-called Gaussian approximation potentialss (GAPs) are notable examples.[88,89] This class of GPs

uses a custom definition of kernel similarity based on three-dimensional geometry to re-create the potential energy surfaces of DFT simulations. GAPs have been used to create low-cost, accurate potentials for various bulk and atomic systems,[90–93] exploiting the built-in uncertainty estimates to decide when new simulations are needed for a particular atomic configuration. GPs have also been used to accelerate geometry optimizations directly,[94] to predict dispersion interactions,[95] to correlation energies[85] and reaction outcomes,[96] and to interpolate between levels of theory for band-gap predictions.[97]

GPR produces similar predictive model forms but does so through distinct reasoning. Similar to KRR, the literature and interpretations of GPR are rich.[98] For completeness, we briefly review how GPR relates to the derivation of kernel methods.

Within the GPs framework, the data-generating process is modeled as a probabilistic relationship between inputs X and outputs y. The probability distribution of the output variable(s) can be determined based on the set of accumulated observations and conditioned on observing a specific new input, x^*. A GP is defined as a collection of random variables where any finite subset has a joint Gaussian distribution, which we assume to have a zero mean. This common and practical assumption is known as a stationary GP.[98]) Modeling the regression function as a stationary GP yields:

$$y_{\mathrm{GP}}(x) \sim \mathcal{N}\left(0, k(x, x')\right) \tag{3.21}$$

The function value at a given position is related to the value at all other inputs by a covariance function, $k(x, x')$. Covariance functions can be selected using the same criteria used for the kernels in Section 3.4, and any of the functions in Table 3.1 are suitable. Although the Gaussian kernel is a popular choice, it is not required or implied that it be a GP; this is a potential point of confusion. Specifying this covariance function completely defines the GP. When it comes to the regression task, one aims to determine the distribution around a given new point, x^*, based on the training data X and y. Their joint distribution may be written as follows:

$$\begin{bmatrix} y_{\mathrm{GP}}(X) \\ y_{\mathrm{GP}}(x^*) \end{bmatrix} \sim \begin{bmatrix} K(X,X) & K(X,x^*) \\ K(x^*,X) & K(x^*,x^*) \end{bmatrix}$$

It is also standard to interpret the observations X as having some inherent noise σ^2. In that case, the self-interaction kernel (i.e., the diagonal elements of the kernel matrix) are replaced with $K(X,X) + I\sigma^2$. The GP need not exactly interpolate the previous observations, thus improving numerical stability in the same manner as ridge regression in KRR. Through manipulation of the above expression for the conditional distribution, where $K = K(X,X)$, one can show:

$$p\left(y_{\mathrm{GP}}(x^*)|X,y\right) = \mathcal{N}\left(K\left(K+I\sigma^2\right)^{-1} y, K(x^*,x^*) - K(X,x^*)\left(K+I\sigma^2\right)^{-1} K(X,x^*)\right) \tag{3.22}$$

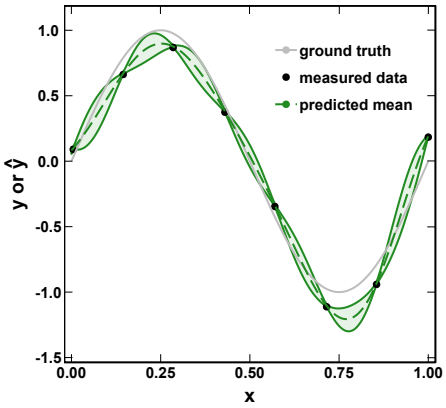

Figure 3.5: Fitting the function $y = \sin(\pi x)$ based on 8 points uniformly sampled in $[0, 1]$ with measurement noise $\mathcal{N}(0, 0.15^2)$ using a GP with a Gaussian kernel and hyperparameters estimated as per Figure 3.4. The mean predicted value is shown as a dashed green line with the shaded region representing one standard deviation above and below the mean. Sampled points are shown in black with the ground truth shown with a gray line.

This expression enables prediction of the average and most likely value for the function at x^*, given by $K\left(K + I\sigma^2\right)^{-1} y$. Setting $a = \left(K + I\sigma^2\right)^{-1} y$ and identifying $\sigma^2 = \lambda$, we also recover the original KRR expressions, equations (3.8–3.9). Thus, one obtains the same prediction for the same choice of kernel from the GP mean value, although with a slightly different interpretation.

What advantage does the GPR framework provide over KRR? The key is that equation 3.22 gives not only a point-value prediction but also a distribution of possible values for the function at each new point. This means that the GPs comes with built-in estimates of system-specific uncertainty, with larger variance predicted for points with which the model is least certain. This predicted uncertainty, given by the variance in equation 3.22, is a direct function of the pairwise distance between a new point and the training data (Figure 3.5). Note that this variance goes to zero if $x^* = X$ and σ^2 is small, meaning that the uncertainty at the training points goes to zero (or at least to σ^2).

This built-in uncertainty estimate makes GPs a popular tool for optimization or exploration problems in which data-driven decisions are needed about which new points to simulate. A well-known example is the "expected improvement" algorithm,[99] also known as efficient global optimization (EGO). This algorithm provides a useful sampling strategy for optimization with a GPs. Expected improvement strategies have been used to search for low-energy conformers[100] and crystals with targeted melting temperatures[101] in an "uncertainty-aware" manner.

Another advantage of the GP probabilistic framework is that the likelihood of observing the training data for a fixed set of hyperparameters can be computed explicitly. This approach provides another metric of how well the proposed model represents that data. Because this probability can be obtained as an analytic, differentiable function of the hyperparameter, one may choose hyperparameters by maximizing this probability directly, rather than relying on the validation-set strategies discussed earlier. In practice, such optimization typically suffers from multiple local minima. If attempted, it should be carried out multiple times with distinct initial guesses.

3.6 Trees and Random Forests

Tree-based models are another family of models that cannot be neatly described by the kernel framework. Nevertheless, they are easy to understand, quick to train, and often highly accurate. Categorical or discrete variables are treated naturally in tree-based models. Single decision trees can be useful for understanding chemical systems[8,102] and identifying important features.[81,103,104] Tree-based models grouped together into random forest (RF) models are often used as a baseline for comparison with other methods[43,75,105] and have been found to outperform other methods in some applications.[104,106–108] They have been extensively applied to pharmaceutical quantitative structure–activity relationship (QSAR) tasks[109–111] and bioinformatics.[112] Although they may be less á la mode than kernel methods or neural networks, they are still used as primary regression models for atomic properties such as partial charges,[113] catalytic reaction yields,[108] and gas molecule absorption energies[114] and in protein-ligand interaction models.[58,115] The training of tree and RF models is often easier than for KRR models. Tree and RF models are less sensitive to hyperparameter selection[106], which makes them a good choice for rapid prototyping.

The basic unit of all tree-based models is a *binary decision tree*, which consists of a series of sequential decisions called "branches." Each branch divides the data based on one input feature. The final branch terminates in a set of nodes called "leaves." Each leaf represents the output that corresponds to a particular chain of binary data partitions. The content of the leaf nodes depends on the type of application: for regression, the leaves represent a single output value; for classification, the leaves correspond to different categorical labels. For example, data that satisfy each of the three successive decisions, shown in blue, in Figure 3.6 results in an output leaf value of 0.5. Tree models are constructed using the training data to define the branches and then can be used to make predictions based on new data, selecting the appropriate output value by following the tree decision chain.

We will briefly compare the type of coupling between input features in RF models with those in the MLR and KRR models seen previously. Because tree models are branching decision processes, the input features interact even though only one is considered at a time (Figure 3.6). This means that the impact of changing one feature depends on features involved in previous decisions in the tree. For example, in Figure 3.6, the effect of $x2$ depends on the value of $x1$ because of its position in the first branch. In contrast, MLR treats each variable as fully independent, so increasing or decreasing one feature always has the same effect regardless of other features. Distance-based kernels used in KRR are fully interacting in that the values of all features contribute to the distance between points; however, the distance is not sensitive to specific features causing the difference between the points—any two points that are d units apart in feature space have the same kernel term.

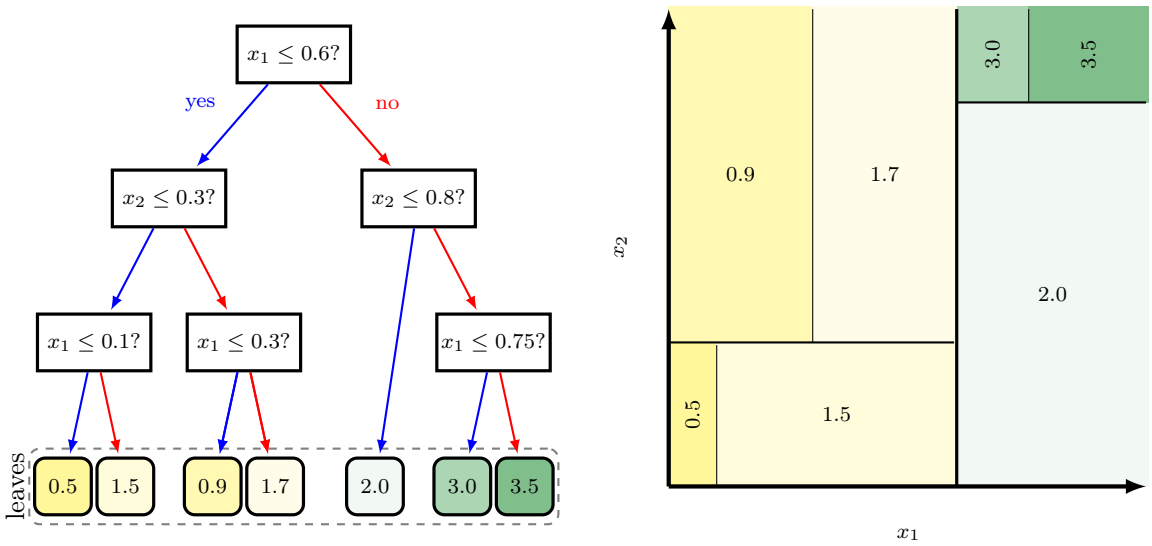

Figure 3.6: Example regression tree model based on two variables x_1, x_2 showing a series of sequential partitions with yes in blue and no in red (left), leading to constant regression values $\hat{y} = f(x_1, x_2)$ in the leaves (colored from yellow to green) resulting in a piecewise constant response surface (right).

Training of tree models involves choosing how to divide the data at each branch, based on the values of the input features. This choice is typically made using the *Classification and Regression Trees* (CART) algorithm.[116] The branches in the model are selected in a "greedy" manner. The best single variable to divide predicted outcomes and to minimize the standard deviation (if using least squares loss function, eq. 2.2) of observations on either side is chosen for each split. This process is considered greedy because it seeks the best local decision at each branch without considering later splits. The procedure is repeated on each subsequent branch and continued up to a user-defined maximum number of branches or until the minimum leaf size is reached. The leaf size is the number of training observations placed in each leaf. Each observation could be placed in its own leaf by adding sufficient branches, but the resulting model would be very complicated and likely overfit to the training data (Section 2.6). The maximum number of branches and the minimum number of observations in each leaf are hyperparameters corresponding to the maximum complexity of the tree model and can be used to control overfitting. The advantage of this model type is that it can be readily understood by humans (i.e., can be interpreted), especially when the number of branch levels is modest (i.e., < 5). It also computes the binary decision tree quickly.

Single trees tend to perform poorly in regression tasks and have limited accuracy. This problem can be addressed by training multiple tree models and aggregating the results.[117] Using a general technique called "bagging," one can train simple regression models on resampled sets of the training data. These resampled sets are generated by uniform sampling with replacement. This means that new data sets of the same size are generated by uniformly choosing points to copy from the original

data set and allowing duplicate points and omissions across the different bags. Each bag is used to train a different tree model, and the results are averaged to give a single prediction that can be substantially more accurate and stable than any of the component models.[117] Because trees are very fast to train, they are good candidates for bagging.

A highly successful[60] model family called *random forests* (RF) uses a bagging variant in combination with tree-based submodels. A RF is composed of a set of tree models that are trained on resampled 'bags of the same size as the training data. RF introduces additional complexity by training models on bags containing different random subsets of possible features (represented as shapes in Figure 3.7). This approach introduces more diversity into the models and improves overall performance of the forest.[118,119] Because the cost of training each tree is low, large numbers of trees (i.e., hundreds to thousands) can be used.

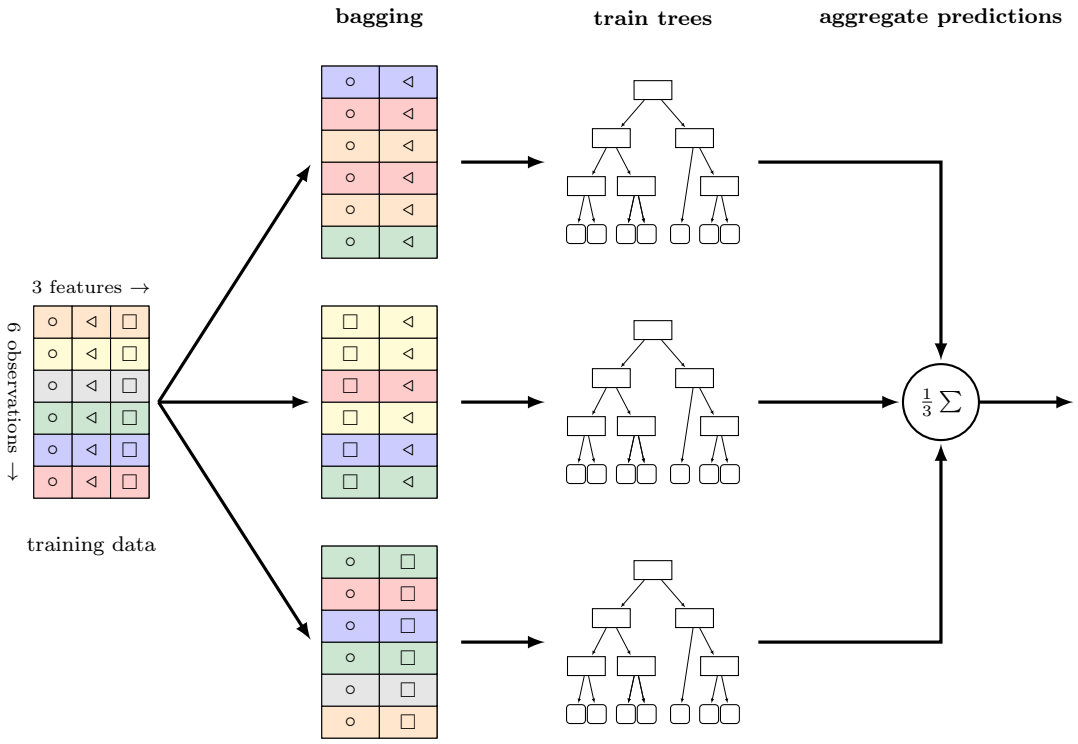

Figure 3.7: Schematic process for training a random forest model with three trees, showing how data (each colored row corresponding to one observation) is partitioned into equal bags by sampling with replacement and random input feature sub-setting (different input features represented by shapes). Each bag is used to train an independent tree model, and the predictions of all trees are averaged to give the final prediction.

Although some of the interpretability of a single tree is lost in RF, there are additional benefits

beyond improved accuracy. An estimate of the out-of-sample error (here called the "out-of-bag" error) can be derived by keeping track of which data points are absent from each tree and using each tree to evaluate points that were absent from its own data. This gives the forest a cross-validation-like error estimator that can be used to assess the impact of hyperparameters without resorting to explicit validation partitions. For example, the performance of the first two trees in Figure 3.7 on the gray input would be used to estimate out-of-bag error.

Similarly, performance of sets of trees that exclude certain features can used to estimate the importance of these excluded features. In brief, a feature may be important if the trees without that feature perform badly, and the opposite applies for features that do not improve predictions noticeably. These built-in estimates are usually calculated on the fly in RF implementations and give immediate insight into the influence of features on model predictions.

3.7 Summary

Supervised machine learning methods can be simple and interpretable:

- Multiple linear regression (MLR) is routinely used to model complex systems. It produces estimates for a quantity of interest as a linear function of dimensional observations. Good practice starts with MLR to establish baseline model performance before attempting nonlinear models.
- Nonlinear regression models the effects of arbitrary functions of features and interactions between functions. The “kernel trick” allows expression of nonlinear models as linear in a transformed space.
- Kernel ridge regression uses training data to obtain an optimal function and parameterize it with a precise number of variables. It works well with limited data to compare different feature sets and to identify which representations best describe the appropriate similarity.
- Hyperparameters are set for a model before the training process begins and have a large impact on model performance. Common hyperparameters for selection are kernel length scale and regularization strength.
- Gaussian process regression is used to recommend data points for tracking model uncertainty. This framework is popular for optimization or exploration problems that require data-driven decisions about new points for simulation.
- Tree-based models grouped together into random forest models are often used as a baseline comparison for other methods. These models are easy to understand, quick to train, and often highly accurate.

Chapter 4

Representations of Atomistic Systems

4.1 Overview

All machine learning models take numbers, vectors, and matrices as inputs and transform them to outputs. Therefore, if we want to apply machine learning to molecules or materials consisting of atoms, we need to translate these chemical structures into mathematical form. This task is referred to as "representation development" or "featurization," in which we assign a set of numerical features, or descriptors, that describe each observation. Although these terms are often used interchangeably, descriptors are commonly associated with multiple linear regression and quantitative structure-property relationship development (e.g., physical properties of molecules and atoms, such as electronegativity or pKa). Featurization typically implies a less intuition-guided and systematic approach. Featurization might involve, for example, applying mathematical operations such as symmetry functions that are unintelligible to humans but that extract detailed descriptions of three-dimensional (3D) atomic environments. Molecules or materials generally correspond to discrete objects in chemical space, $c^{(i)}$, that we may then featurize in a manner that produces discrete vectors, $x^{(i)}$, in some d-dimensional vector space $\mathcal{X} \subset \mathbb{R}^d$ (Figure 4.1). It will be desirable to build a feature space in which molecules or materials with similar properties are proximal to each other (i.e., a small $d(x^{(i)}, x^{(j)})$ in Figure 4.1), but it can be challenging to know a priori how to build such a feature space. This chapter tackles these challenges.

Beyond kernel models, featurization that preserves similarity in property space greatly simplifies the construction of regression models and increases the likelihood that they generalize well. In addition, the choice of representation is closely linked to the interpretability of the resulting models. More complex, high-dimensional featurizations may lead to better model performance in some cases but will make understanding the model predictions more challenging.

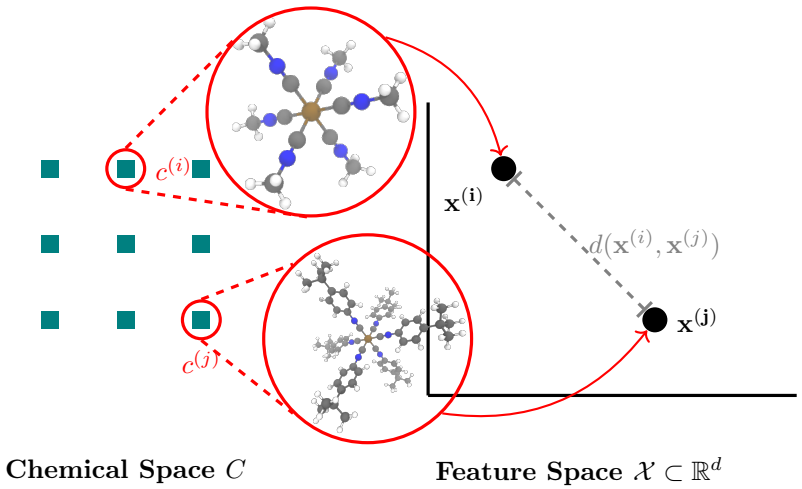

Figure 4.1: Basic overview of the featurization process, mapping elements in chemical space to vectors that represent them in descriptor space. The choice of the featurization governs the geometric similarity and overall geometry of the descriptor space.

For these reasons, a significant body of work has focused on the development of representations for atomistic systems[120–122]. The earliest developments date back to the 1960s–1980s in the cheminformatics community, and featurization continues to be an area of active research.[91,121] Unfortunately, one cannot expect a single, ideal featurization for the broad range of challenges of interest to researchers in the chemical sciences. A secondary factor to consider is the computational cost of the featurization. When the model must be evaluated rapidly, as in a neural network potential (NNP),[2] the featurization must be computed efficiently and focus on encapsulating the stretch, bend, and torsion of local atomic environments. Conversely, prediction of reaction outcomes, equilibrium properties from electronic structure calculations, or adsorption of gas molecules on catalyst surfaces will motivate distinct featurization strategies. In fact, early quantitative-structure property relationships[120] and even more recent models[123] have made use of electronic structure descriptors obtained from density functional theory calculations, which generalize well but would be cost-prohibitive to compute for rapid ML model evaluation. The choice of representation must be paired with the application and objectives in mind. This is also known as "feature engineering" in the ML literature, and more than in any other area of machine learning, the application specialist (e.g., the chemical scientist) is most easily able to incorporate their knowledge and intuition. This chapter will review some representative strategies for approaching feature engineering.

4.2 What Makes a Good Feature Space?

Before addressing specific featurization strategies for chemical systems, we will consider what makes a feature set broadly suitable for a given problem. Some ideal characteristics of representations are typically realized imperfectly by real feature sets. In particular, the ideal the representation should be:

1. Invariant to equivalent inputs that correspond to the same physical system and conversely should should provide a unique encoding for each distinct input.

2. Able to encode property (i.e., chemical) similarity by placing molecules that have similar properties relatively close in the feature space, and vice-versa.A more mathematical way of stating this concept is to say that the distribution of observations in the feature space should correspond with the distribution in property space because this makes the learning task simpler, as is especially apparent in the kernel models.

3. Affordable and straightforward to compute as needed based on the available data and the desired number of model evaluations.

4. Easy to interpret by the scientist to enable extraction or encapsulation into derived heuristics or principles.

5. Able to be decoded (or "inverted") back into a real molecule.

The need to correctly account for invariances in the physical system can be easily understood in the context of representing 3D geometries, where system properties are invariant to global translations and rigid rotations. Ideally, the representation should be robust to these changes and map them to the same point in the feature space. This rotational and translational invariance is easily obtained using an internal coordinate representation that depends solely on interatomic distances and angles, rather than working with Cartesian coordinates directly. This approach has been used in the development of force fields in chemistry for decades. However, the representation must not encode false invariances (by mapping two actually distinct systems to the same point), because if two points in chemical space are mapped to the same point in feature space, they will be indistinguishable to the model.

However, creating a unique representation for each input is not enough. The relationship between the features should preserve the chemical similarity of the input: the worse the input-output correspondence is, the more nonlinearity is required of the model to make predictions. This input-output mismatch can be quantified by the Lipschitz constant L, such that $\|\Delta y\| \leq L \|\Delta x\|$. The lower the quality of the feature space, the larger the values of L that are needed to rearrange the data. A challenge is that chemical similarity is poorly defined and will vary depending on what property is being predicted. For example, the Fe(II) substituted-isocyanide complexes in Figure 4.1 have quite different numbers of atoms (37 to 151), and thus atomization energies (e.g., as evaluated with density functional theory [DFT]), but nearly identical separation between the lowest quantum mechanical ground-state spin and next lowest spin state. Both are strongly low-spin directing (the high-spin to low-spin energy difference is ~ 40 kcal/mol, as calculated with hybrid DFT) because this property is dictated more by the direct metal coordination environment than the overall size of the molecule.

Nevertheless, spending too much computational effort constructing a perfect feature space can be counterproductive. If the trained model is to be useful, computation of the representation should be easier than direct computation of the output property that model is intended to predict.[68] This requirement is particularly applicable to training of machine learning models based on first principles calculation in which attributes of the wave function may correlate well with output properties but may be too expensive to compute over large compound spaces. A good example of high-cost descriptors is the use of the d-band center (i.e., energy of the valence electrons) in a bulk metal to correlate to the DFT-evaluated binding energy of an adsorbate to that metal.[124] The first

quantity requires a DFT determination of the energy levels of electrons in the material and thus is a relatively high-computational-cost descriptor. When a representation requires 3D coordinates, this representation is inherently more costly than those that can be inferred from connectivity-level information. For many data sets, 3D coordinates may not be available, or their generation would require the use of force fields or semiempirical theories to generate the structures. The cost of simple structure-based representations may be prohibitive When working with experimental data sets for which 3D geometry may be unavailable or with data for which efficient molecular modeling methods will not generate a reliable structure in a short amount of time.

Items 4 and 5 above are desirable but frequently incompatible with complex, high-dimensional representations. Although interpretability is not critical in terms of model performance,it helps the scientist understand why models make a certain prediction or how to improve predictions when features are intuitively connected to chemical properties. Predictive models trained on interpretable features can also guide suggested changes to composition or structure of molecules or materials to improve properties. In many cases, the scientist may prefer an interpretable model rather than a black box model with slightly better performance. Representations that can be converted back into molecules would enable so-called inverse design strategies in which optimization in feature space helps identify a single optimal molecule or material. Given that the representations are continuous, whereas the space of feasible chemical compounds is discrete, few representations come close to enabling such strategies in practice. A notable recent exception is generative models, which get around this issue with specially trained neural networks. These models are discussed in Chapter 5.

These challenges often cannot be addressed simultaneously for a single feature set. Instead, one must make choices among feature complexity, evaluation cost, and interpretability. Simpler features are usually easily understood and related back to real systems but can have difficulty representing all inputs uniquely. For example, raw counts of the number of atoms of each type cannot distinguish isomers. One should also consider the available data and how the data relates to the resolution of the representation chosen. Simpler representations may work better for small data sets because their lower dimensionality makes input-output mapping simpler. However, simpler representations can quickly stop improving with new training data because they lack sufficient dimensions to describe more nuanced variations in the data set. Conversely, a high-dimensional representation will require larger data sets but, given the large data set limit, can exceed the predictive power with less precise featurization.

Another important distinction must be drawn between what system, or portion of a system, the descriptors represent. In chemistry, it is natural to focus on the properties of individual molecules, as has historically been done in quantitative structure–property relationship (QSPR) modeling. Features are computed on a per-molecule basis, even when the ultimate property being predicted may depend on other molecules present, such as solvent or the human body (e.g., quantitative-structure activity relationship building for therapeutic drug design). In many applications, representations must be computed for multiple molecules at a time or for systems in which the chemical bonding is not strictly covalent. In these cases, the definition of a molecule becomes challenging.—for example, predicting reaction outcomes from multiple reactant species[9,125] to one or more products or predicting the binding energy of reactant species on catalysts on surfaces[7,8].

For periodic systems, such as metal-organic frameworks[126] or crystalline materials,[6] challenges in choosing how to represent the unit cell can be apparent. If multiple molecules are involved, an option is to consider all molecules present as parts of the single, partially disconnected system. These issues can be handled naturally by representing the systems through their connectivity in a molecular graph representation (introduced later in this chapter).

Alternatively, in analogy with classical molecular mechanics force fields, local atomic environments can be featurized. This process essentially converts each observation of a molecule into a large number of atomic features. The output property used in training machine learning models is usually obtained on a per-molecule basis, but some properties, especially the atomization energy [1] or local atomic partial charges,[113] can be naturally broken down into a sum of contributions of individual atoms. Thus, mapping to local atomic environments may be more suitable for some property predictions than others. The use of local atomic environments is central to NNPs.

Increasing ease of training larger neural networks with modern computational resources has motivated the use of *learned representations*, which sidestep complex, precomputed features. Instead, the neural network learns an internal representation as the model is trained, starting from only simple atomic identity and connectivity information.

4.3 Feature Sets for Chemical Systems

Now we turn to discussing some proposed featurization strategies for chemical systems. There is no single way to classify features, we break them down into three broad categories (Figure 4.2), moving from the simplest heuristics to high-dimensional functions that include detailed 3D geometric information. We also provide a few brief worked examples in Section 4.6.

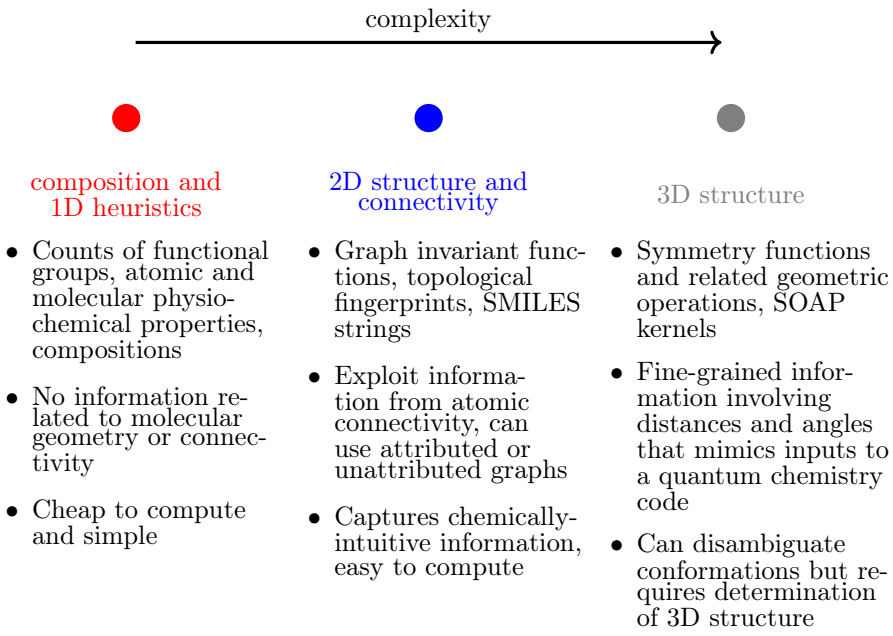

Figure 4.2: Comparison of some common featurization strategies, organized by the level of detail included/complexity.

