Università degli Studi di Milano - Bicocca
Scuola di Scienze
Dipartimento di Informatica, Sistemistica e Comunicazione
Corso di Laurea Magistrale in Informatica

Free-energy calculations using Graph Convolutional Networks

Relatore: Prof. Vittorio Limongelli

Tesi di Laurea Magistrale di:
Demetrio Carrara
Matricola 807894

Anno Accademico 2019-2020

Contents

Chapter 1

Introduction

1.1 Importance and significance

Understanding the complex and intertwined net of biologically relevant mechanisms is the holy grail of biology and biochemistry. The more we dig up to unveil new important pathways, the more difficult understanding the entire picture becomes, meaning that nature has yet a lot to teach us in this regard. In the last decades, computational scientists greatly contributed in extending our knowledge by providing information otherwise difficult or outright impossible to obtain by *in vitro* assays [1]. Nonetheless, the amount of data produced every year by the different -omics sciences (e.g., genomics, proteomics, etc.) requires new ways to extract meaningful information [2].

The methods presented in this thesis aim at introducing a way to obtain thermodynamics and kinetics estimates for a well-known case biological case study from selected molecular features by using graph theory and machine learning (ML) techniques.

Knowledge of these relevant properties (e.g., free energy of processes, transition states, energy barriers, etc.) allows scientists to formulate theories and aid the design of *ad hoc* molecules for different tasks, like peptides, antibodies or drugs for specific molecular targets. The study of the thermodynamics and kinetics properties of a biological system is a non-trivial problem since these are often difficult to capture by the experimental approaches [3]. On the other hand, computational approaches are generally a trade-off between accuracy in the results and simulation time [4]. They could take from days to weeks of calculations to converge on the final value depending on the complexity of the problem and the number of simulated atoms. In this respect, our goal is to build a neural network (NN) model general enough

to estimate relative free-energy values in the problem of conformational sampling of molecules. This topic is related to protein folding and drug/protein interaction, which have been tackled several times by ML approaches [5, 6].

In the "Background" section, a comprehensive list of ML approaches is offered so as to outline how these powerful techniques have been employed to solve biologically relevant questions so far. Moreover, the next chapters will give the reader a closer look to preparatory notion and ML approaches necessary to understand our procedural choices and interpret the results. A special focus will be given to graph convolutional networks, which are part of a new, mostly unexplored research field. Here, the same assumptions for image-recognition-based ML approaches, one of the most addressed research topic, cannot be fully applied. Therefore, method properties and related problems will be further discussed. Successively, the obtained results will be presented, discussing with more details the different use cases where our models might be applied. Finally, future perspectives will be presented on how to improve the work done so far, listing important features to add or revise and possible improvements of our models to solve the current limitations, such as the use of hypergraph NN. This work also offers a broad analysis of the problems that free-energy calculations with ML must deal with, in order to achieve meaningful results.

1.2 Background

The powerfulness and flexibility of the NNs make them a model that is used in a variety of tasks such as speech recognition, image and video processing, and video games [7, 8, 9]. The breakthrough of NN was brought by the first successful work in recognizing handwritten digits [10], in which weight sharing and backpropagation were firstly implemented. The evolution of the design of NNs led to the birth of deep learning (DL), which refers to the multi-layer architecture for NN. DL is good at approximate very complex data structures and it is applicable to many fields, such as disease recognition [11] and particle accelerator data analysis [12].

The improvements in ML methods have been successfully applied also to lots of reserch fields in the chemical and physical areas. The molecular modeling is one of the most investigated topics. There, ML is used for several complex tasks like coarse-grained molecular dynamics, kinetics and thermodynamics calculations. The advent of deep NN was a breakthrough in these research fields, bringing fresh air and new, different point of views. The Quantum Mechanics (QM) approaches received a lot of attention, with the development of models to accelerate computations, investigate electronic configurations, and analyze material properties in the quantum realm [13, 14, 15]. One of the first approaches tried also to improve Molecular Dynamics (MD)

simulations by computing potential energy surfaces from QM [16]. Improvements in the QM calculations brought new, more precise, force fields [17] that lowered the error of some MD simulations compared to experimental assays. ML approaches were also employed to analyse MD trajectories with the purpose of defining optimal Collective Variables (CVs) [18] or isolating transition states [19]. Moreover, they were used in enhanced sampling techniques [20], in order to accelerate the MD sampling. With regard to the free energy problem, while the potential energy function associated with the atomistic representations is known, the problem of computing the free energy of a system as a function of the CVs has no analytical solution and represents the goal of the presented work. Enhanced sampling methods have been assisted by ML methods in the bias potential approximation [21].

1.3 Preparatory notions

In this section, a brief description of the fundamentals of chemistry, physics and biology is presented in order to introduce the reader to concepts and words that will be used in the upcoming chapters.

