Deep Learning Strategies for Enhanced Molecular Docking and Virtual Screening

Matheus Müller Pereira da Silva[†], Isabella Alvim Guedes[†], Fábio Lima Custódio[†], Eduardo Krempser[‡], and Laurent Emmanuel Dardenne[†*]

[†] Laboratório Nacional de Computação Científica, Petrópolis, Rio de Janeiro, 25651-075, Brazil
[‡] Fundação Oswaldo Cruz, Rio de Janeiro, 21040-361, Brazil
[*] E-mail: dardenne@lncc.br

Abstract

Over the last few years, machine learning (ML) and deep learning (DL) have been revolutionising the computer-aided drug discovery landscape. With the recent availability of the so-called ultra-large virtual libraries (libraries with up to billions of readily available virtual compounds), new ML and DL approaches have been developed to enable the exploration of these large chemical spaces, achieving promising results. Molecular docking is one of the most widely used computational methods for performing *in silico* screenings of virtual libraries. The two primary goals of molecular docking are to predict the correct binding pose of small molecules inside the binding pocket of a protein target and also estimate the binding affinity of the protein-ligand complex. In particular, DL methods have been applied in all aspects of protein-ligand molecular docking, from pose and binding affinity prediction to virtual screening campaigns, improving computational costs and accuracy. This chapter introduces the core aspects of the molecular docking methodology and some fundamental concepts of machine learning and deep learning. We also describe different types of molecular representations and DL architectures commonly employed in the field, such as convolutional and graph neural networks. Furthermore, we provide insights into potential applications by presenting related works from the scientific literature. Finally, we discuss the current limitations, challenges, and biases of DL applied to molecular docking.

Table of Contents

1 Introduction

Combining experimental and computational methods has revolutionised how the scientific community and pharma companies develop new drug candidates and treatments for different health problems. The overall drug development process — from early target

identification and clinical trials to, finally, an approved drug reaching the market — is exceedingly expensive and time-consuming. Recent estimates show that this process can take over a decade and cost over one billion US dollars (Schlander et al., 2021). Thereby, computer-aided drug design (CADD) approaches are essential in improving the chances of finding newer and safer drug candidates more efficiently.

Among these computational tools, molecular docking is widely used in the early stages of drug discovery. Molecular docking experiments aim to predict crucial details of the molecular recognition between a therapeutic target (usually a protein) and a ligand molecule (typically a small organic molecule). These details include the spatial orientation adopted by the ligand upon binding (its pose), the intermolecular interactions, and how favourable the binding event is – *i.e.,* the protein-ligand binding affinity (Guedes et al., 2014, Ferreira et al., 2015).

In recent years, machine learning (ML) approaches, including deep learning (DL), have achieved promising results in several scientific fields, and computational chemistry and biology are no exceptions. ML is a subarea of artificial intelligence concerned with building algorithms capable of solving problems using a data set about the phenomena of interest. DL, on the other hand, is a subfield of ML that uses models with many transformation layers. One such example of a successful DL application is AlphaFold, a model that presented significant advancements to a 50-year-old challenge: predicting the three-dimensional structure of proteins from their amino acid sequence with accuracy similar experimentally determined structures (Jumper et al., 2021). Following the increase in structure-activity data in recent years, DL have also led to significant progress in CADD methodologies, including molecular docking, bringing improvements over traditional approaches.

This chapter aims to overview DL applications in the context of molecular docking, highlighting different approaches in the main aspects of the methodology, such as pose prediction, binding affinity prediction and virtual screening. Additionally, it gives a brief summary of the fundamental principles of molecular docking and machine learning. This chapter also highlights relevant DL architectures for structure-based methods, such as convolutional and graph neural networks. Finally, the chapter discusses the current limitations, challenges, and biases of DL methodologies applied to molecular docking.

2 Protein-ligand Docking

Understanding and predicting critical aspects of the interaction between a protein and a ligand molecule is essential to develop computational tools for drug discovery and

development. Methodologies that use the three-dimensional (3D) structure of the target receptor — such as molecular docking — fall within the structure-based drug design (SBDD) category. In contrast, the ligand-based drug design (LBDD) approach encompasses methods that rely primarily on small-molecule libraries known to be active/inactive without considering any 3D structural information from the target. This chapter will focus primarily on SBBD strategies.

Protein-ligand docking is one of the most used *in silico* methodologies in SBDD (Anderson et al., 2003, Guedes et al., 2014). This strategy aims to obtain ligands with physicochemical and stereochemical characteristics that enable target modulation with high binding affinity, ultimately leading to the desired pharmacological and therapeutic effects. Molecular docking has two interconnected goals: to predict (1) the ligand binding poses and (2) the protein-ligand binding affinity. Nowadays, the capability of docking methods to predict the native pose of small molecules in the binding site of proteins is accurate enough for most protein receptors (given a well-done system preparation and low influence of receptor flexibility) (Guedes et al., 2014). Fig. 1 shows the result of a docking experiment performed using the DockThor program (freely available at https://dockthor.lncc.br/).

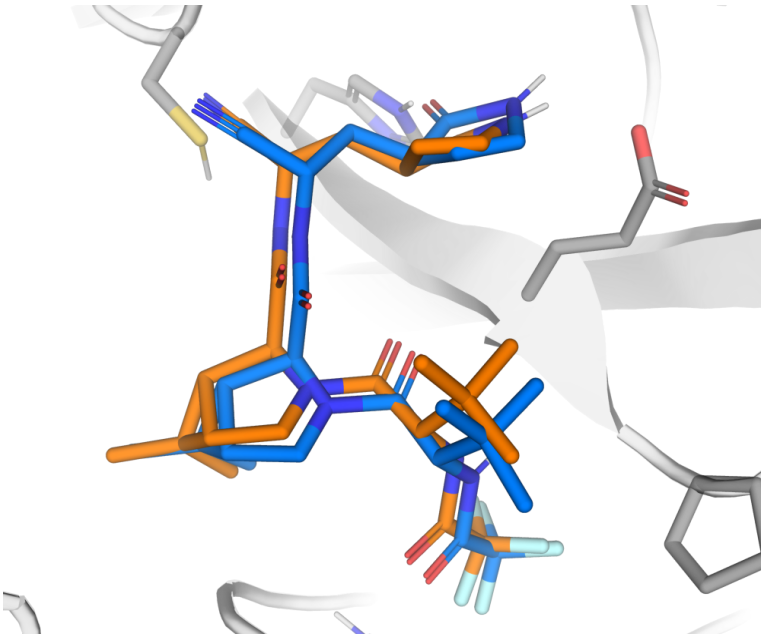

Fig. 1. Example of a protein-ligand docking experiment using the DockThor program. The protein target is the SARS-CoV-2 main protease (Mpro, PDB ID: 7VH8), a key enzyme mediating viral replication and transcription. The ligand is nirmatrelvir, a drug used to treat COVID-19. Its native pose (determined experimentally) is the conformation shown in orange. Depicted in blue is the predicted pose obtained from the docking program.

An aspect of molecular docking that remains a significant challenge is the protein-ligand binding affinity prediction. The binding affinity is a measure of how favourable the binding of a ligand to its target is. The binding affinity can be written as a function of the balance between enthalpy (*H*) and entropy (*S*), *i.e.,* $\Delta G_{bind} = \Delta H_{bind} - T\Delta S_{bind}$, or as a function of some equilibrium constant that can be measured experimentally, *i.e.,* $\Delta G_{bind} = RTln(K_d) = RTln(K_i)$, where R, T, K_d and K_i are the ideal gas constant, temperature and dissociation and inhibition constants, respectively.

The calculation of ΔG_{bind} is a complex task that has been a focus of scientific research. Various methods have been developed to estimate ΔG_{bind}, each with its own advantages and limitations depending on factors such as the number of molecules being evaluated and the desired level of accuracy. Physics-based free energy calculations, which typically involve molecular dynamics (MD) simulations, are considered more advanced techniques that take into account the flexibility of the system. However, these methods often require significant computational resources and can typically only achieve errors of around 1.0 kcal/mol near the experimental value (Parenti et al., 2012, Cournia et al., 2017). Examples of more sophisticated methods for estimating ΔG_{bind} include Free Energy Perturbation (FEP) and Thermodynamic Integration (TI) (Foloppe et al., 2006, Woo et al., 2005). Linear Interaction Energy (LIE) and Molecular Mechanics-Poisson-Boltzmann Surface Area (MM-PBSA) are other methods that utilise molecular dynamic simulations but are less computationally intensive and accurate than FEP and TI (Genheden et al., 2015, Foloppe et al., 2006). Overall, MD-based methods may not be suitable for rapid evaluation of binding affinity for a large number of compounds due to their high computational cost. On the other hand, *scoring functions* (SFs), which consider only a single state of the protein-ligand complex, are employed as a simpler and more efficient alternative in these cases.

Conventional SFs are based on relatively simple descriptors related to the intermolecular interactions, entropy and desolvation effects. As they are generally formed by simple descriptors and do not require MD simulations, they are a suitable choice to be used in virtual screening (VS) experiments, mainly due to their speed and usage simplicity.

VS aims to identify new bioactive compounds for a target of interest by screening large virtual databases of molecules. The VS methodology usually consists of (1) performing large-scale docking experiments with candidate molecules against a target of interest and (2) evaluating the resulting poses using SFs designed explicitly for binding affinity (or

bioactivity) prediction (Lyne, 2002, Guedes et al., 2018, Maia et al., 2020, Kimber et al., 2021). Recent advances in VS methodologies paired with machine learning techniques enabled the screening of ultra-large virtual databases of billions of compounds (Gentile et al., 2020, Gentile et al., 2022).

The simplified modelling assumptions adopted in linear and nonlinear SFs to reduce the computational cost (e.g., implicit solvent, simplified entropy models and rigid protein) imply that they exhibit less predictive accuracy. In this context, affinity predictions on virtual screening experiments are still a challenging problem, and there are several efforts on different fronts of research to develop even more sophisticated and accurate models, such as deep learning-based approaches (Guedes et al., 2018, Meli et al., 2022).

2.1 Conformational Search

The goal of the docking conformational search is to generate geometric orientations of a small molecule within the binding site of a receptor. This process involves the use of a search algorithm and an objective function. The search algorithm searches through the conformational space of possible binding modes, while the objective function estimates the correctness of each potential binding mode, usually by estimating the binding/receptor intermolecular interaction energy.

Search algorithms must navigate a rough energy landscape to find the global energy minimum. Supposing the enthalpic and entropic contributions of the system are modelled correctly by the objective function, the global minimum of this landscape will correspond to the native binding mode. Moreover, local minima will represent alternative binding modes. Unfortunately, considering complex descriptors such as entropy and solvation effects is not straightforward, and current methods employ only rough approximations. Therefore, the global minimum within the energy landscape is not guaranteed to represent the native binding mode (Guedes et al., 2014). Fig. 2 illustrates the energy landscape concept.

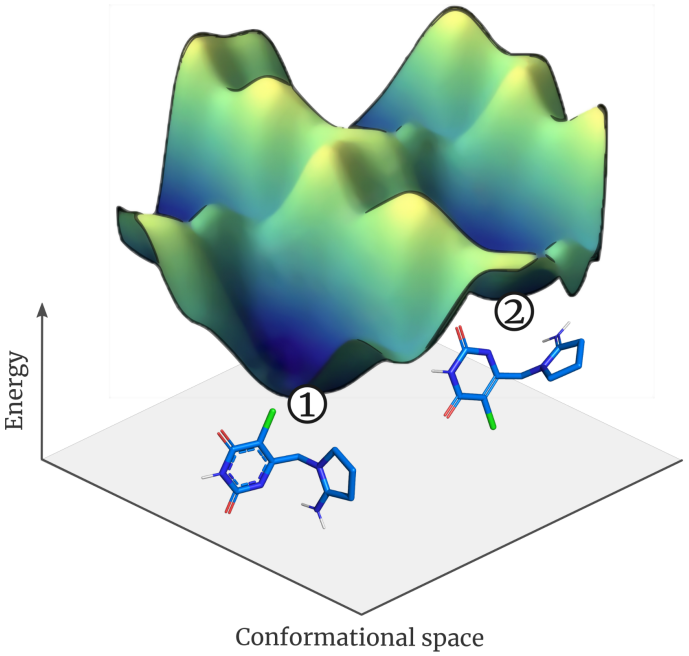

Fig. 2. An hypothetical energy landscape of molecular docking. The goal of the search algorithm is to generate ligand poses while minimising the objective function, typically by modifying the degrees of freedom of the compound (*i.e.*, translational, rotational and torsional), to identify the native binding pose of the ligand in the target binding site. Point 1 represents a ligand conformation at the global minimum, while point 2 corresponds to a ligand conformation resulting in a local minimum. Image adapted from (Ros et al., 2021).

When considering a rigid-protein docking, the search algorithm considers only the ligand's degrees of freedom. Search algorithms are typically divided into three categories: (1) stochastic, (2) systematic, and (3) deterministic (Guedes et al., 2014, Ferreira et al., 2015).

Stochastic methods generate a collection of conformations by randomly modifying ligand parameters, ideally covering a wide range of the energy landscape. Stochastic procedures determine what poses will be kept or discarded. Many stochastic algorithms are biologically-inspired. Some examples of this class of algorithms are evolutionary algorithms, particle swarm optimization, and Monte Carlo optimization. Examples of docking software that implement stochastic algorithms include GOLD (Jones et al., 1997) and DockThor (Magalhães et al., 2004).

On the other hand, the systematic algorithms explore various ligand parameter values via slight changes, gradually changing the ligand pose. The algorithm iteratively

probes the energy landscape until converging to a minimal energy solution. This procedure can be carried out by systematically exploring each degree of freedom within a range of values in a combinatorial fashion. Examples of this type of docking program are Glide (Friesner et al., 2004) and DOCK 4.0 (Erwing et al., 2001).

Finally, deterministic methods are a class of techniques where the current state of a system dictates the modifications that must be made to arrive at the next state. Two notable examples of deterministic methods are energy minimization (EM) and molecular dynamics (MD). EM aims to explore the energy landscape by following the direction associated with the potential energy gradient. On the other hand, MD simulates the movement of a system over time by taking into account thermodynamic variables such as temperature and pressure. These methods possess certain advantages, such as incorporating explicit solvent and considering all degrees of freedom of both the protein and ligand. However, a common drawback of these methods is the tendency for solutions to become trapped in local minima, and they also often entail a high computational cost. Deterministic methods are frequently utilised in conjunction with other docking strategies. CDOCKER is an example of a docking software that employs MD in association with simulated annealing (Wu et al., 2003).

2.2 Scoring Functions

Molecular docking programs use scoring functions (SFs) to evaluate the poses generated by the search algorithm and to estimate the binding affinity. An ideal SF would effectively achieve these two goals simultaneously: (1) pose prediction, which involves distinguishing good poses from bad ones and correctly identifying the native one, and, from these poses, (2) correctly estimating the binding affinity. However, due to simplifications and assumptions made in the modelling process, SFs often exhibit varying levels of accuracy in these tasks. As a result, SFs are more often explicitly designed for just one of these two objectives (Guedes et al., 2014). In this sense, a docking protocol can adopt distinct SFs at different stages. For instance, a fast and accurate SF can be used to accurately predict the pose of millions or possibly billions of candidate compounds on a screening campaign in a reasonable time, without necessarily predicting activity with high accuracy. Finally, the top ranked compounds can be re-scored with a more sophisticated SF to predict binding affinity.

SFs can be classified into three categories, depending on their underlying principles: (1) force field-based, (2) knowledge-based, and (3) empirical (Guedes et al., 2018, Meli et al., 2022). Force field-based SFs have functional forms that are a sum of classical force field terms. They usually consider protein-ligand interaction energies (non-bonded terms) and internal ligand energies (bonded terms). Eq. 1 below is an example of a SF based on the

MMFF94 force field employed in the DockThor docking program for pose prediction (Magalhães et al., 2014):

$$E_{dockthor} = \sum ET_{ijkl} + \sum Evdw_{pq} + \sum EQ_{pq}, \quad (1)$$

where ET represents torsional interactions between the i, j, k, l atoms of the ligand, $Evdw$ is the Buff-14-7 van der Waals potential for non-bonded interactions between atoms p, q from the protein and the ligand respectively, and EQ is the electrostatic potential.