The simplest feature sets are collections of ad hoc system properties that are counted or inferred

directly from the input molecule. Examples of these features range from the number of each element type present to tabulated, semiempirical, or quantum mechanically calculated, physicochemical properties of molecules such as molecular weight or lipophilicity. These types of properties have been used in the development of quantitative structure property relationship (QSPR) or quantitative structure activity relationship (QSAR) models for decades.[120] When empirical ad hoc features are used, they are typically cheap to compute. However, because constructing these feature sets assumes some knowledge of the most important predictors of a chemical property a priori, models built on these features will not be unique or can be insensitive to small changes in chemistry. Therefore, these feature sets can be susceptible to the biases of the practitioner. To minimize the impact of bias, a large number of potentially useful descriptors should be tested. Testing is convenient with open source cheminformatics toolkits such as RDKit[127] or Open Babel.[128] We will discuss ways to identify which features are important and which are not in Section 4.4. These features have shown good baseline results in many areas, including in challenging problems, especially those that are not well described by covalent bonding. Some examples are periodic systems,[97,129] catalysis,[7,130], and open-shell transition metal chemistry[102].

Fingerprints are an important subset of classical cheminformatics descriptors that are vectorial representations of molecules computed by breaking the molecule into fragments. There is a great degree of flexibility in how this is done and, as a result, a large number of fingerprints. The most basic molecular fingerprints (e.g., the FP2 fingerprint[128]) are binary vectors that count if a specific functional group (e.g., a phenyl ring) is present. Each position in the vector corresponds to a functional group or collection of functional groups. If the group occurs in the molecule, the value is set to 1, otherwise it is zero. In this simplest form, additional occurrences of the functional group are not counted, nor is there any accounting of how proximal functional groups are to each other in the molecule.

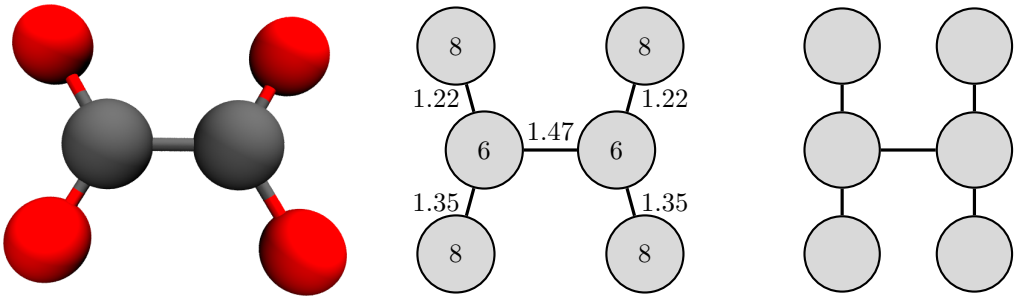

Figure 4.3: Graph representations of oxalate ion, $C_2O_4^{2-}$ (left); a weighted representation with atomic numbers and bond distances (center); and an unweighted representation (right).

To include additional information about functional group proximity, *topological* features are used. These feature sets use information about the connectivity of atoms in the molecule, sometimes including information about the bond order between atoms, without depending on any specific 3D arrangement of atoms. We will also refer to these as features based on the *molecular graph*, which is a mathematical graph with atoms as vertices and bonds as the edges (Figure 4.3). Graph representations may include vertex and edge attributes. The representation can be modified by adding more attributes to the graph (e.g., atomic nuclear charges, bond orders, or even bond

lengths from 3D geometry). Graphs are an attractive mathematical representation for machine learning in chemistry because they inherently reflect the locality of chemistry and the proximity or separation between different components of a molecule or material.

For the simplest representations, graphs may be unattributed, with indistinguishable vertices that only map connectivity and no information about elemental identities. Even this simplified graph model generates useful descriptors. The Randić[131] index, which was proposed more than 40 years ago, is a measure of molecular branching known to correlate well with boiling points of alkanes and is given by the equation:

$$\mathrm{r}_\alpha := \sum_i \sum_j (\deg(i)\deg(j))^\alpha \tag{4.1}$$

Here, $\deg(i)$ refers to the degree, or the number of bonds that node/atom i has in the graph; the sum is taken over all bonds i, j in the graph; and α is a constant, originally $-\frac{1}{2}$ or -1. Other well-known features based on unattributed graphs are the Wiener[132] index and the many descriptors proposed by Kier and Hall,[133–135] which are variants of bond-counting methods that differentiate bonding patterns in molecules.

Attributed graphs readily incorporate more chemical information by associating nodes and edges with specific chemical properties. For example, autocorrelation descriptors[136,137] are calculated as functions of physical properties of the atoms in the graph that are separated by fixed d bonds:

$$\mathrm{AC}_d = \sum_i \sum_j P_i P_j \delta(d_{ij}, d) \tag{4.2}$$

where P refers to an atomic property (e.g., nuclear charge or electronegativity) and $\delta(d_{ij}, d)$ is 1 if atoms i and j are separated by d bonds. By varying the number of bonds considered and using different atomic properties, a large number of possible features can be generated. Autocorrelations can be extended by adjusting their focus on a specific location in a molecule (e.g., metal-proximal groups in inorganic chemistry[102]), by restricting the ranges of the sums to certain subgraphs, or by changing the applied function of the properties, also called the "kernel." For example, one could use a linear combination of P_i and P_j instead of a product. Graph convolutions[138] represent a recent generalization of these graph-kernel methods that learns the function to be applied on neighbouring atoms at training time (as described in Section 5.5.2). The text-based SMILES representation[139] contains the same information as the atom-type attributed molecular graph and has been used as a representation (e.g., in reaction outcome prediction[140,141]). Machine learning models trained on SMILES representations typically exploit progress that has been made in the field of natural language processing.

Many fingerprint methods that use similar topological information are also available, such as Morgan or extended connectivity fingerprints[41,142]. These fingerprints work by collecting a description of the atomic types of a certain number of bonds from each atom. However, rather than producing a numerical value, extended connectivity fingerprintss (ECFPs) concatenate the identifiers of adjacent atoms. This process results in a dynamically derived list of fragments for each atom that can be hashed into a binary vector of fixed length. This is similar to the FP2 fingerprints introduced earlier, but the key difference is that there is no predefined library of substructures. Instead, the ECFPs is dynamically generated based on the molecular graph.

No featurizations introduced so far incorporate 3D structural information; therefore, they would not be suitable for property predictions that require 3D information (e.g., NNPs). As noted previously, most 3D structural descriptors make use of internal coordinates to maintain rotation and

translation invariance, although some models include symmetry operations as part of the model and are able to use raw Cartesian coordinates as inputs.[143] In addition, features should not be sensitive to the atomic ordering in the molecule. Thus, pairwise Euclidean distances $\mathbf{r}_{ij}$ between atoms i and j are a natural starting point. The Coulomb matrix,[144] named for its relation to the Coulomb operator, is a simple example of a 3D structural descriptor. For a molecule with m atoms, the Coulomb matrix is defined as an $m \times m$ matrix with elements:

$$C_{i,j} := \begin{cases} 0.5\left(Z_i Z_j\right)^{2.4} & i = j \\ \frac{Z_i Z_j}{\mathbf{r}_{ij}} & \text{otherwise} \end{cases} \tag{4.3}$$

where Z_i is the nuclear charge of atom i. The exponent of the diagonal elements in the Coulomb matrix was obtained from an empirical fitting procedure on organic molecules. By definition, matrix elements decay with $1/r$, leading to stronger matrix elements between chemically bonded atoms. Although simple and intuitive to build, this representation has some limitations. The labeling order of atoms alters the structure of the matrix, and the Coulomb matrix changes when the number of atoms changes. The first limitation can be solved in a number of ways, for example, by using the eigenvalues of the matrix instead of the elements, by sorting the rows and columns, or by randomly sampling different orderings. The issue of dependence on molecule size must be solved by padding smaller matrices with zeros to match the size of the largest molecule in the training data or on which model prediction would be performed.

Many extensions of the basic idea of equation 4.3 have been proposed, most commonly to include angular information in a similar manner. The "Bag of Bonds"[145] descriptor explicitly factorizes and sorts the elements in the matrix by the atom types involved in each pairwise distance (e.g., grouping all C–C terms). The "bonds, angles" machine learning (BAML) descriptor[121] replaces the distances that have Morse and Lennard-Jones potentials with parameters from force fields. It also includes explicit three- and four-body angular terms. A slightly different family of features can be obtained by inspecting the histograms of all pairwise distances and three- and four-body angles for each group of atom types (i.e., one histogram of C–C distances, one histogram of C–H distances per molecule). Discretizing these histograms into a fixed-length vector results in the histogram of distances, angles, and dihedral angles (HDAD[43]) descriptor. It is intrinsic to the discussion so far that the majority of chemical systems to which these featurizations have been applied are organic molecules with few elements (e.g., C, N, O, H, and F) and atom or bonding types (i.e., all covalent bonds with a range of bond orders).

A different set of features can be obtained by consideration of atomic environments, instead of molecules. As mentioned, these methods are typically associated with NNPs. The atomic environment is defined by a set of *symmetry functions* that encode pairwise distances to neighbors using a square-exponential kernel[1,2], giving a scalar G_i^r for each atom i. A cutoff function, $f_c\left(\mathbf{r}_{ij}\right)$, is used to enforce locality, decaying smoothly to zero for distances larger than a given threshold. These thresholds are typically around 6Åor less to reduce the computational cost of the featurization. This cutoff is within a notably shorter range than electrostatic and van der Waals force length scales typically considered in molecular mechanics force fields. Although good performance has been achieved with only such short-range terms, other approaches have focused on incorporating long-range interactions (e.g., with explicit physics-derived terms[57,146]) alongside these necessarily short-range features. An additional angular term obtained by replacing the square-exponential with trigonometric functions is also used to give angular dependence. These terms then account for local

interactions at each atom by measuring how crowded the local environment is:

$$k_{ij}^{r} := e^{-\eta(\mathbf{r}_{ij}-r_s)^2} \tag{4.4}$$

$$G_i^r := \sum_{i \neq j} k_{ij}^r f_c\left(\mathbf{r}_{ij}\right) \tag{4.5}$$

Typically, a number of different values of η and r_s are used to give a vector of symmetry functions for each atomic environment that decay more or less rapidly and that modulate the degree of locality in 3D space. A difficulty is that this approach does not naturally distinguish atom types. To get around this limitation, a separate environment fingerprint can be constructed for each possible atom type (e.g., one for carbon and one for hydrogen). Angular symmetry functions use two interatomic distances for a three-body angle, so one symmetry function is needed for each pair of atom types. A typical NNP implementation[2] in organic chemistry (i.e., C, N, O, H, F elements) uses 32 radial symmetry functions for each atom type and 8 angular symmetry functions for each atom type pair, leading to a final dimension of approximately 750 for each atomic environment.

A distinct approach called the "smooth overlap of atomic densities"[89] generalizes the square-exponential basis used in symmetry functions to a position-dependent Gaussian function, as shown in equation 4.6, over values of r_s. The density of neighbors around atom i is defined as:

$$\rho_i(r_s) := \sum_j e^{-\eta(\mathbf{r}_{ij}-r_s)^2} \tag{4.6}$$

This is a continuous function of r_s. Rather than constructing a fixed, finite-dimensional fingerprint from this function, the overlap of two environments can be explicitly calculated by integration, giving a similarity score between the two atomic environments $\tilde{K}_{ij}$. Although dependence on pairwise distances ensures translational invariance, this approach is not rotationally invariant by default. Therefore, it requires additional integration over all possible rotations in 3D space, $\hat{R}$. Practical computation of the integral in equation 4.7 is achieved with a set of radial basis functions and spherical harmonic functions, which allows the rotational integral to be computed in an efficient way[89].

$$\tilde{K}_{ij} = \int d\hat{R} \left| \int dr_s \rho_i(r_s) \rho_j(\hat{R} r_s) \right|^2 \tag{4.7}$$

$$K_{ij} = \frac{\tilde{K}_{ij}}{\sqrt{\tilde{K}_{ii}\tilde{K}_{jj}}} \tag{4.8}$$

A final normalization in equation 4.8 provides a scaled similarity measure that is suitable for direct use as the kernel term between any two environments. This approach allows the kernel methods to be applied without the need to formulate the feature space explicitly via equations 3.8–3.9. Extension to systems with multiple chemical species can be accomplished by treating each pairing of atom types separately. This example further emphasizes that the fundamental ingredient in kernel regression methods is the definition of similarity.

Some proposed approaches exploit properties of the molecular wave function calculated with quantum chemical methods. Such features can be highly informative because the wave function describes the electronic configuration of the system of interest, and historical QSPR relationships have exploited this link for decades.[70,147] In recent examples, atomic partial charges and vibrational

modes,[108] bond orders,[123] or matrix elements from the electronic Hamiltonian[46,85,148] have been used in machine learning models as features. Samples of the raw electron density grid[149] have also been used as input to convolutional ANNs (see Section 5.5.1). These approaches incur the additional cost of solving for the wave function at the Hartree-Fock (HF) or DFT level; however, they provide a rich source of information for statistical learning because they transfer some difficulty of the inference task to the electronic structure calculation. Such approaches are beneficial when the target property is more costly than the HF or DFT evaluation, which may take minutes to hours on its own. Geometry optimizations, for example, consist of many such evaluations, so wave function properties have been used to evaluate and predict outcomes of such calculations as they were being performed.[123] Similarly, HF orbitals have been used as inputs to estimate higher scaling CCSD(T) energies.[85] Another advantage of this type of featurization is that it may lead to more transferable models than those based on more basic geometric or graph-based features. Beyond the examples given, promising results have been obtained for predicting DFT exchange-correlation functional energies,[46,148] direct relationships between electron density and energy for orbital-free DFT,[150] and experimental reaction yields.[108]

4.4 Feature Selection

Many approaches exist to generate features for a given chemical system, and all can be combined to generate a very large number of possible feature sets. Having too many features can be problematic if they do not correlate with the regression target—thatis, they are not relevant to the prediction task or are linear combinations of existing features (i.e., redundant with other features). Incorporating such features increases the computational cost of model prediction without improving prediction accuracy. In practice, inclusion of such features introduces noise into the definition of similarity, degrades model performance, and can cause numerical instability.[60] Feature reduction from a large feature space also improves the ratio of training points to the dimension of the feature space, decreasing training time and complexity for nonlinear models. The relationship to feature interpretability is just as important.[151] Having a large number of interrelated features makes extracting trends from models more difficult. Given a set of possible features, *feature selection* is the broad approach by which the most essential features are selected to produce a predictive model. Although feature selection can improve the accuracy of simple models, it may have little benefit for RF or ANN models[152]. The primary benefit in these cases is increased interpretability.

4.4.1 Univariate linear filtering

The simplest approach to feature selection is to test each feature individually for its correlation with the target property and with other features. In univariate filtering, features that fail specific statistical criteria are discarded. The easiest statistical test to conduct is measurement of the linear correlation between each descriptor x_j and the target output:

$$\rho_j = \frac{\frac{1}{n}\sum_{i=1}^{n}\left(x_j^{(i)} - \bar{x}_j\right)\left(y^{(i)} - \bar{y}\right)}{\sqrt{\mathrm{var}(x_j)\mathrm{var}(y)}} \tag{4.9}$$

where $\bar{x}_j$ and $\bar{y}$ are the average values of feature j and the output across the training set, respectively, and *var* is variance. We evaluate the correlation between different feature dimensions of the input space by using $x_{k\neq j}$ in place of y in equation 4.9. A correlation plot (Figure 4.4), colored by

input-output and input-input correlations, can be helpful to quickly visualize relationships between features. Highly correlated features are likely redundant, whereas features that are uncorrelated with the output may be uninformative and could be discarded. More statistically grounded tests (e.g. t- or F-tests[153]) can be used to judge how significant the linear relationship is between each feature and the outcome. These tests provide a consistent, statistically grounded threshold for judging whether a feature should be included. All features falling below a certain threshold could be eliminated. The choice of the exact test tends not to be important because the results of linear filtering based on distinct tests tend to be similar across data sets[154].

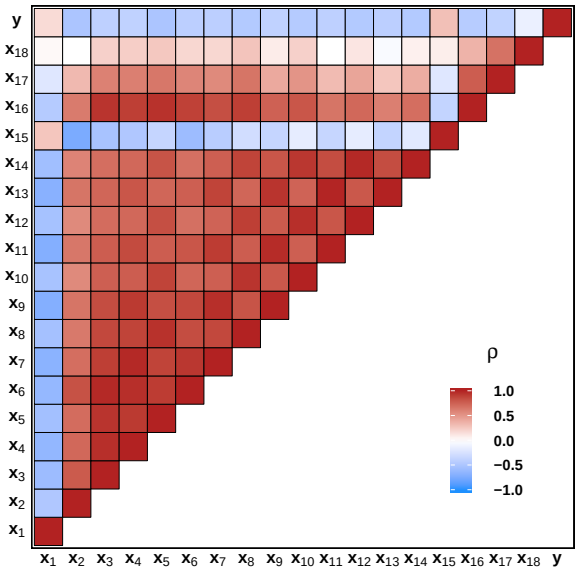

Figure 4.4: Correlation plot showing the linear correlation between 18 graph-based descriptors (x_1–x_{18}) calculated with RDKit[127] and DescriptaStorus[155] for the ESOL dataset[50] from MoleculeNet[49], where the targeted property y is the experimentally-measured aqueous solubility of 1128 organic compounds.

The univariate filtering approach based on linear correlations is amenable to very high-dimensional problems because of the low cost of fitting 1D linear models. Univariate filtering has been shown[154] to perform similarly to more sophisticated feature selection methods when the data set sizes are small. The ease of computation makes this method attractive in bioinformatics and genomics,[156] where the number of possible features is very large. Notably, this type of feature selection is agnostic to the final model used and is a form of data preprocessing. The drawback of this approach is that linear filtering neglects any interactions between features that may occur, and the significance test in a linear model may not relate well to the final machine learning model[151].

Some attempts have been made to extend filters to account for interaction terms. Examples are ReliefF and RReliefF,[157,158] which rank features by how well they can increase pairwise distances between points with different target values (i.e., to separate high and low values of y), while remaining straightforward to calculate. These extended approaches have shown good performance in

protein structure[159] and genomics[47] property prediction.

4.4.2 Iterative subset methods

To consider interactions between multiple features, it is necessary to allow for simultaneous selection of subsets (i.e., groups of multiple features). These subset selection methods are called "wrapper" methods because they iteratively use an internal model to score different subsets of features.[151] This internal model can be any regression model in principle but, ideally, with the same complexity as the final regression model. However, the need to train and evaluate the internal model many times means that a cheaper model such as MLR or KRR may be preferred (e.g., over an artificial neural network).

4.4.3 Best subset selection

The process of selecting a feature set within this framework is referred to as the "best subset selection problem." The process is general is illustrated here for linear models. In MLR, we can "eliminate" a particular feature by setting the corresponding coefficient to zero. This means we can express the linear subset selection problem as a type of regularization on top of normal MLR:

$$w = \arg\min_{w} \left[\mathcal{L}\left(y, Xw\right)\right] \tag{4.10}$$

$$\text{s.t. } \|w\|_0 \leq K,\ K \in \mathbb{N}_0 \tag{4.11}$$

Here, $\|w\|_0 = \sum_i \mathbb{1}_{|w|>0}$ is the ℓ_0 norm of w, the number of nonzero coefficients. Unfortunately, the ℓ_0 norm is not differentiable, meaning that solving equations 4.10–4.10 is difficult. As always, we cannot fairly choose K based on the training error only. For a linear model with the least squares loss, the loss function will not increase as parameters are added. The only way to find the best subset of prediction variables is by combinatorially trying all 2^d possible subsets and estimating out-of-sample error with cross-validation. This exhaustive feature subset comparison is possible for simple, linear models and small (≤ 30) feature sets.[160] However, the approach becomes intractable for more complex models that require hyperparamater tuning.[60] Using KRR with a one-hyperparameter, Gaussian kernel, for example, still requires that this hyperparameter be evaluated for each subset. Two alternative methods can be used to approximate the best subset that circumvents this adverse scaling—stepwise methods and shrinkage operators (Section 4.4.6)—and will be discussed later in this chapter.

4.4.4 Greedy stepwise feature selection

To overcome scaling challenges, we can aim to find a "good," but not necessarily the best, feature set. Theoretically, the best subset of K variables could be quite distinct from the best set of $K+1$ variables, but one can intuitively expect them to be similar. Assuming this expectation holds in general, one might try to find the single best feature and then grow the descriptor space from there. This idea underlies greedy or recursive feature addition,[60,151] which starts with the single most important feature. This first stage of recursive feature addition requires testing d models with cross-validation. Once the first feature has been selected, $(d-1)$ two-variable models are trained next, each with a different choice of the additional variable. This approach is repeated until the optimal K features are obtained. This amount can be detected when the validation error stops decreasing with subsequent variable addition. As opposed to an exhaustive search, this method

requires only $\mathcal{O}(Kd)$ model evaluations, although it is not guaranteed to find the globally optimal subset of features. The linear scaling with d allows this method to be applied to much larger sets of features than can be handled combinatorially. A similar method, known as "recursive feature elimination," starts with a model trained on all d of the available features. Features are then removed by examining improvements in the validation scores among all $(d-1)$-variable models. This process is then repeated, each time removing additional features. Of the two methods, feature addition has lower computational cost and thus is more amenable to more complex internal models because it starts with lower dimensional models.

4.4.5 Random stepwise methods

Feature selection is a discrete optimization problem over all possible subsets. Greedy selection approaches can become stuck in local minima when added features stop improving the out-of-sample error locally and will require many evaluations in high-dimensional spaces. Methods from discrete optimization have been used to add randomness to the process [151,161] to avoid these pitfalls. These methods can have creative names, such as "ant colony optimization," [162,163] which was inspired by how ants behave. A representative method in this class uses genetic algorithms for feature selection, [164,165] although specifics can vary with the implementation. In the overall approach, a population of many regression models is initialized with randomly sampled sets of features, and the worst-performing models are removed in subsequent iterations. The method of scoring the models must depend on out-of-sample evaluation (e.g., by cross-validation). The population is then updated with models trained on combinations of features from the best-performing sets as well as random mutations of features. The process is repeated multiple times, and the feature set of the surviving model can offer better performance than fully randomized searches. Genetic algorithms have been successfully applied in cheminformatics [165,166] or genomics/proteomics [167] problems with a large number of features [168] that would have prevented an exhaustive search. Contemporary use of these methods has declined with increasing computational power available for implementing stepwise methods and the emergence of shrinkage methods, which allow very large feature spaces to be analyzed efficiently.

4.4.6 Shrinkage methods: LASSO and elastic net

A different approach to tackling the subset selection problem in equations 4.10–4.11 is to relax the ℓ_0 norm to the ℓ_1 norm, $\|w\|_1 = \sum_i |w_i|$, which is the best smooth, convex approximation [169] to ℓ_0. This approach is referred to as the least absolute shrinkage and selection operator (LASSO), [23] although the method was used with different terminology previously. [170] LASSO was originally introduced in the context of linear models:

$$\hat{y}_{\text{LASSO}}(X) := Xw \tag{4.12}$$

$$w = \arg\min_w \left[\mathcal{L}\left(y, Xw\right) + \lambda \|w\|_1\right] \tag{4.13}$$

After solving equation 4.13, the uninformative coefficients w_j will be exactly zero, in contrast to using ℓ_2 regression, which makes all coefficients smaller. These zeroed-out coefficients correspond to the uninformative dimensions of the feature space (i.e., the features not useful in our subset). The extent of feature pruning can be controlled by varying λ. Because only some coefficients are non-zero, the resulting models will have fewer features than the original subset. Larger values of λ correspond to a greater degree of regularization. In the context of LASSO, this result corresponds to

fewer nonzero coefficients. The optimal value of λ can be selected through CV. LASSO is primarily used in the context of linear models, for which it was introduced. Extension to nonlinear models is also possible.[171]

The derivative of equation 4.13 with respect to w_i is not defined at $w_i = 0$ because of the absolute value, and we cannot obtain an analytic solution as per equation 3.3. Therefore, the optimal w values must be found using an iterative, numerical procedure. Fortunately, this problem falls into a class of convex optimization problems for which many suitable algorithms exist. Possible approaches include treating this challenge as a constrained quadratic programming problem[23] or using variants of steepest descent optimization algorithms.[172,173] The result is that LASSO achieves both model fitting and feature selection in a single step. The earlier combinatorial challenges observed in wrapper methods are alleviated, meaning that LASSO can be easily applied in a one-shot fashion to a very large set of possible features.[23].

It has been observed that LASSO convergence and stability could be eroded when candidate input features are highly correlated.[171,174] As a result of the correlation of features, the selected subset can change dramatically if there is perturbation or noise in the input or output data. To overcome this potential pitfall, generalize the LASSO approach to include a term from ridge regression. Doing so yields a more general approach, referred to as "elastic net"[174]:

$$\hat{y}_{\text{elastic net}}(X) := Xw \tag{4.14}$$

$$w = \arg\min_{w} \left[\mathcal{L}\left(y, Xw\right) + \lambda \left((1-\alpha) \left\|w\right\|_1 + \alpha \left\|w\right\|_2^2 \right) \right] \tag{4.15}$$

By varying $\alpha \in [0, 1]$, elastic net interpolates between LASSO ($\alpha = 0$) and ridge regression ($\beta = 1$). As with LASSO, features that correlate poorly to the property prediction task will have zero coefficients if $\alpha < 1$. Here, the extent of feature reduction is controlled by either decreasing the value of α or increasing the value of λ. Thus, elastic net has two hyperparameters to be estimated using CV. While no analytic solution exists, the same algorithms that solve LASSO problems can solve elastic net.[174] Adding a small fraction of ℓ_2 character (by $\alpha \neq 0$) solves many of the stability issues observed with standard LASSO[169,175] and ensures a single, well-defined minimum. A proof of the advantages of elastic net using convex analysis is described by Mol et al.[169]

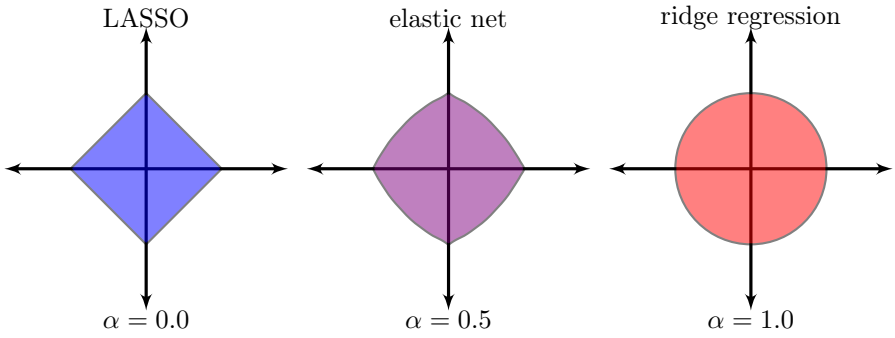

Figure 4.5: Comparison of common regularization penalties, illustrated as the region within one unit of the origin in the plane measured with the ℓ_1 (LASSO), a hybrid elastic net constraint, and ℓ_2 (ridge).

Figure 4.5 shows areas of a two-dimensional parameter space where $\|w\| \leq 1$ for the LASSO,

ridge, and elastic net penalties. These types of regularization penalties are equivalent to selecting coefficients inside a constrained region of parameter space. The size of the constrained region is controlled by the magnitude of λ, where high λ implies a small region. The different shapes of these regions intuitively help to explain why LASSO allows for feature selection. For example, consider two different 2D linear models: a model with $w_1 = w_2 = 0.5$ and a sparse model that has only one dimension, with $w_1 = 1$ and $w_2 = 0$. Under the ℓ_2 norm, the first model has a regression penalty $\|w\|_2^2 = 0.5$, lower than the ℓ_1 norm value of 1.0. This helps explain why ridge regression does not lead to sparse solutions with zero coefficients but favors many small nonzero terms. Using the LASSO penalty, both proposed models have an equal penalty. In theory, taking norms < 1 would lead to more extreme sparsity patterns but would sacrifice convexity, making the optimization problem more difficult.

Both linear LASSO and elastic net have been used in chemistry to select descriptors[102] including applications with more than 10^5 features.[48] Even though LASSO typically produces a linear model, the selected features have been used as inputs into nonlinear models[81]. Elastic net-selected linear models often show better predictive power[174] than MLR without regularization. LASSO or elastic net models are also often used as baselines for comparisons with nonlinear models.[102,105,108,176,177]

4.4.7 Random forests feature selection

Beyond LASSO, some nonlinear regression models are able to perform integrated regression and feature selection. One prominent example is RF models[119,178]. There are two main approaches to using RF models to carry out feature selection. Because each tree in the model is trained on a random subset of all available features, the performance of trees that lack a single feature can be measured (e.g., with the default out-of-bag error metric of the RF model). Averaging these scores for a feature measures its importance, as judged by how much error is introduced when this feature is removed from the feature set. The second, more complex method is called the "variable importance score" or "permutation accuracy importance."[179] This score is constructed[178] by considering the out-of-bag (OOB) errors of the i^{th} tree in the forest trained on the data matrix X be $\varepsilon^{(i)}_{\text{OOB}}(X)$. By randomly permuting the rows of the j^{th} column of X (i.e., randomly shuffling the values of the j^{th} feature), we obtain $X^{[j]}$ and measure the error for the i^{th} tree again without retraining. This produces the $\varepsilon^{(i)}_{\text{OOB}}(X^{[j]})$, and a feature j that has variable importance (VI) determined by the average increase in error over all m trees is the RF under this permutation:

$$\text{VI}(j) = \frac{1}{m}\sum_{i=1}^{m}\left(\varepsilon^{(i)}_{\text{OOB}}(X^{[j]}) - \varepsilon^{(i)}_{\text{OOB}}(X)\right) \tag{4.16}$$

Intuitively, if swapping the values of a feature has little impact, it can be safely ignored. Conversely, if the predictions are severely degraded by the permutation, the feature is important for prediction. Note that the units of VI will be the same as those used to measure the OOB error. For example, the mean squared error is typically used in a regression task, and the VI Will have the units of the predicted quantity squared. An example of VI from a RF model for a chemistry application is shown in Figure 4.6. It is apparent that permuting the variable x_8 degrades model performance about $6\times$ more than x_1. The same data are shown in Figure 4.4. We observe that strong correlations between features (e.g., x_3, x_6, and x_9) leads to "plateaus" of VI with fairly interchangeable values. This observation holds even if the correlation is negative, (e.g., x_2 and x_{15}).