The complexity of nature represents an impervious forest with hidden secrets that the scientific community is trying to unveil. Each new path reveal invaluable information that allows us to shed light on important processes. In the case of biology, countless pathways and mechanisms regulate our daily life. Misregulation or interrumption in one of these processes can lead to dangerous diseases or, in the worst case, death. For this and many more reasons, an incessant progress in research and development of new methods must be pursued. In this regard, computational sciences are providing invaluable information to the scientific community by simulating *in silico* relevant systems, encompassing several disciplines such as biochemistry, material science, fluid dynamics, etc. These simulations produce data that replicate with a certain accuracy the behaviour of atoms, allowing to estimate physico-chemical properties and energetic values that can be complex or time-consuming to obtain through experimental methods. In detail, in the present work we focus on alanine dipeptide, which is a **peptide**, structurally similar to **proteins** even though much smaller. Proteins are machineries that allow cells to work and process the countless chemicals essential for life. They can widely vary in dimensions and function and are made of building blocks called **amino acids**. The process of synthesizing a protein is called translation and it is promoted by ribosomes and a messenger ribonucleic acid (mRNA), which in turn is composed of nucleosides. There exist 4 kind of nucleosides for the mRNA (adenosine, uridine, cytidine and guanosine) and each triplet of nucleosides (also known as codon) translates for one of the possible 20 natural

amino acids. It is worth noting that the number of combinations using 4 bases far exceeds the number of unique amino acids ($3^4 > 20$), thus more than one codon can translate for the same amino acid.

All amino acid share common features: a central carbon atom, known as carbon α (C_α), bonded to an amino group (NH_2), a carboxyl group (COOH), a hydrogen (H) atom, and a side chain (R group). The latter is what distinguishes the amino acids among themselves and it may vary in length and structure. The sequence, the three-dimensional disposition and the number of aminoacids determine the kind of peptide or protein and its function. The covalent bonds formed between two amino acids are called peptide bonds. They are formed when the carboxyl group of the first aminoacid and the amino group of the second one combine, releasing a molecule of water during the process.

Up to a certain amount of amino acids, we speak about peptides (< 5000 Dalton), whereas long chains of amino acids (or many chains) compose the protein. All the bonds that connect one aminoacid to the other form the main skeleton of the molecule, also called backbone. Its disposition and movement outlines important characteristics and behaviours of the system. As it will be discussed later (section 2.2), these properties have important implications in the evaluation of the conformational free energy and recognition of metastable states during the sampling.

Proteins can be organized depending on the number of subunits they are composed of. They can assume different conformations (conformational ensemble), although the most important one is the conformation that allows the protein to fulfil its role (native). The process in which the protein folds to reach its native conformation is called **folding** and it is to date an unresolved task that computational approaches tried to tackle several times [5]. The structure assumed by a protein can be divided in 4 categories in increasing order of complexity:

- primary: the plain sequence of amino acids.
- secondary: the three-dimensional arrangement populated by amino acids. These shapes are built based on the H-bonds that amino acids form among themselves. The most common shapes of secondary structure are the α-helix and the β-sheet, playing an important structural role in most proteins. This is the highest level of complexity in peptides.
- tertiary: the three-dimensional disposition of all the secondary structures in the chain. Here, one may observe interactions between side chains of distant aminoacids. As a general rule of thumb, in aqueous solution the hydrophilic R groups tend to face the solvent, whereas the hydrophobic ones cluster together forming the core of the protein.

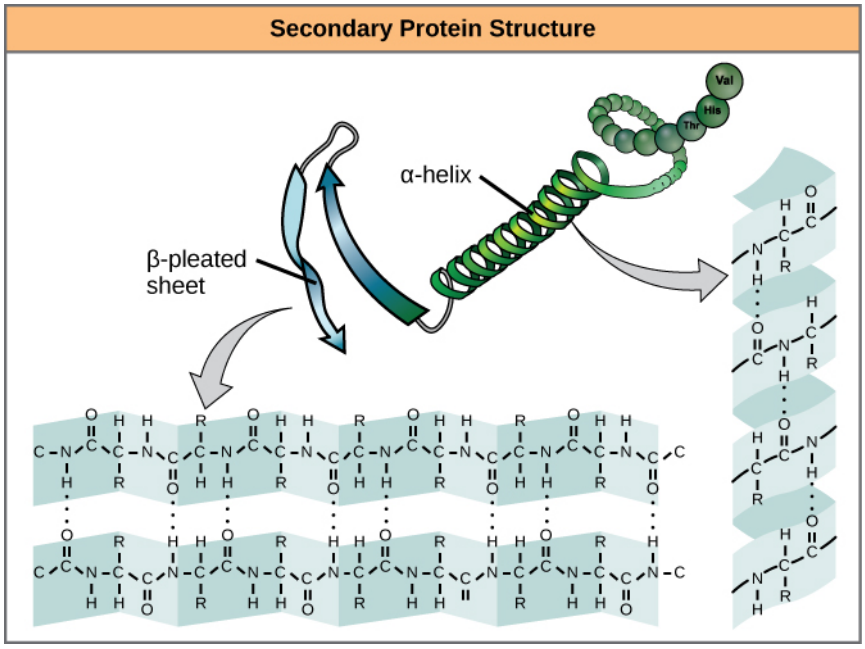


Figure 1.1: α-helix and the β-sheet H-bonds interactions. Credits to OpenStack Biology [22]

- quaternary: the combination of more chains or subunits. In a system where there are multiple chains, they are held together by non-bonded interactions. Not all proteins reach this level of complexity.