Knowledge-based SFs, on the other hand, use large databases of protein-ligand complexes to infer rules and models from information on atomic interactions/distances (Meli et al., 2022). Basically, these functions assume that, in experimentally determined structures, favourable binding modes have similar distributions in the frequency of interatomic contacts between pairs of atoms. Examples of this type of SF are DrugScore (Velec et al., 2005) and PMF (Muegge et al., 2006).

Lastly, empirical SFs are based on the supposition that it is possible to correlate the binding affinity with a group of variables that describe the protein-ligand complex. These variables are usually termed descriptors or features. SFs built using machine learning algorithms usually belong to this category. Three main components are necessary for building an empirical SF: (1) a data set of protein-ligand complexes and their respective binding affinities determined experimentally (or bioactivity label – *i.e.*, active or inactive); (2) descriptors that numerically capture structural and physicochemical characteristics of the complex; (3) a ML algorithm to establish a quantitative relationship between the descriptors space and the experimental affinities (Guedes et al., 2018).

Empirical SFs can be further divided into linear and nonlinear. Linear SFs are formulated as a weighted sum of descriptors, where the coefficients are adjusted to optimise the correlation with the binding affinity on the training set. This type of SF assumes a linear relationship between descriptors and affinities. Below is an example of a linear SF:

$$\Delta G = c_0 + c_1 \Delta G_{vdw} + c_2 \Delta G_{hb} + c_3 \Delta G_s, \quad (2)$$

where $c_0, \cdots, c_3 \in \mathbb{R}$ are the equation's coefficients; ΔG_{vdw} is the van der Waals potential; ΔG_{hb} is a term to account for hydrogen bonds; and ΔG_s is a descriptor for entropic losses due to the binding event.

Nonlinear SFs, on the other hand, use nonlinear ML techniques, such as random forests, support vector machines, gradient boosting trees, and deep artificial neural networks. The scientific literature has reported the superior performance of nonlinear scoring functions (SFs) in predicting binding affinities compared to classical (linear) SFs (Wójcikowski et al., 2017, Wang et al., 2017, Khamis et al., 2016).

It is important to state that, regarding the prediction task, both linear and nonlinear SFs can be developed to estimate the absolute binding affinity (*e.g.*, pK_d values) or binary labels of bioactivity (*e.g.*, active or inactive). These prediction objectives are formulated in terms of regression or classification, respectively. Section 3.1 will cover these concepts in more detail.

2.3 Benchmarking Sets for Molecular Docking

Gathering protein-ligand structural and binding data is essential to develop and assess the performance of docking methodologies. One of the most widely used benchmark datasets for developing and evaluating the performance of docking methods is the PDBbind (Wang et al., 2004, Liu et al., 2015).

PDBbind's goal is to provide binding affinity data for the largest possible number of biomolecular complexes from the Protein Data Bank (PDB). The PDB is the most extensive structural database for biomolecules available to the general public. The PDBbind is generally updated annually, and its latest version (v.2020) has 19,443 protein-ligand complex instances with structural and binding affinity data available.

The PDBbind database is divided into three subsets resulting from several increasingly restrictive filters: the general set, which has all the available data; the refined set, a high-quality subset of protein-ligand complexes only, with 5,316 instances; and the core set. In particular, the PDBbind core set is a subset explicitly designed to assess the performance of docking methods and SFs. The core set serves as a test set for the evaluation protocol called CASF (comparative assessment of scoring functions), designed to assess the capacity of docking methodologies to predict correct poses and binding affinities. In its latest version (v.2016), the core set has 285 high-quality and highly-curated protein-ligand complexes with associated binding affinity (Su et al., 2018).

Another relevant data set for evaluating docking methodologies is the DUD-E (Database for Useful Decoys - Enhanced). This data set has 102 molecular targets, with 22,886 compounds experimentally identified as bioactive. For each bioactive compound,

there are 50 presumably non-binding molecules (called *decoys*) with similar properties to the bioactive ones but topologically distinct. This data set seeks to emulate the practical reality of VS assays, where most compounds are inactive, with a minority of active molecules (Mysinger et al., 2012).

Many recent studies have pointed out that the use of widely employed protein-ligand datasets, such as DUD-E and PDBbind, can lead to the construction of biased binding affinity predictors with over-optimistic performances (see section 7). For that reason, some authors devised datasets specifically crafted to avoid such biases and limitations and provide more rigorous benchmarks that can better estimate the performance of scoring functions in prospective assays. Some of these datasets include D-COID (Adeshina et al., 2020), DUDE-Z (Stein et al., 2021), Deepcoy (Imrie et al., 2021) and TocoDecoy (Zhang et al., 2022). These sets use different strategies, such as property-unmatched decoys and topology and conformation-based decoys, to mitigate previously described liabilities of other datasets.

2.4 Ultra-large Virtual Libraries

In the context of VS assays, novel and highly potent compounds have been identified by screening ultra-large virtual libraries containing up to billions of virtual compounds (Crunkhorn, S., 2022, Sadybekov et al., 2022, Lyu et al., 2019, Graff et al., 2021, Gentile et al., 2022). Virtual compounds are computer-generated representations of chemical structures that do not necessarily exist in physical reality—usually small organic molecules in this context. They are often crafted using building blocks and a number of possible chemical reactions to ensure a vast chemical diversity and high synthetic feasibility (Warr et al., 2022). Furthermore, some commercial ultra-large libraries, such as the Enamine (Enamine, 2023), offer the possibility to synthesise compounds on demand.

Based on their characteristics, these sets of compounds can be classified into spaces and libraries (Warr et al., 2022). Spaces are collections of compounds constructed in a combinatorial fashion. These spaces can be substantially large, with up to trillions of possible compounds. Due to their size, it may not be computationally viable to enumerate all molecules (*i.e.*, count each distinct chemical entity). Exploring these spaces requires more sophisticated techniques like genetic algorithms or generative models (Warr et al., 2022). On the other side, libraries are collections of fully described and enumerated compounds, with sizes up to the order of 10^9. Examples of ultra-large virtual libraries include ZINC20 (1.4 B) (Irwin et al., 2020), GBD-17 (166.4 B) (Ruddigkeit et al., 2012), SAVI (1.75 B) (Patel et al., 2020), and Enamine REAL collection (6 B) (Enamine, 2023).

To perform docking experiments using such large libraries is computationally intractable. Therefore, computational approaches based on AI have emerged in an attempt to explore these collections in less time while retaining most of the best-performing candidates. One of these approaches is the framework of active learning, in which a surrogate model (such as a machine learning model) is trained to predict the actual docking scores using only subsets of the entire collection. Based on the predicted scores, new subsets are then selected for docking experiments to be performed, iteratively updating the training set of the surrogate model. Finally, after a certain number of iterations, the entire library is reduced to a smaller subset of highly enriched compounds presumed to be active against the target of interest (Graff et al., 2021, Gentile et al., 2022, Gentile et al., 2020).

Ultra-large virtual libraries hold promising potential due to their size, vast chemical diversity and high probability of compound synthesizability. Developing computational methods for effectively leveraging this potential is a significant technological and scientific challenge that could lead to new and innovative therapeutic drugs for various diseases.

2.5 Deep Learning in Molecular Docking

Deep learning (DL) is a subfield of machine learning that has gained much attention across many scientific fields in the past few years. DL has been employed in all aspects of protein-ligand molecular docking, from pose prediction to virtual screening campaigns (Crampon et al., 2022). Publications in the scientific literature have shown deep learning-based strategies enabling faster docking protocols, more accurate scoring functions and the virtual screening of libraries with billions of compounds (Gentile et al., 2020, Gentile et al., 2022, Zhang et al., 2022).

To achieve the goals of molecular docking, DL methodologies, ideally, should learn important aspects of the binding event. For this, detailed knowledge and proper representation of the 3D protein-ligand interfaces are determinant factors. Concerning the structured-based approach, two of the most prominent ways of computationally describing molecules in DL applications are voxel-based and graph-based representations. These representations aim to leverage the essential features inherent to the 3D structure of proteins and ligands in a way that can be effectively processed by DL algorithms.

Table 1 is a non-exhaustive list of DL applications across the different objectives of molecular docking. These applications are classified according to their goals in the protein-ligand molecular docking: pose prediction only, binding affinity (or bioactivity)

estimation only and virtual screening. They can also be classified by the type of representation they use (*e.g.*, voxels grids, graphs).

Table 1: DL strategies that use different molecular representations applied to molecular docking tasks (pose prediction, binding affinity prediction, and virtual screening). [a]Prediction of absolute binding affinity value (*e.g.* kcal/mol or pK_i). [b]Prediction of binding affinity class (*e.g.*, binders vs. non-binders).

Method	**Docking task**	**Representation**
GNINA 1.0 (McNutt et al., 2021)	Pose prediction	Voxel grids
DeepDock (Méndez-Lucio et al., 2021)	Pose prediction and virtual screening	Graphs
EquiBind (Stärk et al., 2022)	Pose prediction	Graphs
DiffDock (Corso et al., 2022)	Pose prediction	Graphs
EGNN (Masters et al., 2022)	Pose prediction	Graphs
TankBind (Lu et al., 2022)	Pose prediction and binding affinity prediction[a]	Graphs
CNN-SF (Ragoza et al., 2017)	Pose prediction, binding affinity prediction[b], and virtual screening	Voxel grids
KDEEP (Jiménez et al., 2018)	Binding affinity prediction[a]	Voxel grids
Pafnucy (Stepniewska-Dziubinska et al., 2018)	Binding affinity prediction[a]	Voxel grids
SIGN (Li et al., 2021)	Binding affinity prediction[a]	Graphs
PaxNet (Zhang et al., 2022)	Binding affinity prediction[a]	Graphs
PIGNet (Moon et al., 2022)	Pose prediction, binding affinity prediction[a], and	Graphs

	virtual screening	
MolPAL (Graff et al., 2021)	Virtual screening	Other
Deep Docking (Gentile et al., 2022)	Virtual screening	Other
DeepVS (Pereira et al., 2016)	Virtual screening	Other
DFCNN (Zhang et al., 2022)	Virtual screening and binding affinity prediction[b]	Other
HASTEN (Kalliokoski, 2021)	Virtual screening	Graphs

3 Machine Learning Concepts

As a subfield of machine learning (ML), deep learning (DL) incorporates many of the same concepts and practices. These include the various types of learning, optimization techniques, and strategies for addressing issues such as bias and variance. In the training and validation of machine learning models, it is essential to consider factors such as data set spliting, the potential for overfitting and regularisation techniques. The following section will explore these concepts in further detail before discussing specific deep learning algorithms such as convolutional and graph neural networks.

3.1 Learning from Data

Generally speaking, the ML process has two steps: (1) obtaining a data set and (2) algorithmically building a statistical model based on the data set to solve a given problem. The step described in (2) is called training: ML models are “trained” instead of explicitly programmed (or formulated). A successfully trained ML model is said to have “learned” to solve the problem (Burkov, 2019).

We can divide ML techniques into four types of learning: (1) supervised, (2) semi-supervised, (3) unsupervised, and (4) reinforcement learning. Supervised learning is often the most used type of machine learning and is also the focus of this section. For more information on other types of learning, refer to (Sutton et. al., 2018, Gerón, 2017).

An ML model represents a mathematical formula or algorithm that, when applied to the data set used for its training (usually called the training set), can produce numerical

outputs that are desired or useful. Ideally, the same model should also be able to produce useful and correct results for other data examples that are not present in the training set — under the condition that these examples come from a distribution that is similar or identical to that of the training set. This capacity — predicting correct outputs for previously “unseen” data — is called generalisation (Burkov, 2019).

Essentially, an ML model transforms its inputs into outputs. These transformations can be coordinate changes, linear projections, translations, nonlinear operations, and others. Typically, the set of all possible transformations is pre-determined for a particular ML technique. This set is called the hypothesis space. (Chollet, 2018).

A crucial element of training ML models is the objective function (also called the loss function): a mathematical function that measures the quality of the model’s predictions (*i.e.*, measure the dissimilarity between the predicted output of a model and the actual label). This function generally outputs a single number that indicates the model's performance on a particular data set. Moreover, since many optimization algorithms require taking gradients of the loss function, this function is generally required to be differentiable. In this sense, the word “learning” in machine learning corresponds to finding optimal parameters (to the transformations) using an optimization algorithm that minimises the loss function, resulting in a useful predictive model (Chollet, 2018).

In supervised learning, the data set is a labelled collection of examples $\{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$, where n is the total number of examples. Each element $\mathbf{x}_i$ of the data set is a list (also called vector) of descriptors (or features) of size m, associated with a label y_i. For example, suppose we are trying to predict the binding affinity of a small molecule to a given target using a list of features such as the number of heavy atoms, the number of hydrogen donors and acceptors, and others. In this case, each element x_{ij} of $\mathbf{x}_i$, for $j = 1, \cdots, m$, represents one of these features.

In cases where every position j of each example always represents the same type of information (*e.g.*, $j = 1$ represents the number of heavy atoms), the data is called structured — in other words, the dataset is a $n \times m$ matrix. However, datasets can also be unstructured, such as text and images. Some ML algorithms are better suited for dealing with unstructured data, such as convolutional neural networks for images and recurrent neural networks for text.

Depending on the problem, the data labels can assume different types of values. In a classification problem, labels belong to a finite set of classes: $y_i \in \{1, 2, \cdots, C\}$ where C is the total number of classes. If $C = 2$, the problem is called binary classification (for example, active and inactive molecules). If $C > 2$, the problem is called multiclass classification (for example, classifying molecules into different ranges of activity such as strong, medium, and weak). Finally, y_i can also be a real number, for example, the absolute protein-ligand binding affinity in pK_i. In this case, the problem is called a regression problem.

To exemplify the general process of supervised ML algorithms, consider the multiple linear regression (MLR) model. In MLR, the hypothesis space is all linear functions f such that $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b = \hat{y}$, where $\mathbf{w} \in \mathbb{R}^m$ and $b \in \mathbb{R}$ represent the model's coefficients, also called parameters or weights. The model's output $\hat{y} \in \mathbb{R}$, given a dataset instance $\mathbf{x}$, is the predicted label for that instance. This model assumes that y is a linear combination of the features.

Next, we can define a loss function for the MLR model that considers all examples of the training set and their actual labels:

$$\mathcal{L}(\mathbf{x}, y) = \frac{1}{n}\sum_{i}^{n}(f(\mathbf{x}_i) - y_i)^2 = \frac{1}{n}\sum_{i}^{n}((\mathbf{w}^T\mathbf{x} + b) - y_i)^2 . \quad (3)$$

This loss function is called the mean squared error (MSE). The objective of the training phase is to minimise the loss function in Eq. 3, determining parameters $\mathbf{w}^*$ and b^* for f that results in the lowest error possible, hopefully resulting in accurate predictions for the training set. It is important to note that even if the model performs well in the training set, this does not mean it will perform equally well on new "unseen" data, as discussed in section 3.3.

3.2 Statistical Measures of Predictive Performance

To assess a model's performance on validation and test sets, we can use various formal metrics. These metrics vary depending on the type of the problem (*e.g.* regression or classification). For regression problems, the most widely used metrics include mean squared error (MSE), root mean squared error (RMSE), and Pearson's correlation coefficient (also known as Pearson's r).

The MSE is calculated as follows:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2, \quad (4)$$

where n is the size of the data sample, y_i is the actual label and $\hat{y}_i$ is the model's prediction for instance i. As the prediction error decreases, the MSE approaches zero.

The RMSE, on the other hand, is just the square root of the MSE. One of the main advantages of the RMSE over MSE as a metric is interpretability since the results are given in the same units as the labels.