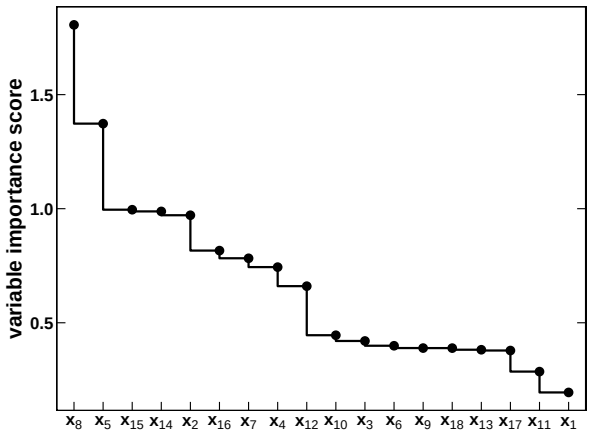

Figure 4.6: Variable importance plot from a 5000-tree random forest model trained using the randomForest package[180] with default thresholds in R 3.6.1[181]. The y-axis shows the relative increase in root mean square error for predictions of solubility for each of 18 graph descriptors (x_1–x_{18}) calculated with RDKit[127] and DescriptaStorus[155] for the ESOL dataset[50] from MoleculeNet[49], ordered from most to least important.

These scores can be used to rank variables from most to least important. They can also provide a sense of how much the out-of-sample error would grow if the number of features were reduced to a target number. This use of RF models is appealing because it provides "free" information about the importance of all variables in the problem at nearly the same cost as training the RF; the models are not retrained on the permuted data. Compared with LASSO methods, RF conducts feature selection in a fundamentally nonlinear manner. The RF feature importance scores have also shown promise for extracting insights from chemical systems[103,109]. Although RF-selected features are optimized for tree models, they can be transferred successfully to other models such as KRR[81]. Alternatively, they can be used as a starting point for more expensive, iterative subset selection methods with other internal regression models by providing ordering for recursive feature addition or elimination methods.[182]

4.5 Dimension Reduction Techniques

Understanding high-dimensional feature spaces can be challenging, especially regarding visualization. A set of methods known as "dimension reduction techniques" provides lower dimensional transformations of high-dimensional data, which are easier to manage and visualize. An important distinction exists between feature selection (Section 4.4) and dimension reduction: in feature selection, the intention is to find the unmodified features that best correlate with the outcome, whereas in dimension reduction, the objective is to find an optimal low-dimensional space that preserves as many of the characteristics of the original feature space as possible. Therefore, the new coordinates found through dimension reduction are not single features but rather transformed variables that can depend on many of the original features. These variables lack any direct interpretability that was present in the underlying features, and the full original feature space vector is required to find the mapping to the lower dimensional space. Similarly, axes from dimension reduction will not

correspond to physical units and should be given arbitrary labels.

The simplest and most computationally efficient method is called principal component analysis (PCA), which dates back over a century [183] and is closely related to singular value decomposition of the data matrix. [72] Essentially, it is a linear transform (i.e., a rotation) of the data onto a new set of orthonormal basis vectors called "principal components." [60] Consider a data matrix $X \in \mathbb{R}^{n \times d}$ of n observations in d dimensions that has been scaled to have zero mean. The PCA is given by the eigenvalue decomposition of the sample covariance matrix, $\frac{1}{n-1} X^T X \in \mathbb{R}^{d \times d}$:

$$\frac{1}{n-1} X^T X = V \Omega V^T = V \begin{bmatrix} \omega_1 & & & \\ & \omega_2 & & \\ & & \ddots & \\ & & & \omega_d \end{bmatrix} V^T \tag{4.17}$$

The matrix $\frac{1}{n-1} X^T X$ is a positive semidefinite matrix and so has eigenvalues ω_i that are all ≥ 0. By convention, we order these eigenvalues from largest to smallest, so $\omega_1 \geq \omega_2 \ldots \geq \omega_d \geq 0$ and call them the singular values or scores. The corresponding eigenvectors, v_i in the matrix V, are an orthonormal basis for the matrix X, and any data point can be written as a projection onto these vectors. Each vector v is a linear combination of the original features obtained by solving the eigenvalue problem; therefore, each principal component vector can depend on every one of the original features. The singular value ω_i represents how much of the variance in the matrix X is encoded by the corresponding principal component vector v_i. This ordering can be be used as a dimension reduction technique by retaining only the vectors with the largest principal component scores. For example, the first two principal components represent more of the variance in the data than any other 2D linear subspace. This makes visualization in the first two principal vectors useful to explore how the data are distributed on a plot.

No 2D plot can capture all information implicit in a higher dimensional space. In addition, the relative importance of the first 2 singular vectors can be determined by comparing ω_1 and ω_2 to the sum of all singular values, $\sum_{i=1}^{d} \omega_i$. If the first few singular values are large relative to the total, the low-dimensional PCA is a good approximation of the full space. If they are small, a large amount of variation is missing from the 2D depiction. This approach can be extended to higher dimensional spaces (e.g., 3D) by including additional principal components.

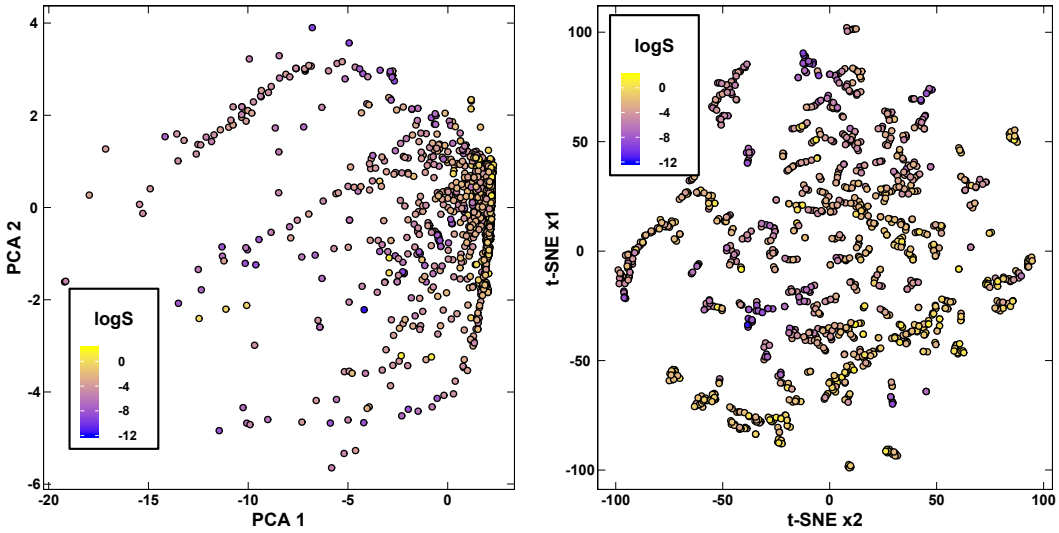

Figure 4.7: Comparison of dimension reduction techniques PCA (left) and t-SNE (right), plotting the distribution points in an 18-dimensional feature space of graph-based descriptors calculated with RDKit[127] and DescriptaStorus[155] for the ESOL dataset[50] from MoleculeNet[49], colored by the output property *logS*, the log of the experimentally-measured aqueous solubility of 1128 organic compounds. All axes units are arbitrary.

PCA is a useful tool that is commonly used to examine chemical feature spaces. It has been used to study different metrics of molecular similarity[184] and to visualize the distribution of output properties in feature space.[4,81,185] We provide an example in Figure 4.7, showing how high- and low-solubility molecules cluster in terms of the first two principal components. It appears that solubility is a stronger function of the first principal component compared with the second. However, PCA is fundamentally linear and does not consider any additional structure in the high-dimenional feature space. Therefore, many nonlinear dimension reduction techniques have been developed that attempt to recreate the relative proximity of points in a higher dimensional feature space. Some popular methods are IsoMap,[186] t-stochastic neighborhood embedding (t-SNE),[187] and uniform manifold approximation (UMAP)[188]. These methods all attempt to create a set of 2D or 3D points that have the same distribution as the points in the high-dimensional space. For example, t-SNE tries to match the pairwise distances between all points[187] in the higher dimensional space to those in the lower dimensional output space, recreating the neighborhoods of points in feature space. These techniques are more opaque than PCA but detect nonlinear structure in feature space that could be missed by a strictly linear method such as PCA. Figure 4.7 compares a t-SNE with PCA, and it appears to provide better clustering in terms of the solubility than that afforded by PCA.

4.6 Worked Example of Some Feature Sets

A number of different methods for computing numerical representations for molecules have been introduced in this chapter. We will compute a few simple alternative feature sets to help clarify these concepts and to provide a comparison between 3D and 2D descriptors as well as the differ-

ences between molecular and atomic approaches. These examples are based on a simple methanol molecule, as shown in Figure 4.8.

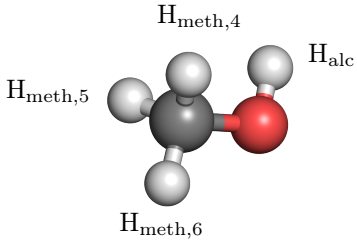

Figure 4.8: Illustration of a methanol structure, showing carbon in gray, hydrogen in red, and hydrogen in white. Labels for the hydrogen atoms used in the text are indicated.

The molecule consists of six atoms: a carbon atom (atom 1), an oxygen atom (atom 2), a hydrogen atom on the alcohol group (H_{alc}; atom 3), and three methyl hydrogen atoms (H_{meth}; atoms 4, 5 and 6). We need to distinguish the three different methyl hydrogens (see Figure 4.8) because they are different distances from the alcohol hydrogen, depending on the conformer. We provide some nominal interatomic distances in Table 4.1. Given the symmetry of our idealized molecule, we provide only one distance between heavy atoms carbon and oxygen and the methyl hydrogens, $r_{CH_{meth}}$ and $r_{OH_{meth}}$, and one distance between any pair of methyl hydrogens, $r_{H_{meth}H_{meth}}$.

$r_{CH_{meth}}$	$r_{CH_{alc}}$	r_{CO}	$r_{OH_{meth}}$	$r_{OH_{alc}}$
1.09	1.94	1.42	2.06	0.97
$r_{H_{meth}H_{meth}}$	$r_{H_{meth,4}H_{alc}}$	$r_{H_{meth,5}H_{alc}}$	$r_{H_{meth,6}H_{alc}}$	
1.78	2.29	2.29	2.84	

Table 4.1: Interatomic distances in Ångströms for all atoms in methanol structure shown in Figure 4.8 .

4.6.1 Coulomb matrix

The first representation considered is the Coulomb matrix,[144] as given in equation 4.3. This representation is a descriptor of the 3D structure of the whole molecule. The Coulomb matrix for this molecule with 6 atoms is 6×6 elements in size and contains terms involving the interatomic

distances and nuclear charges of the atoms:

$$
C = \begin{bmatrix}
\frac{Z_{\mathrm{C}}^{2.8}}{2} & \frac{Z_{\mathrm{C}}Z_{\mathrm{O}}}{r_{\mathrm{CO}}} & \frac{Z_{\mathrm{C}}Z_{\mathrm{H_{alc}}}}{r_{\mathrm{CH_{alc}}}} & \frac{Z_{\mathrm{C}}Z_{\mathrm{H_{meth,4}}}}{r_{\mathrm{CH_{meth,4}}}} & \frac{Z_{\mathrm{C}}Z_{\mathrm{H_{meth,5}}}}{r_{\mathrm{CH_{meth,5}}}} & \frac{Z_{\mathrm{C}}Z_{\mathrm{H_{meth,6}}}}{r_{\mathrm{CH_{meth,6}}}} \\
\frac{Z_{\mathrm{O}}Z_{\mathrm{C}}}{r_{\mathrm{CO}}} & \frac{Z_{\mathrm{O}}^{2.8}}{2} & \frac{Z_{\mathrm{O}}Z_{\mathrm{H_{alc}}}}{r_{\mathrm{OH_{alc}}}} & \frac{Z_{\mathrm{O}}Z_{\mathrm{H_{meth,4}}}}{r_{\mathrm{OH_{meth,4}}}} & \frac{Z_{\mathrm{O}}Z_{\mathrm{H_{meth,5}}}}{r_{\mathrm{OH_{meth,5}}}} & \frac{Z_{\mathrm{O}}Z_{\mathrm{H_{meth,6}}}}{r_{\mathrm{OH_{meth,6}}}} \\
\frac{Z_{\mathrm{H_{alc}}}Z_{\mathrm{C}}}{r_{\mathrm{CH_{alc}}}} & \frac{Z_{\mathrm{H_{alc}}}Z_{\mathrm{O}}}{r_{\mathrm{OH_{alc}}}} & \frac{Z_{\mathrm{H_{alc}}}^{2.8}}{2} & \frac{Z_{\mathrm{H_{alc}}}Z_{\mathrm{H_{meth,4}}}}{r_{\mathrm{H_{alc}H_{meth,4}}}} & \frac{Z_{\mathrm{H_{alc}}}Z_{\mathrm{H_{meth,5}}}}{r_{\mathrm{H_{alc}H_{meth,5}}}} & \frac{Z_{\mathrm{H_{alc}}}Z_{\mathrm{H_{meth,6}}}}{r_{\mathrm{H_{alc}H_{meth,6}}}} \\
\frac{Z_{\mathrm{H_{meth,4}}}Z_{\mathrm{C}}}{r_{\mathrm{CH_{meth,4}}}} & \frac{Z_{\mathrm{H_{meth,4}}}Z_{\mathrm{O}}}{r_{\mathrm{OH_{meth,4}}}} & \frac{Z_{\mathrm{H_{meth,4}}}Z_{\mathrm{H_{alc}}}}{r_{\mathrm{H_{alc}H_{meth,4}}}} & \frac{Z_{\mathrm{H_{meth,4}}}^{2.8}}{2} & \frac{Z_{\mathrm{H_{meth,4}}}Z_{\mathrm{H_{meth,5}}}}{r_{\mathrm{H_{meth,4}H_{meth,5}}}} & \frac{Z_{\mathrm{H_{meth,4}}}Z_{\mathrm{H_{meth,6}}}}{r_{\mathrm{H_{meth,4}H_{meth,6}}}} \\
\frac{Z_{\mathrm{H_{meth,5}}}Z_{\mathrm{C}}}{r_{\mathrm{CH_{meth,5}}}} & \frac{Z_{\mathrm{H_{meth,5}}}Z_{\mathrm{O}}}{r_{\mathrm{OH_{meth,5}}}} & \frac{Z_{\mathrm{H_{meth,5}}}Z_{\mathrm{H_{alc}}}}{r_{\mathrm{H_{alc}H_{meth,5}}}} & \frac{Z_{\mathrm{H_{meth,5}}}Z_{\mathrm{H_{meth,4}}}}{r_{\mathrm{H_{meth,5}H_{meth,4}}}} & \frac{Z_{\mathrm{H_{meth,5}}}^{2.8}}{2} & \frac{Z_{\mathrm{H_{meth,5}}}Z_{\mathrm{H_{meth,6}}}}{r_{\mathrm{H_{meth,5}H_{meth,6}}}} \\
\frac{Z_{\mathrm{H_{meth,6}}}Z_{\mathrm{C}}}{r_{\mathrm{CH_{meth,6}}}} & \frac{Z_{\mathrm{H_{meth,6}}}Z_{\mathrm{O}}}{r_{\mathrm{OH_{meth,6}}}} & \frac{Z_{\mathrm{H_{meth,6}}}Z_{\mathrm{H_{alc}}}}{r_{\mathrm{H_{alc}H_{meth,6}}}} & \frac{Z_{\mathrm{H_{meth,6}}}Z_{\mathrm{H_{meth,4}}}}{r_{\mathrm{H_{meth,6}H_{meth,4}}}} & \frac{Z_{\mathrm{H_{meth,6}}}Z_{\mathrm{H_{meth,5}}}}{r_{\mathrm{H_{meth,6}H_{meth,5}}}} & \frac{Z_{\mathrm{H_{meth,6}}}^{2.8}}{2}
\end{bmatrix}
$$

$$
= \begin{bmatrix}
\frac{6^{2.8}}{2} & \frac{6\times 8}{1.42} & \frac{6\times 1}{1.94} & \frac{6\times 1}{1.09} & \frac{6\times 1}{1.09} & \frac{6\times 1}{1.09} \\
\frac{8\times 6}{1.42} & \frac{8^{2.8}}{2} & \frac{8\times 1}{0.97} & \frac{8\times 1}{2.06} & \frac{8\times 1}{2.06} & \frac{8\times 1}{2.06} \\
\frac{1\times 6}{1.94} & \frac{1\times 8}{0.97} & \frac{1^{2.8}}{2} & \frac{1\times 1}{2.29} & \frac{1\times 1}{2.29} & \frac{1\times 1}{2.84} \\
\frac{1\times 6}{1.09} & \frac{1\times 8}{2.06} & \frac{1\times 1}{2.29} & \frac{1^{2.8}}{2} & \frac{1\times 1}{1.78} & \frac{1\times 1}{1.78} \\
\frac{1\times 6}{1.09} & \frac{1\times 8}{2.06} & \frac{1\times 1}{2.29} & \frac{1\times 1}{1.78} & \frac{1^{2.8}}{2} & \frac{1\times 1}{1.78} \\
\frac{1\times 6}{1.09} & \frac{1\times 8}{2.06} & \frac{1\times 1}{2.84} & \frac{1\times 1}{1.78} & \frac{1\times 1}{1.78} & \frac{1^{2.8}}{2}
\end{bmatrix}
$$

$$
= \begin{bmatrix}
75.47 & 33.80 & 3.10 & 5.50 & 5.50 & 5.50 \\
33.80 & 168.90 & 8.25 & 3.88 & 3.88 & 3.88 \\
3.10 & 8.25 & 0.50 & 0.44 & 0.44 & 0.35 \\
5.50 & 3.88 & 0.44 & 0.50 & 0.56 & 0.56 \\
5.50 & 3.88 & 0.44 & 0.56 & 0.50 & 0.56 \\
5.50 & 3.88 & 0.35 & 0.56 & 0.56 & 0.50
\end{bmatrix}
$$

Note that larger nuclear charge of the oxygen and the carbon heavily weighs the first few elements of the matrix. Note also that the hydrogen/hydrogen contributions (the lower 4×4 submatrix) are relatively small. This matrix represents a combination of structural and chemical information because it directly involves interatomic distances; therefore, it will be able to distinguish between different conformations of the same molecule. However, this matrix is not invariant to our choice of atom ordering. This choice requires either consistent sorting or extracting of an invariant property of the matrix, such as the eigenvalues.[144] In addition, to compare molecules with differing numbers of atoms, such representations must pad all of the matrices to the size of the largest system by adding rows and columns of zeros.

4.6.2 Autocorrelations

Next we will compute a single element of an autocorrelation descriptor for the methanol molecule. This is a 2D whole-molecule descritptor. Unlike the Coulomb matrix, we will use connectivity information only—what each atom is bonded to—instead of the interatomic distances in Table 4.1. Therefore, this descriptor will be insensitive to conformational changes in the input. We will compute the autocorrelation of depth 1, which means we consider all pairs of atoms that are separated by exactly one bond, and use the nuclear charge as the property. A standard autocorrelation[136,137] is a sum over the correlation functions for each atom in the system. For the carbon atom, this can

be computed as follows:

$$\begin{aligned}\mathrm{AC}^{C}_{d=1} &= \sum_{i\in\text{neighbors of C}} Z_{\mathrm{C}}Z_i = 3\times Z_{\mathrm{C}}Z_{\mathrm{H_{meth}}} + Z_{\mathrm{C}}Z_{\mathrm{O}} \\ &= 3\times 6\times 1 + 6\times 8 = 66\end{aligned}$$

For each of the three methyl hydrogen atoms bonded to the carbon center and the one hydrogen in the alcohol group, there is only one neighboring atom:

$$\begin{aligned}\mathrm{AC}^{\mathrm{H_{meth}}}_{d=1} &= \sum_{i\in\text{neighbors of } H_{\mathrm{H_{meth}}}} Z_{\mathrm{H_{meth}}}Z_i \\ &= Z_{\mathrm{H_{meth}}}Z_{\mathrm{C}} \ = 1\times 6 = 6 \\ \mathrm{AC}^{\mathrm{H_{alc}}}_{d=1} &= \sum_{i\in\text{neighbors of } H_{\mathrm{H_{alc}}}} Z_{\mathrm{H_{alc}}}Z_i \\ &= Z_{\mathrm{H_{alc}}}Z_{\mathrm{O}} = 1\times 8 = 8\end{aligned}$$

Since distances are not used, there is no need to treat the methyl hydrogens distinctly. For the oxygen, there are two neighbors:

$$\begin{aligned}\mathrm{AC}^{O}_{d=1} &= \sum_{i\in\text{neighbors of O}} Z_{\mathrm{O}}Z_i = \times Z_{\mathrm{O}}Z_{\mathrm{C}} + \times Z_{\mathrm{O}}Z_{\mathrm{H_{alc}}} \\ &= 8\times 6 + 8\times 1 = 56\end{aligned}$$

The total nuclear charge autocorrelation of depth 1 for methanol is 148. By choosing different properties and numbers of bonds or by modifying the scope of these sums [81], many different autocorrelations can be computed and stacked together to create an information-rich fingerprint.

4.6.3 Symmetry functions

Finally, we will compute a radial symmetry function [1] for this system. Recall that symmetry functions are defined for individual atoms and can be computed separately for each possible pair of atom types (e.g., between carbon and carbon, carbon and hydrogen, etc.). Instead, we will compute a symmetry function for the central carbon atom that sums all possible partner atoms as a demonstration. In practice, [1,189] 40 or more unique symmetry functions are used, corresponding to a range of values of η and r_s. Here, we set $\eta = 1$ and $r_s = 1.0$ for simplicity. Recall equation 4.4, with our selected values of η and r_s applied:

$$G^{r}_{\mathrm{C}} := \sum_{j\neq \mathrm{C}} e^{-1(r_{\mathrm{C}j}-1.0)^2} f_c(r_{\mathrm{C}j}) \tag{4.18}$$

The summation will run over all other atoms. f_c is the thresholding function that dampens the effect of distant atoms. In our case, all atoms are within a small radius of the carbon atom anyway. We will set $f_c = 1$ for all of them, although smoothly decaying functions are used in real examples that prevent atoms seeing further than ~ 4Å. Applying this simplification and writing out the sum yields:

$$G^{r}_{\mathrm{C}} = e^{-1(\mathbf{r}_{\mathrm{CO}}-1.0)^2} + e^{-1\left(\mathbf{r}_{\mathrm{CH_{alc}}}-1.0\right)^2} + 3\times e^{-1\left(\mathbf{r}_{\mathrm{CH_{meth}}}-1.0\right)^2} \tag{4.19}$$

$$\approx 0.84 + 0.41 + 3\times 0.99 = 4.22 \tag{4.20}$$

This would make up one of the elements of a vector fingerprint that would be computed for all atoms. The nearer the atom is to the carbon, the larger the effect. Therefore, the value of G^r gives a measure of how crowded the local environment is around the carbon atom. By selecting different values of η and r_s, the sharpness of the decay of the symmetry function with distance can be controlled. This allows for a lot of information about 3D structure to be encoded. A few observations should be made at this point. First, there is no angular dependence in G^r, which is addressed by supplementing it with angular symmetry functions that depend on these angles explicitly but otherwise can be computed similarly. Second, this expression does not distinguish between different types of atoms—only the distance to the carbon is encoded, and this result motivates the use of different symmetry functions for different atomic pairs.

4.7 Summary

Featurization is the process of translating observations into mathematical form. The choice of featurization must be paired with the application and objectives in mind:

- The ideal feature representation should (1) be invariant to equivalent inputs, (2) place similar properties relatively close in the feature space, (3) be affordable and straightforward to compute, (4) be easy for the scientist to interpret, and (5) be able to be decoded to the original property.
- Feature sets for chemical systems fall into 3 broad categories, from least to most complex: (1) composition and 3D heuristics provide counts that are simple and cheap to compute, with no information about connectivity; (2) 2D structures use graphs and topologies that capture intuitive connectivity information that is easy to compute; (3) 3D structures are symmetry functions and geometric operations with detailed information about distances and angles for disambiguating conformations.
- Feature selection is the broad approach by which the most essential features are selected to produce a predictive model with optimized accuracy and interpretability. Different characteristics are optimized by different techniques.
- Dimension reduction techniques are used with high-dimensional feature spaces to provide lower dimensional transformations that are easier to manage and visualize.

Chapter 5

Neural Networks and Learned Representations

5.1 Overview

Artificial neural networks are a popular type of regression and classification model with a long history. ANNs were first developed in the 1960s,[190–192] although their simpler predecessor, the *perceptron*, was introduced in the 1950s.[193] Development of the original, underlying theories can be traced back to the 1940s.[194–196] ANNs are so named because they were initially conceptualized as a model of how neurons in the brain communicate by collecting input signals from other neurons and propagating signals to other neurons. A detailed history of the development of ANNs is provided by Schmidhuber.[190] Beyond their original inception, the biological analogy provides limited insight. ANNs are best thought of as a flexible class of nonlinear models that can have billions of parameters[197] and still be efficient to train.

These models have been applied to chemistry problems for decades. Interest in ANN models in chemistry and beyond is driven in part by the excellent progress obtained for some benchmarks.[198] AlexNet,[199] an ANN developed in 2012, achieved state-of-the-art performance on the 1000-class ImageNet Large-Scale Visual Recognition Challenge,[200] beating the previous best-known model by 60% and revolutionizing the field of computer vision. AlexNet is special type of ANN known as a convolutional neural network, as explained in Section 5.5.1. ANN models also exceeded the performance of other models for natural language processing with long short-term memory (LSTM) layers,[201,202], a type of recurrent neural network described in Section 5.5.4. These advances were due to both algorithmic improvements (i.e., convolutional and long-term short-term memory (LTSM) layers) and increased computational power through the use of graphics processing units (GPUs) that made it possible for larger networks to be trained[24,190] on larger data sets. These developments in ANN architectures led to increasing interest in the chemistry community.[203]

This chapter reviews the basic structure of ANN models and how they can be trained to produce any target output. Then we review specialized ANNs that have been used in chemistry. Theoretical understanding of neural networks is still an area of active research, and Goodfellow et al.[204] provide additional technical details beyond the scope of this chapter.

5.2 Anatomy of the Multilayer Perceptron

The simplest ANN model is the multilayer perceptron. An multilayer perceptron (MLP) is a series of transformations from one vector space to another. A chemist can interpret this process as a transformation from the structure of a molecule to its property, which mathematically corresponds to a map from a vector in feature space to a scalar property (e.g., energy or toxicity). Although common, this restriction is not necessary because the output could also be a vector (e.g., of forces on an atom) or even new points in the feature space (see Section 5.7). The terminology for neural networks originates from analogies to neurons in the brain; therefore, each of these transforms is called a "neuron" or "node." The basic model of a node is as follows[204]: the node calculates a weighted sum over the input vector, combining elements of the feature space, x_i, with unique weights, w_i. This is shown in Figure 5.1. The output of an idealized neuron[60] is zero unless the total computed sum exceeds a certain threshold, at which point the node "turns on" and propagates a signal downstream.

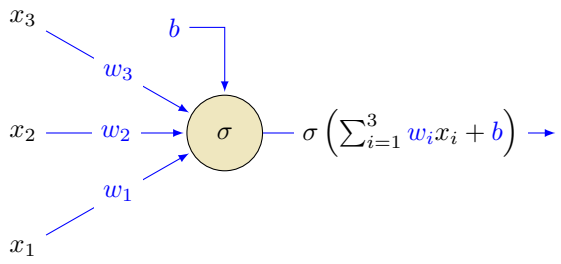

Figure 5.1: Diagram of a single neuron, showing a combination of inputs x_i, corresponding weights w_i, and a possible bias term b. The output of the neuron is given by passing the summation to the activation function, σ. Parameters of the model that can be learned to change the output are indicated in blue.