One last concept that is important to introduce is intermolecular forces. Few of these were already mentioned in the previous paragraph (i.e., non-bonded interactions). Although they can not compete in energy with covalent bonds (formed by sharing electon pairs between atoms), they are vital in a plethora of processes (e.g., maintaining protein structure, non-covalent drug-target interactions, protein-protein interactions, etc.). We will divide them in two families (i.e., electrostatic and Van der Waals interactions), since this will help in explaining how they are treated in MD simulations. The **electrostatic** interactions are present when charges are involved and the applied forces can be attractive or repulsive depending on the charge of the interacting groups. On the other hand, the **Van der Waals** interactions consider the correlations in the fluctuating polarizations of nearby atoms, therefore including all multipoles starting from induced or fixed dipole. Also in this case, we can have repulsive or attractive forces that are generally very low in energy. Nonetheless, the huge amount of these kind of interactions allow them to influence the behaviour

of molecules. Hydrogen bonds (also known as H-bonds), are a particular class of electrostatic interactions since their energy is highly dependent on the geometry of the interaction. As previously mentioned, these interaction are very important in peptide and protein secondary structure formation.

Chapter 2

Molecular Dynamics

MD simulations [23] were born as a way to reproduce many-particle systems and compute particular properties of interest. These properties are tipically measured as an average over a large number of particles and time. At the base of MD stands the ergodic hypothesis, which states that: considering long periods of time, supposing that the system state does not depend on the starting particle position and the system has constant energy, volume, and particles (microcanonical ensemble), then the average in time of a certain property can be measured as the average on the phase space of positions and momentum of particles, or ensemble average. To reproduce the motion of atoms in a molecule, it is necessary to replicate their behaviour by creating a force field, a set of properties and potentials, that rules the interactions among atoms. Ideally, an high accuracy description of the properties of molecular systems is given by the time-dependent Schrödinger equation [24]. However, such a detailed level of information is unfeasible for many-particle systems, thus MD generally follows the classical physics laws. This approximation is acceptable for big systems (greater number of atoms involved), where the contribute of quantum effects is negligible. However, the increase in particles also causes the exponential growth in the number of computations to solve and the time necessary for the simulation to compute the desired property. To date, MD simulations have been successfully employed in various fields and the improvements in hardware performances allow to simulate long and complex processes, such as drugs targeting receptors involved in important biological pathways, protein polymerization, signal translation, etc. [23]

2.1 Molecular Dynamics fundamentals

Given a system of N interacting atoms [25], MD simulations solve Newton's equations of motion:

$$m_i \frac{\delta^2 r_i}{\delta t^2} = F_i \tag{2.1}$$

where the forces F_i are negative derivative of a potential function $V(r_1, r_2, ..., r_N)$:

$$F_i = -\frac{\delta V}{\delta r_i} \tag{2.2}$$

In turn, forces and energy are derived based on a set of atom properties and functions collected inside a force field. Nowadays, there are a lot of different force fields, each of them specialized on subsets of systems: proteins, organic molecules, ligands and so on [26, 27, 28]. In an MD simulation, atoms are just hard spheres with no information about hybridization and geometry. In order to reproduce the correct behaviour, constraints are contained in the force fields and applied during the simulation, namely bonds, angles and dihedrals. **Bonds** control the relative distance between two atoms, **angles** in the planar space form a relationship among 3 connected atoms and approximate the atom hybridization, **dihedrals** or torsionals, are rotational angles of a set of 4 connected atoms. Specifically, given 4 atoms, the torsional angle is the angle between the plane formed by the first 3 atoms and the plane formed by the last 3 ones (Fig. 2.1).

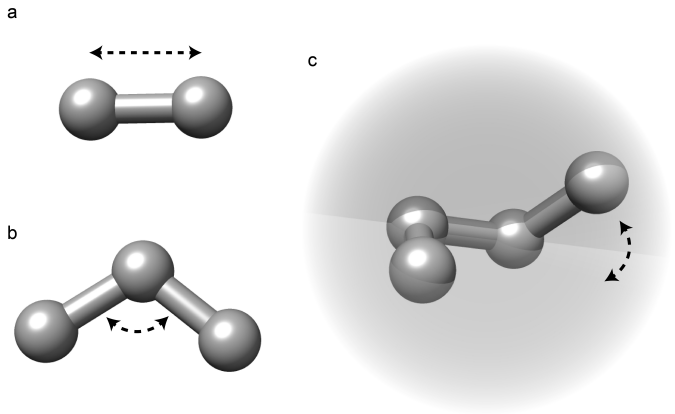


Figure 2.1: Example of constraints or bonded interactions to replicate molecular behaviours, namely bonds (a), angles (b), and dihedral angles (c).