At last, Pearson's r is a measure of the linear correlation between two sets of variables, in this case y and $\hat{y}$. The formula for Pearson's r is given by:

$$r = \frac{\sum_{i=1}^{n}(y_i - \bar{y})(\hat{y}_i - \bar{\hat{y}})}{\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}\sqrt{\sum_{i=1}^{n}(\hat{y}_i - \bar{\hat{y}})^2}}, \quad (5)$$

where $\bar{y}$ and $\bar{\hat{y}}$ are the average of y and $\hat{y}$, respectively. The correlation coefficient ranges from -1 to 1. An absolute value of 1 means that both measurements are perfectly correlated (i.e., a linear equation describes precisely the relationship between the two variables). Conversely, if r = -1, there is a perfect correlation in the opposite direction (as one variable increases, the other decreases). Lastly, if r = 0, there is no linear correlation between the variables.

Regarding classification problems, the most widely used metrics to assess model performance are: confusion matrix, accuracy, precision/recall, and area under the receiver operator characteristic (ROC) curve (Burkov, 2019).

The confusion matrix summarises how successful the model is at predicting the different classes of a data set sample, with each axis representing the predicted and actual labels, respectively. This metric lets us easily compare how many examples from a specific class were correctly predicted or mislabeled. For example, consider a model that is trying to predict the binding activity of protein-ligand complexes. A possible confusion matrix for this model's predictions on a data sample is shown in Table 2.

Table 2. Example of a confusion matrix for protein-ligand complex activity predictions.

	active (predicted)	*inactive (predicted)*
active (actual)	**662** (TP)	**63** (FN)
inactive (actual)	**8** (FP)	**721** (TN)

In the confusion matrix above, TP means true positives, TN stands for true negatives, FP means false positives, and FN means false negatives. In the case of a multiclass classification problem, the confusion matrix has as many rows and columns as the number of classes. A model that does not mislabel any data set instance produces a confusion matrix with zeros in all its off-diagonal entries (*i.e.*, no false positives nor false negatives).

The precision and recall metrics can be calculated from the confusion matrix. Precision is defined as the ratio of the correctly predicted positive instances (TP) over the total number of positive predictions (TP + FP). On the other hand, recall is the ratio of correct positive predictions (TP) over all instances of the positive class (TP + FN).

There are some problems where high recall is more critical than high precision and vice versa. For example, consider that we are trying to determine the toxicity of a substance (toxic being the positive class and non-toxic the negative class). In this context, a high recall is critical because it means that from all possible toxic substances, most of them were correctly labelled as so.

Another traditional metric employed in classification model assessment is accuracy. Accuracy is the number of correctly classified examples over the total number of predictions, defined as:

$$\text{accuracy} = \frac{TP+TN}{TP+FP+TN+FN}. \quad (6)$$

Accuracy is especially valuable when errors in predicting all classes are equally important.

Finally, the ROC curve is a graph that allows us to assess the performance of a classification model at various classification thresholds. Thus, this metric can only be used with classifiers that output some confidence score or probability measure, such as logistic regression or neural networks (Burkov, 2019).

The ROC curve uses two measures: the true positive rate (TPR), which is precisely the same as the recall, and the false positive rate (FPR), which is defined as $FP/(FP + TN)$. The graph is created by plotting the TPR against the FPR at the predefined thresholds.

Lowering the classification threshold classifies more items as positive, thus increasing TPR and FPR. Conversely, raising the threshold will make TPR and FPR tend to zero.

The area under the ROC curve (AUC) measures the two-dimensional area underneath the ROC curve. It's an aggregate measure that summarises the ROC curve. The higher the AUC, the better the classifier is. A random classifier has AUC equal to 0.5. An AUC below 0.5 typically indicates something is wrong with the model, data labels or choice of train/test data set.

3.3 Bias and Variance in Machine Learning

Two relevant concepts to the ML field are bias and variance. To better understand it, it is helpful to introduce the notion of training, validation, and test sets. Usually, to apply any ML technique, it is recommended to divide the available data set into three disjoint subsets: training, validation, and test. The training set's purpose is to adjust the model's parameters (*i.e.*, to train the model). The validation and test set goal is to estimate the model's performance in previously "unseen" data. However, the validation set is used to assist in the choice of different models (or different hyperparameters, which are parameters of the ML algorithm itself, *e.g.,* the number of iterations). On the other hand, the test set should be reserved to estimate the performance of the final models.

Therefore, in simple terms, bias refers to a systematic error or tendency of a model to consistently learn the wrong pattern from the training data. A low-bias model designates a model with low prediction error in the training set. In contrast, a high-bias model has a large training error (it cannot reproduce the training set's labels very well). High bias is also termed underfitting.

Conversely, the variance indicates the model's sensitivity to small fluctuations in the training set. Consider that we sample different training data sets S from a universe of possible data sets from the same distribution. Due to finite-sample effects, a high-variance model would exhibit significant fluctuations from one set S to another. This variance can result from variation in the training sample, random noise in the data, or even random behaviour in the learning algorithm itself (Dietterich et al., 1995).

A validation set comes in handy for estimating the model variance. A high-variance model could do well in the training set but will perform poorly in the validation set. This scenario is called overfitting: a model with low bias and high variance. Simpler models (*e.g.,* models with fewer parameters) often present a lower variance since they are less dependent

on a particular realisation of S — but usually at the expense of a higher bias (Mehta et al., 2019). Finally, a model that does not overfit nor underfit exhibits good performance on training and validation/test sets.

Among different reasons that can be responsible for underfitting, the most important are (Burkov, 2019): (1) the model is too simple for the data. For example, when trying to model a nonlinear relationship with a linear function. Another reason could be (2) the chosen features used to represent the data need to be more informative. For example, when trying to predict the protein-ligand binding affinity using only the number of atoms from the ligand and the protein.

When it comes to overfitting, its most important causes are (Burkov, 2019): (1) the model is too complex for the data. Complexity, in this sense, usually refers to the model's number of trainable parameters, *e.g.,* big decision trees or neural networks with too many layers. With many trainable parameters (especially if the data set is small), the model could reproduce small details that are particular to a specific realisation of the data set (*e.g.*, random noise), impacting the model's generalisation capability. Another reason for overfitting is (2) a large number of features coupled with few data available. Adding too many features to describe the data can lead to overfitting because the model can find relationships between the features specific to a particular dataset.

Using a more complex model may reduce bias. However, if the model becomes too complex, the generalisation error becomes large due to high variance. Thus, in order to improve generalisation, it may be necessary to find a balance between a biased model with a slight variance and a less-biased model with a more significant variance. This struggle is known as the bias-variance tradeoff and is an essential factor in ML projects, especially if the amount of data is limited (Mehta et al., 2019). Fig. 3 demonstrates the relation between the generalisation error (error in the validation/test set) and the training error as model complexity increases.

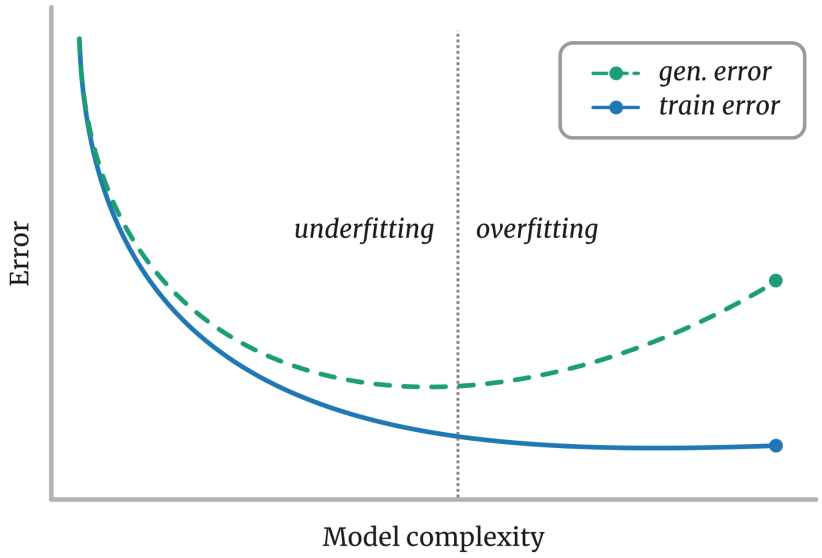

Fig. 3. The generalisation error (dashed line) and training error (solid line) are plotted as a function of model complexity, considering a fixed-size training set. To the left of the vertical line, generalisation and training errors are high, characterising an underfitting scenario. Conversely, to the right of the vertical line, the gap between the training and generalisation error increases as the model complexity increases (low bias and high variance), indicating an overfitting scenario. The vertical line represents an ideal model complexity. Adapted from (Goodfellow et al., 2016).

3.4 Regularisation in Machine Learning Models

Regularisation is a term that encompasses different ML methods and, simply put, is a technique that forces the ML algorithm to construct less complex models (Burkov, 2019). In practice, this could lead to a slight increase in bias but with a noticeable reduction in model variance. In a definition by Goodfellow et al., regularisation is "any modification we make to a learning algorithm that is intended to reduce its generalisation error but not its training error."

There are many regularisation strategies; some can be generally applied to most ML (and even DL) algorithms, and others are specific to a class of algorithms. In order to reduce the validation/test error, these strategies aim to limit the algorithm's complexity: for example, by constraining the parameter values or adding terms corresponding to soft constraints to the loss function.

Two types of regularisation are widely used across different ML techniques: L1 and L2. The basic idea behind L1 and L2 regularisation is to modify the loss function by adding a

term that imposes penalties on more complex models. For example, consider the loss function from Eq. 3:

$$\mathcal{L} = \frac{1}{n}\sum_{i}^{n}(f(\mathbf{x}_i) - y_i)^2 = \frac{1}{n}\sum_{i}^{n}((\mathbf{w}^T\mathbf{x}_i + b) - y_i)^2. \quad \text{(3 revisited)}$$

The application of L1 regularisation would result in modifying the loss like this:

$$\mathcal{L} = \frac{1}{n}\sum_{i}^{n}(f(\mathbf{x}_i) - y_i)^2 + \lambda||\mathbf{w}||. \quad (7)$$

In the modified version of the loss function above, $||\mathbf{w}|| := \sum_{j}^{m}|w_j|$ is L1 norm applied to the weights and λ is a hyperparameter that controls the importance of the regularisation (must be adjusted by the user). If $\lambda = 0$, no regularisation is applied. Conversely, if λ has a high value, the optimization algorithm will prefer low values for the weights $\mathbf{w}$ to minimise $\mathcal{L}$. In practice, by choosing an appropriate value for λ, the algorithm will apply a form of feature selection, deciding which features have greater predictive power (Burkov, 2019). Finally, L2 regularisation is similar to L1, the difference being the use of the L2 norm $||\mathbf{w}||^2 := \sum_{j}^{m}(w_j)^2$ instead of the L1 norm.

3.5 A Basic Protocol for Iteratively Developing Machine Learning Models

Training and validation datasets help diagnose if an ML model has problems with bias or variance. A high-bias model will perform poorly in the training data, *i.e.,* it will underfit the data. Solutions to underfitting problems may vary for different ML techniques but generally include: using a more complex model (bigger neural networks, bigger decision trees, etc.), using features with greater predictive power, using more training iterations, and using different ML algorithms (Ng, 2017).

Once the model's bias has been reduced to an acceptable level, the next step is to investigate its variance. As stated before (section 3.3), a model that does well in the training set but poorly in the validation set has high-variance problems (it overfits the data). Solutions to overfitting include: getting more data, applying regularisation techniques, using less complex models, and reducing the number of features (Ng, 2017).

The balance between bias and variance is a tradeoff. Reducing one may increase the other. Finding the right balance is usually done through trial and error. Fig. 4 shows an iterative process for developing machine learning models in a more systematic way.

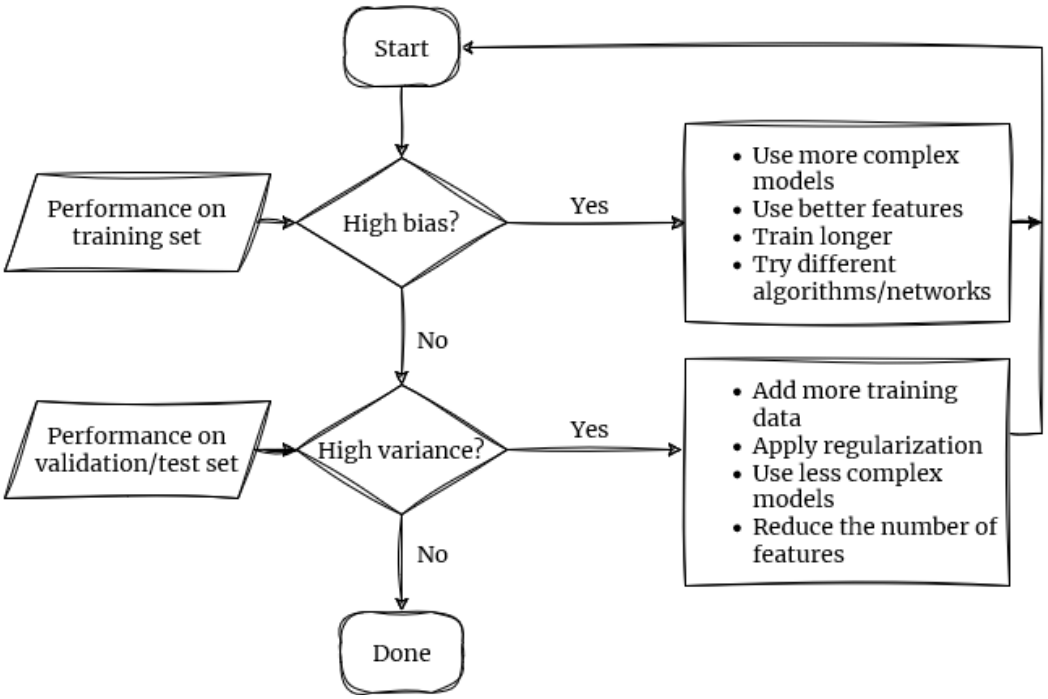

Fig. 4. A flowchart for systematically approaching the development of ML models. The performance of the model on the training and validation or test set should be used to identify high bias or high variance. Iterative experimentation with different approaches should be employed until the model exhibits satisfactory performance.

4 Deep learning

The word "deep" in deep learning concerns an essential characteristic of this class of ML algorithms: the existence of successive layers of representation used to transform the data. Moreover, the number of layers used designates the "depth" of the model. This trait marks a distinction between shallow and deep ML models. Shallow models usually have only one transformation layer — *e.g.*, a single linear transformation in the case of the MLR model. Deep models, on the contrary, have two or more transformation layers.

Successive layers of nonlinear transformations in DL are better suited for extracting valuable representations from data throughout the learning process. For this reason, in DL, it is common to use data features closer to the raw data without performing much feature

extraction — *i.e.,* preprocessing the data (so it is more informative and non-redundant). Feature extraction usually requires a high degree of technical knowledge in the application field, and it is generally a necessary step to successfully train shallow models (Chollet, 2018).

4.1 Basic Components of Deep Artificial Neural Networks

A neural network (NN) is a composite function f where each function composing f corresponds to a different layer. For example, a three-layer model has three components such that $f = f_3(f_2(f_1(\mathbf{x})))$. In a feedforward NN (the typical case), information flows from data set examples $\mathbf{x}$, through the computations defined by f, to the output $\hat{y}$.

The intermediate layers — those between the inputs $\mathbf{x}$ and the last layer — are denominated hidden layers. Considering the example above, the hidden layers correspond to f_1 and f_2. A crucial aspect of the layer transformations in a NN is that they should be differentiable, which means they should be smooth and continuous. This is important because the optimization methods employed to minimise the loss function are generally gradient-based and depend on the partial derivatives of the loss w.r.t. the NN's parameters.

Neural networks have three basic components (Chollet, 2018): (1) layers and their specifications, which define the NN architecture and the hypothesis space, (2) the loss function to be minimised; and (3) the optimization algorithm employed to adjust the NN's parameters during training. The training process is iterative, and the parameters are updated at each iteration. Fig. 5 illustrates the relationship between the different components of a NN.