This nonlinear response is governed by an activation function σ that receives the weighted sum of inputs $w_i x_i$ and a possible bias term b as inputs (Figure 5.1). There are many choices of activation function. Originally, nondifferentiable step functions were used to mimic biology[60], for example:

$$\sigma\left(\sum w_i x_i + b\right) = \begin{cases} 0 & \sum w_i x_i + b \leq 0 \\ 1 & \sum w_i x_i + b > 0 \end{cases}$$

These nonsmooth functions were replaced with smooth approximations such as the sigmoid or hyperbolic tangent functions (Figure 5.2). Recently, these functions have been supplanted[198] by so-called rectified linear units (ReLUs),[205] which have the following form:

$$\sigma\left(\sum w_i x_i + b\right) = \begin{cases} 0 & \sum w_i x_i + b \leq 0 \\ \sum w_i x_i + b & \sum w_i x_i + b > 0 \end{cases} \tag{5.1}$$

This shift to ReLUs has been motivated mainly by the lower computational cost of evaluating the ReLU function and its derivatives. This is true especially in modern "deep learning," where a great many activation functions are used simultaneously,[206]. There is not a universal best choice of activation function. Although it could be selected through cross-validation, ReLU is often recommended by default.[198,204]

Regardless of the form used, the activation function is a single, fixed function that does not change during model training. Importantly, only the activation function can make the neural network become nonlinear. If we chose the linear activation function $\sigma(x) \propto x$, the entire MLP model would collapse to multiple linear regression. However, the incorporation of nonlinearities via the form of the activation function makes MLPs extremely flexible, powerful regression functions. Instead, we control the output of nodes by changing the input weights w_i and bias term b (blue labels in Figure 5.1). As long as the activation function is differentiable, the derivative of the node output with respect to any of the weights can be obtained by:

$$\frac{\partial \sigma\left(\sum w_i x_i + b\right)}{\partial w_i} = x_i \sigma'\left(\sum w_i x_i + b\right) \tag{5.2}$$

$$\implies \frac{\partial \sigma\left(\sum w_i x_i + b\right)}{\partial w} = x \sigma'\left(\sum w_i x_i + b\right) \tag{5.3}$$

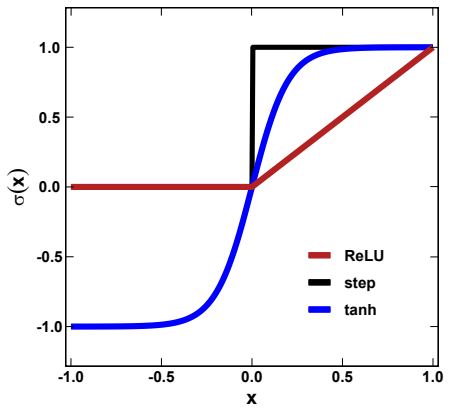

Figure 5.2: Activation functions for neural networks, showing initially-proposed step function (black) and approximations hyperbolic tangent (tanh, blue) and rectified linear (ReLU, red) functions.

Because this derivative is easily computed, determining how to change the weights to increase or decrease the output of the node is straightforward. However, the capacities of a single neuron are fairly limited. Therefore, we group many such nodes together into a layer and then group multiple layers into a network (Figure 5.3). The input molecular representation is represented by input nodes, denoted as I, one per dimension of the feature space. They are passed into multiple layers of nodes, called "hidden layers" because their output is not directly observed, denoted with an H. Networks with more than one hidden layer[60] are called deep neural networks (DNNs). In practice, this term usually implies a large[190] number of layers. For example, DNNs such as ResNet[207] have hundreds of hidden convolutional layers (see Section 5.5).

Each hidden layer in a simple MLP consists of multiple nodes that all receive the same input signals. Each neuron has its own vector of input weights, denoted $w_{i,j}^{[l]}$ for the weight of input i to hidden node $H_j^{[l+1]}$ at layer $l+1$. Regarding notation, we use superscripts with brackets, $^{[l]}$, to refer to the sequence of layers in the network; this approach avoids confusion with indexes of the vector, given by subscripts, and different training examples, indicated by superscript parentheses, $^{(i)}$. An individual bias term is also included for each node, although it is not shown in Figure 5.3 for clarity. The output of each hidden node in layer $(l-1)$ is used as the input for all of the nodes in layer l, which is why the MLP model is also called a "fully connected network." We call the collection of outputs from all the nodes in layer l the "latent state," $z^{[l]}$. The latent state is a vector space of the dimension of the number of nodes in layer l. The neural network as a whole can be understood as mapping from the input, x, through a series of learned latent representations $x \mapsto z^{[1]} \mapsto z^{[2]} \ldots \mapsto y$.

The final output of the network is constructed from a linear combination of the outputs of the final layer. This means that an additional set of weights converts the final latent space to the output ($w_{i,o}^{[2]}$ in Figure 5.3). Because the relationship between the final latent space and the output is linear, inspecting how the latent representation of an input (i.e., molecule or data) varies for different inputs provides insight into how the ANN operates. Latent representations have been exploited to determine a model-specific notion of chemical similarity that can determine how similar a new point is to the training data.[208]. This process allows for interpolation in the model space between molecules[209] and enriches data sets optimally[185] in generative models (see Section 5.7).

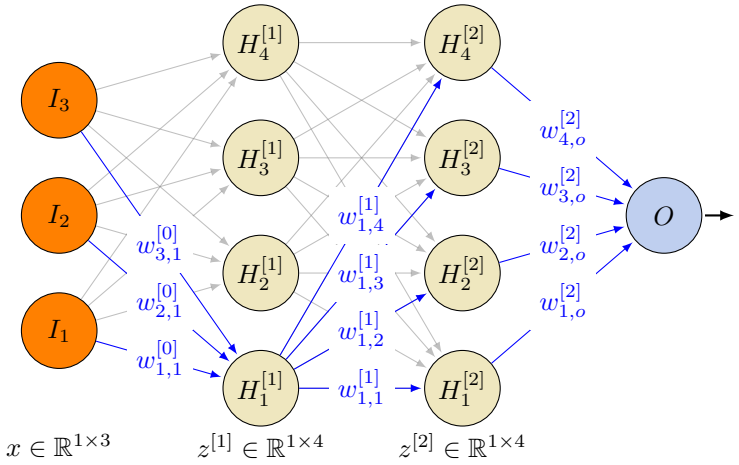

Figure 5.3: Overview of a multilayer perceptron model, showing how three input nodes, I, connect to two layers of four hidden nodes, H, and a scalar output node O. One possible pathway through the network passing through node $H_2^{[1]}$, and the associated learnable weights, w, is highlighted in blue. Other pathways through the other hidden nodes in layer one are shown in gray. The dimensions of the representation at each stage are is shown below the nodes: x for the feature space and $z^{[1]}$ and $z^{[2]}$ for the two latent representations.

The type of ANN just described is called a "feed-forward" neural network because information is sequentially passed from the input to the output layer. In Figure 5.3, one of many paths in the feed-forward network is visualized with blue annotation: all of the inputs contribute the activation of hidden node $H_1^{[1]}$ with the weights $w_{1,1}^{[0]}$ to $w_{3,1}^{[0]}$. The output from this node feeds into all of the hidden nodes in the second layer, $H_1^{[2]}$ to $H_4^{[2]}$, with weights $w_{1,1}^{[1]}$ to $w_{1,4}^{[1]}$, respectively. Each node in the second layer contributes to the output through a final sets of weights. In addition, explicit expressions for the latent representations can be formulated—for example, $z^{[1]}$ can be expressed as:

$$z^{[1]} = \begin{bmatrix} \sigma\left(\sum_{k=1} w_{k,1}^{[0]} x_k\right) \\ \sigma\left(\sum_{k=1} w_{k,2}^{[0]} x_k\right) \\ \sigma\left(\sum_{k=1} w_{k,3}^{[0]} x_k\right) \\ \sigma\left(\sum_{k=1} w_{k,4}^{[0]} x_k\right) \end{bmatrix}^T = \sigma\left(xW^{[0]}\right) \in \mathbb{R}^{1\times 4} \tag{5.4}$$

In equation 5.4, we have used the $W^{[0]}$ notation to represent all of the initial weight vectors as one matrix, and the activation function is applied element-wise to $xW^{[0]}$, which allows compact expression of the model output. Recalling that we orient each observation or latent vector as a row, we can rewrite the output of the MLP shown in Figure 5.3 in terms of the weight matrices

$W^{[l]} \in \mathbb{R}^{N^{[l-1]} \times N^{[l]}}$:

$$y = \sum_{i=1}^{4} w_{i,o}^{[2]} \sigma \left(\sum_{j=1}^{4} w_{j,i}^{[1]} \sigma \left(\sum_{k=1}^{3} w_{k,j}^{[0]} x_k \right) \right) \tag{5.5}$$

$$= z^{[2]} w^{[2]} = \sigma(z^{[1]} W^{[1]}) w^{[2]} = \sigma \left(\left[\sigma \left(x W^{[0]} \right) \right] W^{[1]} \right) w^{[2]} \tag{5.6}$$

By combining a large number of nodes, highly nonlinear models can be constructed. The number of parameters for a fully connected network of L layers, each with N nodes, a single scalar output, and a d-dimensional input, is $\mathcal{O}(N^2 L + dN)$. In principle, the number of layers should be selected based on fit to validation data. In practice, small changes to the number of nodes has limited effect. Typical implementations [1,2,75,102,150,210] of MLP in chemistry use 50–500 nodes per layer and ≤ 3 layers, corresponding to around 10^4–10^5 parameters.

Figure 5.3 has a single scalar output, which makes it also a so-called single-task ANN), and the final layer weights are a vector, $w^{[2]}$. Multiple output nodes to predict multiple properties can also be used in what is called a "multitask" ANN to predict multiple properties, and the final weight layer is set to a matrix with one column per target output. By predicting multiple properties simultaneously (e.g., dipole moments and energies [57,211]), multitask ANNs increase utility and sometimes show improved performance over multiple, single-task models. For example, the DeepTox [212] multitask ANN predicts binding affinity of drug-like molecules to 12 types of receptors.

Because the output of the MLP model is a nonlinear function of the weights in the previous layers, the weights must be optimized numerically instead of being obtained from a direct solution that is possible for other models (e.g., with eq. 3.9). To carry out this optimization, we need to be able to compute the derivative of the output of the model with respect to each parameter. Fortunately, the structure of the connections in the MLP makes calculating derivatives easy and efficient. The derivative of equation 5.6 with respect to the weights in the final layer is clearly $\frac{\partial y}{\partial w_i^{[2]}} = z_i^{[2]}$. It is also possible to compute the derivative with respect to the interior layer weights as follows:

$$\frac{\partial y}{\partial w_{i,j}^{[1]}} = w_i^{[2]} \frac{\partial z_i^{[2]}}{\partial w_{i,j}^{[1]}} \tag{5.7}$$

$$= w_{i,o}^{[2]} \sigma' \left(\sum_{j=1}^{4} w_{j,i}^{[1]} \sigma \left(\sum_{k=1}^{3} w_{k,j}^{[0]} x_k \right) \right) \sigma \left(\sum_{k=1}^{3} w_{k,j}^{[0]} x_k) \right) \tag{5.8}$$

$$= w_{i,o}^{[2]} \sigma' \left(z^{[1]} W^{[1]} \right) \sigma \left(x W^{[0]} \right) \tag{5.9}$$

$$\frac{\partial y}{\partial w_{k,j}^{[0]}} = \sum_i w_i^{[2]} \frac{\partial z_i^{[2]}}{\partial w_{k,j}^{[0]}} = \sum_i w_i^{[2]} \frac{\partial z_i^{[2]}}{\partial z_j^{[1]}} \frac{\partial z_j^{[1]}}{\partial w_{k,j}^{[0]}} \tag{5.10}$$

$$= \sum_i^4 w_{i,o}^{[2]} \sigma' \left(\sum_{j=1}^{4} w_{j,i}^{[1]} \sigma \left(\sum_{k=1}^{3} w_{k,j}^{[0]} x_k \right) \right) w_{j,i}^{[1]} \sigma' \left(\sum_{k=1}^{3} w_{k,j}^{[0]} x_k \right) x_k \tag{5.11}$$

$$= \sum_i^4 w_{i,o}^{[2]} \sigma' \left(z^{[1]} W^{[1]} \right) w_{j,i}^{[1]} \sigma' \left(x W^{[0]} \right) x_k \tag{5.12}$$

These expressions can be extended to an arbitrary number of layers and nodes. The expressions

are simplest for the final layer and increase in complexity as the derivatives propagate back through the network. Therefore, the gradient calculation should start at the final layer and move progressively back through the network, reusing already calculated terms. However, derivatives for the weights at layer l depend on the hidden state variables at early layers $z^{[l-1]} \dots z^{[1]}$. Consequently, during training, the states and outputs of the network are calculated in a forward pass by inputting the features of the training data to enable the loss function and all latent states to be calculated. Then the gradients of the loss function with respect to the weights are calculated in a backward pass from the final layer to the input and used to update the weights. This process is known as "back propagation." It is critical to the efficiency of modern neural network training [204] and is an original driver of interest in neural network models.

It is possible to prove that any continuous function on a compact domain can be approximated arbitrarily well using even a single-layer MLP with a sufficient number of nodes (for a full proof, see Hornik [213] and Csaji [214]). Although full proofs are beyond the scope of this work, we will review how this nonlinearity arises by analyzing sigmoid activation functions in one dimension with observations that also hold for other activation functions. Consider the difference of two "steep" sigmoid functions given by an MLP with one layer, two nodes, and output weights $w_1^{[1]} = 1$ and $w_2^{[1]} = -1$ (Figure 5.4). The sigmoid function is given by the following equation for a scalar input x that has one weight, w, and one bias b:

$$\sigma(x) = \frac{1}{1 + \exp(-wx + b)} \tag{5.13}$$

If we make the sigmoids steep by setting $w = 20$ for both but separate them by choosing offsets $b_1 = -7$, $b_2 = 7$, the output of the network approximates a square pulse (dashed curve in Figure 5.4). We can sharpen this pulse by tweaking the weights and shift the pulse location by changing the bias terms. By considering a sufficient number of square pulses, each made up of a pair of nodes, we can approximate any continuous function $y(x)$ fairly well.

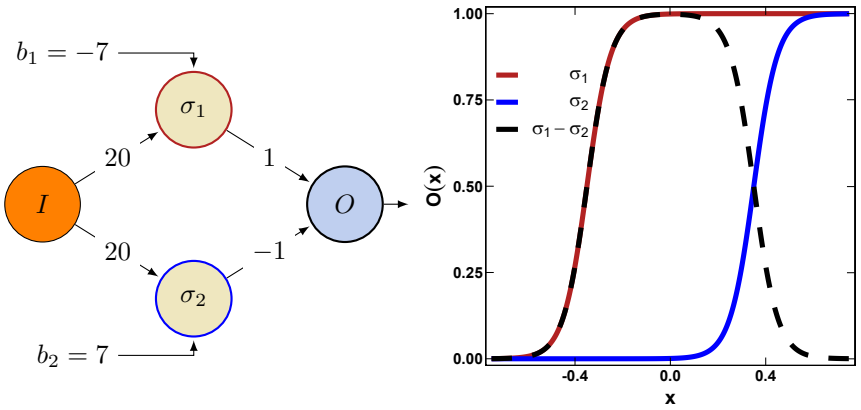

Figure 5.4: Example of 1D universal approximator construction: an ANN with one dimensional input I, two hidden sigmoid units σ_1 and σ_2 and scalar output O, with weights indicated on the connections and bias terms b_1 and b_2 explicitly shown (left) along with output of individual hidden nodes, shown in red and blue, and the overall output shown with black dashed line for inputs $x \in [-0.75.0.75]$ (right).

5.3 Optimization and Training

As suggested in the previous section, access to computationally affordable derivatives of the model $f(X, w)$ with respect to the weights enables us to iteratively update the weights to produce the desired input-output relationship between X and $f(X, w)$. Success is measured using a loss function, for example, the least squares loss in equation 2.2. Any numerical optimization procedure could be applied to update the model parameters to reduce this loss function. In practice, the most common[198,204] is *stochastic minibatch gradient descent*. The basic idea of gradient descent is to update the weights of the network iteratively by taking a step of length α, commonly known as the "learning rate," in the negative direction of the loss function:

$$w^{[t+1]} = w^{[t]} - \alpha \sum_{i=1}^{n} \nabla \mathcal{L}_w(x_i, w^{[t]}) \tag{5.14}$$

This algorithm simply follows the gradient of the loss function in a descent direction as if it were linear with fixed gradient, $\nabla \mathcal{L}_w(X, w_i)$, calculated at the current weight and summed over all training points. This approach ensures that the loss function will have a lower value for a sufficiently small step. The dependence on only $\nabla \mathcal{L}$, instead of second or higher derivatives, is important because higher derivatives are not as readily computed.

The model is nonlinear, so this linear approximation to the derivative may not be very good, and the step size may have to be fairly small. More significantly, because the optimization problem is not convex,[204,215] this type of approach can find only a *local minimum*. There is no guarantee that starting in a different location would lead to the same set of model parameters. In fact, because the number of parameters in the model is often extremely large and the model is highly nonlinear, the optimization landscape is generally challenging, and local minima and saddle points

occur frequently. The number of local minima can be expected to grow exponentially with the dimension,[216] and the number of saddle points may be even larger.[217] The simple iterative scheme given in equation 5.14 could easily converge to a local minimum that produces a model with worse accuracy than could be achieved at the global minimum.

Many strategies have been developed to address this issue. The most widely used approach is to add noise to the optimization. Intuitively, if the optimization trajectory is noisy, it should be easier to escape shallow local minima and become trapped in a deeper, more stable minimum that may produce a model with better generalization performance. This noise is introduced in practice by evaluating the gradient with respect to a randomly sampled subset of the data in each step[215,218]:

$$w^{[t+1]} = w^{[t]} - \alpha \sum_{i \in \text{batch}} \nabla \mathcal{L}_w(x_i, w^{[t]}) \tag{5.15}$$

The size of the subset that is used is called the "batch size." It can range from a single observation, called stochastic gradient descent (SGD), to the full data set, which is the same as ordinary gradient descent. Generally, the larger the batch size, the smoother the optimization and the faster it will converge, but the more susceptible the procedure is to getting stuck in unproductive minima. Another advantage of using training batches is that the code used for training needs to load only one batch of examples at a time into system memory. Loading all examples into memory can become a bottleneck in model training when using very large data sets or large-footprint inputs (e.g., high-resolution images[199]). In the course of training an ANN using equation 5.15, the full set of training data is randomly partitioned into batches of the selected size, and the batches are used sequentially to update the weights. A full pass through all the batches is called one "epoch." Between epochs, the batches are reshuffled. Training should continue until convergence, either because the weight updates are sufficiently small, or the change in loss function is within some tolerance, which may take hundreds of epochs. Exceptions to this process, when "early stopping" is used, are discussed in Section 5.4.

The selection of the learning rate, α, can have a significant effect on the stability of the training process and the quality of the final model.[206] If the step size is too large, the network optimization may overstep entire regions of the parameter space and even move uphill. Very small learning rates will slow training and may lead to a failure to converge if the balance between the noise introduced by the minibatch procedure is too large relative to the step size. The learning rate is often decreased during training time,[218] a process called "learning rate decay." This helps the model optimization process travel quickly to productive regions in the initial steps and then fine-tune the estimates with smaller step sizes later in the training process.

Many additional "tricks" are used to accelerate and improve the convergence of neural network training. These include the addition of momentum terms by blending the current gradient with the previous iterations in close analogy to the widely used conjugate-gradient approach and effectively "ball-rolling" down the error surface. In addition, there is ongoing interest in the use of approximate, second-order (i.e., quasi-Newton) methods, which construct an estimate of the second derivative and use it to accelerate optimization.[218]

Equation 5.15 is the underlying method. Modern optimizers for ANN training, such as Adam[219] (a portmanteau of "adaptive moments") and AdaDelta,[220] make use of these and other enhancements to offer better performance compared with raw SGD. Adam is a good out-of-the-box choice. It uses multiple learning rates that are algorithmically reduced during optimization based on estimates of the noise in the gradient. This makes Adam fairly robust for hyperparameter selection and requires less tweaking than other optimizers such as naive SGD, which is highly sensitive to

how the learning rate is set.

Another concern is that the average weights should be initialized in a way that leads to stable model optimization. Starting with large, nonzero weights makes model optimization highly unstable, but starting with zero weights would mean that the initial gradient is zero.[218] Weights should be initialized to small nonzero values from a narrow uniform distribution around zero.[215] Comprehensive advice on training neural networks is given by Montavon et al.[218] and Goodfellow et al.[204]. In addition, some more concrete recommendations are provided in chapter 6.

5.4 Regularization and Hyperparameter Selection

Because neural networks are highly nonlinear, they are at significant risk of overfitting training data in the same way as any regression model. Therefore, some familiar ideas, such as Tikhonov regularization in equation 2.9, are applied to control overfitting.[218] However, the unique architecture of neural networks offers some additional approaches for reducing model complexity. Two of the most important are *dropout regularization* and *early stopping*.

Dropout regularization[221] is the process of removing or "dropping out" some nodes in the neural network during the training process. The user provides a probability or *rate* of dropout, p, that is a hyperparameter through which the degree of regularization can be adjusted. Then a Bernoulli random variable[1] is drawn for each node in the network at each step of the optimization process. All nodes that are dropped in a given step do not contribute to the prediction or the gradient. Effectively, each step of the training process thus uses a slightly different model, adding noise to the optimization process. This noise is intended to prevent the neural network becoming too dependent on a particularly pathway or set of nodes and can be interpreted as an average across models. Dropout can be applied to hidden and/or input nodes. It is usually applied during model training but not when making final predictions. Because the model has fewer nodes during training than during its final application, the weights should be rescaled by $\frac{1}{1-p}$ to compensate. This adjustment is typically handled automatically by software packages used for training ANNs.

Early stopping is another regularization technique that is widely used for its simplicity and ease of use.[218,222] Because the initial weights are usually set near zero at the beginning of training, weights will typically increase in magnitude during training. This increase in magnitude corresponds to the degree of ANN nonlinearity increasing during optimization (i.e., the learning process). Early stopping involves tracking the error on an OOS validation set during gradient descent and stopping the training process "early" when the error on this validation set starts to increase. This validation set is typically 5%-10% of the training data, and the increase in error is a diagnostic of overfitting to the training data. The optimization can be noisy because of the stochastic minibatch selection method or from passing through local minima. Therefore, optimization continues for a user-specified number of steps after the error starts increasing, in case it decreases again later. If the error remains flat or increases, the optimization is terminated, and the weights can be rolled back to the values that gave the lowest validation error. "Patience" is the hyperparameter for the number of iterations (i.e., epochs) to wait after no improvement is observed in the validation set.

ANNs depend on a large number of hyperparameters. The practitioner's most important choices are the architecture of the network itself (i.e., how many nodes and how they should be connected), the learning rate, the batch size, and the degree of regularization. The degree of regularization is controlled by the strength of the regularization terms, the dropout probability, and the patience for

[1] This variable is basically a coin flip weighted by the dropout rate.

early stopping. All of these factors create an extremely high-dimensional space of possible models. Choosing the optimal model is much more difficult and time consuming than training a network in a one-shot manner. For this reason, exhaustive grid searches of this landscape based on CV, as can be done for kernel methods, are not practical. Instead, algorithms have been developed to explore this hyperparameter space efficiently by narrowing the search for space for good hyperparameter choices in conjunction with a validation set to monitor OOS error. For example, HyperOpt [35,62] is a widely used software package that uses Bayesian optimization to accelerate ANN model optimization. Nevertheless, hyperparameter selection for ANNs remains a challenge and an art. An important diagnostic that can assist in deciding how to tweak hyperparameters is the *learning curve*—that is, how the loss function decreases with the number of training iterations (i.e., epochs). If the loss function oscillates significantly without stabilizing, it could be beneficial to reduce the noise added to the gradient by using a lower dropout rate, a larger batch size, or a slower learning rate. Conversely, if the loss function reaches an asymptotic limit quickly, decreasing the regularization strength, increasing the size of the model (i.e., more layers and nodes), or increasing the learning rate could be helpful.

5.5 Other Types of Layers

The fully connected layers described in Section 5.2 are only one type of neural network model. Alternative layers leading to distinct architectures [204] have been proposed and used, including in the chemical sciences. Each layer type has the same inputs, weights, nonlinear activation functions, and optimization challenges as the MLP. For example, an image classification ANN (e.g., to distinguish a car from a dog) may start with a number of layers that help process the image in a format more readily interpreted by the model (commonly referred to as a "convolutional model"). These layers are then followed by fully connected, MLP-like layers to classify the preprocessed vector. [199]. Next, we briefly outline these different layers and how they are most commonly applied in the chemical sciences.

5.5.1 Convolutional neural networks

Convolutional neural network (CNN) layers are a type of ANN layer that encodes spatial invariance and reduces the number of free parameters in the model. They are most commonly applied in image recognition tasks. CNNs were originally proposed in analogy to the visual cortex in the 1980s, in a model known as the "neocognitron," [25] and gained prominence in image-recognition tasks. [223,224]. CNNs have spurred some of the increased interest in deep learning by powering record-setting image-classification models such as AlexNet [199] and ResNet. [207] The CNN uses *convolutional filters* to extract predictive information from a raw input. A convolutional filter is composed of a small receptive field that is repeatedly applied to different parts of the input, a process called "convolution." In an image context, a block of the image at a time is fed into a small MLP until the process has covered the entire image. The parameters of the filter are fixed throughout the translation across the image (i.e., the convolution process). Therefore, each filter in a CNN applies the same operation to each subregion. In the context of image classification, this filter, once trained, can be interpreted as a means of detecting a particular input signature (e.g., a cat's ear) wherever it is located in the image. Multiple filters can be defined to detect the presence of different features. Figure 5.5 shows the application of a single convolutional filter to a 2D input. The receptive field is translated across the input image, producing a scalar output each time. Therefore, the output of

each single feature is an array of reduced dimension compared with the input.

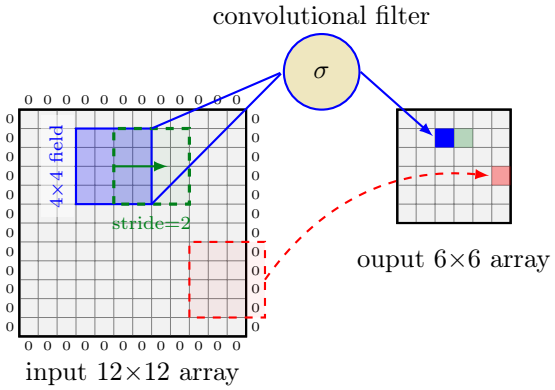

Figure 5.5: Illustration of a single 2D convolutional filter applied to an input array of dimension 12×12 units with a zero-padding of one 1. The filter has a receptive field of 4×4 units and stride of 2 units, illustrated being applied to the blue sub-region of the input array. The unit applies its activation function, σ, to this 4×4 sub-region and stores the scalar result in the corresponding unit of the output array, illustrated with a blue square. Next, the receptive field will be translated to the right by the stride, and this is shown in with a dashed green region, whose output will occupy the green square in the output array. An example of what happens when the receptive field overhangs the edge of the array and utilizes the zero padding is shown by the red dashed region, with the corresponding output region shown with a red square.

Some considerations when using CNNs include the size of the filter (i.e., how many pixels or elements should be considered) and the "stride," which refers to the number of units by which the filter is translated. If the stride is smaller than the filter size, the filter inputs will be partially overlapping. In addition, the user must specify how to handle the edges of the image where the filter would overlap with the end of the input array. The typical approach is to pad with zeros,[204] as shown in the red region in Figure 5.5. These choices are all effectively hyperparameters that govern the length scales of the features that can be distinguished; a filter cannot detect a feature that is larger than its filter size.

Relative to a fully connected layer, which will have the same number of nodes and thus weights as the input vector has dimensions, a convolutional filter has far fewer parameters. In the convolutional layer, only the set of small filters needs to parameterized to detect the same input signature no matter where it appears in the input field. Convolutional layers can be stacked together, and they are often interspersed with *pooling* layers that combine inputs into a single intermediate state. This pooling can be thought of as the equivalent of the latent states in MLPs. The purpose of stacking convolutional layers is to build more complex transformations hierarchically. In image classification, for example, the first layers could be used to identify basic features such as lines or corners. Subsequent filters could be used to identify shapes from combinations of these features, leading to detection of complex objects or textures.[204]

Because CNNs explicitly encode translational invariance, they are most appropriate for processing

2D images or 3D coordinates, where the absolute location of an object or feature is not important to the model prediction. In three dimensions, voxels or point-cloud data[225] are used in place of pixels, and the filters are cubes instead of rectangles. Three-dimensional CNNs have been used in processing medical imaging data.[226] These 3D models have also been applied to combinations of two spatial dimensions with time in an approach known as "spatiotemporal convolutions."[227] Convolutions in one dimension are also sometimes useful. For example, if the dimension is time, they can be used to detect a feature in the same way at all times. They have also been applied in analyzing medical electrocardiography data.[228]

In chemistry, convolutional networks have been applied to input images or 3D geometries. Some examples include protein docking,[229] energies of atoms in planar materials,[230] and the relationship between electron density and energies in orbital-free DFT[150] or exchange correlation energies.[149] Other examples include making predictions of molecular properties from images of 2D chemical skeleton structures[231] or, alternatively, converting images of 2D skeleton structures into machine-readable SMILES strings.[232] These approaches differ from features based on 3D coordinates that use symmetry functions, which have an atom- or molecule-centric view. The input to CNNs is a full 2D or 3D domain, reducing the effort required by the researcher to prepare a representation of the molecule. At the same time, this approach discards some information about the structure of chemical bonding. Next we consider a type of CNN that preserves some of this structure.