All these constraints are included in the calculation of the energy in force fields, together with the electrostatic and Van der Waals contributions. For example the

Amber force field equation [29] reads as follow:

$$E = \sum_{bonds} K_r(r - r_{eq})^2 + \sum_{angles} K_\theta(\theta - \theta_{eq})^2 + \sum_{dihedrals} \frac{V_n}{2}[1 + cos(n\phi - \gamma)] + \sum_{i<j} \Big[\frac{A_{ij}}{R_{ij}^{12}} - \frac{B_{ij}}{R_{ij}^{6}} + \frac{q_i q_j}{\epsilon R_{ij}}\Big] \tag{2.3}$$

where K_r is the spring constant defining the strength of the bond, r is the distance between two atoms, r_eq is the equilibrium distance in the bond, K_θ is the spring defining the strength of an angle, θ is the angle among 3 atoms, θ_eq is the equilibrium value for the angle, V_n is the potential for a given dihedral angle of multeplicity n, ϕ is the angle of the dihedral defined by 4 atoms, and γ is the phase of the dihedral. The last term in equation 2.3 contains the two interactions that were presented in the previous chapter. In particular, the Van der Waals interactions [30] are generally described by the Lennard Jones potential with the following formula:

$$U^{LJ} = 4\epsilon\left(\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right) \tag{2.4}$$

where σ is the distance at which the potential is zero and r is the distance between the atoms. The shape of the function can be appreciated in Figure 2.2. This form is slightly different from the one reported in equation 2.3, where the terms A and B can be seen as $4\epsilon\sigma^{12}$ and $4\epsilon\sigma^{6}$, respectively.

On the other hand, the electrostatic (or Coulomb) forces are described by Coulomb's law:

$$F = k_e \frac{q_1 q_2}{r^2} \tag{2.5}$$

where k_e is the Coulomb's constant, q_i and q_j are the partial charges of the two particles and r is their distance. In equation 2.3 the two charges q_i and q_j are processed in internal units that incorporate the Coulomb's constant.

All the algorithms developed for integrating the equations of motion assumes that positions, velocities and accelerations are approximated by a Taylor series expansion [32]:

$$r(t + \delta t) = r(t) + v(t)\delta t + \frac{1}{2}a(t)\delta t^2 + ... \tag{2.6}$$

$$v(t + \delta t) = v(t) + a(t)\delta t + \frac{1}{2}b(t)\delta t^2 + ... \tag{2.7}$$

$$a(t + \delta t) = a(t) + b(t)\delta t + ... \tag{2.8}$$

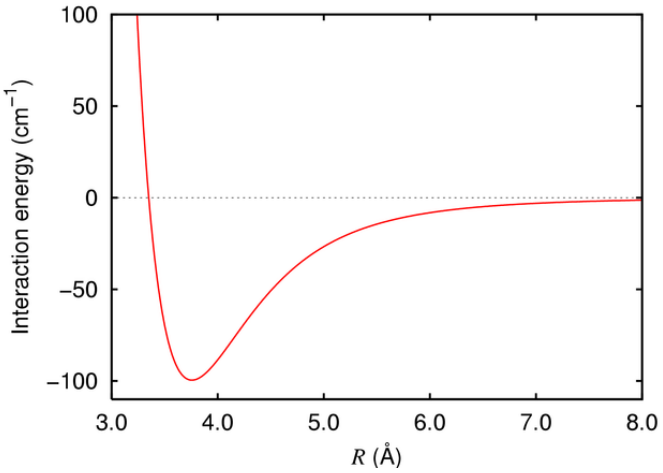


Figure 2.2: An example [31] of interaction energy-distance relation graph obtained from argon dimer

where r is the position, v the velocity and a the acceleration. It is worth noting that the equations are solved in tiny time steps, in the magnitude of femtoseconds ($10^{-15}s$), since integration of forces must happen in the same time scale of the fastest molecular degree of freedom (i.e, vibrations). After each integration step, the atomic coordinates are written to an output file, that describes the trajectory of the simulation in function of time.

It is mandatory to underline that MD simulations are based on a series of approximations in order to make the simulation sustainable in terms of computational complexity and matematical feasibility. However, this does not mean that results are flawed and can not be trusted. In fact, MD protocols and force fields have been constantly fine-tuned to replicate reality with an acceptable discrepancy (e.g., current calculations with MD allow to obtain energetic estimations with an error of 1-2 kcal/mol). We have to keep in mind that these approximations are inevitable and the real world is too complex to be studied on its entirety. Using MD as a preliminary observation tool can to save a lot in terms of money and time, therefore allowing to test hyphotesis [33] before planning further work.

2.1.1 Preparing the simulation

MD simulations need to be initialized and configured properly for each system we want to study. There is no one-fit-all system configuration because, depending on the parameters we choose, a different desired behaviour may be highlighted from the

resulting trajectory. Below, the most important concepts to successfully carry out a simulation are listed.

The first step consists of preparing the simulation box and the environment:

1. Choose the force field that should be used to assign properties to atoms. It is worth noting that while MD treats atoms as hard spheres, each kind of atom has unique characteristics, such as atomic number, mass, radius, charge, number of electrons and more, defined in the force field;

2. Create a box around your system. If the simulation is not performed *in vacuum*, the program needs to know boundaries for the simulation. The *periodic boundary conditions* (PBC) [34] are a way to substitute the boundaries of an isolated system with periodic images of itself. In other words, the box is surrounded with other translated copies of itself avoiding artifacts caused by hard walls limiting the simulation space. When a molecule oversteps the box limit, a copy of it will enter from the opposite side, maintaining constant the number of particles.

3. Fill the box with a solvent (could be water, or nothing if we want to simulate the molecule *in vacuum*). A molecule may act differently based on the solvent it is inserted into, so it is necessary to recreate an accurate copy of the environment where to observe a certain behaviour;

4. Add ions in order to neutralize the charge of the system. Ions are a particular set of atoms which have a net electrical charge that is non-zero (formally it is the difference between the number of protons, positive, and the number of electrons, negative). Depending on the simulation conditions, MD might require the system to have a total net charge of 0. This is needed if we use PBC because the electrostatic evaluation algorithm will add to infinity (due to the infinite periodic images). By having a total charge of 0, the electrostatics sum to a finite value (i.e, 0).