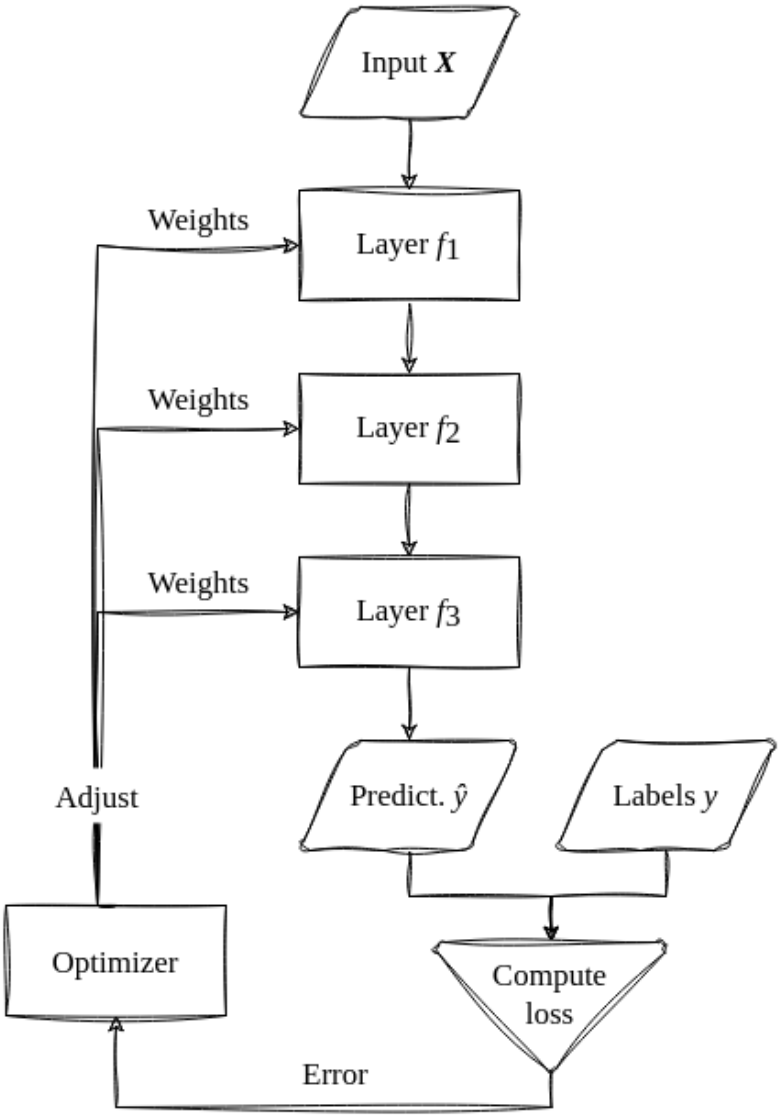

Fig. 5. Components of a neural network with two hidden layers. The input examples $\mathbf{X}$ are processed by the f_1, f_2, and f_3 layers, and the resulting predictions $\hat{\mathbf{y}}$ are assessed using the cost function. The resulting error serves as a signal that, in conjunction with the optimizer, is used to adjust the weights of each layer to minimise the cost function through an iterative process. The commonly employed stopping criterion is the achievement of a predetermined number of iterations.

Each neural network layer has one or more computational units called neurons (Nielsen, 2015). A neuron is a processing unit that receives inputs and produces a single real-valued output. Fig. 6a demonstrates a single neuron representation with three input values x_1, x_2 and x_3 and its output a. The neuron computes a in two steps: (1) a linear and

(2) a nonlinear transformation. In the first step, input variables are linearly combined with weights $\mathbf{w} \in \mathbb{R}^m$ plus a coefficient $b \in \mathbb{R}$ (called bias) to produce $z \in \mathbb{R}$, similar to the MLR model (section 3.1):

$$z = \mathbf{w}^T\mathbf{x} + b = w_1x_1 + \cdots + w_mx_m + b. \quad (8)$$

The second step involves the computation of a function $g(z)$. The nonlinear function g is called the activation function. Different activation functions can be employed, the most common being the sigmoid and the ReLU (rectified linear unit). The ReLU simply corresponds to the maximum value between zero and z — *i.e.*, $\text{ReLU}(z) = max(0, z)$.

The sigmoid function, defined below (Eq. 10), maps z to the range (0, 1):

$$\sigma(z) = \frac{1}{1+exp(-z)}. \quad (9)$$

Therefore, the computation done by a single neuron with activation g is defined as follows:

$$a = g(z) = g(\mathbf{w}^T\mathbf{x} + b). \quad (10)$$

Each layer of the network can have a different number of neurons, represented by n_i for $i = 1, \cdots, l$ with l being the total number of layers. The transformation applied by a layer is simply a repetition of the computation described in Eq. 11 for all n_i neurons in a given layer i. Hence, the computation performed by the first hidden layer of a NN can be described in a compact manner by the following equation:

$$\mathbf{a}^{(1)} = g(\mathbf{z}^{(1)}) = g(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}), \quad (11)$$

where $\mathbf{W}^{(1)}$ represents the weight matrix with dimensions $n_1 \times m$ (lines correspond to the weights of every neuron in layer $i = 1$); $\mathbf{x} \in \mathbb{R}^m$ represents a data set example, and $\mathbf{b}^{(1)} \in \mathbb{R}^{n_1}$ is the bias vector of the first hidden layer. The activation function g is applied individually to each element of the vector $\mathbf{z}^{(1)} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}$. The superscript (1) indicates the corresponding layer (which in this case is the first one).

Consequently, we can interpret Eq. 12 as the multiplication of the feature vector $\mathbf{x}$ by a weight matrix plus a bias vector $\mathbf{b}$, followed by the activation function. The result of this computation is called the activation vector $\mathbf{a}$, which corresponds to the output of each neuron of that layer.

In the case of a NN with l layers, the operations described in Eq. 12 are repeated l times, replacing $\mathbf{x}$ with the activation vector of the previous layer:

$$\begin{aligned} \mathbf{a}^{(2)} &= g(\mathbf{z}^{(2)}) = g(\mathbf{W}^{(2)}\mathbf{a}^{(1)} + \mathbf{b}^{(2)}) \\ &\vdots \\ \mathbf{a}^{(l)} &= g(\mathbf{z}^{(l)}) = g(\mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}). \end{aligned} \tag{12}$$

Fig. 6b shows a graphical representation of a NN with two hidden layers.

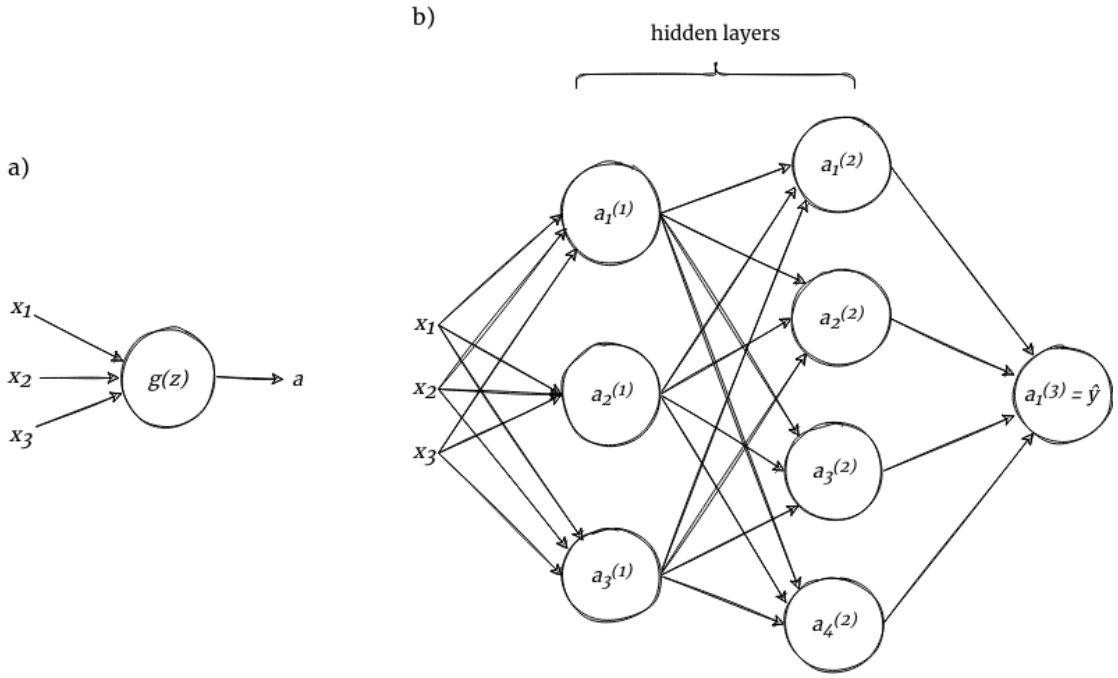

Fig. 6. In (a), a single artificial neuron which takes as inputs three values x_1, x_2 and x_3 and calculates $g(z)$ to produce the output a. In (b), an artificial neuron network with two hidden layers. The variables $a_j^{(i)}$ represent the output value produced by neuron j in layer i. The output of the last layer is the label predicted by the network. The arrows represent the weights of each neuron. For example, the arrows connecting inputs x_1, x_2 and x_3 to the first neuron in the first layer of the neural network represent the weights w in the equation $a_1^{(1)} = g(w_1x_1 + w_2x_2 + w_3x_3 + b_1)$.

The sequence of equations above (Eq. 11 and 12) shows how information is propagated forward in a multilayer neural network. For regression or binary classification problems, a single neuron is usually used in the last layer, producing a single output value. Concerning the activation function of the last layer, in the case of regression, the usual practice is to perform just the linear part of the neuron computation (no activation function). Conversely, in a classification problem, the sigmoid activation function is typically employed in the last layer. The sigmoid output is interpreted as a probability of the example belonging to one class or another.

4.2 Optimization and Backpropagation

In order to train a neural network to make valuable predictions, the model weights must be adjusted to fit the training data set. Once we have defined a network architecture and a loss function according to the type of problem (*e.g.*, classification or regression), we can train our model — *i.e.,* find values for the weights that minimise the loss function. One of the most widely used methods for performing this minimization is gradient descent and its variations (Mehta et al., 2019).

The loss function to be optimised during training is usually very complex, rugged, non-convex, and has many local minima. Besides, in most modern applications, data sets can have millions of examples, and NNs can have millions or even billions of adjustable parameters, which makes this a hard optimization problem (Mehta et al., 2019).

The basic idea behind the gradient descent algorithm is to update the model's weights in the opposite direction of the gradient of the loss function. Denote the loss function as $\mathcal{L}(f(\theta), \mathbf{X})$, where $f(\theta)$ is the NN model (a function of the adjustable parameters θ), and $\mathbf{X}$ is the data set. The standard gradient descent algorithm updates the parameters as follows:

$$\theta_{t+1} = \theta_t - \gamma \nabla \mathcal{L}(f(\theta), \mathbf{X}), \tag{13}$$

where $\nabla \mathcal{L}(f(\theta), \mathbf{X})$ is the gradient of the loss function w.r.t. θ, and $\gamma \in \mathbb{R}$ is the *learning rate*. The learning rate controls the size of the step in the direction of the gradient at iteration t. The appropriate value for the learning rate is usually problem-dependent, and it is a crucial hyperparameter that the user must adjust. A very small γ leads to a high computational cost since too many steps are necessary to reach a local minimum. Alternatively, if γ is too large, the algorithm may become unstable, oscillating around or even diverging from the minimum (Mehta et al., 2019).

Since data sets can be very large, computing the gradient for the whole data set may be impractical. The mini-batch stochastic gradient is the most common variant of the gradient descent algorithm employed to overcome this limitation. It is called stochastic because it approximates the gradient on a subset of the data called a mini-batch. The mini-batch size is typically much smaller than the data set size, from dozens of examples to a few hundred. At each training step, the algorithm computes the gradients of the loss function using a different mini-batch and updates the weights accordingly. A complete iteration through the whole data set is called an epoch.

Optimising neural networks can be tricky, and numerous adaptations to the gradient descent have been proposed to improve the performance of these algorithms. Modern deep learning frameworks such as TensorFlow, Caffe, and PyTorch already implement different optimization algorithms; some of the most widely used are Adam (Kingma et al., 2017) and RMSprop (Graves, 2013).

As stated above, adjusting the weights (or parameters) of a NN requires the computation of the gradient w.r.t. millions or even billions of parameters. An algorithm called backpropagation enables the efficient calculation of these derivatives. This algorithm computes partial derivatives by propagating information backwards through the network, hence the name backpropagation (Goodfellow et al., 2016).

The backpropagation procedure is based on the chain rule of calculus to compute the gradient of the loss function. Let j represent the index of neurons in the last layer l, and k the index of neurons in layer $l-1$. Let $w_{jk}^{(l)}$ denote the weight connecting the k-th neuron in layer $l-1$ to the j-th neuron in layer l. Suppose $C_0 = \sum_{j=1}^{n_l} (a_j^{(l)} - y_j)^2$ is the loss function for a single example of the data set, where y_j is the j-th component of the actual label for that example and $a_j^{(l)}$ is the activation value for neuron j in layer l. Using the chain rule is possible to calculate the derivative of C_0 with respect to any weight $w_{jk}^{(l)}$ of the last layer as follows:

$$\frac{\partial C_0}{\partial w_{jk}^{(l)}} = \frac{\partial C_0}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial w_{jk}^{(l)}}. \tag{14}$$

Similarly, it is possible to calculate the partial derivative of C_0 w.r.t. any bias $b_j^{(l)}$ as follows:

$$\frac{\partial C_0}{\partial b_j^{(l)}} = \frac{\partial C_0}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}}. \quad (15)$$

Finally, it is also possible to calculate the partial derivative of the loss function with respect to the activations $a_k^{(l-1)}$ of the layer $l - 1$:

$$\frac{\partial C_0}{\partial a_k^{(l-1)}} = \sum_{j=1}^{n_l} \frac{\partial C_0}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial a_k^{(l-1)}}. \quad (16)$$

In possession of the partial derivatives with respect to the activations of the previous layer, we can repeat this process (repetitively apply the chain rule) to obtain the gradient of C_0 with respect to all weights and biases of the NN. These calculations can be implemented into a simple and computationally efficient algorithm, enabling the training of large NNs in a feasible period. Finally, the average of the gradients over an entire mini-batch is usually used at each gradient descent step (Nielsen, 2015).

4.3 Regularisation in Deep Neural Networks

Besides the L1 and L2 regularizations (section 3.4), other regularisation techniques are widely used with NNs, such as dropout, batch normalisation, and early stopping (Burkov, 2019). Dropout works by randomly excluding some neurons (zeroing out their activations) layer-wise, with a probability defined by the user for each layer separately. This probability is denominated dropout rate and corresponds to the chance of a neuron being excluded at each forward pass. Typical values for the dropout rate are between 20% to 50%. Dropout is applied only during the training step, while inference is made with all neurons. This technique has a regularisation effect because the NN cannot rely upon any specific neuron for its predictions, avoiding spurious correlations.

The early stopping technique consists in stopping the training when no improvement is observed on the validation set — according to the loss score or some other metric of choice. For this technique, it is necessary to choose the value for a hyperparameter called "patience," which defines how many epochs/steps the training will continue before stopping due to no improvement. Early stopping is a simple procedure that prevents the generalisation error from rising too much (overfitting) while also avoiding wasting computational resources.

Finally, batch normalisation (Szegedy et al., 2015) works by normalising each layer's input by fixing its mean and variance. This process is performed for each mini-batch of

inputs in the training process. Normalising the layers makes the training faster and more stable and also mitigates vanishing or exploding gradient problems (when gradients become increasingly too large or too small).

4.4 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are NNs with specialised layers that perform a type of linear transformation called convolution. This type of network is widely used in problems where data sets are 2D images (for image recognition, for example) or even 3D images (volumetric image data, such as in medical images) (Goodfellow et al., 2016).

A 2D image can be represented in the computer as a matrix. Each matrix entry is designated a *pixel*; the value of each pixel indicates the colour intensity in that region of the image. Despite being 2D objects, colour images are represented in the computer by using 3D tensors with dimensions $w_{in} \times h_{in} \times c_{in}$ (where *w* and *h* are the width and the height of the image, respectively) . Specifically, colour image uses three matrices (*i.e.*, $c_{in} = 3$) with dimensions $w_{in} \times h_{in}$ each, while grey images use $c_{in} = 1$. Each matrix of a colour image corresponds to the intensity of the colours red, green, and blue (RBG). The quantity c_{in} is also denominated *channels*. Ultimately, the sum of the three channels produces the final image. Fig. 7 illustrates the representation of grey and colour images on the computer.