5.5.2 Graph convolutions, message-passing, and continuous filter convolutions

A subfamily of CNNs that is particularly relevant to chemistry is graph convolutional neural networks (GCNNs).[138] This type of partially connected network is similar to standard CNNs except that it uses the connections in the molecular graph (i.e., chemical bonds) to define proximity rather than 2D or 3D space. In an image convolution, nearby pixels are assessed by the same filter; in a GCNN, atoms that are connected or near each other in the molecular graph are treated together. GCNNs assign a fingerprint vector to each atom.[138,233] This fingerprint is updated by comparing it with all the atoms bonded to it, and the process is repeated multiple times. It is a convolution because the same operation is applied to all atoms in the molecule at each stage. This framework of iterative updates based on graph connectivity can be more broadly classified as a message passing neural networks (MPNNs)[233], and many varieties are possible. The "message" is the information passed to each atom from its neighbors, a function of the neighbors' own fingerprints. Naively, a carbon atom might receive messages from three hydrogen atoms and one other carbon atom, updating its own fingerprint to identify it as a $C{-}CH_3$ motif. The basic procedure of a graph convolutional network can be broken down as follows:

1. Initialize all atoms with a fixed size vector, $h_i^{[0]}$, with one unique vector per atom type. This vector can be a series of existing atomic properties (e.g., atom type and formal charge[234]) or it could be randomly initialized for each atom type.[42] A representative vector of each bond as $e_{i,j}$ is also needed. This can be the bond order or a quantity that depends on the pairwise atomic distance.[233] The bond representation can also include binary variables to indicate whether the bond is aromatic.[234]

2. For each atom, h_i, at each step, t:

   (a) Compute a message, $m_i^{[t]}$, as a function of the fingerprints, $h_j^{[t]}$, of all atoms bonded to

atom i, and the bond features, $e_{i,j}$, using a regression function or MLP model, M:

$$m_i^{[t+1]} = \sum_j M\left(h_i^{[t]}, h_j^{[t]}, e_{i,j}\right) \tag{5.16}$$

The sum in equation 5.16 is taken over all neighbors of atom i. This is called the "message-passing phase."[233] Simple functions are often used for M including linear functions,[6,176,234] concatenation,[138] or taking the maximum value.[235]

(b) Update the atom's fingerprint based on the messages it receives, using another learned function U. This update function could be another MLP model[6,9,176]:

$$h_i^{[t+1]} = U\left(h_j^{[t]}, m_i^{[t]}\right) \tag{5.17}$$

The update function is sometimes[6,236] configured to be an explicit update to the existing atomic feature map, for example $h_i^{[t+1]} = h_i^{[t]} + U\left(h_j^{[t]}, m_i^{[t]}\right)$, which is essentially a specialized type of ANN layer discussed in Section 5.5.3).

3. Repeat step 2 for several iterations (e.g., ≤ 5).

It is worth noting that information only propagates one bond away per iteration. For a molecule with a longest bond path of N atoms, $N/2$ iterations of the convolutional operation are required for the most distant atoms to see each other and share information. Some alternative formulations known as weave[237] or wave[238] networks attempt to address this shortcoming by passing information more non-locally. Many extensions, for example updating bond properties and atom properties at each step[236,237] or treating molecules as directed graphs in which each bond has a start and end atom[176,239] have been attempted. Crystalline systems can be handled by considering periodic versions of the molecular graph[6]. The form of both the message and the self-update function can also be tuned as desired. Geometric information can be included through edge features or incorporating additional bond types for non-covalent interactions[240]. Another related class of models uses 3D distances in place of graph bonding information, and so these models pass information between regions of 3D space centered on atoms instead but have similar update functions based on the state of their neighbors. Some examples of this approach are *continuous filter convolutions* in SchNet[42,241], the Hierarchically Interacting Particle Neural Network (HIP-NN)[242] or the atoms-in-molecule network (AIMNet)[211]. These approaches augment atom-based symmetry functions with a learned atomic representation.

Applying these specialized convolutional filters on the molecular graph produces a set of atomic features for either direct use in property prediction, such as atomic energies[241,242], or to compute a representative 'molecule fingerprint' from all of the atoms in the graph in conjunction with another neural network[233]. The molecular fingerprint can be used as an input to a final MLP to give some of the most accurate predictions of molecular properties[176,211,234] to date, or to predict reaction sites and outcomes[9]. Overall, MPNNs enable accurate machine learning in data-rich chemical applications.

The procedure to train a GCNN involves many unknown parameters including the weights to construct the message, the weighted combination of the message and current fingerprint, and the mapping from atoms to a molecule vector. These parameters are selected during training at the same time as the final predictive model by minimizing the loss function. This approach has some

conceptual similarities with the construction of force fields in which the same element can have distinct atom types that depend on the atom's local bonding environment. However, in this case, these distinct atom types are not pre-defined, and the representation and model are learned simultaneously. Analysis of the resulting atomic fingerprints can provide insight into both the model and the chemistry being modeled. For example, learned atomic fingerprints showed similar clustering as the corresponding elements in the periodic table[211] or by crystal structure and composition[243]. This analysis provides some support for interpretable and reasonable processing of chemical structures.

5.5.3 Residual and skip layers

One issue with DNNs is the so-called vanishing gradient problem[204,218], wherein the derivatives of DNNs become very small and the learning process stalls. This problem occurs most frequently in conjunction saturating activation functions (e.g., tanh) that level out (Figure 5.2) and can be avoided somewhat through the use of ReLUs. However, this issue can still occur, especially in the later layers of deep networks with many layers that are only making incremental refinements to the latent representation of the data. This motivates the introduction of residual layers[207], which mix the unmodified input and output before passing the combination as input to the next layer (Figure 5.6). The resulting model is trained to learn a mapping that is:

$$z^{[l+1]} = z^{[l]} + \sigma\left(z^{[l]}W^{[l]}\right)$$

To add these two vectors, the dimensions of $z^{[l]}$ and $z^{[l+1]}$ must be the same. Note that if all of the weights in a standard MLP are set to zero, the output will also be zero. However, for a residual layer with zero weights, the output is the unmodified input. In this way, a residual layer that is initialized close to zero will resemble the identify operator, which is clear from Figure 5.6. If the input representation already provides a 'good' mapping to the target property, a residual layer could be able to learn small improvements to the initial representation more easily than a standard layer. *Skip connections* are a related approach that simply concatenates the input and output of the hidden layers, meaning both are fed as inputs into the next layers (Figure 5.6).

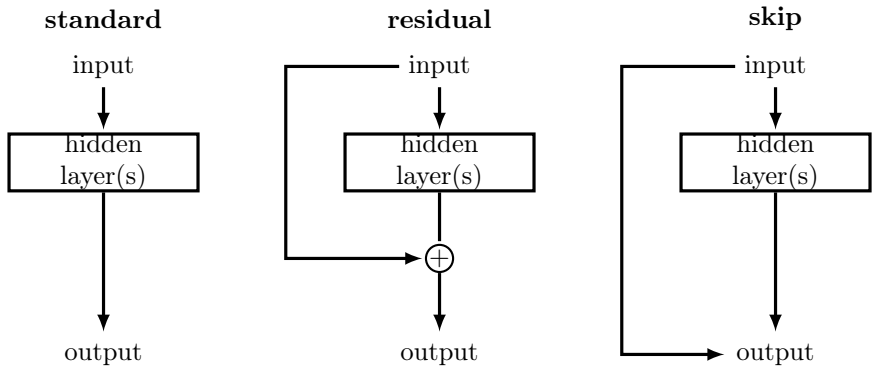

Figure 5.6: Comparison between standard (left), residual (center), and skip (right) layers, where $\oplus$ represents addition. Note that the hidden layer block may consist of multiple hidden layers or other operations. In the case of bypass or skip connections, both the unmodified input signal and the output signal are passed to the next layer.

Residual and pass-through layers are commonly used in chemistry, especially NNPs e.g. PhysNet[57] and SchNet[42] but also in whole-molecule, atomization energy prediction based on graph representation[208]. These layer types are also useful when combining information from non-proximal atoms in graph-convolution type models[233,236] (see Section 5.5.2). This combination has been suggested to be useful in aligning the model structure to better weight the relative importance of proximal and distant atom-atom interactions[242].

5.5.4 Recurrent layers and attention mechanisms

Many data sources have additional structure that limits the independence of individual observations. One example is *sequential data*, where inputs have a natural order, as occurs in the natural language processing of sentence structures. In this case, the meaning of words depends strongly on their order in sentences. Here, we distinguish between having one input vector $x \in \mathbb{R}^{1\times d}$ for each observation and having a sequence of input vectors for each observation $x^{[i]} \in \mathbb{R}^{1\times d}$, $i = 1, 2 \ldots$. As with MLP layers, we will use square brackets to indicate sequence elements. If all the sequences are the same length, it is possible to treat the sequences as simple vectors and 'stack' them to the input of a MLP. However, this approach may not be desirable[204] because 1) the number of parameters grows with sequence length, 2) handling sequences of variable length is not straightforward, and 3) any notion of order in the sequence is not preserved. Recurrent neural networks (RNNs)[244] are a way to encode this sequential nature into ANNs in a manner that is robust to varying sequence length and more parameter efficient than MLPs.

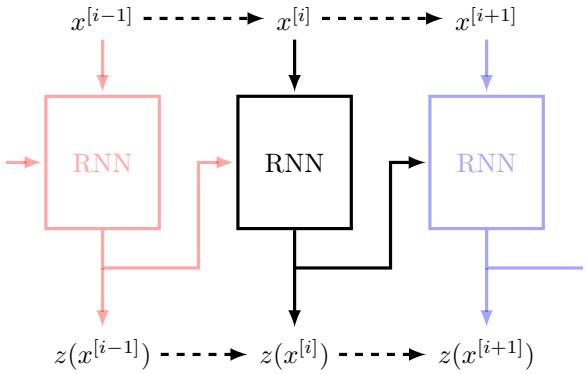

Figure 5.7: Schematic of RNN operation: inputs arrive as a sequence $x^{[1]}, x^{[2]} \ldots$. The RNN block is applied to each element i sequentially (shown in solid black), and receives input from both $x^{[i]}$ and from the output of the same RNN applied to the previous input ($z\left(x^{[i-1]}\right)$, shown as desaturated red). The block produces output, $z\left(x^{[i]}\right)$, which is passed to the downstream evaluations (desaturated blue).

The basic unit of an RNN, i.e., the equivalent of the neuron in a MLP, receives two, distinct input channels, as shown for the central black block in Figure 5.7. The first is from the i^{th} element of the input sequence, and the second is the hidden state from the same RNN unit applied to the $(i-1)^{\text{th}}$ input, which can have distinct weights, w_{memory} and w_{input}:

$$z^{[i]} = \sigma\left(x^{[i]} w_{\text{input}} + z^{[i-1]} w_{\text{memory}}\right) \tag{5.18}$$

Recall here that each $x^{[i]}$ in the sequence is still a vector, as is each $z^{[i]}$. The output state of the RNN block, $z^{[i]} = f\left(x^{[i]}, z^{[i-1]}\right)$, is then used in the calculation of the next input in the sequence. In this way, each time the RNN is applied, it receives a 'message' that contains information from the previous element in the sequence. This is similar to the message encountered earlier in MPNNs, but instead of coming from the neighboring atoms, this message comes from the same node, applied to the previous element in the sequence. The same weights (i.e. the coefficients of the RNN) are used each time the RNN is applied, although a model will often use many independent RNN channels [204], analogous to using multiple neurons in an MLP or filters in a CNN. Downstream functions can then be used to process either the final latent vector, which encodes some memory of the entire sequence in order, or can act on the full sequence of latent vectors. A natural choice is to map each latent vector to a character, which allowing the generated sequence to be interpreted as a word or sentence.

The equation above for a true 'recurrent unit' (eq. 5.18) represents only the simplest possible type. Notably, the direct memory only looks back one unit, which is often not long enough to understand context, e.g., one adjacent word in a sentence. In order to circumvent this problem, many other types of recurrent layers have been proposed. Examples include gated recurrent units (GRUs) [245] or long-term short-term memory units [246,247], which are similar in spirit to simple RNNs but have additional 'forget gates' that regulate the influence of past states. These may provide better ability

to understand long-term dependency in sequences, and LTSM models demonstrated state-of-the-art performance on natural language tasks[190,201,202], although they have been generally overtaken in this application by the attention-based models[248] described next.

Since RNN layers are useful for interpreting sequences, their chemical applications have included interpreting SMILES strings[141,249] or in DNA-sequence-based predictions of DNA-protein interactions[250]. Recurrent units have also been used to generate SMILES strings from images interpreted by convolutional layers[232]. More applications of RNNs can be found in generative models (Section 5.7).

Attention mechanisms[251], which are also called attention layers, are another tool mainly used for handling sequences of data, especially in natural language tasks[248]. In analogy with human cognition and attention, attention layers enable the model to focus on different parts of the input at different times, depending on the internal state of the model. These layers can be combined with other types of layers, for example LTSM layers[252] or graph convolutional layers[9]. A single attention layer can be thought of as assigning an importance between zero and one for each element in the input sequence. In practice, one predicts an output vector using an ANN and then applies the Softmax function, a differential version of scaling by the maximum element that is given by[204]:

$$\sigma(x_i) = \frac{\exp(x_i)}{\sum_{i=1}^{m} \exp(x_i)} \tag{5.19}$$

Softmax functions are used in many other contexts, most notably for transforming the output of ANNs in classification problems into probabilities over the possible classes[60]. However, Softmax functions are used here to generate a normalized importance from the output of the attention mechanism. This set of relative, normalized importance values is then multiplied by the input to compute what is called the context vector. Summing over all elements in the sequence gives an average of the initial inputs weighted by their importance, with those elements with high importance being emphasized over those with low importance, for example certain words in sentence or atoms in a molecule[9]. One can use multiple attention units to extract different parts of inputs in an approach known as multi-headed attention[251], for example, having different units to focus on the subject or object of a sentence. In practice, there are some difficulties associated with handling vectors of variable size that are discussed further in Ref.[253] and Ref.[251]. The broad picture of this approach is that an attention layer can emphasize or de-emphasize parts of the inputs to improve subsequent components of a model in a regression or classification task.

Attention layers have begun to be applied to chemistry, in the prediction of reaction outcomes. One example involved the IBM-developed 'Molecular Transformer' model[140,249], which predicts the SMILES strings of products from the SMILES strings of reactants using the same attention mechanisms used in language translation. In MPNNs models[9], the attention has been applied to a sum over the learned atomic representations to select most important reaction sites in the molecule.

5.6 Neural network potentials

The most frequent application of modern neural networks in the contemporary chemistry literature are is neural network potentials. Therefore, we will dedicate some discussion to their specific properties. The modern crop of 'big data' NNPs first appeared in the literature in their current form in 2007[1], and the basic ideas have not changed substantially. The idea of an NNP is to create a mapping between the 3D arrangement of atoms and the energy or forces of these atoms. These

NNPs have been developed with the aim to enable high-fidelity dynamics near the cost of fixed analytical form classical molecular dynamics after parameterization with limited quantities of more expensive first-principles data. If achieved in stable models that conserve energy and reproduce first-principles data well, NNPs have the promise of enabling the simulation of long time dynamics near the cost of classical force fields but with the accuracy of higher-cost, physics-based models that are intractable at long times[189].

Because NNPs provide fine-grained information about the dependence of energy on atomic positions, they must use features that encode detailed geometric information. It is standard practice to consider atomic descriptors, such that a molecule is considered as a collection of distinct individual atoms[189]. Each atom, j, is described by a set of descriptors, $g_j(r, Z)$, depending on its position, r_j, and type, Z_j, that encode its environment, and on the position and nature of all atoms relative to atom j (Figure 5.8). We have provided a detailed discussion of some common forms of $g(r, Z)$ in Chapter 4, although machine-learned representations described in Section 5.5.2 are also used[241]. It worth noting that g_j can depend on all atomic positions and types (r, Z) and must be updated at each time step. In practice, real space cutoffs[2] on the order of 5Åare used to make the calculation of g more affordable.

Although such distances fall short of the length scales known to be needed to encode electrostatic and van der Waals interactions, this approach has proven relatively successful. In some cases, physics-based terms are added on top of the atomic energy terms to recover the longer range interactions neglected by the typically short real-space cutoffs[57,146]. Each atomic representation is then fed into its own copy of the neural network, generally using a different network for each atomic species[2,44,45]. This means that each atom is effectively one independent training point. The energy output for each atom is then summed over all atoms to give the full energy of the system. This general approach synergistically produces derivatives on a per-atom basis, which can be recognized as the forces needed for geometry optimization or molecular dynamics. An example of an NNP is represented in Figure5.8. In addition, because the model inputs are local atomic environments, it is possible to train on atoms in small systems of organic molecules, which can be computed affordably with high accuracy first-principles methods, and apply the model to larger systems (e.g., $\sim$ 300 atoms in Trp-cage in Ref.[254]). Such an approach requires that all of the relevant chemical bonding and non-covalent interactions are faithfully captured in the small molecule training data.

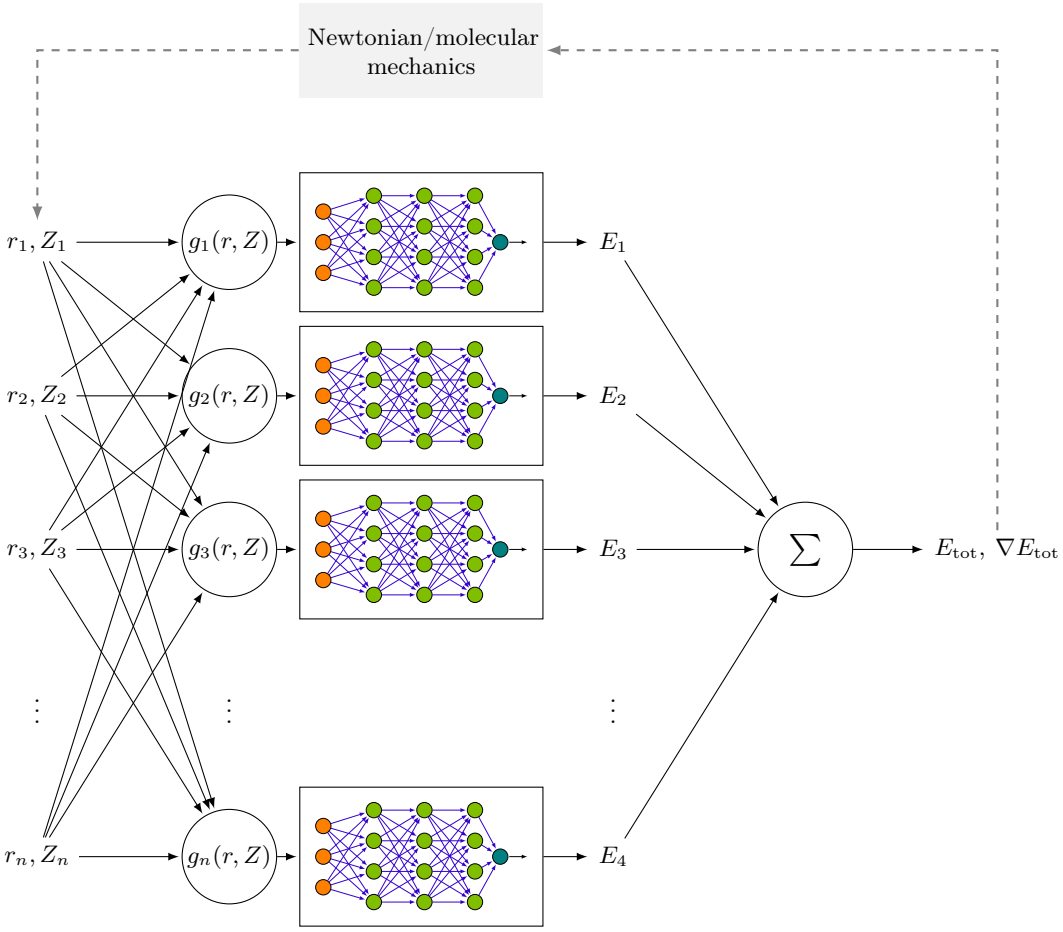

Figure 5.8: Overview of a NNP implementation. Atomic coordinates (r_j) and types (Z_j) are used to generate a local descriptor for each atom (g_j) that is passed into a copy of the neural network, which produces a corresponding atomic energy E_j. Energies are summed to provide a single, differentiable function of the system energy E_{tot} and gradient ∇E_{tot} (with respect to atomic coordinates), which are used to drive atomic position updates via standard molecular dynamic integrators.

The most computationally-demanding aspect of evaluating an NNP is calculating and updating the atomic representation, which requires significantly more computation than a standard, fixed charge force field but with the promise of potentially being more accurate[2]. In terms of training data, NNPs are usually trained on many snapshots from *ab initio* molecular dynamics or DFT calculations on a combination of equilibrium and distorted geometries (e.g., through sampling displacements along normal modes). Thus, these models are highly data intensive in a manner that scales with distinct atom types. For example, the ANI-1 NNP was trained on over 20M DFT

calculations[2,31]. However, careful selection of data through active learning has been shown to improve NNPs training[189,254] and reduce the size of needed data sets. In this approach, one takes the variance of a number (~ 5) of distinct NNP models each trained on equal-sized re-sampling of the training data with replacement in a method called bootstrapping[60]. When the variance of the models' predictions on a new molecular configuration is high, a new DFT calculation is run on that geometry, and the result is added to the training data.

5.7 Generative models

While the focus of this text is on supervised regression tasks, neural networks also support some unique types of modeling that have garnered substantial interest. One of these that is of particular relevance to chemical sciences[255] is that of *generative modeling*, which we will briefly discuss. The principle goal of generative modeling is not to assign values to previously unknown input samples but rather to generate new inputs that are similar and yet still distinct from the original inputs[204] (Figure 5.9). The model should 'imagine' new samples that are diverse and representative of the original data. In chemistry, this would correspond to the generation of new molecules, based on but distinct from existing sets that largely follow the chemical rules implicitly encoded in the source data. In theory, such models could enable significant augmentation of such databases. By controlling the generation process, one can also steer the generation process toward specific regions of chemical space[255] that are contain synthetically accessible or soluble molecules[256]. This approach works by designing a 'reward function' that provides a higher score to generated samples that have a desired property[257].

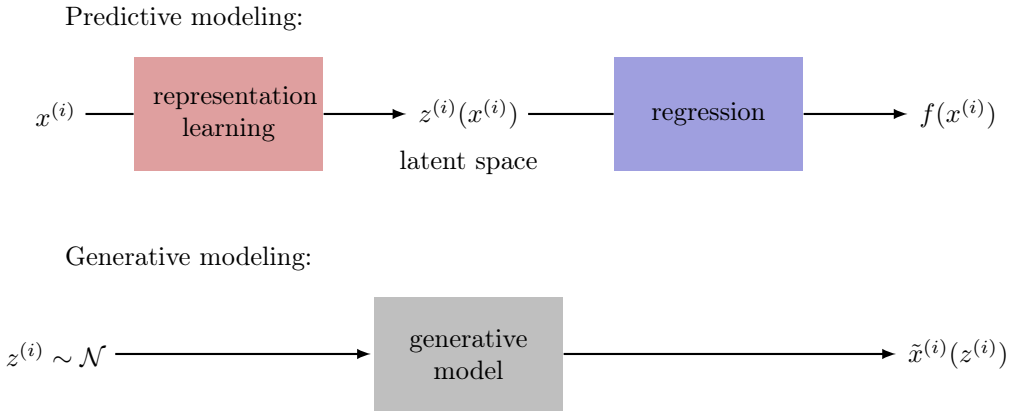

Figure 5.9: Comparison between predictive (top) and generative (bottom) modeling approaches. In predictive modeling, the learned function maps from the input $x^{(i)}$ to the output $f(x^{(i)})$ through the latent representation $z^{(i)}$. By contrast, generative modeling attempts to map a (usually Gaussian) distribution of vectors in latent space, $z^{(i)} \sim \mathcal{N}$, to a distribution of synthetic inputs $\tilde{x}^{(i)}(z^{(i)})$), such that these synthetic inputs should be indistinguishable from samples from the original data distribution.

There are two primary types of generative model: variational autoencoders (VAEs) and generative adversarial networks (GANs). These two types differ primarily in how the generative model is trained, but, once trained, both models function similarly. The trained generative model takes in samples from a random latent variable, which is almost always drawn from a Gaussian distribution[204,258]. The mean and covariance are learned during training in VAEs, but are typically set at the outset when using GANs. This use of random variables is essential to ensure that each time the model is called it will generate distinct output. The generative model can be interpreted as a mapping between a distribution over the latent space to a distribution over the space of real input data (e.g., in chemistry, molecules). The sampled latent vectors are passed through a neural network model that outputs vectors that imitate the distribution of the 'real' data as closely as possible[204]. This is the inverse of the standard representation learning process used to derive chemically meaningful latent spaces from input data. Here, we are extracting syntactic meaning from variations in the latent vector and reconstructing real data (e.g., molecules from implicit chemical rules). This difference is illustrated in Figure 5.9. The size of the latent space can be chosen freely, with a larger latent space potentially leading to more variation in the generated data at the cost of likely being more difficult to train.

VAEs are related to a simpler class of models called autoencoders. A standard autoencoder, as shown in the top pane of Figure 5.10, consist of an encoder, which is a neural network that maps the inputs to a latent vector, just like a standard ANN. However, this encoder is followed by another neural network that acts as a decoder and converts points from the latent space back into their input format[204]. During training, the loss function is set as the difference between the original data points and the reconstructed data $\tilde{x}$. This loss function is minimized by updating the weights in the encoder and decoder using the same type of optimization methods described in Section 5.3. While the utility of this approach may not be intuitive, it creates low-dimensional latent representation that can be used to interpret high-dimensional data more readily. Autoencoders have been used in chemistry to cluster molecular dynamics trajectories[259,260] and extract collective variables.

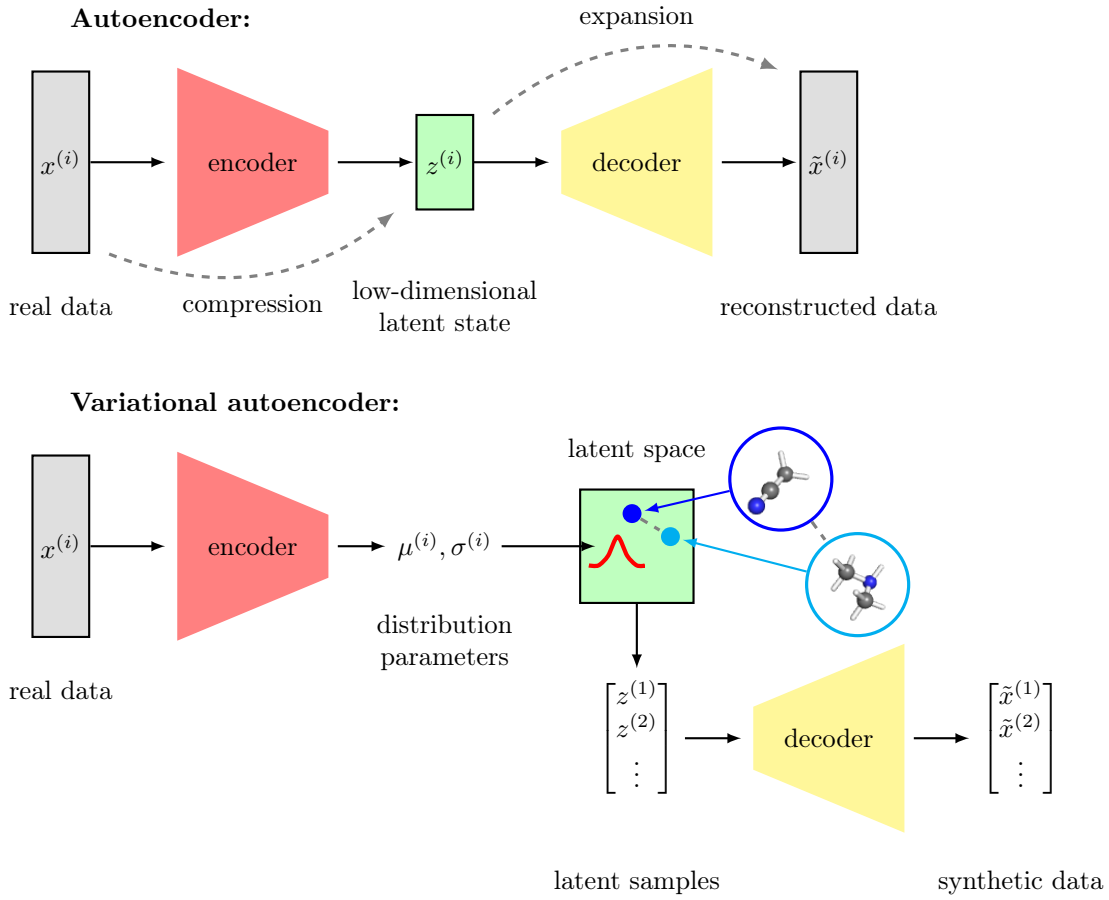

Figure 5.10: Schematic of autoencoder (top) and variational autoencoder (bottom) architectures. In both cases, samples of real data, $x^{(i)}$, are passed through an encoder ANN, to create a latent representation, $z^{(i)}$, that is then decoded back to the input space using a second ANN, yielding reconstructed or synthetic data $\tilde{x}^{(i)}$. In the variational case, instead of outputting the latent state directly, the encoder produces parameters, $\mu^{(i)}$ and $\sigma^{(i)}$ that define a probability distribution over latent space, shown as a red curve. This distribution is repeatedly sampled to generate a distribution of latent vectors, which are decoded into synthetic data. The latent space should encode chemical or structural similiarity, such that adjacent points decode into related molecules, as illustrated with the insets of two related structures, dimethylamine (cyan) and acetonitrile (blue).