All information proovided in the previous points are stored in two files, the coordinate files and the topology. The latter contains the properties of atoms, with bonded and non-bonded interactions. Instead the former contains the position in Cartesian coordinates of all the atoms. Depending from the MD engine, format and position of the information can change.

System equilibration

Before beginning with dynamics, one must assert that there are no inappropriate geometry or steric clashes in the system, otherwise the forces computed will lead to structural distortion and possible explosion fo the system.

The process that takes care of correcting these kinf of problems is the energy minimization procedure [35] which can use several techniques, such as steepest descent or conjugate gradient techniques.

Given a vector r, containing the coordinates, the initial forces and the potential energy [25]; the new atomic positions are calculated as:

$$r_{n+1} = r_n + \frac{F_n}{\max(|F_n|)} h_n \tag{2.9}$$

where h_n is the maximum displacement and F_n is the negative gradient of the potential V.

The algorithm stops when either it reaches a maximum number of iterations, or when the absolute values of the gradient falls under a certain threshold ϵ.

After minimization, the system must be brought to the correct environmental conditions. This procedure, called thermalisation, follows several steps intended to reach the correct values of temperature, volume and pressure of the system without causing artifacts. In order to explain these steps, we need to define a list of ensembles. The study of the real world thermodynamics can be described throught a set of observable variables [36]. Three important ensembles, environments that maintain constant some macroscopical variables, have been defined by Gibbs [37]:

- NVE (microcanonical): Number of atoms (N), the system volume (V) and the energy (E) are conserved.
- NVT (canonical): Number of atoms (N), the system volume (V) and the temperature (T) are conserved. The energy must be exchanged with a thermostat.
- NPT (isothermal-isobaric): Number of atoms (N), the system pressure (P) and the temperature (T) are conserved. There is the need of a thermostat and a barostat that exchange pressure and temperature with the environment in order to let the system vent.

It is worth noting that based on the chosen ensemble, different real system may be described.

The effect of applying the correct variable values all of a sudden might create artifacts in the minimized structure that will be carried throughout the whole simulation,

flawing the results. Therefore, thermalisation consist in gradually ramping up temperature, while regulating the volume of the system and monitoring its energy, until the requested conditions are met. It is common practice also to employ high-energy position constraints to preserve molecules inside the box that are susceptible to temperature gradients. These constraints are gradually removed upon reaching the final conditions. A combination of NVT and NPT ensembles at increasing temperatures are used to stabilize volume and pressure of the system.

2.1.2 The production run

The production run is the last step in the MD protocol that produces a time-dependent system's trajectory by integrating the classical Newton's equations of motion [38, 25]. The production run represents the simulation that will be considered statistically relevant for the successive analysis. Depending on the complexity of the studied system and the desired behaviour one would like to observe (e.g. folding, binding free energy, etc.), the production run can take from nanoseconds to milliseconds. One real example of a specific supercomputer, created for running simulations is the machine Anton 2 [39], able to perform, on a 23'588-atom system, 85μs/day (180 times faster than any general-purpose supercomputer nowadays).

To avoid spending months while waiting for the production run to finish, one can employ enhanced sampling methods that accelerate selected slow degrees of freedom in the system to improve the sampling. In this way the system does not remain stuck for long times in states of minimum and convergence in results is reached faster. One of such techniques is called MetaDynamics (MetaD) and it was used for this project, thus it will be presented in the following sections.

2.2 Free energy

In chemistry, direction of a specific reaction can be defined by its free energy value. In the hypothesis of having a reversible process [40], the free energy of a system can be defined as the maximum amount of work that the system can exert [41]. Depending on the conditions of the simulation, we have different definitions of free energy. For the purpose of this work, we will use the Gibbs free energy (G), which is obtained for systems with fixed temperature and pressure (general conditions in experiments). The Gibbs free energy is defined as:

$$G = H - TS, \tag{2.10}$$

where H is the enthalpy, T the temperature, and S is the entropy of the system. However, it is generally the relative value of free energy that is interesting and holds physical meaning. In other words, it would be very informative to estimate the difference in free energy (ΔG) between reagents and products, to assess if a process is favourable or not (i.e., if its ΔG is negative or positive, respectively). Due to the complicated nature of the free-energy formula, there is no analytical solution for it, but it must be solved numerically. With respect to this work, the free energy of a conformation is considered as a relative value with respect to the absolute minimum of that system, meaning that all computed free energies represent differences in free energy.

Exploring and quantifying the Free Energy Surface (FES) of a system is an important topic in several fields (chemistry, biophysics, bioinformatics) [42]. One can obtain important thermodynamics and kinetics parameters that might be difficult or time-consuming to be evaluated by means of experimental assays [43]. Moreover, few information can only be gathered through *in silico* simulations, such as particular metastable states, saddle points for transition states, etc. Analysis of energy barriers [44] and low energy pathways in reactions allow us to tweak and engineer new molecules with *ad hoc* features for specific problems (e.g., drug optimization).