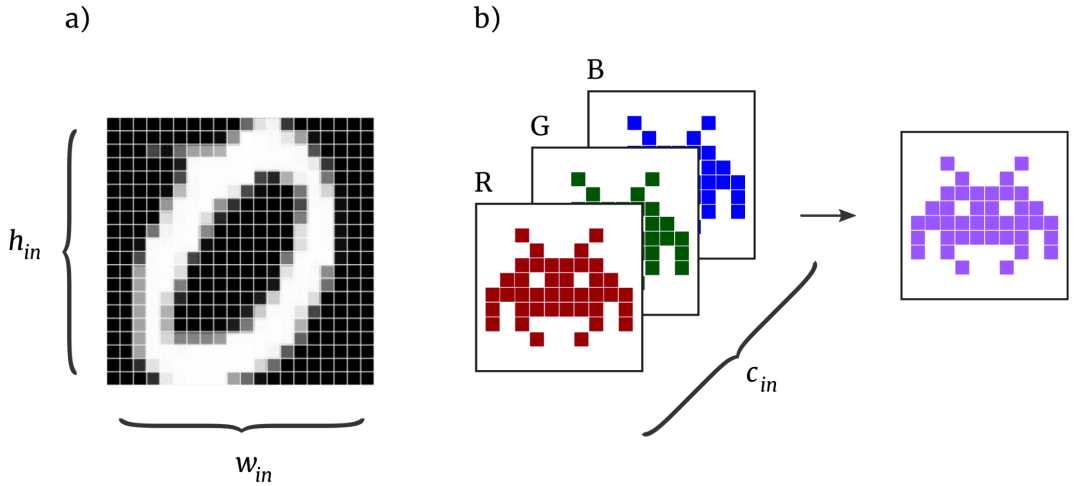

Fig. 7. Representation of grey and colour images on the computer. In (a) a single matrix is used to represent a grayscale image. Each matrix entry indicates the light intensity in that region of the image. The colour image in (b) uses three matrices, also called channels, that are added together to produce the final colour result. Each channel matrix represents the

intensity of red, blue, and green (RGB) respectively. Values between 0 and 255 can be used to represent the intensity of the primary colours and grey.

Consider $\mathbf{X}$ a matrix with dimensions $n_{\mathbf{X}} \times n_{\mathbf{X}}$ as a grayscale image. The convolution, in this context, corresponds to the application of a filter $\mathcal{F}$ over the image $\mathbf{X}$. In this case, $\mathcal{F}$ is also a matrix (typically squared) with dimensions $m_{\mathcal{F}} \times m_{\mathcal{F}}$, with $m_{\mathcal{F}} \leq n_{\mathbf{X}}$. The convolution operation is denoted $\mathcal{F} * \mathbf{X}$. In simple terms, what the convolution operation does is to "slide" the filter $\mathcal{F}$ over all the image space. At each step, $\mathcal{F}$ is multiplied component-wise by the values of the submatrix of $\mathbf{X}$ formed by stacking $\mathcal{F}$ over $\mathbf{X}$, and the results are summed. The result of each step is a single real value stored in an output matrix $\mathbf{Z}$, usually called a feature map. Fig. 8 illustrates the convolution operation on a 2D matrix.

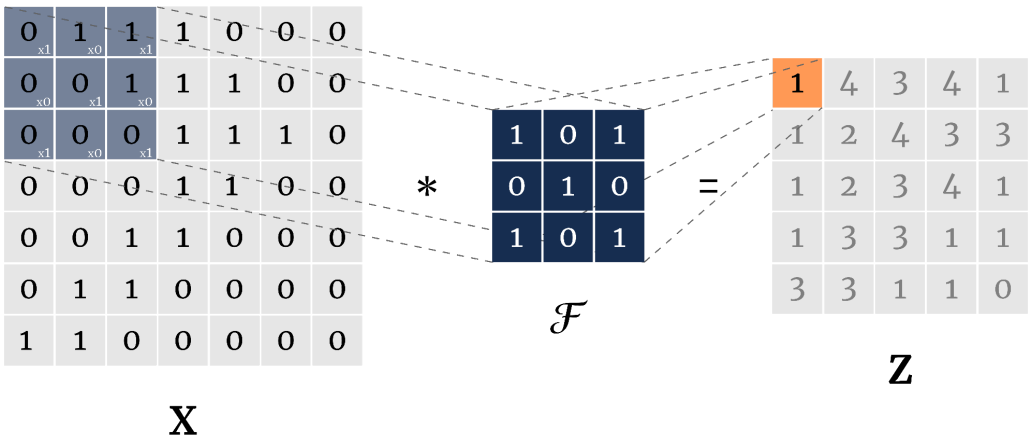

Fig. 8. Example of a 2D convolution operation. A single 3×3 filter $\mathcal{F}$ is applied to a 7×7 input matrix $\mathbf{X}$ to produce the 5×5 feature map $\mathbf{Z}$. This operation used a stride of 1 and no padding.

A convolutional layer has three important hyperparameters: padding, stride, and kernel size. The padding creates a border around $\mathbf{X}$ with a size defined by the user. Usually, the border is filled with zeros. This "empty" border is useful, for example, when we want the input and output dimensions after the transformation to match (since convolutions can downsample the input image).

The stride is the step size when moving the filter $\mathcal{F}$. A stride of 1 means the filter advances 1 pixel at each step. At last, the kernel size defines the dimensions of the filter. Moreover, a single convolutional layer can have multiple filters with the same dimensions, and their output is grouped as channels. The convolutional operation can also be generalised to tensors in higher dimensions, with filters having dimensions in accordance with the input.

During the training phase, the values in the filters are adjusted similarly to the neurons' weights in traditional feedforward NNs. Furthermore, activation functions (such as ReLUs and sigmoids) can also be applied to the feature maps.

Some hallmarks of convolutional layers for image processing include (Chollet, 2018): (1) translational invariance. Specific patterns can be recognized in different parts of the image since the same convolutional filter is applied in different regions. Another advantage is (2) a hierarchical representational relationship between different layers. As the different convolutional transformations are applied sequentially, increasingly more complex patterns can arise based on simpler ones identified in the first layers.

One downside to the CNNs — including in the case of using voxel-based representations of molecules (see section 5.1) — is their lack of rotational invariance with respect to the input (Jaderberg et al., 2016), *i.e.*, even though the underlying object contains the same information, it looks different to the network when it is rotated. To overcome this limitation, it is common to employ data augmentation strategies that rotate the input in different angles while training the neural network (Jiménez et al., 2018, Stepniewska-Dziubinska et al., 2018).

4.5 Graph Neural Networks

A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ can be defined as a set of nodes $\mathcal{V}$ and a set of edges $\mathcal{E}$ between those nodes. An edge between two nodes $u, v \in \mathcal{V}$ can be denoted as $(u, v) \in \mathcal{E}$. In the simplest case, a graph has always at most one edge between a pair of nodes, no edges between a node and itself, and all edges are undirected — *i.e.*, $(u, v) \leftrightarrow (v, u)$ (Hamilton, 2020).

Graphs are a very natural way of representing many objects and natural phenomena, such as social networks, maps and molecules. Moreover, graph nodes can be endowed with m-dimensional feature vectors. For example, in the case of using graphs to represent molecules, each node (representing an atom) could have features such as hydrogen bond

acceptor or donor and charge. It is also possible to endow edges with features; for example, the edges of a molecule graph could represent different bond types between atoms (Bronstein et al., 2021).

The typical predictive tasks we can perform on graphs include (1) node classification, (2) edge classification, and (3) graph classification or regression. In node classification, the objective is to predict a label associated with each node. These labels could be, *e.g.*, a type, category or attribute. For example, in an online social network graph, we may want to identify accounts (nodes) that are bots.

On the other hand, in edge classification, the goal is to infer information about the edges of the graph (for example, to infer missing edges between nodes). Finally, in graph classification or regression, the goal is to summarise information about the entire graph into classes or a single number. This task is the most direct analogue of standard supervised learning with NNs (Hamilton, 2020).

In order to apply NNs directly to graph-structured data, a new kind of deep learning architecture is needed — one that is broadly called graph neural network (GNN). The layers in GNNs apply transformations to the features $\boldsymbol{h}_u$ of each node u in the graph to generate new feature embeddings $\boldsymbol{h}'_u$ for these nodes. A particular property of these transformations is that they should be permutation invariant, meaning they must not depend on the order in which nodes and their neighbours are presented.

The basic idea behind a GNN transformation is to aggregate information from a node's local neighbourhood to update its feature vector, similarly to the CNN layer (that aggregates information from neighbouring pixels). The set of neighbours of a node $u \in \mathcal{V}$ is denoted as $\mathcal{N}_u = v \in \mathcal{V} : (u, v) \in \mathcal{E}$. Fig. 9 exemplifies a general GNN transformation applied to a node.

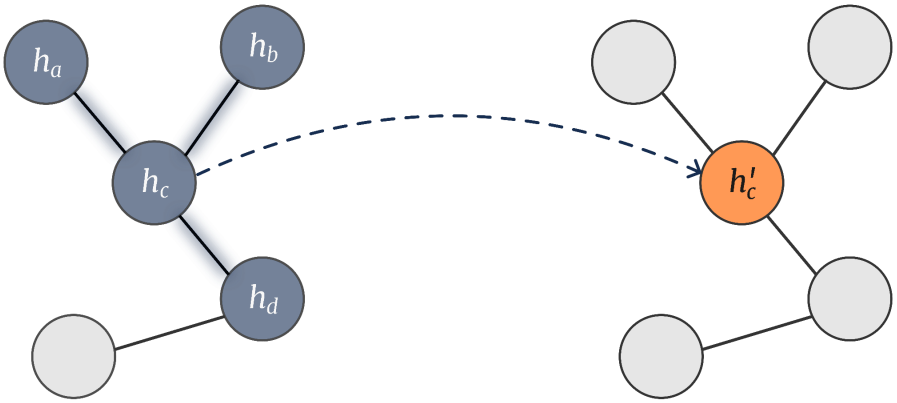

Fig. 9. Example a generic GNN transformation applied to an arbitrary graph. The node features of node $\boldsymbol{h}_c$ are updated based on its own feature vector and the features of its neighbouring nodes $\boldsymbol{h}_a$, $\boldsymbol{h}_b$, and $\boldsymbol{h}_d$. The resulting updated node features is denoted $\boldsymbol{h}'_c$.

One of the most common types of rules used to update the feature vector $\boldsymbol{h}_u$ of a node $u \in \mathcal{V}$ is called the graph convolutional layer (Kipf et al., 2016):

$$\boldsymbol{h}'_u = g(\mathbf{W} \sum_{v \in \mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u||\mathcal{N}_v|}} \boldsymbol{h}_v). \tag{18}$$

In the equation above, $\boldsymbol{h}'_u$ is the updated feature vector of node u, g is any nonlinearity (such as ReLU), $\mathbf{W}$ is a trainable weight matrix, $|\mathcal{N}_u|$ and $|\mathcal{N}h_v|$ represents the number of neighbours of u or v respectively, and $\boldsymbol{h}_v$ is the feature vector of the neighbour nodes $v \in \mathcal{N}_u$. Noteworthy, in Eq. 18 it is common to consider the summation over not just the set of neighbours $v \in \mathcal{N}_u$, but also the node u itself, *i.e.*, the set $\mathcal{N}_u \cup \{u\}$.

5 Molecular representations for Geometric Deep Learning

In order to successfully apply DL methods to molecular docking, we must be able to represent the protein-ligand structures appropriately. Different molecular representations are best suited for specific NN architectures (*e.g.*, graphs for GNNs and voxels for CNNs). Two

of the most widely employed classes of molecular representations in the scientific literature are voxel grids and graphs. The following sections briefly describe these approaches to molecular representation.

5.1 Voxel Grids

Voxel grids treat the molecular representation problem from a computer vision perspective. A *voxel* (volume element) represents a value on a regular grid in the three-dimensional space. A voxel grid is a 4D tensor with dimensions $w \times h \times d \times c$, where w is the width, h is the height, d is the depth, and c is the number of different channels (analogously to the representation of colour images). Voxel grids are suitable inputs for CNNs with 3D convolutional layers — a generalisation of the typical 2D convolutional layer (used for images) for 3D volumes.

In the case of biomolecules, channels can be used to represent different physicochemical properties of the atoms. Examples include channels representing atoms with specific properties such as hydrophobicity, hydrogen bond acceptor or donor, positive or negative ionizable, partial atomic charges, and metallic (Jiménez et al., 2018, Stepniewska-Dziubinska et al., 2018). Another example consists of each channel representing different chemical elements, such as carbon, hydrogen, oxygen, and nitrogen (Li et al., 2019, Ragoza et al., 2022).

In voxel grids, the space is discretized at regular intervals. The typical size for the intervals employed in the scientific literature is usually from 0.25 Å to 1.5 Å. The specific values that each voxel assumes depend on the occupancy model employed. The simplest case is to use a binary representation, where each voxel indicates the presence or absence of an atom (or atomic property) in that region. This type of occupancy usually leads to a very sparse grid. Fig. 10 illustrates a voxel grid with this type of occupancy.

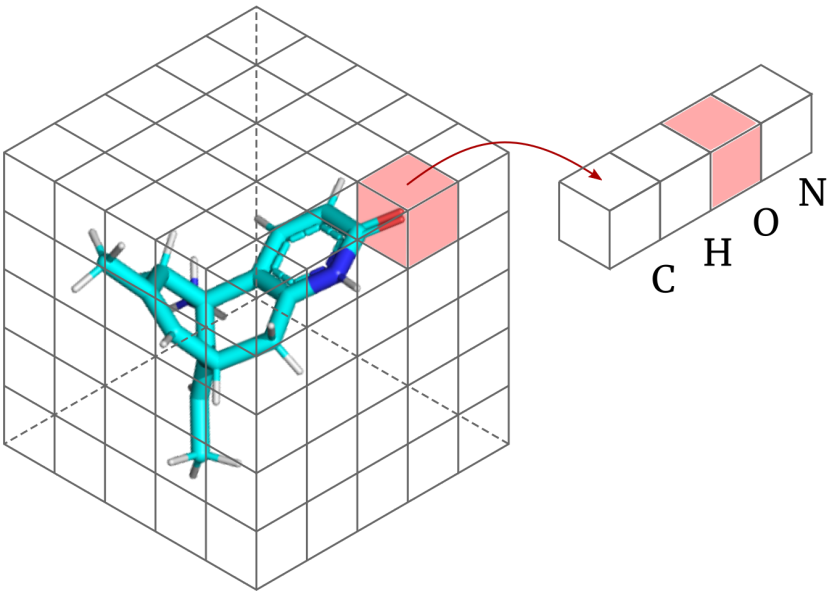

Fig. 10. Illustration of a voxel grid where voxels represent atoms of a given small molecule in the 3D space. In this example, each voxel indicates the presence of a different atom (carbon, oxygen, and nitrogen) in that region. Four separate channels are used to display each atom type. The red shaded block indicates which channel is being used in a particular grid position.

Another example of a (less sparse) occupancy model represents the volume of each atom by considering its van der Waals radius (Jiménez et al., 2018, McNutt et al., 2021, Li et al., 2019). In this model, the occupancy of each voxel can be calculated, for example, as:

$$v(d) = 1 - exp(-(\frac{r_{vdw}}{d})^{12}), \quad (19)$$

where d is the distance between an atom and a voxel and r_{vdw} is the van der Waals radius of that atom. Usually, the distances between all the atoms and the voxels in the grid are calculated, but only the maximum value of $v(d)$ for each voxel is kept to represent that voxels' value. For a given channel, only atoms present in that channel are considered. An example of grids built with this kind of occupancy is shown in Fig. 11. An example source code written in Python is also available[1] for generating and plotting voxel grids with this occupancy model.