The VAE also consists of an encoder, but this encoder maps the inputs to parameters of the latent space distribution instead of the latent vectors directly. In practice, the mapping is to the mean and variance or covariance of a normal distribution. This distribution is then sampled to

generate latent points that are passed to the decoder or generator, which produces synthetic inputs as described above. In order to train the model, real inputs are fed to the encoder network, and the resulting latent distribution is sampled and used to generate a distribution of synthetic data. This distribution is scored by how similar it is to the original distribution, and the gradient of this loss function can be used to update the network weights for both the encoder and generator. Care must be taken in selecting a suitable metric for measuring the similarity between the synthetic and original data distributions. It must be differentiable for training. The metric should also encourage a diverse synthetic distribution, as opposed to perfectly reconstructing the input as in the standard autoencoder. The approach typically taken is to optimize the *variational lower bound* on the probability of observing the original data under the proposed distribution from the generator, the theory behind this choice is detailed more in Ref. [258] and practical advice is provided in Ref. [261]. Careful choice produces a loss function that balances learning a good reconstruction of the data *and* encourages a sufficiently broad learned distribution that will generate diverse data.

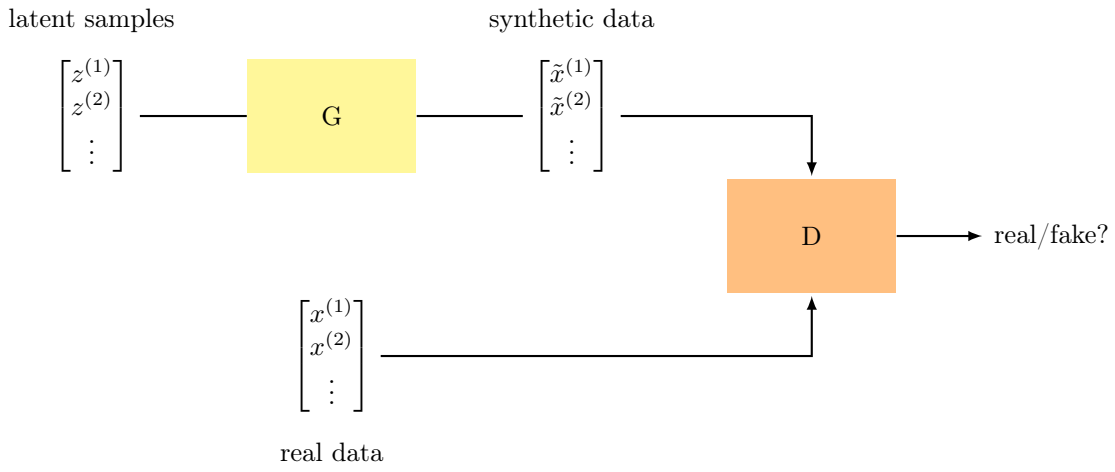

Figure 5.11: Schematic of GAN training process. Samples from the latent space, $z^{(i)}$ are drawn from a fixed distribution and then passed through the generator ANN, G, which converts them into synthetic data, $\tilde{x}^{(i)}$. A second ANN, called the discriminator, D, receives both the synthetic data and the real data, $x^{(i)}$, and must classify each observation as real or synthetic. When training G, the objective function is to make D classify synthetic data as fake.

GANs are the other main type of generative model. Similar to VAEs, GANs also use a decoder network, here, called a *generator*, which maps samples of the latent space to elements in the input or feature space. Unlike VAEs, there is no encoder stage, and the generator is not trained to reproduce the existing data. Instead GANs are trained simultaneously with a second network called the *discriminator*. The objective of the discriminator is to distinguish between the synthetic samples from the generator and samples from the original data [262], as shown in Figure 5.11. These two networks are trained in an *adversarial* fashion in stages. The generator is trained to make the discriminator confuse its output with the real data while the discriminator is trained to correctly distinguish real

and fake data. During the training of one model, the parameters of the other are held constant. Both the generator and discriminator are typically fully-connected MLPs, although in some cases convolutional, message passing, or recurrent layers may be used. This simple approach can generate very convincing synthetic results, for example realistic human faces using convolutional layers[263]. One issue with GANs is that if the generator finds one output that can fool the discriminator, the entire synthetic data distribution can concentrate on that output regardless of the latent space input. This produces a very low diversity of synthetic data, which is called 'mode collapse'. This issue can be partly addressed by attempting to match the distribution of the synthetic data to the real data using the Wasserstein loss function[264] instead of the the binary true/false labeling from the discriminator.

Generative models have begun recently to be applied in chemistry. One advantage of autoencoders over GANs is that any molecule can be placed in the latent space with the encoder. Doing so makes it possible to identify paths between known molecules in the latent space[38] or to optimize molecular properties in latent space[209] and to then decode them to real molecules. They have been used to generate organic molecules as either SMILES strings or as graphs with both VAEs[265–268] and GANs[256,269,270]. Both representations of molecular structures are challenging to generate in comparison to fixed size vectors or images because the size of the molecules can vary substantially. SMILES strings have been generated sequentially, with the starting character generated first and then subsequent letters added to the string until a termination key is selected[256]. This sequential generation method make use of a technique known as policy gradient optimization[271], which involves learning a probability distribution over the possible letters that can be added based on the current content of the string. Due to the probabilistic nature of the process, taking the derivative of the so-called policy is not as simple as the deterministic derivatives in Section 5.3, but an expectation or average gradient can be calculated using multiple samples. Initially it was not clear how to extend generative models to varying-length sequences such as SMILES strings, but this has been achieved (e.g., in SeqGAN[272]) through a combination of policy gradient and recurrent layers (Section 5.5.4) to generate variable length text output. These techniques have been adapted for SMILES-generating models[256,269,273] in chemistry.

Generation of molecular graphs, as opposed to SMILES strings, can be realized 'all-at-once' by outputting a full graph[257,266] up to a maximum fixed size or by learning step-wise construction rules[267,274,275], as with the sequential assembly of SMILES. One potential problem with generative models is the ease with which one can generate invalid structures. This issue can be resolved though enforcing grammar in SMILES strings[276,277], introducing text-based SMILES alternatives that have simpler grammar[278], or limiting the types of graphs that can be generated by only adding chemically-reasonable bonds[274]. Generative models are usually assessed based on the fraction of valid and unique generated molecules as well as the diversity of the generated molecules. In the case that targeted molecules have been generated, they could also be judged on the distribution of the molecules' targeted physical properties such as molecular weight and solubility. Since generative modeling for chemistry is a relatively new field, it is not immediately apparent which, if any, of the presented strategies is best. In a comparison of SMILES-based GANs and SMILES- and graph-based VAE, similar performance was obtained for all models[279]. In a comparison between an all-at-once graph GAN and a sequence-based SMILES GAN, it was reported[257] that the graph approach resulted in higher validity and better ability to steer generation towards specific properties but at the cost of a much lower uniqueness score compared to SMILES.

Chapter 6

Applying Machine Learning Models in Chemistry

6.1 Overview

The techniques outlined in the previous sections represent a powerful and flexible set of tools that can help a chemist save time and interpret data. However, machine learning is not a panacea that can solve every problem. Even where machine learning could be beneficial, the diversity of models and representations can be daunting to new practitioners. In this chapter, we provide some practical steps for starting to apply machine learning models in chemical science (as summarized in Figure 6.1).

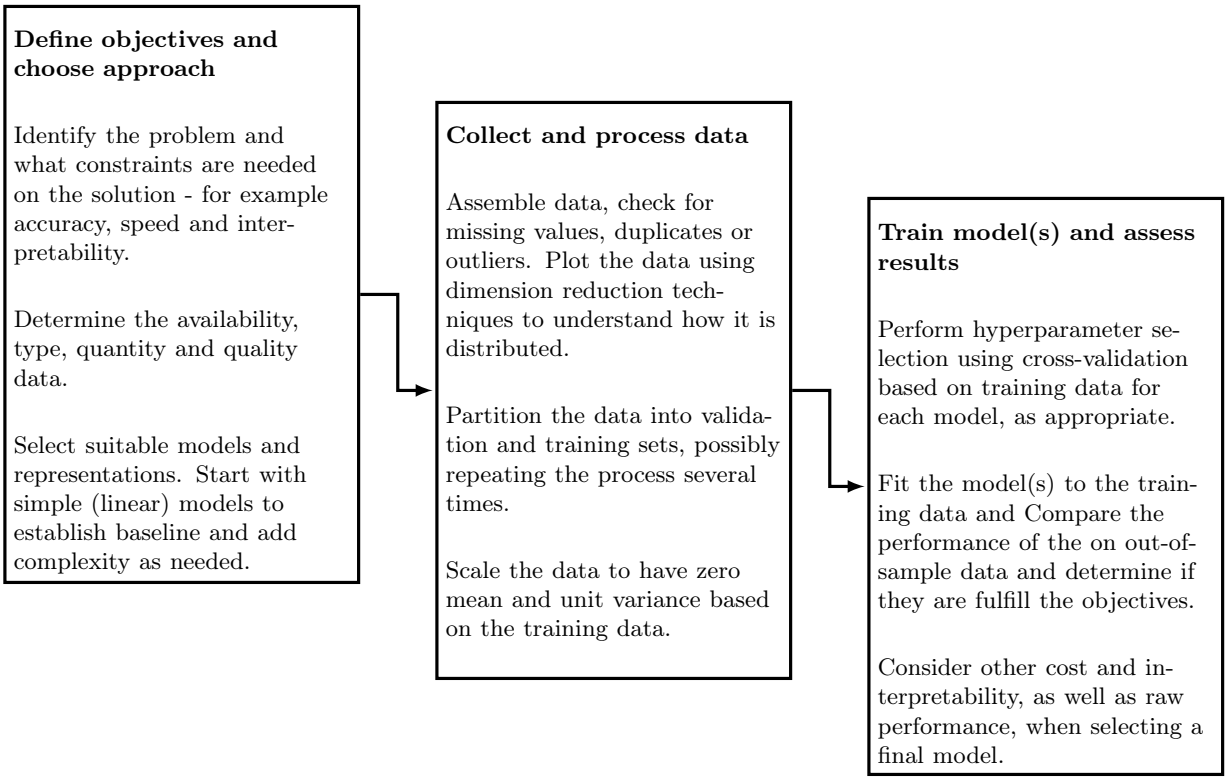

Figure 6.1: Summary of practical steps in developing machine learning predictions for chemical systems.

Disclaimer: This chapter provides references to specific software tools, from academic codes and open-source projects to propriety software from commercial vendors. These are provided in order to collect some of the tools that the authors are aware of in a central location, and should not be taken as endorsement that any specific tool is appropriate for a given task. In a rapidly evolving field, new tools will continue to be developed and the capacity and availability of tools listed here may change. The authors and publisher makes no claims, promises, or guarantees about the capacity of any tools listed here, and expressly disclaim liability for use of any software listed here.

6.2 Defining objectives

In order to benefit from a machine learning toolbox, it is important to lay out which goals you would like to achieve beforehand. Carefully considering objectives at the outset can help you to select appropriate methods. Some questions that should be considered are:

1. What do you hope to achieve with ML? For example, are you aiming to bypass time-intensive simulations or experiments, enable design in large chemical spaces, build self-driving labora-

tory experiments, or create accurate potential energy surfaces for a single species? A clear idea of the intended application can guide how much emphasis to put on different factors such as accuracy and interpretability. This choice of focus is also important because some favorable model characteristics must be chosen at the expense of others (e.g., ease of model training and interpretation vs. model generalization).

2. How much data is available and is it intended or possible to generate more easily or rapidly? The quantity of data is important for model selection, as explained below, and some methods, such as GPR or ANNs with confidence estimators, can be helpful in recommending new data points to acquire.

3. Is there noise in the data, and how accurate do models need to be to be useful? It may not be worthwhile to predict simulation outcomes to a greater degree of accuracy than the reliability of the underlying data source. For example, modern methods can predict atomization energies of small organic molecules to within tenths of a kcal/mol[242] relative to DFT, which exceeds the typical accuracy of the DFT calculations with respect to available experimental data[43].

6.3 Choosing a representation and a model

The main choices made by a chemist applying ML to their problem are in selecting a model and a representation. These choices are interwoven and must be related back to the objectives and the amount of data that is available. Trial-and-error is an important part of machine learning. For example, CV is nothing more than testing many different model parameters and selecting the best values. This is a useful general attitude to apply broadly towards ML models and representations as well. Once data has been collected, it is usually straightforward to train a range of models and test a variety of representations. Beyond a trial-and-error approach, some general considerations when selecting models and representations are:

1. The choice of representation is the easiest part of the ML for chemists with expert domain knowledge to contribute about the systems of interest[68,121]. At the same time, one of the greatest benefits of working with large data sets and ML tools is to appreciate and develop understanding of the limitations of 'chemical intuition'. Indeed, there is evidence suggesting that learned representations for simple organic molecules can outperform carefully 'engineered' representations when the representations are learned on large training set sizes[176]. However, this approach is not suitable for cases where less data is available or greater chemical complexity arises. For example, tailored representations have been beneficial in heterogeneous catalysis[280], coordination chemistry[81,281], and materials[282]. Overall, it can be beneficial to use experience to guide candidate subsets of features but ultimately apply statistical tests to select the most promising feature set.

2. Always start with a simple ML model first, as this helps validate a proposed representation quickly. Simple models can also prevent issues, such as redundant, noisy, or missing data early on. One can quickly calculate the mean absolute deviation (MAD) of the predicted property, i.e., that is the average difference from the mean value, $\frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})$. One can also calculate the average out-of-sample/CV error for a regularized linear model (e.g., LASSO). In order for more complex models to be justified, these two metrics from the linear model should be improved upon.

3. For accurate predictions that depend on 3D structure, a representation that incorporates intermolecular distances is essential. A choice should be made between whole-molecule representations, for example HDAD[43] and others[87] or the SOAP kernel[89], and atomic representations, such as symmetry functions[1] or continuous filter convolutions[42]. Both approaches have yielded good accuracy for predicting molecular properties. If the property in question can be readily decomposed into atomic contributions (e.g., atomization energy or partial charges), use of atomic representations can increase the number of observations available from the training data and provide greater promise of application to systems beyond the size distribution in the training data.

4. The size of the available data set is also important. Complex ANN architectures handle large numbers ($> 10^4$) of training observations more readily than kernel methods. Kernel methods, such as KRR or GPR, can provide effective models with fewer training examples, particularly if the training examples are chosen to give good coverage of the expected chemical space. KRR models can typically match or exceed the accuracy of other models on small data sets[87] and are easier to train and interpret.

5. GPR is a natural choice for active learning where the decision about which data points to study next must be iteratively repeated. In active learning, one starts from a small initial pool of data and can use the built-in uncertainty estimates of a GP to identify which new data would be most informative[88,93,94] to the model. However, ANNs have also shown good performance for active learning[189,254] based on quantified uncertainty from the variance of a small number (< 10) of different models or based on latent-space-derived uncertainty[208]. As with the GPR, the ANNs are improved upon retraining with the acquired uncertain data points.

6. When available data set sizes are limited, an approach known as transfer learning can be beneficial. Transfer learning refers to the process of training a model (e.g., an ANN) on a large quantity of cheap-to-acquire data and then retraining the model on a smaller amount of more challenging to acquire but related data. In computational chemistry, this approach has been applied by training on a large number of affordable hybrid DFT calculations and then retraining with more computationally-demanding correlated wavefunction theory calculations[283]. This approach works because the ANN first learns a mapping between molecular input and property outputs that is at least partially transferable to the less abundant, higher quality data. In particular, the model training starting point is with that of the original, affordable data, and the model need only to learn a small adaption to match the higher quality data[284,285]. The first few layers of the model that accomplish the feature engineering are sometimes 'frozen' and only the later layers are trained[286].

7. If extracting physical insight from the data is a priority, feature selection techniques can be combined with interpretable representations, for example from heuristics or graph-based descriptors[81]. RF models are very quick to train and provide implicit feature importance information scores. Thus, even if RF models are not predictive on the data set, they are useful for analysis of feature importance. The visualization techniques presented below can also aid analysis of feature importance.

6.4 Data processing

Data preparation is an important step in the ML pipeline that is easy to overlook but in practice can be fairly time- and resource-intensive[60]. Processing raw data from data from calculation outputs, experiments, or existing databases can be time-consuming. Mistakes in this stage can lead to incorrect results when training models. So, it makes sense to consider how the data will be transformed and partitioned upfront. Here is some general guidance on getting data ready for feeding to an ML model:

- Normalize both features and output properties by subtracting the average value of each feature or property and dividing by the standard deviation[60]. This step is implemented by default in popular software. This results in each column of the data matrix and output having a mean value of zero and unit variance and removes challenges related to scaling of different quantities. It can also make model training easier and more stable. For example, an ANN initialized with weights near zero will produce output near zero. If the outputs are normalized around zero, this means the initial model will be producing outputs close to the average. We also explored the relationship between the regression constant and this average for linear models in Section 3.2.1. Remember to store the normalization values because they are needed to convert the output back into 'real' units after training.

- Separate the available data into train and test fractions at the initial stages of model training and exclude test data from model training. This distinction applies to normalizing the data as well. A common mistake is to standardize all data before partitioning it, which causes the OOS data distribution to influence the model[60]. A typical fraction of training data is $70 - 90\%$ of the original data set[60]. It is good practice to conduct multiple divisions of the data into train and test and repeat the entire process, especially when the training data set size is small (e.g., $\leq 10^3$ points). Remember to shuffle data before dividing it to avoid any impact of data ordering (e.g., alphabetical or chronological).

- Be aware of the diversity of the data used. It has been observed[287,288] that chemical data sets are often strongly clustered. These data sets are comprised of a number of few, distinct domains. A uniform train/test division could produce a test set too similar to training data and result in spuriously good test set errors. This can be countered by examining clustering in the feature space directly and choosing to exclude[287] specific clusters from the training data.

- For the above reasons, external data sets that are not involved in the modeling training process are excellent tests for trained models and are strongly recommended. For example, ANN models can be trained on DFT calculations on transition metal complexes and then tested on experimentally-characterized complexes[102] or trained on a library of small organic molecules[31] and tested on a diverse set of larger molecules[254].

- Look for imbalances in the data, which can represent the over-abundance of one atom type or one type of system. All ML models are inherently statistical, so they will inherit any biases from their training data. This makes modeling rare or uncommon values very challenging and affects many materials[289] and pharmacological[290] applications. The techniques to construct representative training and test data are known as stratified sampling methods, as described in greater detail in Ref.[153] or Ref.[290]. One can test for imbalance in regression tasks by plotting

the distribution of features and output values in test and train sets, with large discrepancies likely to result in poor performance.

- Be cautious when model predictions are implausibly accurate. It is easy to unintentionally provide the outputs to the model, e.g., by leaving an unintended column in the matrix. One should ideally validate any result on new inputs and explore how the model behaves on radically different inputs. Another strategy is to permute columns of the data randomly and re-apply the model, which should degrade performance[291].

- Conversely, pay close attention to observations that are poorly-predicted by the model, particularly any training data that is not well modeled. This can signal that there is something wrong with the data. If one is a computational chemist, this data point could be from a calculation that failed to converge properly. If one is an experimentalist, this could correspond to a contaminated or incorrectly interpreted experimental result. Such data is called *mislabeled*. Kernel methods are especially good for identifying mislabeled data because they map transparently between outputs and feature space distances. Any badly-predicted training data will have output values very different from their neighbors in the feature space. If these points are correctly-labeled, it is a signal the feature space (i.e., representation) needs to be revised to place these points closer to the appropriate neighbors.

- Discrete variables or *factors* can be used as features with a technique known as one-hot encoding[60]. This works by representing a single discrete feature as several binary features with one less than the number of values that can be assigned to the factor. For example, atom identity with four possible values of C, H, O, and N can be represented by a three-element vector: 000 for C, 001 for H, 010 for O, and 100 for N. Hence, one discrete feature becomes many binary features. Some ML models handle discrete variables better[60] (e.g., ANNs and RFs) than others.

- Visualization of high-dimensional spaces is challenging but can be very useful in interpreting data, feature space, and model performance. In order to visualize data in the feature space, one must convert from 10s–100s of dimensions to a 2D image, as discussed in Section 4.5. Coloring this image by the target property can show how well feature spaces correlate with output properties and identify any outlier points that have very different properties with respect to neighbors. It can be useful to overlay model errors on this plot as well, which can help indicate if the performance of the model varies based on feature space location.

6.5 Training and hyperparameter selection

Going from a selected model family, representation, and some appropriately-processed data to a useful predictive model involves both hyperparameter selection and actual training of the final model. Hyperparameter optimization is usually the most computationally expensive part of the process since it involves fitting many prospective models to the data and evaluating which is best. Here we present some general advice on tackling these problems in both kernel and ANN models:

- For kernel methods, start hyperparameter optimization by searching over a broad range of values approximately. If the optimal values are on the boundaries of the grid search, extend the range until the minimum (or minima) is included in the range. The search can then

be refined around regions of interest. Plotting the CV error response as a function of the hyperparameter values, as per Figure 3.4, can help you to interpret model behavior and identify where to refine the search.

- We recommend using isotropic kernels, i.e., those that have one single characteristic length hyperparameter, before trying more complex kernels. In some packages (e.g., mlegp[292]), more complex kernels with one hyperparameter per dimension such as eq. 3.20 are sometimes the default for GPR. While the resulting model with multiple length scales is potentially better able to fit data, hyperparameter estimation becomes much more challenging.
- Use multiple metrics to evaluate model errors, both for training and OOS data. We recommend the root-mean-square error (RMSE) (i.e. the root of the mean-square loss function, as in eq. 2.2), the mean absolute error (MAE), the mean absolute percentage error (MAPE), and the maximum error. These metrics give slightly different information about the error distribution because they are differently-sensitive to outliers. For example, a large discrepancy between the RMSE and MAE is indicative of outlier points with high error relative the mean.
- When training ANNs, use Hyperopt[35,62] to conduct the initial hyperparameter selection with at least 10% validation data. Use the Adam optimizer[219] with the default parameters for this first pass and focus on tuning the architecture, i.e., the number of layers and nodes. Insight into the training process can be obtained by observing how the loss function decreases during training. Early leveling-off of the loss function or rapid oscillations in the loss function indicate that the hyperparameter selection can be improved, as indicated in Section 5.4.
- Most packages will allow the user to track validation accuracy during training. Employ early stopping if the OOS error begins to increase. In the authors' experience, dropout can help overcome over-fitting issues with small data sets.
- Instabilities in training ANN models, which manifest as large oscillations in the loss function, can also be managed with gradient clipping[218], which essentially 'caps' gradients at a maximum value.
- Complex ANN models are often successfully based closely on those from other applications. For example, the SMILES-generating ORGAN[256] is based on SeqGAN[272]. In those cases, hyperparameter and training configuration can be modeled on what worked in the prior context.

6.6 Frameworks

While most ML methods could be implemented by hand, a large number of commercial and open source software tools provide highly-optimized implementations that make model training more straightforward. Here, we provide a brief list of some of these tools that can be used to apply any of the models described in the previous sections.

- ML models can be trained in most common scientific programming languages including Python, R[181], MatLab© and Julia[293]. Most tasks can be achieved in any language. The choice of which language to train models in can be made based on preference and previous experience, although Python is popular for its flexibility and well-developed ML ecosystem. The broad, cross-platform compatibility of Python also makes it easy to share code.

- Useful functions and models are available through open-source packages. Scikit-learn[294] is a Python package that provides basic routines for all of the models discussed in this book, including built-in functions for CV and model assessment. Some R packages that may provide convenient functionality include caret[295] for model assessment, randomForest[180] for RF and Kernlab[296] for KRR.
- While these systems are good for prototyping small neural networks as well, construction of large ANNs benefits from optimized code and acceleration with heterogeneous computing (i.e., graphics processing units or GPUs). For this reason, training of ANNs models should be conducted using an optimized library that automates model creation, derivative evaluation, and weight updates. Two popular libraries are TensorFlow[34] and PyTorch[297]. These libraries provide highly-efficient, low-level 'backends' that can compute operations, such as matrix products and back-propagation, quickly to accelerate ANN training and evaluation. Since TensorFlow is mainly a low-level library, many practitioners prefer to use the simpler 'frontend' Python package Keras[33], which provides convenient wrapper functions around the underlying backend, instead of TensorFlow functions directly. Pytorch includes such wrappers by default and is originally based on the Torch library[298], which is no longer developed separately.
- GPU computing can offer substantially faster training and prediction times in classical ML tasks such as image recognition[198] and has also been used to accelerate NNPs in chemistry[2]. This acceleration is likely to be especially important when data sets are large.

In addition to these general ML frameworks, a large number of chemistry-specific tools are available. While we expect the available tools to enlarge and evolve over the coming years, a few contemporary examples are provided below:

- RDKit[127] and OpenBabel[128] are open-source libraries for general manipulation and conversion of molecular data. These tools can convert from SMILES to 3D geometries for organic molecules, and have various routines to manipulate molecules and modify them on an atom-by-atom basis. These tools can also calculate multiple different simple features, from physiochemical properties to various fingerprints.
- DeepChem[299] aims to provide a 'one-stop-shop' Python library for machine learning on atomic systems, which includes routines based on TensorFlow for creating various types of ANN models including MPNNs. It also includes MoleculeNet[49], a suite of 21 benchmark datasets.
- SchNetPack[42] is a Python toolbox and command-line tool for Pytorch-based NNP training and evaluation, implementing both symmetry functions and continuous-filter convolutions (see Section 5.5.2).
- MatMiner[300] is a Python toolbox for calculation of representations, from simple counts to detailed 3D information, aimed at periodic systems. It connects with the pymatgen materials automation suite[301] for integration with first-principles calculations.
- ChemOS[302] is a Python project for coupling ML models with experimental setups to drive chemical discovery, including tools for connecting robotics with measurement, analysis and prediction tools.
- AFLOW-ML[303] provides a standard way of accessing the AFLOW library[52,304] of calculations on periodic systems with the aim of making/evaluating ML models on these systems simpler and faster.

Bibliography

[1] Behler, J.; Parrinello, M. Generalized Neural-Network Representation of High-Dimensional Potential-Energy Surfaces. *Physical Review Letters* **2007**, *98*, 146401.

[2] Smith, J. S.; Isayev, O.; Roitberg, A. E. ANI-1: an extensible neural network potential with DFT accuracy at force field computational cost. *Chemical Science* **2017**, *8*, 3192–3203.

[3] Rupp, M.; Tkatchenko, A.; Müller, K.-R.; von Lilienfeld, O. A. Fast and Accurate Modeling of Molecular Atomization Energies with Machine Learning. *Physical Review Letters* **2012**, *108*, 058301.

[4] Montavon, G.; Rupp, M.; Gobre, V.; Vazquez-Mayagoitia, A.; Katja,; Tkatchenko, A.; Müller, K.-R.; von Lilienfeld, O. A. Machine learning of molecular electronic properties in chemical compound space. *New Journal of Physics* **2013**, *15*, 095003.

[5] Janet, J. P.; Chan, L.; Kulik, H. J. Accelerating Chemical Discovery with Machine Learning: Simulated Evolution of Spin Crossover Complexes with an Artificial Neural Network. *The Journal of Physical Chemistry Letters* **2018**, *9*, 1064–1071.

[6] Xie, T.; Grossman, J. C. Crystal Graph Convolutional Neural Networks for an Accurate and Interpretable Prediction of Material Properties. *Physical Review Letters* **2018**, *120*, 145301.

[7] Ma, X.; Li, Z.; Achenie, L. E. K.; Xin, H. Machine-Learning-Augmented Chemisorption Model for CO2 Electroreduction Catalyst Screening. *The Journal of Physical Chemistry Letters* **2015**, *6*, 3528–3533.

[8] Nandy, A.; Zhu, J.; Janet, J. P.; Duan, C.; Getman, R. B.; Kulik, H. J. Machine Learning Accelerates the Discovery of Design Rules and Exceptions in Stable Metal- Oxo Intermediate Formation. *ACS Catalysis* **2019**, *0*, 8243–8255.

[9] Coley, C. W.; Jin, W.; Rogers, L.; Jamison, T. F.; Jaakkola, T. S.; Green, W. H.; Barzilay, R.; Jensen, K. F. A graph-convolutional neural network model for the prediction of chemical reactivity. *Chemical Science* **2019**, *10*, 370–377.

[10] Jurs, P. C.; Kowalski, B. R.; Isenhour, T. L. Computerized learning machines applied to chemical problems. Molecular formula determination from low resolution mass spectrometry. *Analytical Chemistry* **1969**, *41*, 21–27.

[11] Zupan, J.; Gasteiger, J. Neural networks: A new method for solving chemical problems or just a passing phase? *Analytica Chimica Acta* **1991**, *248*, 1–30.

[12] Gasteiger, J.; Zupan, J. Neural Networks in Chemistry. *Angewandte Chemie International Edition in English* **1993**, *32*, 503–527.

[13] Burns, J. A.; Whitesides, G. M. Feed-forward neural networks in chemistry: mathematical systems for classification and pattern recognition. *Chemical Reviews* **1993**, *93*, 2583–2601 0009–2665.

[14] Sumpter, B. G.; Getino, C.; Noid, D. W. Theory and applications of neural computing in chemical science. *Annual Review of Physical Chemistry* **1994**, *45*, 439–481.

[15] Venkatasubramanian, V. The promise of artificial intelligence in chemical engineering: Is it here, finally? *AIChE Journal* **2019**, *65*, 466–478.

[16] Kowalski, B. R.; Bender, C. F. Pattern recognition. Powerful approach to interpreting chemical data. *Journal of the American Chemical Society* **1972**, *94*, 5632–5639.

[17] Stuper, A. J.; Jurs, P. C. ADAPT: A computer system for automated data analysis using pattern recognition techniques. *Journal of Chemical Information and Computer Sciences* **1976**, *16*, 99–105.

[18] Zander, G. S.; Stuper, A. J.; Jurs, P. C. Nonparametric feature selection in pattern recognition applied to chemical problems. *Analytical Chemistry* **1975**, *47*, 1085–1093.

[19] Hopfield, J. J. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences* **1982**, *79*, 2554–2558.