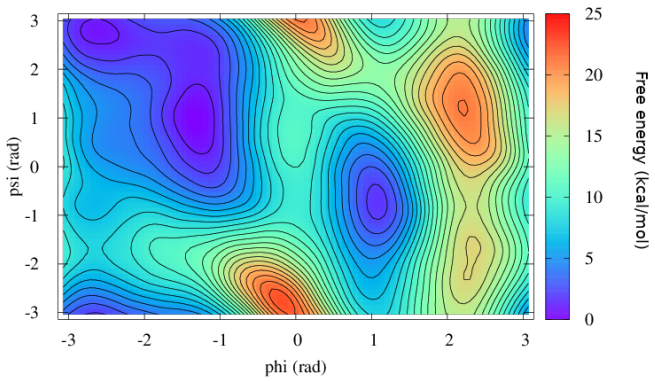


Figure 2.3: An example of Free Energy Surface: Alanine Dipeptide, in function of ϕ (phi) and ψ (psi) dihedrals.

2.3 MetaDynamics

Molecular Dynamics has some important limitations due to the nature of its algorithms. The more complex the system and the higher the number of degrees of

freedom, making the process very slow. Moreover, a molecule is not favoured to leave a minimum, overcome an energy barrier, and explore another minimum, unless enough time and energy are provided. What if we would like to explore the entire Free Energy Surface (FES) in a quicker way, by forcing the escape from the minimum?

MetaD [45] enhances the sampling by adding an external bias to selected reaction coordinates (also known as collective variables) of the system in the form of small Gaussians [46]. Under the MetaD regime, the system overcomes even large energy barriers and explores the FES in a computationally affordable simulation time. This also provides a quantitative determination of the FES as an implicit product of the process by rescaling the known amount of additional bias deposited.

In MetaD, the selection of the **Collective Variables** (CV) is a crucial step. By representing the system with a finite number of relevant CVs s_i, where $i \in (1, n)$ with n small, the resulting FES $F(s_i)$ can be defined as a manifold of order n. Each CV should represent an important degree of freedom of the system that must be accelerated. However, complex FESs require long time to be explored and in addition, the convergence time of the calculation exponentaially increases with the number of the CVs biased during the simulation. Moreover, selection of unnecessary CVs may lead to meaningless results, thus methods have been developed for a clever selection of the most important CVs [18].

Exploration of the FES is driven by the forces applied along the chosen CV. To correctly estimate these forces, a number of replicas are launched in parallel, each one respecting a constraint applied to the CV $s_i^{(0)}$. By writing this constraint in a form familiar to MD simulations [47], it is possible to add the following term to the Lagrangean:

$$\sum_{i=i,n} \lambda_i (s_i - s_i^0) \tag{2.11}$$

where λ_i are the Lagrange multipliers. By averaging in time and over the replicas, it is possible to obtain the forces $F_i^{(0)} = \langle \lambda_i \rangle$ [45].

But this alone does not guarantee the exploration of the whole FES; that is the reason why a history-dependent term must be applied to force the system to explore new regions.

The introduced potential V, in function of the CVs $\vec{s}$ and time t, is built as a sum of Gaussian kernels with the following form:

$$V(\vec{s}, t) = \sum_{k\tau<t} W(k\tau) \exp\left(- \sum_{i=1}^{n} \frac{(s_i - s_i^{(0)}(k\tau))^2}{2\sigma_i^2} \right) \tag{2.12}$$

where:

- τ is the Gaussian deposition stride
- σ_i is the width of the Gaussian for the ith CV
- $W(k\tau)$ is the height of the Gaussian, $k \in \mathbb{Z}$

Depending on the width and height of the Gaussian functions and given enough time, the bias potential will completely cover the underlying FES, which can be reconstructed following the formula:

$$V(\vec{s}) = -F(\vec{s}) \tag{2.13}$$

The correct choice of the hyperparameters W, σ_i and τ allows a swift and easy completion of a MetaD simulation. Unfortunately, the perfect estimates can be chosen only for systems with well known characteristics. A way on how to assess acceptable values in case of unknown systems are reported in the original Metad paper [45].

Although the technique demonstrated its ability to highly accelerate the sampling, a well-known problem in MetaD is the error between the real underlying FES and the complemented potential $V(\vec{s})$. The ideal situation would be that the added potential completely covers all energy barriers and the sum of the FES and the potential should give a uniformely flat surface. However, by adding constant Gaussian potentials of height W, MetaD keeps creating an irregular surface, oscillating around the real value with an error proportional to the value of the chosen parameters.

This behaviour was corrected few years later with the development of the Well-Tempered MetaDynamics.

2.3.1 Well-tempered MetaDynamics

Well-tempered MetaDynamics (WT-MetaD) [48] is an evolution of plain MetaD, developed to correct the oscillating behaviour at the end of the simulation. Using a chosen temperature as reference, it computes dynamically the height of the Gaussians potentials to be placed in order to decrease it over time when sampling already visited states. It has been demonstrated mathematically that WT-MetaD is able to converge to the correct free-energy value given an infinite amount of time [49]. From a practical point of view, the function for the Gaussian potential is very similar but its height is rescaled according to the following term:

$$W(k\tau) = W_0 \exp\left(-\frac{V(\vec{s}_i^{(0)}(k\tau), k\tau)}{k_B \Delta T}\right) \tag{2.14}$$

where:

- W_0 is the initial Gaussian height
- ΔT is the reference temperature
- k_B is the Boltzmann constant which puts in relationship the average relative kinetic energy of particles in a gas with the thermodynamic temperature of the gas. $k_B = 1.380649 \cdot 10^{-23} J \cdot K^{-1}$

It is worth outlining that the bias potential with a dynamic height converges in the long time limit but it does not guarantee to fully compensate the underlying free energy. This is a desired effect since one may not want to explore physically unfeasible states.