[1] https://gist.github.com/mpds/058d5310f9e6a3b21d353d22599c6760

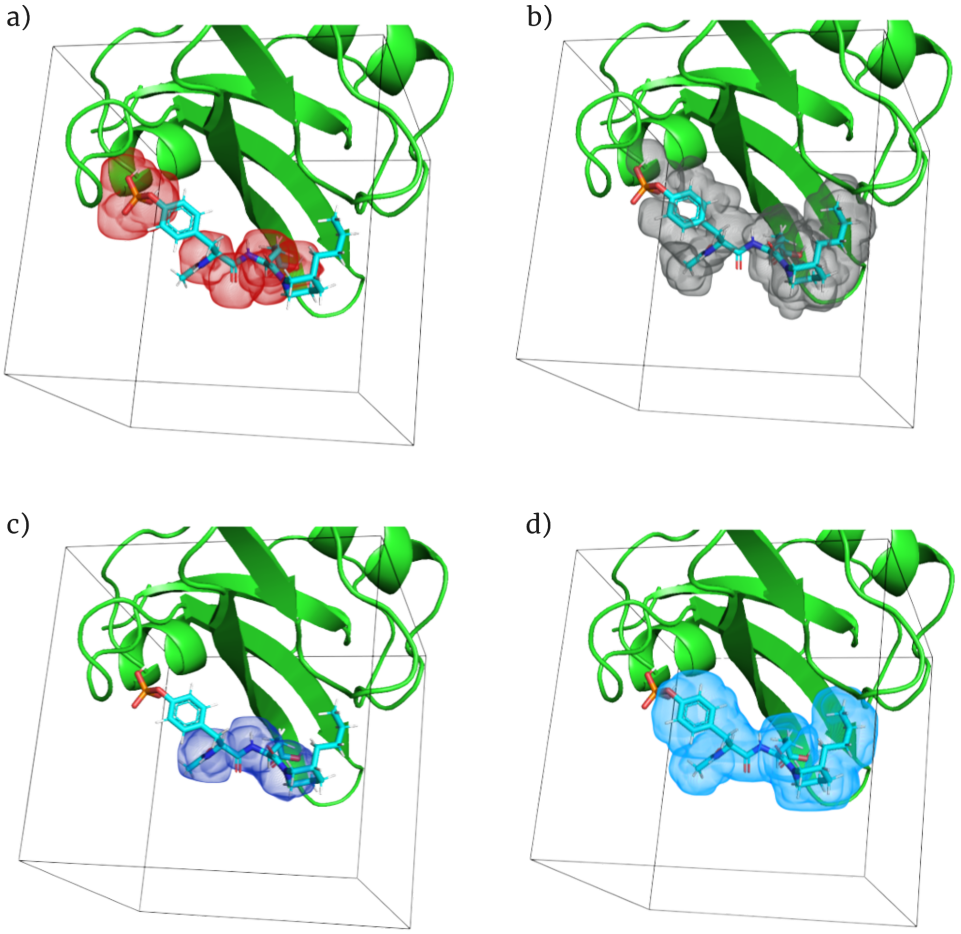

Fig. 11. Voxel-based representation of a protein-ligand complex. Grids a, b, c, and d represent oxygen, hydrogen, nitrogen, and carbon channels for the ligand only, respectively. Each grid shows only atoms belonging to that specific channel. The protein (green) and ligand (cyan) molecules are represented to aid the visualisation. For further information on related research using this type of representation, refer to Table 1.

Compared to graph-based molecular representations (section 5.2), voxel grids tend to represent volumes and protein cavities more straightforwardly. Additionally, current research on CNN architectures is relatively more extensive in image processing, and the knowledge gained there could be applied to the drug design particular field.

5.2 Graphs

As stated in section 4.5, a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ can be defined as a set of nodes $\mathcal{V}$ and a set of edges $\mathcal{E}$ between those nodes. Both nodes and edges can be endowed with feature

vectors describing relevant properties. Graph-based molecular representations have some advantages when compared to voxel grid-based representations. Most graph representations are invariant to translations and rotations in the input data (which is not the case for the voxel representation). Moreover, voxel grid representations tend to be computationally inefficient since grids are reasonably sparse, with most voxels carrying no information. Finally, training CNNs on voxel grids tends to be very demanding in terms of memory usage and GPU time, while GNNs are faster and require less memory to train (Volkov et al., 2022; Moon et al., 2022). In practice, however, for some tasks (such as binding affinity prediction), the performance of graph-based representations trained with GNNs does not significantly outperforms that of voxel-based CNNs (Volkov et al., 2022).

The construction of graph representations differs among the various implementations in the scientific literature (Isert et al., 2022). Edges can distinguish covalent bonds from intermolecular interactions between protein and ligand, usually using a distance threshold to differentiate them (Moon et al., 2022, Volkov et al., 2022, Jiang et al., 2021). Edges can also encode bond types, such as single, double, triple or aromatic bonds (Méndez-Lucio et al., 2021). Moreover, 3D information can be used as edge features, for example, bond lengths and statistics about angles and areas (Jiang et al., 2021, Volkov et al., 2022).

Regarding the node features, different atom types (*e.g.*, carbon, hydrogen, metals, etc.) can be used as one-hot encodings (Méndez-Lucio et al., 2021, Moon et al., 2022). Other features include hybridization, partial atomic charge, number of hydrogens and aromaticity (Lim et al., 2019, Moon et al., 2022). Fig. 12. illustrates a small molecule graph representation with generic node and edge features.

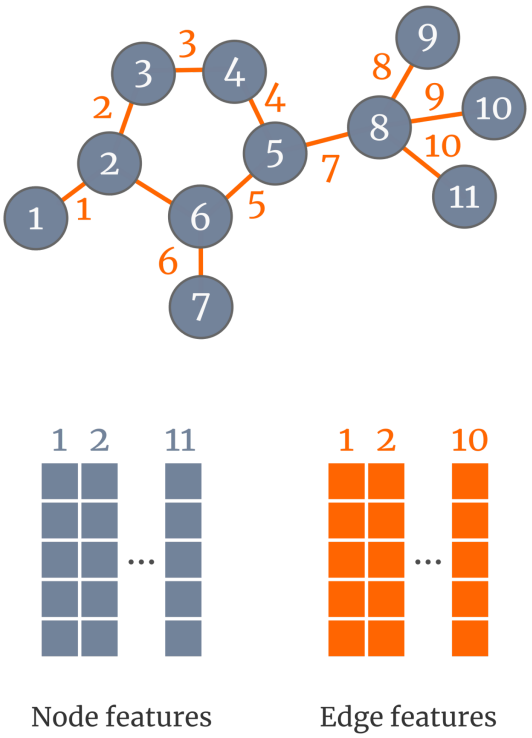

Fig. 12. Example of a graph used to represent a molecule. Each node represents a given atom, while the edges represent bonds between them. Both nodes (in grey) and edges (in orange) can be endowed with feature vectors (illustrated by the squares in the image) to describe the different characteristics of these components (atom types, bond types, *etc.*). For further information on related research using this type of representation, refer to Table 1.

6 Deep Learning Applications in Molecular Docking

This section will review selected papers from the scientific literature on deep learning techniques applied to molecular docking. Each subsection will describe approaches in the three main aspects of the docking methodology: pose prediction, binding affinity prediction and virtual screening, providing a brief overview of how these methods can contribute to the improvement of molecular docking.

5.3 Deep Learning for Pose Prediction

GNINA 1.0, a fork of the docking software called SMINA, is a molecular docking software that incorporates CNNs as an integral part of the docking workflow (McNutt et al.,

2021). GNINA uses CNN-based scoring functions trained on voxel-based molecular representations. The conformational search is performed using Monte Carlo simulation. The user can choose to employ CNN scoring functions at different steps of the optimization process, *i.e.*, just for rescoring poses at the end of the docking procedure; during a refinement step; or even for the entire docking pipeline (at the expense of more computational time). GNINA outperforms AutoDock Vina (Eberhardt et al., 2021) on redocking (evaluated using the PDBbind refined set v.2019) and cross-docking (evaluated using a benchmark proposed in Wierbowski et al., 2020) tasks, with the success rate considering the top 1 selected pose increasing from 58% to 73% (redocking) and from 27% to 37% (cross-docking).

The CNN scoring functions take as input voxel grid representations with a gaussian-like occupancy model to represent the density of each atom. In total, 28 different channels are used, with 14 separated channels for the ligand and another 14 channels for the protein only. Distinct atom types were used as channels, such as oxygen/nitrogen, hydrogen donor/acceptors, and aliphatic/aromatic carbons. A cubic grid with dimensions 23.5 and 0.5 Å discretization is used.

The user can select five different CNN models to be used within the program (including ensemble combinations of them): crossdock_default2018, dense, general_default2018, redock_default2018, and default2017. Each model is trained using different training data and/or a different model architecture (Francoeur et al., 2020). The parameter size of the models range from ~300k to ~1 million parameters. The default2017 model is the originally proposed CNN architecture, while the others were derived via an extensive hyperparameter search. All models consist of a series of 3D convolutional and/or pooling layers followed by two separate fully connected layers whose outputs are the pose score and affinity prediction. The pose score is the probability that the pose has a low RMSD (<2 Å) to the native binding pose. The affinity prediction, on the other hand, is the binding affinity given in pK units.

By default, the GNINA 1.0 software uses CNNs for rescoring the final poses obtained after the optimization and refinement processes. The default CNN scoring function is an ensemble of five different variations of the available models, which was shown to outperform all of the single models on both the redocking task and the cross-docking task. GNINA 1.0 is publicly available at https://github.com/gnina/gnina.

Another approach to pose prediction, called DeepDock, is based on GNNs for predicting the binding conformation of ligands to protein targets (Méndez-Lucio et al., 2021). DeepDock uses two graph representations, one for the protein and the other for the ligand.

The protein is represented by a collection of nodes and edges that define a polygon mesh of the binding pocket surface, as described by (Gainza et al., 2020). Basically, each node of the mesh is represented by a four-element feature vector with the following properties: poisson-boltzmann continuum electrostatics, free electrons and proton donors, hydropathy, and shape index. The edges, on the other hand, are represented by the relative distance between two connected nodes on the mesh. Only nodes within 10 Å or less from any ligand atom were considered.

Furthermore, the ligand is represented using a 2D graph where nodes and edges represent atom and bond types, respectively. Each ligand node uses a one-hot vector that indicates the atom type among 28 possibilities (*e.g.,* C, N, O, F, P, S, etc.). Similarly, the edges are endowed with feature vectors that indicate one of the bond types: single, double, triple or aromatic). Noteworthy is the fact that no 3D information of the ligand was used in this representation.

The model can be divided into three stages: feature extraction, feature concatenation and a mixed density network (MDN). Both the protein mesh and the ligand graph are processed by independent residual GNNs (with the same architecture). First, three GNN layers were used to update the nodes and edges embeddings, followed by 10 residual GNN blocks. In the following step, the processed node features from the target and ligand were combined, *i.e.*, all node features were concatenated in a pairwise manner, meaning each ligand atom was paired with each node in the protein mesh. Finally, these concatenated features were processed by the MDN. The MDN is a feed forward neural network that predicts a set of means, standard deviations and mixing coefficients needed to parametrize a mixture density model for each ligand-protein node pair. These learned probability density functions are aggregated into a single statistical potential that can be minimised to find the most likely distance between ligand atoms and the protein surface (*i.e.*, the correct binding pose).

The data used to train the model was the PDBbind dataset. In this work, differential evolution (DE) was employed as the search algorithm, but other types of algorithms could also be used, such as particle swarm optimization (PSO), simulated annealing (SA) or even gradient-based algorithms. The authors evaluated the docking and screening power of DeepDock using the CASF-2016 benchmark (285 protein-ligand complexes). The complexes from the CASF-2016 were not included in the training set. DeepDock was able to find conformations corresponding to a minimum for 225 of the compounds in the CASF-2016. Success rates among top 1 ranked poses (with an RMSD < 2 Å) were comparable or superior to other methods described in the literature, for both docking and

screening power tests. Noteworthy, optimization failed for most of the compounds with more than 10 rotatable bonds, due to inefficiency of the optimization algorithms when dealing with a large number of degrees of freedom. Finally, no correlation was found between the compound binding affinity and the score of the predicted or real binding pose produced by their model. This indicates that this method may not be suitable for binding affinity prediction tasks.

Finally, EquiBind is another GNN-based model proposed by (Stärk et al., 2022). Their method is developed considering the blind docking scenario (*i.e.*, no knowledge of the protein's binding site)—even though the method could be easily adapted to scenarios where the binding pocket is known. Furthermore, like other programs in the literature, they assume a rigid target and model only the ligand flexibility. The model directly predicts the protein-ligand complex structure without relying on population-based optimization methods. Therefore, each prediction happens significantly faster than other methods, taking less than 1 second.

Both ligand and protein are represented using spatial k-nearest neighbour graphs. The ligand graph uses atoms as nodes with features that include: atomic coordinates from the unbound conformer, atomic number, chirality, degree, formal charge, implicit valence, the number of connected hydrogens, hybridization type and more. The edges have two attributes: the interatomic distances encoded with Gaussian basis functions and local frame orientation encodings. On the other hand, the protein graph uses residues as nodes (their coordinates given by the α-carbon location), and also the residue type as a feature. Each node is connected in the graph to the closest 10 other nodes at less than 30 Å distance. The edges are represented similarly to the ligand graph.

The authors used Independent E(3)-Equivariant Graph Matching Networks, which jointly transforms both features and 3D coordinates to perform intra and inter neural graph message passing. The core property of this GNN architecture is that stacking any number of such layers guarantees equivariance, *i.e.*, that any independent rotation and translation of the original input structures will be exactly reflected in the outputs. The model achieves the final pose by predicting the translation and rigid rotation transformations that dock the ligand in the proper position and orientation. The model is also trained to predict changes to the coordinates of the ligand atoms to account for ligand flexibility. More specifically, ligand flexibility is accounted for by first predicting an atomic point cloud of the deformed molecule and then performing a fast algorithm procedure to infer changes in torsion angles that would match the point cloud as well as possible.

The dataset used was the PDBbind v2020, with a temporal split schema for defining training, validation and test sets. The developed method was compared with regard to its redocking capabilities to other docking programs described in the scientific literature, achieving comparable or superior results with lower CPU/GPU execution times. For example, while GNINA (McNutt et al., 2021) took on average 146 seconds to dock a single ligand-receptor pair, EquiBind took only 0.02 seconds on average in the same computational environment.

5.4 Deep Learning for Binding Affinity Prediction

KDEEP is a CNN-based scoring function for predicting the absolute binding affinity of protein-ligand complexes (Jiménez et al., 2018). The model uses a voxel-based molecular representation with a 24 Å³ ligand-centred grid and 16 channels, with separated channels for protein and ligand, respectively (same 8 channels for each). The channels account for different atomic properties, such as aromaticity, hydrophobicity, positive and negative formal charge and metallic atoms. The occupancy model considers the van der Waals radius to model the atomic volume. Since CNNs lack rotational invariance, 90° rotations in the input during the training step were used for data augmentation to reduce this problem and improve model generalizability.

The model was trained using the PDBbind v2016 refined set. The model architecture was inspired by SqueezeNet (Iandola et al., 2016), with convolution layers followed by max and average pooling operations, ReLU activations and a final linear dense layer. The total number of trainable parameters used was 1,340,769. The KDEEP model achieves a Pearson's correlation coefficient (R) of 0.82 on the PDBbind core set v. 2016 (N = 290), and it is freely available through the web portal https://playmolecule.com/Kdeep/.

Pafnucy is a CNN-based scoring function for protein-ligand binding affinity estimation (Stepniewska-Dziubinska et al., 2018). Pafnucy uses 19 channels to describe the protein-ligand complex. These channels describe features such as atom types (carbon, nitrogen, phosphorus, etc.), atom hybridization, hydrophobicity, aromaticity, etc. In contrast to the KDEEP model, Pafnucy uses a smaller grid with 20 Å³ centred at the ligand, with a 1 Å discretization. Moreover, the occupancy model considers the atoms as points in the grid instead of their volumes.