[20] Minsky, M.; Papert, S. A. *Perceptrons: An introduction to computational geometry*; MIT press, 2017.

[21] Rasmussen, C. E. *Summer School on Machine Learning*; Springer, 2003; pp 63–71.

[22] Cortes, C.; Vapnik, V. Support-vector networks. *Machine learning* **1995**, *20*, 273–297.

[23] Tibshirani, R. Regression Shrinkage and Selection Via the Lasso. *Journal of the Royal Statistical Society: Series B (Methodological)* **1996**, *58*, 267–288.

[24] LeCun, Y.; Bengio, Y. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks* **1995**, *3361*, 1995.

[25] Fukushima, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics* **1980**, *36*, 193–202.

[26] Rumelhart, D. E.; Hinton, G. E.; Williams, R. J. Learning representations by back-propagating errors. *Nature* **1986**, *323*, 533–536.

[27] Curry, B.; Rumelhart, D. E. MSnet: A neural network which classifies mass spectra. *Tetrahedron Computer Methodology* **1990**, *3*, 213–237.

[28] Caruthers, J. M.; Lauterbach, J. A.; Thomson, K. T.; Venkatasubramanian, V.; Snively, C. M.; Bhan, A.; Katare, S.; Oskarsdottir, G. Catalyst design: knowledge extraction from high-throughput experimentation. *Journal of Catalysis* **2003**, *216*, 98–109.

[29] Potyrailo, R.; Rajan, K.; Stoewe, K.; Takeuchi, I.; Chisholm, B.; Lam, H. Combinatorial and high-throughput screening of materials libraries: review of state of the art. *ACS combinatorial science* **2011**, *13*, 579–633.

[30] Ramakrishnan, R.; Dral, P. O.; Rupp, M.; von Lilienfeld, O. A. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific Data* **2014**, *1*, 140022.

[31] Smith, J. S.; Isayev, O.; Roitberg, A. E. ANI-1, A data set of 20 million calculated off-equilibrium conformations for organic molecules. *Scientific data* **2017**, *4*, 170193.

[32] Kim, E.; Huang, K.; Saunders, A.; McCallum, A.; Ceder, G.; Olivetti, E. Materials Synthesis Insights from Scientific Literature via Text Extraction and Machine Learning. *Chemistry of Materials* **2017**, *29*, 9436–9444.

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

[34] Abadi, M. et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. 2015; `https://www.tensorflow.org/`, Software available from tensorflow.org.

[35] Bergstra, J.; Cox, D. D.; Yamins, D. Hyperopt: A Python Library for Optimizing the Hyperparameters of Machine Learning Algorithms. *Proceedings of the 12^{th} Python in science conference* **2013**, 13–20.

[36] Paesani, F. Getting the Right Answers for the Right Reasons: Toward Predictive Molecular Simulations of Water with Many-Body Potential Energy Functions. *Accounts of Chemical Research* **2016**, *49*, 1844–1851.

[37] Ponder, J. W.; Wu, C.; Ren, P.; Pande, V. S.; Chodera, J. D.; Schnieders, M. J.; Haque, I.; Mobley, D. L.; Lambrecht, D. S.; DiStasio Jr, R. A. Current status of the AMOEBA polarizable force field. *The journal of physical chemistry B* **2010**, *114*, 2549–2564.

[38] Sanchez-Lengeling, B.; Aspuru-Guzik, A. Inverse molecular design using machine learning: Generative models for matter engineering. *Science* **2018**, *361*, 360–365.

[39] Duch, W.; Diercksen, G. H. F. Neural networks as tools to solve problems in physics and chemistry. *Computer Physics Communications* **1994**, *82*, 91–103.

[40] Ren, F.; Ward, L.; Williams, T.; Laws, K. J.; Wolverton, C.; Hattrick-Simpers, J.; Mehta, A. Accelerated discovery of metallic glasses through iteration of machine learning and high-throughput experiments. *Science Advances* **2018**, *4*.

[41] Rogers, D.; Hahn, M. Extended-Connectivity Fingerprints. *Journal of Chemical Information and Modeling* **2010**, *50*, 742–754.

[42] Schütt, K. T.; Sauceda, H. E.; Kindermans, P.-J.; Tkatchenko, A.; Müller, K.-R. SchNet – A deep learning architecture for molecules and materials. *The Journal of Chemical Physics* **2018**, *148*, 241722.

[43] Faber, F. A.; Hutchison, L.; Huang, B.; Gilmer, J.; Schoenholz, S. S.; Dahl, G. E.; Vinyals, O.; Kearnes, S.; Riley, P. F.; von Lilienfeld, O. A. Prediction Errors of Molecular Machine Learning Models Lower than Hybrid DFT Error. *Journal of Chemical Theory and Computation* **2017**, *13*, 5255–5264.

[44] Artrith, N.; Morawietz, T.; Behler, J. High-dimensional neural-network potentials for multicomponent systems: Applications to zinc oxide. *Physical Review B* **2011**, *83*, 153101.

[45] Morawietz, T.; Behler, J. A Density-Functional Theory-Based Neural Network Potential for Water Clusters Including van der Waals Corrections. *The Journal of Physical Chemistry A* **2013**, *117*, 7356–7366.

[46] Li, H.; Collins, C.; Tanha, M.; Gordon, G. J.; Yaron, D. J. A Density Functional Tight Binding Layer for Deep Learning of Chemical Hamiltonians. *Journal of Chemical Theory and Computation* **2018**, *14*, 5764–5776.

[47] Moore, J. H. *Epistasis: Methods and Protocols*; Springer New York: New York, NY, 2015; pp 315–325.

[48] Gu, G. H.; Plechac, P.; Vlachos, D. G. Thermochemistry of gas-phase and surface species via LASSO-assisted subgraph selection. *Reaction Chemistry & Engineering* **2018**, *3*, 454–466.

[49] Wu, Z.; Ramsundar, B.; Feinberg, E.; Gomes, J.; Geniesse, C.; Pappu, A. S.; Leswing, K.; Pande, V. MoleculeNet: a benchmark for molecular machine learning. *Chemical Science* **2018**, *9*, 513–530.

[50] Delaney, J. S. ESOL: Estimating Aqueous Solubility Directly from Molecular Structure. *Journal of Chemical Information and Computer Sciences* **2004**, *44*, 1000–1005.

[51] Wang, R.; Fang, X.; Lu, Y.; Yang, C.-Y.; Wang, S. The PDBbind Database: Methodologies and Updates. *Journal of Medicinal Chemistry* **2005**, *48*, 4111–4119.

[52] Curtarolo, S.; Setyawan, W.; Hart, G. L. W.; Jahnatek, M.; Chepulskii, R. V.; Taylor, R. H.; Wang, S.; Xue, J.; Yang, K.; Levy, O.; Mehl, M. J.; Stokes, H. T.; Demchenko, D. O.; Morgan, D. AFLOW: An automatic framework for high-throughput materials discovery. *Computational Materials Science* **2012**, *58*, 218–226.

[53] Lowe, D. M. Extraction of chemical structures and reactions from the literature. Ph.D. thesis, University of Cambridge, 2012.

[54] Jain, A.; Ong, S. P.; Hautier, G.; Chen, W.; Richards, W. D.; Dacek, S.; Cholia, S.; Gunter, D.; Skinner, D.; Ceder, G.; Persson, K. a. The Materials Project: A materials genome approach to accelerating materials innovation. *APL Materials* **2013**, *1*, 011002.

[55] for Advancing Translational Sciences, T. N. C. The Tox21 Challenge. 2014; `https://tripod.nih.gov/tox21/challenge/`.

[56] Ramakrishnan, R.; Hartmann, M.; Tapavicza, E.; von Lilienfeld, O. A. Electronic spectra from TDDFT and machine learning in chemical space. *The Journal of Chemical Physics* **2015**, *143*, 084111.

[57] Unke, O. T.; Meuwly, M. PhysNet: A Neural Network for Predicting Energies, Forces, Dipole Moments, and Partial Charges. *Journal of Chemical Theory and Computation* **0**, *0*, null.

[58] Zilian, D.; Sotriffer, C. A. SFCscoreRF: A Random Forest-Based Scoring Function for Improved Affinity Prediction of Protein–Ligand Complexes. *Journal of Chemical Information and Modeling* **2013**, *53*, 1923–1933.

[59] Vapnik, V. *The nature of statistical learning theory*; Springer-Verlag New York, 200.

[60] Hastie, T.; Tibshirani, R.; Friedman, J. H. *The elements of statistical learning: data mining, inference, and prediction, 2nd Edition*; Springer series in statistics; Springer, 2009.

[61] Efron, B.; Tibshirani, R. Improvements on Cross-Validation: The 632+ Bootstrap Method. *Journal of the American Statistical Association* **1997**, *92*, 548–560.

[62] Snoek, J.; Larochelle, H.; Adams, R. P. In *Advances in Neural Information Processing Systems 25*; Pereira, F., Burges, C. J. C., Bottou, L., Weinberger, K. Q., Eds.; Curran Associates, Inc., 2012; pp 2951–2959.

[63] Egan, W. J.; Merz, K. M.; Baldwin, J. J. Prediction of Drug Absorption Using Multivariate Statistics. *Journal of Medicinal Chemistry* **2000**, *43*, 3867–3877.

[64] Livingstone, D. J.; Salt, D. W. Judging the Significance of Multiple Linear Regression Models. *Journal of Medicinal Chemistry* **2005**, *48*, 661–663.

[65] Guo, J.-Y.; Minko, Y.; Santiago, C. B.; Sigman, M. S. Developing Comprehensive Computational Parameter Sets To Describe the Performance of Pyridine-Oxazoline and Related Ligands. *ACS Catalysis* **2017**, *7*, 4144–4151.

[66] Robinson, S. G.; Yan, Y.; Hendriks, K. H.; Sanford, M. S.; Sigman, M. S. Developing a Predictive Solubility Model for Monomeric and Oligomeric Cyclopropenium-Based Flow Battery Catholytes. *Journal of the American Chemical Society* **2019**, *141*, 10171–10176.

[67] Santiago, C. B.; Guo, J.-Y.; Sigman, M. S. Predictive and mechanistic multivariate linear regression models for reaction development. *Chemical Science* **2018**, *9*, 2398–2412.

[68] Ghiringhelli, L. M.; Vybiral, J.; Levchenko, S. V.; Draxl, C.; Scheffler, M. Big Data of Materials Science: Critical Role of the Descriptor. *Physical Review Letters* **2015**, *114*, 105503.

[69] Janet, J. P.; Gani, T. Z. H.; Steeves, A. H.; Ioannidis, E. I.; Kulik, H. J. Leveraging Cheminformatics Strategies for Inorganic Discovery: Application to Redox Potential Design. *Industrial & Engineering Chemistry Research* **2017**, *56*, 4898–4910.

[70] Amat, L.; Carbó-Dorca, R.; Ponec, R. Simple Linear QSAR Models Based on Quantum Similarity Measures. *Journal of Medicinal Chemistry* **1999**, *42*, 5169–5180.

[71] Kaneko, H. Discussion on Regression Methods Based on Ensemble Learning and Applicability Domains of Linear Submodels. *Journal of Chemical Information and Modeling* **2018**, *58*, 480–489.

[72] Trefethen, L. N.; Bau, D. *Numerical Linear Algebra*; SIAM, 1997.

[73] Hawkins, D. M.; Basak, S. C.; Shi, X. QSAR with Few Compounds and Many Features. *Journal of Chemical Information and Computer Sciences* **2001**, *41*, 663–670.

[74] Williams, C. K. I. Learning Kernel Classifiers. *Journal of the American Statistical Association* **2003**, *98*, 489–490.

[75] Hansen, K.; Montavon, G.; Biegler, F.; Fazli, S.; Rupp, M.; Scheffler, M.; von Lilienfeld, O. A.; Tkatchenko, A.; Müller, K.-R. Assessment and Validation of Machine Learning Methods for Predicting Molecular Atomization Energies. *Journal of Chemical Theory and Computation* **2013**, *9*, 3404–3419.

[76] Bartók, A. P.; De, S.; Poelking, C.; Bernstein, N.; Kermode, J. R.; Csányi, G.; Ceriotti, M. Machine learning unifies the modeling of materials and molecules. *Science Advances* **2017**, *3*.

[77] Wilkins, D. M.; Grisafi, A.; Yang, Y.; Lao, K. U.; DiStasio, R. A.; Ceriotti, M. Accurate molecular polarizabilities with coupled cluster theory and machine learning. *Proceedings of the National Academy of Sciences* **2019**, *116*, 3401–3406.

[78] Bogojeski, M.; Brockherde, F.; Vogt-Maranto, L.; Li, L.; Tuckerman, M. E.; Burke, K.; Müller, K.-R. Efficient prediction of 3D electron densities using machine learning. *arXiv e-prints* **2018**, arXiv:1811.06255.

[79] Noh, J.; Back, S.; Kim, J.; Jung, Y. Active learning with non-ab initio input features toward efficient CO2 reduction catalysts. *Chemical Science* **2018**, *9*, 5152–5159.

[80] Meyer, B.; Sawatlon, B.; Heinen, S.; von Lilienfeld, O. A.; Corminboeuf, C. Machine learning meets volcano plots: computational discovery of cross-coupling catalysts. *Chemical Science* **2018**, *9*, 7069–7077.

[81] Janet, J. P.; Kulik, H. J. Resolving Transition Metal Chemical Space: Feature Selection for Machine Learning and Structure–Property Relationships. *The Journal of Physical Chemistry A* **2017**, *121*, 8939–8954.

[82] Schölkopf, B.; Herbrich, R.; Smola, A. J. A Generalized Representer Theorem. Computational Learning Theory. Berlin, Heidelberg, 2001; pp 416–426.

[83] Scholkopf, B.; Smola, A. J. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*; MIT Press: Cambridge, MA, USA, 2001.

[84] Hofmann, T.; Schölkopf, B.; Smola, A. J. Kernel Methods in Machine Learning. *The Annals of Statistics* **2008**, *36*, 1171–1220.

[85] Welborn, M.; Cheng, L.; Miller, T. F. Transferability in Machine Learning for Electronic Structure via the Molecular Orbital Basis. *Journal of Chemical Theory and Computation* **2018**, *14*, 4772–4779.

[86] Huang, B.; Anatole von Lilienfeld, O. The DNA of chemistry: Scalable quantum machine learning with amons. *arXiv e-prints* **2017**, arXiv:1707.04146.

[87] Faber, F. A.; Christensen, A. S.; Huang, B.; von Lilienfeld, O. A. Alchemical and structural distribution based representation for universal quantum machine learning. *The Journal of Chemical Physics* **2018**, *148*, 241717.

[88] Bartók, A. P.; Payne, M. C.; Kondor, R.; Csányi, G. Gaussian Approximation Potentials: The Accuracy of Quantum Mechanics, without the Electrons. *Physical Review Letters* **2010**, *104*, 136403.

[89] Bartók, A. P.; Kondor, R.; Csányi, G. On representing chemical environments. *Physical Review B* **2013**, *87*, 184115.

[90] Szlachta, W. J.; Bartók, A. P.; Csányi, G. Accuracy and transferability of Gaussian approximation potential models for tungsten. *Physical Review B* **2014**, *90*, 104108.

[91] Imbalzano, G.; Anelli, A.; Giofré, D.; Klees, S.; Behler, J.; Ceriotti, M. Automatic selection of atomic fingerprints and reference configurations for machine-learning potentials. *The Journal of Chemical Physics* **2018**, *148*, 241730.

[92] Nguyen, T. T.; Székely, E.; Imbalzano, G.; Behler, J.; Csányi, G.; Ceriotti, M.; Götz, A. W.; Paesani, F. Comparison of permutationally invariant polynomials, neural networks, and Gaussian approximation potentials in representing water interactions through many-body expansions. *The Journal of Chemical Physics* **2018**, *148*, 241725.

[93] Fujikake, S.; Deringer, V. L.; Lee, T. H.; Krynski, M.; Elliott, S. R.; Csányi, G. Gaussian approximation potential modeling of lithium intercalation in carbon nanostructures. *The Journal of Chemical Physics* **2018**, *148*, 241714.

[94] Denzel, A.; Kästner, J. Gaussian process regression for geometry optimization. *The Journal of Chemical Physics* **2018**, *148*, 094114.

[95] Proppe, J.; Gugler, S.; Reiher, M. Gaussian Process-Based Refinement of Dispersion Corrections. *arXiv e-prints* **2019**, arXiv:1906.09342.

[96] Simm, G. N.; Reiher, M. Error-Controlled Exploration of Chemical Reaction Networks with Gaussian Processes. *Journal of Chemical Theory and Computation* **2018**, *14*, 5238–5248.

[97] Pilania, G.; Gubernatis, J. E.; Lookman, T. Multi-fidelity machine learning models for accurate bandgap predictions of solids. *Computational Materials Science* **2017**, *129*, 156–163.

[98] Williams, C. K. I.; Rasmussen, C. E. *Gaussian processes for machine learning*; MIT press Cambridge, MA, 2006.

[99] Jones, D. R.; Schonlau, M.; Welch, W. J. Efficient Global Optimization of Expensive Black-Box Functions. *Journal of Global Optimization* **1998**, *13*, 455–492.

[100] Carr, S.; Garnett, R.; Lo, C. BASC: Applying Bayesian Optimization to the Search for Global Minima on Potential Energy Surfaces. Proceedings of The 33$^{\text{rd}}$ International Conference on Machine Learning. New York, New York, USA, 2016; pp 898–907.

[101] Seko, A.; Hayashi, H.; Nakayama, K.; Takahashi, A.; Tanaka, I. Representation of compounds for machine-learning prediction of physical properties. *Physical Review B* **2017**, *95*, 144110.

[102] Janet, J. P.; Kulik, H. J. Predicting electronic structure properties of transition metal complexes with neural networks. *Chemical Science* **2017**, *8*, 5137–5152.

[103] Teixeira, A. L.; Leal, J. P.; Falcao, A. O. Random forests for feature selection in QSPR models - An application for predicting standard enthalpy of formation of hydrocarbons. *Journal of Cheminformatics* **2013**, *5*, 9.

[104] Cano, G.; Garcia-Rodriguez, J.; Garcia-Garcia, A.; Perez-Sanchez, H.; Benediktsson, J. A.; Thapa, A.; Barr, A. Automatic selection of molecular descriptors using random forest: Application to drug discovery. *Expert Systems with Applications* **2017**, *72*, 151–159.

[105] Zahrt, A. F.; Henle, J. J.; Rose, B. T.; Wang, Y.; Darrow, W. T.; Denmark, S. E. Prediction of higher-selectivity catalysts by computer-driven workflow and machine learning. *Science* **2019**, *363*.

[106] Palmer, D. S.; O'Boyle, N. M.; Glen, R. C.; Mitchell, J. B. O. Random Forest Models To Predict Aqueous Solubility. *Journal of Chemical Information and Modeling* **2007**, *47*, 150–158.

[107] Himmetoglu, B. Tree based machine learning framework for predicting ground state energies of molecules. *The Journal of Chemical Physics* **2016**, *145*, 134101.

[108] Ahneman, D. T.; Estrada, J. G.; Lin, S.; Dreher, S. D.; Doyle, A. G. Predicting reaction performance in C–N cross-coupling using machine learning. *Science* **2018**, *360*, 186–190.

[109] Svetnik, V.; Liaw, A.; Tong, C.; Culberson, J. C.; Sheridan, R. P.; Feuston, B. P. Random Forest: A Classification and Regression Tool for Compound Classification and QSAR Modeling. *Journal of Chemical Information and Computer Sciences* **2003**, *43*, 1947–1958.

[110] Lewis, R. A.; Wood, D. Modern 2D QSAR for drug discovery. *Wiley Interdisciplinary Reviews: Computational Molecular Science* **2014**, *4*, 505–522.

[111] Riddick, G.; Song, H.; Ahn, S.; Walling, J.; Borges-Rivera, D.; Zhang, W.; Fine, H. A. Predicting in vitro drug sensitivity using Random Forests. *Bioinformatics* **2010**, *27*, 220–224.

[112] Boulesteix, A.-L.; Janitza, S.; Kruppa, J.; König, I. R. Overview of random forest methodology and practical guidance with emphasis on computational biology and bioinformatics. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* **2012**, *2*, 493–507.

[113] Bleiziffer, P.; Schaller, K.; Riniker, S. Machine Learning of Partial Charges Derived from High-Quality Quantum-Mechanical Calculations. *Journal of Chemical Information and Modeling* **2018**, *58*, 579–590.

[114] Panapitiya, G.; Avendaño-Franco, G.; Ren, P.; Wen, X.; Li, Y.; Lewis, J. P. Machine-Learning Prediction of CO Adsorption in Thiolated, Ag-Alloyed Au Nanoclusters. *Journal of the American Chemical Society* **2018**, *140*, 17508–17514.

[115] Li, H.; Leung, K.-S.; Wong, M.-H.; Ballester, P. J. Improving AutoDock Vina Using Random Forest: The Growing Accuracy of Binding Affinity Prediction by the Effective Exploitation of Larger Data Sets. *Molecular Informatics* **2015**, *34*, 115–126.

[116] Breiman, L. *Classification and regression trees*; Routledge, 1984.

[117] Breiman, L. Bagging predictors. *Machine Learning* **1996**, *24*, 123–140.

[118] Ho, T. K. Random decision forests. Proceedings of 3rd International Conference on Document Analysis and Recognition. 1995; pp 278–282.

[119] Breiman, L. Random Forests. *Machine Learning* **2001**, *45*, 5–32.

[120] Polishchuk, P. Interpretation of Quantitative Structure–Activity Relationship Models: Past, Present, and Future. *Journal of Chemical Information and Modeling* **2017**, *57*, 2618–2639.

[121] Huang, B.; von Lilienfeld, O. A. Communication: Understanding molecular representations in machine learning: The role of uniqueness and target similarity. *The Journal of Chemical Physics* **2016**, *145*, 161102.

[122] Willatt, M. J.; Musil, F.; Ceriotti, M. Feature optimization for atomistic machine learning yields a data-driven construction of the periodic table of the elements. *Physical Chemistry Chemical Physics* **2018**, *20*, 29661–29668.

[123] Duan, C.; Janet, J. P.; Liu, F.; Nandy, A.; Kulik, H. J. Learning from Failure: Predicting Electronic Structure Calculation Outcomes with Machine Learning Models. *Journal of Chemical Theory and Computation* **2019**, *15*, 2331–2345.

[124] Lima, F. H. B.; Zhang, J.; Shao, M. H.; Sasaki, K.; Vukmirovic, M. B.; Ticianelli, E. A.; Adzic, R. R. Catalytic Activity–d-Band Center Correlation for the O2 Reduction Reaction on Platinum in Alkaline Solutions. *The Journal of Physical Chemistry C* **2007**, *111*, 404–410.

[125] Jin, W.; Coley, C.; Barzilay, R.; Jaakkola, T. In *Advances in Neural Information Processing Systems 30*; Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R., Eds.; Curran Associates, Inc., 2017; pp 2607–2616.

[126] Fernandez, M.; Boyd, P. G.; Daff, T. D.; Aghaji, M. Z.; Woo, T. K. Rapid and Accurate Machine Learning Recognition of High Performing Metal Organic Frameworks for CO2 Capture. *The Journal of Physical Chemistry Letters* **2014**, *5*, 3056–3060.

[127] Landrum, G. RDKit: Open-source cheminformatics. `http://www.rdkit.org`.

[128] O'Boyle, N. M.; Banck, M.; James, C. A.; Morley, C.; Vandermeersch, T.; Hutchison, G. R. Open Babel: An open chemical toolbox. *Journal of Cheminformatics* **2011**, *3*, 33.

[129] Lu, S.; Zhou, Q.; Ouyang, Y.; Guo, Y.; Li, Q.; Wang, J. Accelerated discovery of stable lead-free hybrid organic-inorganic perovskites via machine learning. *Nature Communications* **2018**, *9*, 3405.

[130] Calle-Vallejo, F.; Martínez, J. I.; García-Lastra, J. M.; Sautet, P.; Loffreda, D. Fast Prediction of Adsorption Properties for Platinum Nanocatalysts with Generalized Coordination Numbers. *Angewandte Chemie International Edition* **2014**, *53*, 8316–8319.

[131] Randić, M. On Characterization of Molecular Branching. *Journal of the American Chemical Society* **1975**, *97*, 6609–6615.

[132] Wiener, H. Correlation of Heats of Isomerization, and Differences in Heats of Vaporization of Isomers, Among the Paraffin Hydrocarbons. *Journal of the American Chemical Society* **1947**, *69*, 2636–2638.

[133] Kier, L. B.; Hall, L. H.; Murray, W. J.; Randić, M. Molecular Connectivity I: Relationship to Nonspecific Local Anesthesia. *Journal of Pharmaceutical Sciences* **1975**, *64*, 1971–1974.

[134] Kier, L. B.; Murray, W. J.; Randić, M.; Hall, L. H. Molecular Connectivity V: Connectivity Series Concept Applied to Density. *Journal of Pharmaceutical Sciences* **1976**, *65*, 1226–1230.

[135] Kier, L. B.; Hall, L. H. Molecular Connectivity VII: Specific Treatment of Heteroatoms. *Journal of Pharmaceutical Sciences* **1976**, *65*, 1806–1809.

[136] Broto, P.; Moreau, G.; Vandycke, C. Molecular structures: perception, autocorrelation descriptor and SAR studies. *European Journal of Medicinal Chemistry* **1984**, *19*, 71–78.

[137] Virshup, A. M.; Contreras-García, J.; Wipf, P.; Yang, W.; Beratan, D. N. Stochastic Voyages into Uncharted Chemical Space Produce a Representative Library of All Possible Drug-Like Compounds. *Journal of the American Chemical Society* **2013**, *135*, 7296–7303.

[138] Duvenaud, D. K.; Maclaurin, D.; Aguilera-Iparraguirre, J.; Gómez-Bombarelli, R.; Hirzel, T.; Aspuru-Guzik, A.; Adams, R. P. *Advances in Neural Information Processing Systems 28:*; Curran Associates, Inc., 2015; pp 2224–2232.

[139] Weininger, D. SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules. *Journal of Chemical Information and Computer Sciences* **1988**, *28*, 31–36.

[140] Schwaller, P.; Laino, T.; Gaudin, T.; Bolgar, P.; Bekas, C.; Lee, A. A. Molecular Transformer for Chemical Reaction Prediction and Uncertainty Estimation. *arXiv e-prints* **2018**, *abs/1811.02633*.

[141] Goh, G. B.; Hodas, N. O.; Siegel, C.; Vishnu, A. SMILES2Vec: An Interpretable General-Purpose Deep Neural Network for Predicting Chemical Properties. *arXiv e-prints* **2017**, *abs/1712.02034*.

[142] Morgan, H. L. The Generation of a Unique Machine Description for Chemical Structures-A Technique Developed at Chemical Abstracts Service. *Journal of Chemical Documentation* **1965**, *5*, 107–113.

[143] Thomas, N.; Smidt, T.; Kearnes, S.; Yang, L.; Li, L.; Kohlhoff, K.; Riley, P. Tensor field networks: Rotation- and translation-equivariant neural networks for 3D point clouds. *arXiv e-prints* **2018**, arXiv:1802.08219.

[144] Montavon, G.; Hansen, K.; Fazli, S.; Rupp, M.; Biegler, F.; Ziehe, A.; Tkatchenko, A.; von Lilienfeld, A.; Müller, K. *Advances in Neural Information Processing Systems 25*; Curran Associates, Inc., 2012; pp 449–457.

[145] Hansen, K.; Biegler, F.; Ramakrishnan, R.; Pronobis, W.; von Lilienfeld, O. A.; Müller, K.-R.; Tkatchenko, A. Machine Learning Predictions of Molecular Properties: Accurate Many-Body Potentials and Nonlocality in Chemical Space. *The Journal of Physical Chemistry Letters* **2015**, *6*, 2326–2331.

[146] Yao, K.; Herr, J. E.; Toth, D.; Mckintyre, R.; Parkhill, J. The TensorMol-0.1 model chemistry: a neural network augmented with long-range physics. *Chemical Science* **2018**, *9*, 2261–2269.

[147] Karelson, M.; Lobanov, V. S.; Katritzky, A. R. Quantum-Chemical Descriptors in QSAR/QSPR Studies. *Chemical Reviews* **1996**, *96*, 1027–1044.

[148] Snyder, J. C.; Rupp, M.; Hansen, K.; Müller, K.-R.; Burke, K. Finding Density Functionals with Machine Learning. *Physical Review Letters* **2012**, *108*, 253002.

[149] Lei, X.; Medford, A. J. Design and analysis of machine learning exchange-correlation functionals via rotationally invariant convolutional descriptors. *Physical Review Materials* **2019**, *3*, 063801.

[150] Yao, K.; Parkhill, J. Kinetic Energy of Hydrocarbons as a Function of Electron Density and Convolutional Neural Networks. *Journal of Chemical Theory and Computation* **2016**, *12*, 1139–1147.

[151] Saeys, Y.; Inza, I.; Larrañaga, P. A review of feature selection techniques in bioinformatics. *Bioinformatics* **2007**, *23*, 2507–2517.

[152] Eklund, M.; Norinder, U.; Boyer, S.; Carlsson, L. Choosing Feature Selection and Learning Algorithms in QSAR. *Journal of Chemical Information and Modeling* **2014**, *54*, 837–843.

[153] Rice, J. *Mathematical statistics and data analysis*; Thomson/Brooks/Cole, 2013.

[154] Hua, J.; Tembe, W. D.; Dougherty, E. R. Performance of feature-selection methods in the classification of high-dimension data. *Pattern Recognition* **2009**, *42*, 409–424.