The potential in the long time limit becomes:

$$V(\vec{s}, t \to \infty) = -\frac{\Delta T}{T + \Delta T} F(\vec{s}) \tag{2.15}$$

where:

1. T is the temperature of the system
2. ΔT is the extension, in reference to the temperature, that we want to explore

In the long time limit, the sampling is in an ensemble at a temperature $\Delta T + T$. With $\Delta T = 0$, it's a standard Molecular Dynamics simulation, with $\Delta T \to \infty$ it's a standard MetaDynamics.

2.4 Alanine Dipeptide

The experiments carried out in this project consider alanine dipeptide as the reference molecule. Alanine dipeptide [50] is a very simple system, well studied in literature and formed by a chain of 1 amino acid and 2 terminal groups (ACE-ALA-NME). This has been chosen as reference because of the wealth of literature data on its FES and its role as stepping stone in methods focused in computing a FES before progressing to more complex systems.

The FES of the Alanine Dipeptide is often described as a function of the two most central backbone dihedrals ϕ and ψ, since they are known slow degrees of freedom for this system. It shows two minima, named C7eq and Cax, located at, (ϕ, ψ) = (-1.45, +1.30) and (+1.22, -1.22) radians [51] (Figure 2.4), respectively. The two metastable states are divided by an energetic barrier of ∼8 kcal/mol [52].

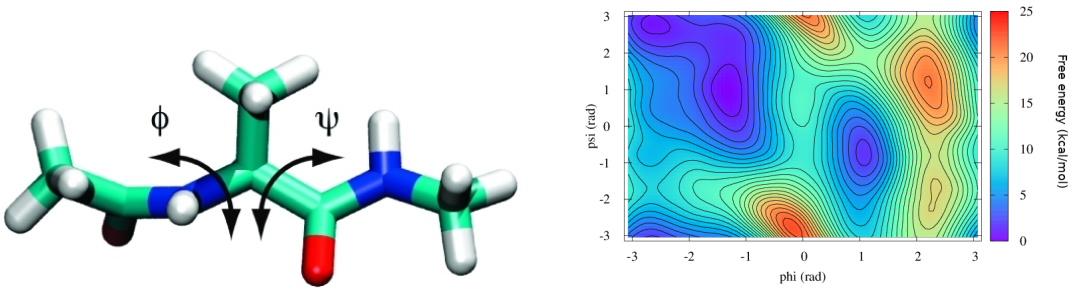


Figure 2.4: Alanine Dipeptide FES in function of ϕ and ψ

The ease of studying the FES of the Alanine Dipeptide comes with the fact that dihedrals by themselves give already a detailed description of the FES, thus they are the perfect CVs. In other, more complex systems, a combination of the dihedrals values might not be enough to explain fully their FES. In such cases, different CVs should be chosen.

Chapter 3

Introduction to neural networks

In this chapter, a general overview is given about functioning and reasoning behind NNs. It starts from the high-level concepts and how they are achieved mathematically, going through the tasks they are used for and explaining the optimization methods that contribute in accelerating the learning problem.

3.1 Neural network fundamentals

The main concept behind NN models is the imitation of the brain connections. In the brain, neurons are communicating among them, producing an output based on the received input signals. The expressivity power of the brain does not come from a single neuron, but from a combination of them. NNs are built mimicking the brain architecture; by showing enough examples with the desired label, NNs learn an approximation (whose parameters are the NN weights) of the function that correlates input features to their respective outputs. Moreover, NNs can learn relative importance among neurons connections by prioritizing one with respect to another based on the received input, therefore becoming a universal function approximator when using a high-enough number of neurons stacked layers [53]. The building block of a NN is the perceptron, represented as a linear combination of input parameters, followed by a non-linear activation function. Imagining a sequence of perceptrons, the output of the i-th one is computed through the following formula [54]:

$$y_i = f_i\big(\textstyle\sum_{j=1}^{n} w_{ij} x_j + \theta_i\big). \tag{3.1}$$

where x_j is the j-th input parameter with j going from 1 to n, w_{ij} is the connection weight that links the j-th input parameter to the i-th neuron, θ_i is the weights-

independent bias, and $f_i(\cdot)$ is the non-linear activation function that transforms the weighted sum to describe abstract relationships among input data.