The Pafnucy model was trained using the PDBbind v2016 general set. The model uses three convolutional layers with 64, 128, and 256 filters. Each layer has a 5x5x5 filter and is followed by a 2x2x2 max pooling layer. The activation function used was the ReLU.

The convolutional layers are followed by three fully-connected layers of 1000, 500 and 200 neurons each. In order to reduce overfitting, dropout was used in all dense layers with a probability of 0.5. Also, L2 normalisation was employed. Moreover, 90° rotations in the input structures are adopted for data augmentation. Pafnucy achieved R = 0.78 on the PDBbind core set v. 2016 (N = 290), and its source code is publicly available at https://gitlab.com/cheminfIBB/pafnucy/.

PaxNet (Zhang et al., 2022) is a fast and memory-efficient GNN-based scoring function to predict binding affinities and other molecular properties such as dipole moment, isotropic polarizability, internal energy, enthalpy and others. The authors propose a novel framework that models different local (intra) and non-local (inter) molecular interactions using a set of multiple graphs, called a *multiplex*. Basically, a multiplex graph consists of multiple types of edges among a set of nodes, where each set of nodes and edges forms a graph (or layer). Different message passing schemes are used to handle different kinds of interactions. In this work, two graphs are used for each type of molecular interaction considered, *i.e.*, local and global interactions respectively. The local plex considers only local, covalent interactions, while the global layer includes non-covalent interactions. These graphs are processed by different networks and later are fused together to be used for downstream tasks. For the binding affinity prediction task, the model was trained using the PDBbind v2016. The PaxNet model achieved R = 0.815 on the PDBbind core set v. 2016 (N = 290).

5.5 Deep Learning Applied to Virtual Screening

With the recent increase in the availability of ultra-large virtual chemical libraries (beyond billions of molecules), new computational methodologies are necessary to explore these databases efficiently (see section 2.4 for more information on ultra-large libraries). Graff et al. developed a surrogate model strategy called MolPAL using Bayesian optimization to decrease the computational cost of docking in identifying top-scoring compounds in virtual libraries. Different types of machine learning and deep learning algorithms (including neural networks and GNNs) were used as surrogate models to predict the objective function value for molecules that had not been docked. Also, different acquisition strategies were evaluated to iteratively select which docking experiments to perform based on the prediction of the surrogate model. The authors evaluated MolPAL using the Enamine 10k, Enamine 50k, and Enamine HTS (2.1 million compounds) data sets. Besides that, they also used two datasets of 99 million and 138 million compounds, respectively, as described in another study (Lyu et al., 2019). The results demonstrated that MolPAL could retrieve more than 70% of the

top-scoring compounds while docking less than 6% of the total library in most cases. MolPAL software is available at https://github.com/coleygroup/molpal.

Gentile et al. developed Deep Docking, a virtual screening protocol that utilises deep neural network models. The models are trained in small increments with explicitly docked compounds, allowing the system to rank the yet-undocked remainder of the library. This strategy efficiently eliminates unfavourable (undockable) molecular structures while conserving computational resources. Deep Docking uses a ligand-base model and represents input molecules using a circular binary Morgan fingerprint of size 1024 bits. The protocol has four iterative steps: (1) ligand preparation, where molecules from the whole database are prepared, and fingerprints are computed; (2) docking experiments, where a reasonably sized training subset is randomly sampled from the database and docked into the target of interest using any molecular docking software (authors used FRED); (3) the deep learning model is trained using the selected sample and their respective docking score; (4) the trained deep learning model is used to predict the docking outcomes of the rest of the database and a predefined number of molecules considered hits (based on their predicted score) are randomly selected and used for training set augmentation. Steps 2-4 are repeated until a predefined number of iterations is reached. The authors used the Deep Docking protocol to screen the ZINC15 database (1.36 billion molecules) against 12 proteins representing four important families of drug targets. The method is capable of retrieving 90% of the best-scoring structures by docking only 1% of the library. Deep Docking is publicly available at https://github.com/jamesgleave/DD_protocol.

Zhang et al. applied their previously developed deep learning model DFCNN, a binary classifier of affinity prediction, to perform large-scale virtual screening using 102 protein targets from the DUD-E dataset (Mysinger et al., 2012) and also a constructed dataset of 10,402,895 compounds. Moreover, they employed this methodology to successfully find five novel active compounds for the target Trypsin I Protease validated experimentally. Since their model only uses sequence information from the ligand and protein, each prediction takes only 2.25×10^{-5} seconds with the computer setup defined by the authors, which is tens of thousands of times faster than AutoDock Vina (Eberhardt et al., 2021), enabling the virtual screening of large compound libraries in a more feasible time. DFCNN has a success rate in selecting active compounds 22 times greater than random selection, being an interesting approach to be used as a filtering strategy to reduce large-scale datasets before docking-based virtual screening or molecular dynamics simulations. A web portal for performing virtual screening using this methodology is available at http://cbblab.siat.ac.cn/DFCNN/.

7 Critical Aspects and Challenges

Despite the promising results and improvements achieved by DL applied to structure-based drug design, several challenges and critical aspects still need to be addressed by these methods.

A series of recent publications have pointed out that DL methods designed for binding affinity prediction and virtual screening predominantly rely on hidden biases in training and test data sets, resulting in overly optimistic performance estimates (Smusz et al., 2013, Wallach et al., 2018, Chen et al., 2019, Sieg et al., 2019, Kanakala et al., 2023). In particular, some findings have shown that CNNs and GNNs trained on protein-ligand complexes could distinguish active from inactive compounds when given only the ligand or protein structure alone. These results suggest that these methods are learning differences between properties of actives and decoys rather than meaningful features of the protein-ligand interactions. When tested on more extensive and challenging external test sets, these methods show a significant decrease in performance, indicating the need for more generalizability (Yang et al., 2020, Scantlebury et al., 2020, Francoeur et al., 2020, Volkov et al., 2022).

Some authors have identified hidden bias in popular docking benchmarks, such as the DUD-E (Mysinger et al., 2004), that arise due to several factors. The *analogue bias* refers to the situation in which active compounds of the same target, homologous, or targets with similar functionality, tend to be correlated in chemical space. Models could learn these correlations to identify active compounds. Another type of bias identified is the *decoy bias*. Most decoy libraries employ specific rules for generating presumably inactive compound instances (named *decoys*). These criteria stipulate, for example, that decoys must have similar properties to active compounds but differ topologically. However, deep learning models could uncover these rules, and decoys could be distinguished solely based on such patterns (Chen et al., 2019). *Inductive bias* arises because experiments are not designed to sample the chemical space uniformly. For example, certain compounds are excluded from drug discovery campaigns for reasons such as high costs, synthetic feasibility and availability. Also, chemists often deliberately construct new molecules that resemble other molecules with desirable properties, which means that the explored chemical space is expanded in regions very close to known binders, instead of uniformly (Sieg et al., 2019). These characteristics can lead to the construction of *biased* data sets.

Another critical but often overlooked limitation of ML methods for molecular property prediction (such as binding affinity) is the performance of these methods on activity cliffs, which are especially important in virtual screening. Activity cliffs are “pairs of molecules that are highly similar in structure but exhibit large differences in potency” (van Tilborg et al., 2022). Arguably, models that exhibit better predictive performance on activity cliffs are overall better, as they capture the underlying structure-activity relationships more accurately. A recent study (van Tilborg et al., 2022) evaluated several machine learning methods, including deep learning, based on strings or graphs. Overall, all methods showed poor performances on activity cliff compounds. Moreover, DL methods, in particular, did not surpass shallow ML methods for binding affinity prediction on affinity cliffs. Although the study did not address structured-based approaches, these results highlight the importance of developing ML methods that adequately capture structure-activity relationships.

The recent availability of ultra-large virtual chemical libraries with billions of compounds bring out the limitations in terms of computational performance of current docking software, which typically operates on the scale of millions of molecules at most (Gentile et al., 2022, Grygorenko et al., 2020). Only a few billion-sized docking campaigns have been conducted until now, usually with the support of elite supercomputing facilities (Gorgulla et al., 2020, Acharya et al., 2020). In order to take advantage of the increasingly more extensive virtual libraries, it will be necessary to develop methods capable of exploring these large virtual spaces efficiently and in a less wasteful manner (many molecules are undockable and discarded in the process). Studies in the scientific literature have employed DL to develop faster docking methodologies (Stärk et al., 2022, Masters et al., 2022) and to decrease the number of total docked compounds while retaining top-scoring ones (Gentile et al., 2022, Graff et al., 2021).

It is important to note that these limitations do not mean DL methods trained on currently available data sets cannot be used in practical applications. However, it is crucial to be aware of such limitations and use performance metrics on benchmarks with reservations. Volkov et al. highlight some critical aspects that should be considered when developing and using DL methods for structure-based drug design:

1. Is the performance of the algorithm *biased* by the chosen descriptors or the protein-ligand training space?
2. Does the model generalise well to external test sets?
3. Does the model achieve good predictions for meaningful reasons?

Since deep learning methods are considered "black boxes," it can be difficult to understand the specific reasons behind a given prediction (Sieg et al., 2019). Some researchers propose incorporating physics-based principles into DL models to take advantage of already established knowledge about the physics of binding events and protein-ligand interactions (Meli et al., 2022, Zhang et al., 2022). Developing tools, techniques, and data sets that can help prevent or reveal bias and limitations are essential for improving our understanding of effectively using and developing more meaningful DL methods in this field.

8 Concluding Remarks

The structure-based computer-aided drug discovery is an exciting and fast-developing scientific field. The rise in DL applications in recent years has revolutionised many aspects and methodologies of the field, bringing improvements over traditional approaches. In particular, DL methods have been applied in all stages of protein-ligand molecular docking, from pose prediction to virtual screening campaigns.

Publications in the scientific literature have shown DL-based strategies enabling faster docking protocols, more accurate scoring functions and the virtual screening of ultra-large virtual libraries of compounds (a strategy that was not computationally viable until recently).

Two of the most prominent types of deep neural networks used in structure-based prediction tasks are CNNs and GNNs. CNNs usually take as inputs the molecular structure of the protein-ligand complex represented by a voxel grid with multiple channels. These channels can encode molecular properties such as atom types and volume. CNNs, however, are not invariant to rotations on the input structures. GNN models, on the other hand, use graphs as a straightforward way of representing molecules and their properties. Besides, GNNs are usually permutation invariant or equivariant, which are valuable characteristics when working with 3D structural data.

Despite these advances, several challenges remain, and essential issues must be addressed to consolidate the success of molecular docking in developing new therapeutic compounds. One of the main challenges to further develop DL applications is the need for larger and more diverse training sets with high-quality data. DL models are prone to overfitting and can capture biases in the data that are difficult to anticipate and/or eliminate. These limitations can result in biased models with overestimated performances that do not perform as expected in real-world scenarios.

Perspectives to solve these problems include directing more efforts into collecting structural and experimental data, with particular attention to diversification and inclusion of inactive compounds (since most of the structural data available are from known active compounds). Developing robust ways to inquire and uncover hidden model biases is also essential. Finally, some research indicates that incorporating physics-based principles into machine learning models may reduce bias and improve the generalizability and interpretability of predictions.

Acknowledgments

The authors thank the Conselho Nacional de Desenvolvimento Científico e Tecnológico − CNPq (grant 309744/2022-9), and the Fundação Carlos Chagas Filho de Amparo à

Pesquisa do Estado do Rio de Janeiro - FAPERJ (E-26/010.001415/2019, E-26/211.357/2021, E-26/210.372/2022, E-26/200.608/2022 and E-26/200.393/2023) for Financial Support.

References

Schlander, M., Hernandez-Villafuerte, K., Cheng, C. Y., Mestre-Ferrandiz, J., & Baumann, M. (2021). How much does it cost to research and develop a new drug? A systematic review and assessment. *PharmacoEconomics*, 39(11), 1243-1269.

Guedes, I. A., de Magalhães, C. S., & Dardenne, L. E. (2014). Receptor–ligand molecular docking. *Biophysical reviews*, 6(1), 75-87.

Ferreira, L. G., Dos Santos, R. N., Oliva, G., & Andricopulo, A. D. (2015). Molecular docking and structure-based drug design strategies. *Molecules*, 20(7), 13384-13421.

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., ... & Hassabis, D. (2021). Highly accurate protein structure prediction with AlphaFold. *Nature*, *596*(7873), 583-589.

Anderson, A. C. (2003). The process of structure-based drug design. *Chemistry & biology*, 10(9), 787-797.

Ros, V., Biroli, G., & Cammarota, C. (2021). Dynamical instantons and activated processes in mean-field glass models. SciPost Physics, 10(1), 002.

Jones, G., Willett, P., Glen, R. C., Leach, A. R., & Taylor, R. (1997). Development and validation of a genetic algorithm for flexible docking. Journal of molecular biology, 267(3), 727-748.

Magalhães, C. S. D., Barbosa, H. J., & Dardenne, L. E. (2004). A genetic algorithm for the ligand-protein docking problem. Genetics and Molecular Biology, 27, 605-610.

Friesner, R. A., Banks, J. L., Murphy, R. B., Halgren, T. A., Klicic, J. J., Mainz, D. T., ... & Shenkin, P. S. (2004). Glide: a new approach for rapid, accurate docking and scoring. 1. Method and assessment of docking accuracy. Journal of medicinal chemistry, 47(7), 1739-1749.

Ewing, T. J., Makino, S., Skillman, A. G., & Kuntz, I. D. (2001). DOCK 4.0: search strategies for automated molecular docking of flexible molecule databases. Journal of computer-aided molecular design, 15(5), 411-428.

Wu, G., Robertson, D. H., Brooks III, C. L., & Vieth, M. (2003). Detailed analysis of grid-based molecular docking: A case study of CDOCKER—A CHARMm-based MD docking algorithm. Journal of computational chemistry, 24(13), 1549-1562.

Parenti, M. D., & Rastelli, G. (2012). Advances and applications of binding affinity prediction methods in drug discovery. Biotechnology advances, 30(1), 244-250.

Cournia, Z., Allen, B., & Sherman, W. (2017). Relative binding free energy calculations in drug discovery: recent advances and practical considerations. Journal of chemical information and modelling, 57(12), 2911-2937.

Foloppe, N., & Hubbard, R. (2006). Towards predictive ligand design with free-energy based computational methods?. Current medicinal chemistry, 13(29), 3583-3608.

Woo, H. J., & Roux, B. (2005). Calculation of absolute protein–ligand binding free energy from computer simulations. Proceedings of the National Academy of Sciences, 102(19), 6825-6830.

Genheden, S., & Ryde, U. (2015). The MM/PBSA and MM/GBSA methods to estimate ligand-binding affinities. Expert opinion on drug discovery, 10(5), 449-461.

Lyne, P. D. (2002). Structure-based virtual screening: an overview. *Drug discovery today*, 7(20), 1047-1055.

Maia, E. H. B., Assis, L. C., De Oliveira, T. A., Da Silva, A. M., & Taranto, A. G. (2020). Structure-based virtual screening: from classical to artificial intelligence. Frontiers in chemistry, 8, 343.

Kimber, T. B., Chen, Y., & Volkamer, A. (2021). Deep learning in virtual screening: recent applications and developments. International Journal of Molecular Sciences, 22(9), 4435.

Gentile, F., Agrawal, V., Hsing, M., Ton, A. T., Ban, F., Norinder, U., ... & Cherkasov, A. (2020). Deep docking: a deep learning platform for augmentation of structure based drug discovery. ACS central science, 6(6), 939-949.

Gentile, F., Yaacoub, J. C., Gleave, J., Fernandez, M., Ton, A. T., Ban, F., ... & Cherkasov, A. (2022). Artificial intelligence–enabled virtual screening of ultra-large chemical libraries with deep docking. Nature Protocols, 17(3), 672-697.

Guedes, I. A., Pereira, F. S., & Dardenne, L. E. (2018). Empirical scoring functions for structure-based virtual screening: applications, critical aspects, and challenges. *Frontiers in pharmacology*, *9*, 1089.