[155] Kelley, B. DescriptaStorus. `https://github.com/bp-kelley/descriptastorus`.

[156] Dessì, N.; Pascariello, E.; Pes, B. A Comparative Analysis of Biomarker Selection Techniques. *BioMed Research International* **2013**, *2013*.

[157] Kira, K.; Rendell, L. A. In *Machine Learning Proceedings 1992*; Sleeman, D., Edwards, P., Eds.; Morgan Kaufmann: San Francisco, CA, 1992; pp 249–256.

[158] Robnik-Šikonja, M.; Kononenko, I. Theoretical and Empirical Analysis of ReliefF and RReliefF. *Machine Learning* **2003**, *53*, 23–69.

[159] Wang, X.; Zhang, Y.; Wang, J. Prediction of Protein Structural Class Based on ReliefF-SVM. *Letters in Organic Chemistry* **2017**, *14*, 696–702.

[160] Furnival, G. M.; Wilson, R. W. Regressions by leaps and bounds. *Technometrics* **1974**, *16*, 499–511.

[161] Jović, A.; Brkić, K.; Bogunović, N. A review of feature selection methods with applications. 2015 38$^{\text{th}}$ International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). 2015; pp 1200–1205.

[162] Dorigo, M.; Di Caro, G. Ant colony optimization: a new meta-heuristic. Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406). 1999; pp 1470–1477 Vol. 2.

[163] Chen, Y.; Miao, D.; Wang, R. A rough set approach to feature selection based on ant colony optimization. *Pattern Recognition Letters* **2010**, *31*, 226–233.

[164] Yang, J.; Honavar, V. *Feature Extraction, Construction and Selection: A Data Mining Perspective*; Springer US: Boston, MA, 1998; pp 117–136.

[165] Cho, S. J.; Hermsmeier, M. A. Genetic Algorithm Guided Selection: Variable Selection and Subset Selection. *Journal of Chemical Information and Computer Sciences* **2002**, *42*, 927–936.

[166] Leardi, R.; Boggia, R.; Terrile, M. Genetic algorithms as a strategy for feature selection. *Journal of Chemometrics* **1992**, *6*, 267–281.

[167] Petricoin, E. F.; Ardekani, A. M.; Hitt, B. A.; Levine, P. J.; Fusaro, V. A.; Steinberg, S. M.; Mills, G. B.; Simone, C.; Fishman, D. A.; Kohn, E. C.; Liotta, L. A. Use of proteomic patterns in serum to identify ovarian cancer. *The Lancet* **2002**, *359*, 572–577.

[168] Leardi, R. Genetic algorithms in chemometrics and chemistry: a review. *Journal of Chemometrics* **2001**, *15*, 559–569.

[169] Mol, C. D.; Vito, E. D.; Rosasco, L. Elastic-net regularization in learning theory. *Journal of Complexity* **2009**, *25*, 201–230.

[170] Santosa, F.; Symes, W. Linear Inversion of Band-Limited Reflection Seismograms. *SIAM Journal on Scientific and Statistical Computing* **1986**, *7*, 1307–1330.

[171] Zhang, Y.; Guo, W.; Ray, S. On the Consistency of Feature Selection With Lasso for Non-linear Targets. Proceedings of The 33$^{\text{rd}}$ International Conference on Machine Learning. New York, New York, USA, 2016; pp 183–191.

[172] Wu, T. T.; Lange, K. Coordinate descent algorithms for lasso penalized regression. *The Annals of Applied Statistics* **2008**, *2*, 224–244.

[173] Bottou, L. *Neural Networks: Tricks of the Trade: Second Edition*; Springer Berlin Heidelberg: Berlin, Heidelberg, 2012; pp 421–436.

[174] Zou, H.; Hastie, T. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* **2005**, *67*, 301–320.

[175] Jin, B.; Lorenz, D. A.; Schiffler, S. Elastic-net regularization: error estimates and active set methods. *Inverse Problems* **2009**, *25*, 115022.

[176] Yang, K.; Swanson, K.; Jin, W.; Coley, C.; Eiden, P.; Gao, H.; Guzman-Perez, A.; Hopper, T.; Kelley, B.; Mathea, M.; Palmer, A.; Settels, V.; Jaakkola, T.; Jensen, K.; Barzilay, R. Are Learned Molecular Representations Ready For Prime Time? *arXiv e-prints* **2019**, arXiv:1904.01561.

[177] Ghiringhelli, L. M.; Vybiral, J.; Ahmetcik, E.; Ouyang, R.; Levchenko, S. V.; Draxl, C.; Scheffler, M. Learning physical descriptors for materials science by compressed sensing. *New Journal of Physics* **2017**, *19*, 023017.

[178] Genuer, R.; Poggi, J.-M.; Tuleau-Malot, C. Variable selection using random forests. *Pattern Recognition Letters* **2010**, *31*, 2225–2236.

[179] Strobl, C.; Boulesteix, A.-L.; Zeileis, A.; Hothorn, T. Bias in random forest variable importance measures: Illustrations, sources and a solution. *BMC Bioinformatics* **2007**, *8*, 25.

[180] Liaw, A.; Wiener, M. Classification and Regression by randomForest. *R News* **2002**, *2*, 18–22.

[181] Team, R. C. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing: Vienna, Austria, 2014.

[182] Nandy, A.; Duan, C.; Janet, J. P.; Gugler, S.; Kulik, H. J. Strategies and Software for Machine Learning Accelerated Discovery in Transition Metal Chemistry. *Industrial & Engineering Chemistry Research* **2018**, *57*, 13973–13986.

[183] Pearson, K. LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* **1901**, *2*, 559–572.

[184] Bender, A.; Jenkins, J. L.; Scheiber, J.; Sukuru, S. C. K.; Glick, M.; Davies, J. W. How Similar Are Similarity Searching Methods? A Principal Component Analysis of Molecular Descriptor Space. *Journal of Chemical Information and Modeling* **2009**, *49*, 108–119.

[185] Iovanac, N. C.; Savoie, B. M. Improved Chemical Prediction from Scarce Data Sets via Latent Space Enrichment. *The Journal of Physical Chemistry A* **2019**, *123*, 4295–4302.

[186] Tenenbaum, J. B.; Silva, V. d.; Langford, J. C. A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science* **2000**, *290*, 2319–2323.

[187] Maaten, L. v. d.; Hinton, G. Visualizing data using t-SNE. *Journal of machine learning research* **2008**, *9*, 2579–2605.

[188] McInnes, L.; Healy, J. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. *arXiv e-prints* **2018**, *abs/1802.03426*.

[189] Behler, J. Perspective: Machine learning potentials for atomistic simulations. *The Journal of Chemical Physics* **2016**, *145*, 170901.

[190] Schmidhuber, J. Deep learning in neural networks: An overview. *Neural Networks* **2015**, *61*, 85–117.

[191] Ivakhnenko, A. G.; Lapa, V. G. *Cybernetics and forecasting techniques*; Modern analytic and computational methods in science and mathematics; American Elsevier Pub. Co., 1967.

[192] Ivakhenko, A. G. The Group Method of Data of Handling ; A rival of the method of stochastic approximation. *Soviet Automatic Control* **1968**, *13*, 43–55.

[193] Rosenblatt, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review* **1958**, *65*, 386–408.

[194] Kleene, S. C. In *Automata Studies. (AM-34)*; Shannon, C. E., McCarthy, J., Eds.; Princeton University Press: Princeton, 1956; pp 3–42.

[195] Hebb, D. O. *The Organization of Behavior: A Neuropsychological Theory*; Taylor & Francis, 1949.

[196] McCulloch, W. S.; Pitts, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics* **1943**, *5*, 115–133.

[197] Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; Sutskever, I. Language models are unsupervised multitask learners. *OpenAI* **2019**, *1*.

[198] LeCun, Y.; Bengio, Y.; Hinton, G. E. Deep learning. *Nature* **2015**, *521*, 436–444.

[199] Krizhevsky, A.; Sutskever, I.; Hinton, G. E. In *Advances in Neural Information Processing Systems 25*; Pereira, F., Burges, C. J. C., Bottou, L., Weinberger, K. Q., Eds.; Curran Associates, Inc., 2012; pp 1097–1105.

[200] Deng, J.; Dong, W.; Socher, R.; Li, L.-j.; Li, K.; Fei-fei, L. Imagenet: A large-scale hierarchical image database. 2009 IEEE conference on computer vision and pattern recognition. 2009.

[201] Graves, A.; Schmidhuber, J. In *Advances in Neural Information Processing Systems 21*; Koller, D., Schuurmans, D., Bengio, Y., Bottou, L., Eds.; Curran Associates, Inc., 2009; pp 545–552.

[202] Graves, A.; Liwicki, M.; Fernández, S.; Bertolami, R.; Bunke, H.; Schmidhuber, J. A Novel Connectionist System for Unconstrained Handwriting Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **2009**, *31*, 855–868.

[203] Mater, A. C.; Coote, M. L. Deep Learning in Chemistry. *Journal of Chemical Information and Modeling* **2019**, *59*, 2545–2559.

[204] Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press, 2016; `http://www.deeplearningbook.org`.

[205] Hahnloser, R. H.; Sarpeshkar, R.; Mahowald, M. A.; Douglas, R. J.; Seung, H. S. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature* **2000**, *405*, 947–51.

[206] Glorot, X.; Bordes, A.; Bengio, Y. Deep Sparse Rectifier Neural Networks. Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics. Fort Lauderdale, FL, USA, 2011; pp 315–323.

[207] He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016. 2016; pp 770–778.

[208] Janet, J. P.; Duan, C.; Yang, T.; Nandy, A.; Kulik, H. J. A quantitative uncertainty metric controls error in neural network-driven chemical discovery. *Chemical Science* **2019**, *10*, 7913–7922.

[209] Gómez-Bombarelli, R.; Wei, J. N.; Duvenaud, D.; Hernández-Lobato, J. M.; Sánchez-Lengeling, B.; Sheberla, D.; Aguilera-Iparraguirre, J.; Hirzel, T. D.; Adams, R. P.; Aspuru-Guzik, A. Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules. *ACS Central Science* **2018**, *4*, 268–276.

[210] St. John, P. C.; Phillips, C.; Kemper, T. W.; Wilson, A. N.; Guan, Y.; Crowley, M. F.; Nimlos, M. R.; Larsen, R. E. Message-passing neural networks for high-throughput polymer screening. *The Journal of Chemical Physics* **2019**, *150*, 234111.

[211] Zubatyuk, R.; Smith, J. S.; Leszczynski, J.; Isayev, O. Accurate and transferable multitask prediction of chemical properties with an atoms-in-molecules neural network. *Science Advances* **2019**, *5*.

[212] Mayr, A.; Klambauer, G.; Unterthiner, T.; Hochreiter, S. DeepTox: Toxicity Prediction using Deep Learning. *Frontiers in Environmental Science* **2016**, *3*, 80.

[213] Hornik, K. Approximation capabilities of multilayer feedforward networks. *Neural Networks* **1991**, *4*, 251–257.

[214] Csáji, B. C. Approximation with artificial neural networks. M.Sc. thesis, Faculty of Sciences, Etvs Lornd University, Hungary, 2001.

[215] Glorot, X.; Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010. 2010; pp 249–256.

[216] Auer, P.; Herbster, M.; Warmuth, M. K. In *Advances in Neural Information Processing Systems 8*; Touretzky, D. S., Mozer, M. C., Hasselmo, M. E., Eds.; MIT Press, 1996; pp 316–322.

[217] Dauphin, Y.; Pascanu, R.; Gulcehre, C.; Cho, K.; Ganguli, S.; Bengio, Y. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *arXiv e-prints* **2014**, arXiv:1406.2572.

[218] Montavon, G., Orr, G. B., Müller, K., Eds. *Neural Networks: Tricks of the Trade - Second Edition*; Lecture Notes in Computer Science; Springer, 2012; Vol. 7700.

[219] Kingma, D. P.; Ba, J. Adam: A Method for Stochastic Optimization. 3$^{\text{rd}}$ International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings. 2015.

[220] Zeiler, M. D. ADADELTA: An Adaptive Learning Rate Method. *arXiv e-prints* **2012**, *abs/1212.5701*.

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

[222] Caruana, R.; Lawrence, S.; Giles, C. L. *Advances in Neural Information Processing Systems 13*; Curran Associates, Inc., 2000; pp 402–408.

[223] LeCun, Y.; Boser, B.; Denker, J. S.; Henderson, D.; Howard, R. E.; Hubbard, W.; Jackel, L. D. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation* **1989**, *1*, 541–551.

[224] Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* **1998**, *86*, 2278–2324.

[225] Maturana, D.; Scherer, S. VoxNet: A 3D Convolutional Neural Network for real-time object recognition. 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). 2015; pp 922–928.

[226] Milletari, F.; Navab, N.; Ahmadi, S. V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation. 2016 Fourth International Conference on 3D Vision (3DV). 2016; pp 565–571.

[227] Ji, S.; Xu, W.; Yang, M.; Yu, K. 3D Convolutional Neural Networks for Human Action Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **2013**, *35*, 221–231.

[228] Kiranyaz, S.; Ince, T.; Gabbouj, M. Real-Time Patient-Specific ECG Classification by 1-D Convolutional Neural Networks. *IEEE Transactions on Biomedical Engineering* **2016**, *63*, 664–675.

[229] Ragoza, M.; Hochuli, J.; Idrobo, E.; Sunseri, J.; Koes, D. R. Protein–Ligand Scoring with Convolutional Neural Networks. *Journal of Chemical Information and Modeling* **2017**, *57*, 942–957.

[230] Jørgensen, M. S.; Mortensen, H. L.; Meldgaard, S. A.; Kolsbjerg, E. L.; Jacobsen, T. L.; Sørensen, K. H.; Hammer, B. Atomistic structure learning. *The Journal of Chemical Physics* **2019**, *151*, 054111.

[231] Goh, G. B.; Siegel, C.; Vishnu, A.; Hodas, N. O.; Baker, N. Chemception: A Deep Neural Network with Minimal Chemistry Knowledge Matches the Performance of Expert-developed QSAR/QSPR Models. *arXiv e-prints* **2017**, *abs/1706.06689*.

[232] Staker, J.; Marshall, K.; Abel, R.; McQuaw, C. M. Molecular Structure Extraction from Documents Using Deep Learning. *Journal of Chemical Information and Modeling* **2019**, *59*, 1017–1029.

[233] Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; Dahl, G. E. Neural Message Passing for Quantum Chemistry. Proceedings of the 34$^{\text{th}}$ International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017. 2017; pp 1263–1272.

[234] Coley, C. W.; Barzilay, R.; Green, W. H.; Jaakkola, T. S.; Jensen, K. F. Convolutional Embedding of Attributed Molecular Graphs for Physical Property Prediction. *Journal of Chemical Information and Modeling* **2017**, *57*, 1757–1772.

[235] Altae-Tran, H.; Ramsundar, B.; Pappu, A. S.; Pande, V. Low Data Drug Discovery with One-Shot Learning. *ACS Central Science* **2017**, *3*, 283–293.

[236] Jørgensen, P. B.; Jacobsen, K. W.; Schmidt, M. N. Neural Message Passing with Edge Updates for Predicting Properties of Molecules and Materials. *arXiv e-prints* **2018**, *abs/1806.03146*.

[237] Kearnes, S.; McCloskey, K.; Berndl, M.; Pande, V.; Riley, P. Molecular graph convolutions: moving beyond fingerprints. *Journal of Computer-Aided Molecular Design* **2016**, *30*, 595–608.

[238] Matlock, M. K.; Dang, N. L.; Swamidass, S. J. Learning a Local-Variable Model of Aromatic and Conjugated Systems. *ACS Central Science* **2018**, *4*, 52–62.

[239] Dai, H.; Dai, B.; Song, L. Discriminative Embeddings of Latent Variable Models for Structured Data. Proceedings of the 33$^{\text{nd}}$ International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016. 2016; pp 2702–2711.

[240] Feinberg, E. N.; Sur, D.; Wu, Z.; Husic, B. E.; Mai, H.; Li, Y.; Sun, S.; Yang, J.; Ramsundar, B.; Pande, V. S. PotentialNet for Molecular Property Prediction. *ACS Central Science* **2018**, *4*, 1520–1530.

[241] Schütt, K.; Kindermans, P.; Felix, H. E. S.; Chmiela, S.; Tkatchenko, A.; Müller, K. *Advances in Neural Information Processing Systems 30*; Curran Associates, Inc., 2017; pp 991–1001.

[242] Lubbers, N.; Smith, J. S.; Barros, K. Hierarchical modeling of molecular energies using a deep neural network. *The Journal of Chemical Physics* **2018**, *148*, 241715.

[243] Xie, T.; Grossman, J. C. Hierarchical visualization of materials space with graph convolutional neural networks. *The Journal of Chemical Physics* **2018**, *149*, 174111.

[244] Hopfield, J. J. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences* **1982**, *79*, 2554–2558.

[245] Cho, K.; van Merrienboer, B.; Gülçehre, Ç.; Bahdanau, D.; Bougares, F.; Schwenk, H.; Bengio, Y. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL. 2014; pp 1724–1734.

[246] Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Computation* **1997**, *9*, 1735–1780.

[247] Gers, F. A.; Schmidhuber, J.; Cummins, F. A. Learning to Forget: Continual Prediction with LSTM. *Neural Computation* **2000**, *12*, 2451–2471.

[248] Devlin, J.; Chang, M.; Lee, K.; Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers). 2019; pp 4171–4186.

[249] Schwaller, P.; Gaudin, T.; Lányi, D.; Bekas, C.; Laino, T. Found in Translation: predicting outcomes of complex organic chemistry reactions using neural sequence-to-sequence models. *Chemical Science* **2018**, *9*, 6091–6098.

[250] Zhang, Y.; Qiao, S.; Ji, S.; Li, Y. DeepSite: bidirectional LSTM and CNN models for predicting DNA–protein binding. *International Journal of Machine Learning and Cybernetics* **2019**,

[251] Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; Polosukhin, I. *Advances in Neural Information Processing Systems 30*; Curran Associates, Inc., 2017; pp 5998–6008.

[252] Peters, M. E.; Neumann, M.; Iyyer, M.; Gardner, M.; Clark, C.; Lee, K.; Zettlemoyer, L. Deep Contextualized Word Representations. Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers). 2018; pp 2227–2237.

[253] Luong, T.; Pham, H.; Manning, C. D. Effective Approaches to Attention-based Neural Machine Translation. Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015. 2015; pp 1412–1421.

[254] Smith, J. S.; Nebgen, B.; Lubbers, N.; Isayev, O.; Roitberg, A. E. Less is more: Sampling chemical space with active learning. *The Journal of Chemical Physics* **2018**, *148*, 241733.

[255] Sanchez-Lengeling, B.; Aspuru-Guzik, A. Inverse molecular design using machine learning: Generative models for matter engineering. *Science* **2018**, *361*, 360–365.

[256] Guimaraes, G. L.; Sanchez-Lengeling, B.; Farias, P. L. C.; Aspuru-Guzik, A. Objective-Reinforced Generative Adversarial Networks (ORGAN) for Sequence Generation Models. *arXiv e-prints* **2017**, *abs/1705.10843*.

[257] Cao, N. D.; Kipf, T. MolGAN: An implicit generative model for small molecular graphs. *arXiv e-prints* **2018**, *abs/1805.11973*.

[258] Kingma, D. P.; Welling, M. Auto-Encoding Variational Bayes. 2$^{\text{nd}}$ International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings. 2014.

[259] Chen, W.; Ferguson, A. L. Molecular enhanced sampling with autoencoders: On-the-fly collective variable discovery and accelerated free energy landscape exploration. *Journal of Computational Chemistry* **2018**, *39*, 2079–2102.

[260] Wang, W.; Gómez-Bombarelli, R. Coarse-Graining Auto-Encoders for Molecular Dynamics. *arXiv e-prints* **2018**, arXiv:1812.02706.

[261] Doersch, C. Tutorial on Variational Autoencoders. *arXiv e-prints* **2016**, arXiv:1606.05908.

[262] Goodfellow, I. J.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A. C.; Bengio, Y. *Advances in Neural Information Processing Systems 27*; Curran Associates, Inc., 2014; pp 2672–2680.

[263] Radford, A.; Metz, L.; Chintala, S. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. 4$^{\text{th}}$ International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings. 2016.

[264] Arjovsky, M.; Chintala, S.; Bottou, L. Wasserstein GAN. *arXiv e-prints* **2017**, *abs/1701.07875*.

[265] Kadurin, A.; Nikolenko, S.; Khrabrov, K.; Aliper, A.; Zhavoronkov, A. druGAN: An Advanced Generative Adversarial Autoencoder Model for de Novo Generation of New Molecules with Desired Molecular Properties in Silico. *Molecular Pharmaceutics* **2017**, *14*, 3098–3104.

[266] Simonovsky, M.; Komodakis, N. GraphVAE: Towards Generation of Small Graphs Using Variational Autoencoders. Artificial Neural Networks and Machine Learning - ICANN 2018 - 27$^{\text{th}}$ International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part I. 2018; pp 412–422.

[267] Jin, W.; Barzilay, R.; Jaakkola, T. S. Junction Tree Variational Autoencoder for Molecular Graph Generation. Proceedings of the 35$^{\text{th}}$ International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018. 2018; pp 2328–2337.

[268] Kang, S.; Cho, K. Conditional Molecular Design with Deep Generative Models. *Journal of Chemical Information and Modeling* **2019**, *59*, 43–52.

[269] Popova, M.; Isayev, O.; Tropsha, A. Deep Reinforcement Learning for De-Novo Drug Design. *arXiv e-prints* **2017**, *abs/1711.10907*.

[270] Maziarka, L.; Pocha, A.; Kaczmarczyk, J.; Rataj, K.; Warchol, M. Mol-CycleGAN - a generative model for molecular optimization. *arXiv e-prints* **2019**, *abs/1902.02119*.

[271] Sutton, R. S.; McAllester, D. A.; Singh, S. P.; Mansour, Y. *Advances in Neural Information Processing Systems 12*; Curran Associates, Inc., 1999; pp 1057–1063.

[272] Yu, L.; Zhang, W.; Wang, J.; Yu, Y. SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient. *arXiv e-prints* **2016**, *abs/1609.05473*.

[273] Segler, M. H. S.; Kogej, T.; Tyrchan, C.; Waller, M. P. Generating Focused Molecule Libraries for Drug Discovery with Recurrent Neural Networks. *ACS Central Science* **2018**, *4*, 120–131.

[274] Popova, M.; Shvets, M.; Oliva, J.; Isayev, O. MolecularRNN: Generating realistic molecular graphs with optimized properties. *arXiv e-prints* **2019**, *abs/1905.13372*.

[275] You, J.; Liu, B.; Ying, Z.; Pande, V. S.; Leskovec, J. *Advances in Neural Information Processing Systems 31*; Curran Associates, Inc., 2018; pp 6412–6422.

[276] Kusner, M. J.; Paige, B.; Hernández-Lobato, J. M. Grammar Variational Autoencoder. Proceedings of the 34$^{\text{th}}$ International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017. 2017; pp 1945–1954.

[277] Dai, H.; Tian, Y.; Dai, B.; Skiena, S.; Song, L. Syntax-Directed Variational Autoencoder for Structured Data. 6$^{\text{th}}$ International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings. 2018.

[278] Krenn, M.; Häse, F.; Nigam, A.; Friederich, P.; Aspuru-Guzik, A. SELFIES: a robust representation of semantically constrained graphs with an example application in chemistry. *arXiv e-prints* **2019**, *abs/1905.13741*.

[279] Polykovskiy, D.; Zhebrak, A.; Sanchez-Lengeling, B.; Golovanov, S.; Tatanov, O.; Belyaev, S.; Kurbanov, R.; Artamonov, A.; Aladinskiy, V.; Veselov, M.; Kadurin, A.; Nikolenko, S. I.; Aspuru-Guzik, A.; Zhavoronkov, A. Molecular Sets (MOSES): A Benchmarking Platform for Molecular Generation Models. *arXiv e-prints* **2018**, *abs/1811.12823*.

[280] Kitchin, J. R. Machine learning in catalysis. *Nature Catalysis* **2018**, *1*, 230–232.

[281] Kulik, H. J. Making machine learning a useful tool in the accelerated discovery of transition metal complexes. *Wiley Interdisciplinary Reviews: Computational Molecular Science* **2019**, *0*, e1439.

[282] Schleder, G. R.; Padilha, A. C. M.; Acosta, C. M.; Costa, M.; Fazzio, A. From DFT to machine learning: recent approaches to materials science–a review. *Journal of Physics: Materials* **2019**, *2*, 032001.

[283] Smith, J. S.; Nebgen, B. T.; Zubatyuk, R.; Lubbers, N.; Devereux, C.; Barros, K.; Tretiak, S.; Isayev, O.; Roitberg, A. E. Approaching coupled cluster accuracy with a general-purpose neural network potential through transfer learning. *Nature Communications* **2019**, *10*, 2903.

[284] Pan, S. J.; Yang, Q. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering* **2010**, *22*, 1345–1359.

[285] Shin, H.; Roth, H. R.; Gao, M.; Lu, L.; Xu, Z.; Nogues, I.; Yao, J.; Mollura, D.; Summers, R. M. Deep Convolutional Neural Networks for Computer-Aided Detection: CNN Architectures, Dataset Characteristics and Transfer Learning. *IEEE Transactions on Medical Imaging* **2016**, *35*, 1285–1298.

[286] Yosinski, J.; Clune, J.; Bengio, Y.; Lipson, H. In *Advances in Neural Information Processing Systems 27*; Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., Weinberger, K. Q., Eds.; Curran Associates, Inc., 2014; pp 3320–3328.

[287] Meredig, B.; Antono, E.; Church, C.; Hutchinson, M.; Ling, J.; Paradiso, S.; Blaiszik, B.; Foster, I.; Gibbons, B.; Hattrick-Simpers, J.; Mehta, A.; Ward, L. Can machine learning identify the next high-temperature superconductor? Examining extrapolation performance for materials discovery. *Molecular Systems Design & Engineering* **2018**, *3*, 819–825.

[288] Wallach, I.; Heifets, A. Most Ligand-Based Classification Benchmarks Reward Memorization Rather than Generalization. *Journal of Chemical Information and Modeling* **2018**, *58*, 916–932.

[289] Kailkhura, B.; Gallagher, B.; Kim, S.; Hiszpanski, A.; Yong-Jin Han, T. Reliable and Explainable Machine Learning Methods for Accelerated Material Discovery. *arXiv e-prints* **2019**, arXiv:1901.02717.

[290] Blagus, R.; Lusa, L. Class prediction for high-dimensional class-imbalanced data. *BMC Bioinformatics* **2010**, *11*, 523.

[291] Anderson, M. J.; Robinson, J. Permutation Tests for Linear Models. *Australian & New Zealand Journal of Statistics* **2001**, *43*, 75–88.

[292] Dancik, G. M.; Dorman, K. S. mlegp: statistical analysis for computer models of biological systems using R. *Bioinformatics* **2008**, *24*, 1966–1967.

[293] Bezanson, J.; Edelman, A.; Karpinski, S.; Shah, V. B. Julia: A fresh approach to numerical computing. *SIAM review* **2017**, *59*, 65–98.

[294] Pedregosa, F. et al. Scikit-learn: Machine Learning in Python . *Journal of Machine Learning Research* **2011**, *12*, 2825–2830.

[295] Kuhn, M. Building Predictive Models in R Using the caret Package. *Journal of Statistical Software, Articles* **2008**, *28*, 1–26.

[296] Karatzoglou, A.; Smola, A.; Hornik, K.; Zeileis, A. kernlab – An S4 Package for Kernel Methods in R. *Journal of Statistical Software* **2004**, *11*, 1–20.

[297] Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; Lerer, A. Automatic differentiation in PyTorch. NIPS 2017 Workshop Autodiff. 2017.

[298] Collobert, R.; Kavukcuoglu, K.; Farabet, C. Torch7: A Matlab-like Environment for Machine Learning. BigLearn, NIPS Workshop. 2011.

[299] Democratizing Deep-Learning for Drug Discovery, Quantum Chemistry, Materials Science and Biology. `https://github.com/deepchem/deepchem`, 2016.

[300] Ward, L. et al. Matminer: An open source toolkit for materials data mining. *Computational Materials Science* **2018**, *152*, 60–69.

[301] Ong, S. P.; Richards, W. D.; Jain, A.; Hautier, G.; Kocher, M.; Cholia, S.; Gunter, D.; Chevrier, V. L.; Persson, K. A.; Ceder, G. Python Materials Genomics (pymatgen): A robust, open-source python library for materials analysis. *Computational Materials Science* **2013**, *68*, 314–319.

[302] Roch, L. M.; Häse, F.; Kreisbeck, C.; Tamayo-Mendoza, T.; Yunker, L. P. E.; Hein, J. E.; Aspuru-Guzik, A. ChemOS: An Orchestration Software to Democratize Autonomous Discovery. *chemRxiv e-prints* **2018**,

[303] Gossett, E.; Toher, C.; Oses, C.; Isayev, O.; Legrain, F.; Rose, F.; Zurek, E.; Carrete, J.; Mingo, N.; Tropsha, A.; Curtarolo, S. AFLOW-ML: A RESTful API for machine-learning predictions of materials properties. *Computational Materials Science* **2018**, *152*, 134–145.

[304] Curtarolo, S.; Setyawan, W.; Wang, S.; Xue, J.; Yang, K.; Taylor, R. H.; Nelson, L. J.; Hart, G. L. W.; Sanvito, S.; Buongiorno-Nardelli, M.; Mingo, N.; Levy, O. AFLOWLIB.ORG: A distributed materials properties repository from high-throughput ab initio calculations. *Computational Materials Science* **2012**, *58*, 227–235.