Meli, R., Morris, G. M., & Biggin, P. (2022). Scoring functions for protein-ligand binding affinity prediction using structure-based deep learning: A review. *Frontiers in bioinformatics*, 57.

de Magalhães, C. S., Almeida, D. M., Barbosa, H. J. C., & Dardenne, L. E. (2014). A dynamic niching genetic algorithm strategy for docking highly flexible ligands. *Information Sciences*, 289, 206-224.

Velec, H. F., Gohlke, H., & Klebe, G. (2005). DrugScoreCSD knowledge-based scoring function derived from small molecule crystal data with superior recognition rate of

near-native ligand poses and better affinity prediction. *Journal of medicinal chemistry*, 48(20), 6296-6303.

Muegge, I. (2006). PMF scoring revisited. *Journal of medicinal chemistry*, 49(20), 5895-5902.

Guedes, I. A., Barreto, A., Marinho, D., Krempser, E., Kuenemann, M. A., Sperandio, O., ... & Miteva, M. A. (2021). New machine learning and physics-based scoring functions for drug discovery. *Scientific reports*, 11(1), 1-19.

Wójcikowski, M., Ballester, P. J., & Siedlecki, P. (2017). Performance of machine-learning scoring functions in structure-based virtual screening. Scientific Reports, 7(1), 1-10.

Khamis, M., Gomaa, W., & Galal, B. (2016). Deep learning is competing random forest in computational docking. arXiv preprint arXiv:1608.06665.

Wang, C., & Zhang, Y. (2017). Improving scoring‐docking‐screening powers of protein–ligand scoring functions using random forest. Journal of computational chemistry, 38(3), 169-177.

Wang, R., Fang, X., Lu, Y., & Wang, S. (2004). The PDBbind database: Collection of binding affinities for protein− ligand complexes with known three-dimensional structures. *Journal of medicinal chemistry*, 47(12), 2977-2980.

Liu, Z., Li, Y., Han, L., Li, J., Liu, J., Zhao, Z., ... & Wang, R. (2015). PDB-wide collection of binding data: current status of the PDBbind database. *Bioinformatics*, 31(3), 405-412.

Su, M., Yang, Q., Du, Y., Feng, G., Liu, Z., Li, Y., & Wang, R. (2018). Comparative assessment of scoring functions: the CASF-2016 update. *Journal of chemical information and modeling*, 59(2), 895-913.

Mysinger, M. M., Carchia, M., Irwin, J. J., & Shoichet, B. K. (2012). Directory of useful decoys, enhanced (DUD-E): better ligands and decoys for better benchmarking. *Journal of medicinal chemistry*, 55(14), 6582-6594.

Adeshina, Y. O., Deeds, E. J., & Karanicolas, J. (2020). Machine learning classification can reduce false positives in structure-based virtual screening. Proceedings of the National Academy of Sciences, 117(31), 18477-18488.

Stein, R. M., Yang, Y., Balius, T. E., O'Meara, M. J., Lyu, J., Young, J., ... & Irwin, J. J. (2021). Property-unmatched decoys in docking benchmarks. Journal of chemical information and modeling, 61(2), 699-714.

Imrie, F., Bradley, A. R., & Deane, C. M. (2021). Generating property-matched decoy molecules using deep learning. Bioinformatics, 37(15), 2134-2141.

Zhang, X., Shen, C., Liao, B., Jiang, D., Wang, J., Wu, Z., ... & Hou, T. (2022). TocoDecoy: a new approach to design unbiased datasets for training and benchmarking machine-learning scoring functions. Journal of Medicinal Chemistry, 65(11), 7918-7932.

Crunkhorn, S. (2022). Screening ultra-large virtual libraries. Nature reviews. Drug Discovery.

Sadybekov, A. A., Sadybekov, A. V., Liu, Y., Iliopoulos-Tsoutsouvas, C., Huang, X. P., Pickett, J., ... & Katritch, V. (2022). Synthon-based ligand discovery in virtual libraries of over 11 billion compounds. Nature, 601(7893), 452-459.

Warr, W. A., Nicklaus, M. C., Nicolaou, C. A., & Rarey, M. (2022). Exploration of ultralarge compound collections for drug discovery. Journal of Chemical Information and Modeling, 62(9), 2021-2034.

Enamine REAL. http://enamine.net/library-synthesis/real-compounds/real-database (accessed 2023-04-04).

Irwin, J. J., Tang, K. G., Young, J., Dandarchuluun, C., Wong, B. R., Khurelbaatar, M., ... & Sayle, R. A. (2020). ZINC20—a free ultralarge-scale chemical database for ligand discovery. Journal of chemical information and modeling, 60(12), 6065-6073.

Ruddigkeit, L., Van Deursen, R., Blum, L. C., & Reymond, J. L. (2012). Enumeration of 166 billion organic small molecules in the chemical universe database GDB-17. Journal of chemical information and modeling, 52(11), 2864-2875.

Patel, H., Ihlenfeldt, W. D., Judson, P. N., Moroz, Y. S., Pevzner, Y., Peach, M. L., ... & Nicklaus, M. C. (2020). SAVI, in silico generation of billions of easily synthesizable compounds through expert-system type rules. Scientific data, 7(1), 384.

Crampon, K., Giorkallos, A., Deldossi, M., Baud, S., & Steffenel, L. A. (2022). Machine-learning methods for ligand–protein molecular docking. Drug discovery today, 27(1), 151-164

Zhang, H., Lin, X., Wei, Y., Zhang, H., Liao, L., Wu, H., ... & Wu, X. (2022). Validation of Deep Learning-Based DFCNN in Extremely Large-Scale Virtual Screening and Application in Trypsin I Protease Inhibitor Discovery. Frontiers in Molecular Biosciences, 9.

McNutt, A. T., Francoeur, P., Aggarwal, R., Masuda, T., Meli, R., Ragoza, M., ... & Koes, D. R. (2021). GNINA 1.0: molecular docking with deep learning. *Journal of cheminformatics*, 13(1), 1-20.

Méndez-Lucio, O., Ahmad, M., del Rio-Chanona, E. A., & Wegner, J. K. (2021). A geometric deep learning approach to predict binding conformations of bioactive molecules. *Nature Machine Intelligence*, 3(12), 1033-1039.

Stärk, H., Ganea, O., Pattanaik, L., Barzilay, R., & Jaakkola, T. (2022, June). Equibind: Geometric deep learning for drug binding structure prediction. *In International Conference on Machine Learning* (pp. 20503-20521). PMLR.

Corso, G., Stärk, H., Jing, B., Barzilay, R., & Jaakkola, T. (2022). DiffDock: Diffusion Steps, Twists, and Turns for Molecular Docking. *arXiv preprint* arXiv:2210.01776.

Masters, M., Mahmoud, A. H., Wei, Y., & Lill, M. A. (2022). Deep learning model for flexible and efficient protein-ligand docking. In ICLR2022 Machine Learning for Drug Discovery.

Lu, W., Wu, Q., Zhang, J., Rao, J., Li, C., & Zheng, S. (2022). TANKBind: Trigonometry-Aware Neural NetworKs for Drug-Protein Binding Structure Prediction. *bioRxiv*.

Ragoza, M., Hochuli, J., Idrobo, E., Sunseri, J., & Koes, D. R. (2017). Protein–ligand scoring with convolutional neural networks. *Journal of chemical information and modeling*, 57(4), 942-957.

Jiménez, J., Skalic, M., Martinez-Rosell, G., & De Fabritiis, G. (2018). Kdeep: protein–ligand absolute binding affinity prediction via 3d-convolutional neural networks. *Journal of chemical information and modeling*, 58(2), 287-296.

Stepniewska-Dziubinska, M. M., Zielenkiewicz, P., & Siedlecki, P. (2018). Development and evaluation of a deep learning model for protein–ligand binding affinity prediction. *Bioinformatics*, 34(21), 3666-3674.

Li, S., Zhou, J., Xu, T., Huang, L., Wang, F., Xiong, H., ... & Xiong, H. (2021). Structure-aware interactive graph neural networks for the prediction of protein-ligand binding affinity. I*n Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining* (pp. 975-985).

Zhang, S., Liu, Y., & Xie, L. (2022). Efficient and Accurate Physics-aware Multiplex Graph Neural Networks for 3D Small Molecules and Macromolecule Complexes. *arXiv preprint* arXiv:2206.02789.

Moon, S., Zhung, W., Yang, S., Lim, J., & Kim, W. Y. (2022). PIGNet: a physics-informed deep learning model toward generalized drug–target interaction predictions. Chemical Science, 13(13), 3661-3673.

Lyu, J., Wang, S., Balius, T. E., Singh, I., Levit, A., Moroz, Y. S., ... & Irwin, J. J. (2019). Ultra-large library docking for discovering new chemotypes. Nature, 566(7743), 224-229.

Graff, D. E., Shakhnovich, E. I., & Coley, C. W. (2021). Accelerating high-throughput virtual screening through molecular pool-based active learning. *Chemical science*, 12(22), 7866-7881.

Pereira, J. C., Caffarena, E. R., & Dos Santos, C. N. (2016). Boosting docking-based virtual screening with deep learning. *Journal of chemical information and modeling*, 56(12), 2495-2506.

Kalliokoski, T. (2021). Machine Learning Boosted Docking (HASTEN): An Open‐source Tool To Accelerate Structure‐based Virtual Screening Campaigns. Molecular Informatics, 40(9), 2100089.

Burkov, A. (2019). The hundred-page machine learning book (Vol. 1, p. 32). Quebec City, QC, Canada: Andriy Burkov.

Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. *MIT press*.

Géron, A. (2017). Hands-on machine learning with scikit-learn and tensorflow: Concepts. Tools, and Techniques to build intelligent systems.

Chollet, F. (2018). Deep learning with Python. *Simon and Schuster*.

Mehta, P., Bukov, M., Wang, C. H., Day, A. G., Richardson, C., Fisher, C. K., & Schwab, D. J. (2019). A high-bias, low-variance introduction to machine learning for physicists. *Physics reports*, 810, 1-124.

Dietterich, T. G., & Kong, E. B. (1995). Machine learning bias, statistical bias, and statistical variance of decision tree algorithms (pp. 0-13). Technical report, Department of Computer Science, Oregon State University.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. *MIT press*.

Ng, A. (2017). Machine learning yearning. URL: http://www.mlyearning.org/.

Nielsen, M. A. (2015). Neural networks and deep learning (Vol. 25). San Francisco, CA, USA: *Determination press*.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint* arXiv:1412.6980.

Graves, A. (2013). Generating sequences with recurrent neural networks. *arXiv preprint* arXiv:1308.0850.

Ioffe, S., & Szegedy, C. (2015, June). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning* (pp. 448-456). PMLR.

Jaderberg, M., Simonyan, K., & Zisserman, A. (2015). Spatial transformer networks. *Advances in neural information processing systems*, *28*.

Hamilton, W. L. (2020). Graph representation learning. *Morgan & Claypool Publishers*.

Bronstein, M. M., Bruna, J., Cohen, T., & Veličković, P. (2021). Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint* arXiv:2104.13478.

Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint* arXiv:1609.02907.

Li, Y., Rezaei, M. A., Li, C., & Li, X. (2019, November). DeepAtom: a framework for protein-ligand binding affinity prediction. In 2019 IEEE International Conference on Bioinformatics and Biomedicine (BIBM) (pp. 303-310). IEEE.

Ragoza, M., Masuda, T., & Koes, D. R. (2022). Generating 3D molecules conditional on receptor binding sites with deep generative models. *Chemical science*, 13(9), 2701-2713.

Volkov, M., Turk, J. A., Drizard, N., Martin, N., Hoffmann, B., Gaston-Mathé, Y., & Rognan, D. (2022). On the Frustration to Predict Binding Affinities from Protein–Ligand Structures with Deep Neural Networks. *Journal of Medicinal Chemistry*.

Eberhardt, J., Santos-Martins, D., Tillack, A. F., & Forli, S. (2021). AutoDock Vina 1.2. 0: New docking methods, expanded force field, and python bindings. *Journal of chemical information and modeling*, *61*(8), 3891-3898.

Wierbowski, S. D., Wingert, B. M., Zheng, J., & Camacho, C. J. (2020). Cross‐docking benchmark for automated pose and ranking prediction of ligand binding. *Protein Science*, *29*(1), 298-305.

Gainza, P., Sverrisson, F., Monti, F., Rodola, E., Boscaini, D., Bronstein, M. M., & Correia, B. E. (2020). Deciphering interaction fingerprints from protein molecular surfaces using geometric deep learning. *Nature Methods*, *17*(2), 184-192.

Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., & Keutzer, K. (2016). SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size. *arXiv preprint arXiv:1602.07360*.

Isert, C., Atz, K., & Schneider, G. (2022). Structure-based drug design with geometric deep learning. *arXiv preprint* arXiv:2210.11250.

Jiang, D., Hsieh, C. Y., Wu, Z., Kang, Y., Wang, J., Wang, E., ... & Hou, T. (2021). Interactiongraphnet: A novel and efficient deep graph representation learning framework for accurate protein–ligand interaction predictions. *Journal of medicinal chemistry*, 64(24), 18209-18232.

Lim, J., Ryu, S., Park, K., Choe, Y. J., Ham, J., & Kim, W. Y. (2019). Predicting drug–target interaction using a novel graph neural network with 3D structure-embedded graph representation. *Journal of chemical information and modeling*, 59(9), 3981-3988.

Smusz, S., Kurczab, R., & Bojarski, A. J. (2013). The influence of the inactives subset generation on the performance of machine learning methods. *Journal of cheminformatics*, 5(1), 1-8.

Wallach, I., & Heifets, A. (2018). Most ligand-based classification benchmarks reward memorization rather than generalization. *Journal of chemical information and modeling*, 58(5), 916-932.

Chen, L., Cruz, A., Ramsey, S., Dickson, C. J., Duca, J. S., Hornak, V., ... & Kurtzman, T. (2019). Hidden bias in the DUD-E dataset leads to misleading performance of deep learning in structure-based virtual screening. *PloS one*, 14(8), e0220113.

Sieg, J., Flachsenberg, F., & Rarey, M. (2019). In need of bias control: evaluating chemical data for machine learning in structure-based virtual screening. *Journal of chemical information and modeling*, 59(3), 947-961.

van Tilborg, D., Alenicheva, A., & Grisoni, F. (2022). Exposing the limitations of molecular machine learning with activity cliffs. Journal of Chemical Information and Modeling, 62(23), 5938-5951.

Grygorenko, O. O., Radchenko, D. S., Dziuba, I., Chuprina, A., Gubina, K. E., & Moroz, Y. S. (2020). Generating multibillion chemical space of readily accessible screening compounds. Iscience, 23(11), 101681.

Gorgulla, C., Boeszoermenyi, A., Wang, Z. F., Fischer, P. D., Coote, P. W., Padmanabha Das, K. M., ... & Arthanari, H. (2020). An open-source drug discovery platform enables ultra-large virtual screens. Nature, 580(7805), 663-668.

Acharya, A., Agarwal, R., Baker, M. B., Baudry, J., Bhowmik, D., Boehm, S., ... & Zanetti-Polzi, L. (2020). Supercomputer-based ensemble docking drug discovery pipeline with application to COVID-19. Journal of chemical information and modeling, 60(12), 5832-5852.

Kanakala, G. C., Aggarwal, R., Nayar, D., & Priyakumar, U. D. (2023). Latent Biases in Machine Learning Models for Predicting Binding Affinities Using Popular Data Sets. *ACS Omega*.

Yang, J., Shen, C., & Huang, N. (2020). Predicting or pretending: artificial intelligence for protein-ligand interactions lack of sufficiently large and unbiased datasets. *Frontiers in pharmacology*, 11, 69.

Scantlebury, J., Brown, N., Von Delft, F., & Deane, C. M. (2020). Data set augmentation allows deep learning-based virtual screening to better generalize to unseen target classes and highlight important binding interactions. *Journal of chemical information and modeling*, 60(8), 3722-3730.

Francoeur, P. G., Masuda, T., Sunseri, J., Jia, A., Iovanisci, R. B., Snyder, I., & Koes, D. R. (2020). Three-dimensional convolutional neural networks and a cross-docked data set for structure-based drug design. *Journal of chemical information and modeling*, 60(9), 4200-4215.