3.1.1 Activation functions

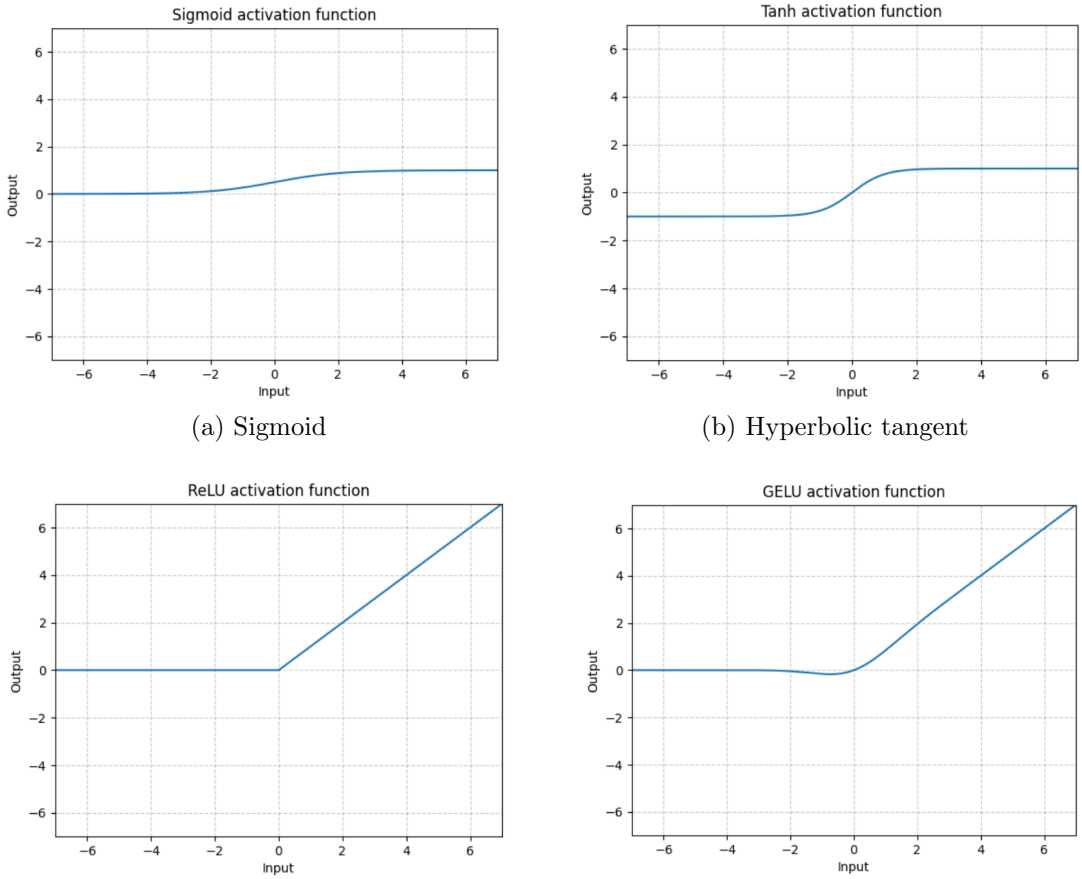
Why do we need a non-linear activation function? Neurons in brains may be in an *active* or *sleepy* state [55], with their output being significant only if there are enough input stimuli. The activation function mimics this behaviour in the perceptron [56]. As a by-product the non-linearity enables the real power of NNs, by allowing to approximate non-linear, thus more complex than the linear combination of inputs, functions.

The common characteristic of an activation function is its derivability since it will be used to train the network with a backpropagation algorithm as explained in subsection 3.1.3. Some of the most used activation function are shown below:

- Sigmoid:
$$\text{Sigmoid}(x) = \frac{1}{1 + e^{-x}} \tag{3.2}$$
- Hyperbolic tangent:
$$\text{Tanh}(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{3.3}$$
- Rectified Linear Unit:
$$\text{ReLU}(x) = \max(0, x) \tag{3.4}$$
- Gaussian Error Linear Unit:
$$\text{GELU}(x) = x * \phi(x) \tag{3.5}$$
where $\phi(x)$ is the cumulative distribution function for Gaussian distribution [57].

3.1.2 Loss functions

NNs can be designed to be supervised or unsupervised, depending if the data set has pre-existing labels or not, respectively. The methods used in this thesis are all supervised ML models, in which a loss function output is computed against the true label and used by the optimization method to update the connections' weights. A loss function should be able to process scalar values as well as vectors and the learning



(a) Sigmoid (b) Hyperbolic tangent

(c) ReLU (d) GELU

Figure 3.1: Most used activation functions

process can be summarized in the minimization of its output, which corresponds to the error between the NN estimate y_i and the data label t_i, with respect to the i-th training istance. The most common loss functions are:

- Mean Squared Error:

$$MSE = \frac{\sum_{i=1}^{n} \left(y_i - t_i\right)^2}{n} \tag{3.6}$$

- Root Mean Squared Error:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} \left(y_i - t_i\right)^2}{n}} \tag{3.7}$$

- Mean Average Error:

$$MAE = \frac{\sum_{i=1}^{n} \left| y_i - t_i \right|}{n} \tag{3.8}$$

where n is the number of output values.

3.1.3 Learning problem

The idea behind training a neural network is simple but very powerful. As previously reported, it can be described as an optimization problem where the loss function is minimized. This is accomplished by using an optimization method and the back-propagation algorithm over the NN weights. During the *forward* step, the input goes through a stack of neurons layers and the network returns an output consisting of one or more values, which are used to compute the loss. In the *backward* step, the gradient of the loss with respect to the weights is computed in all the layers. Then, weights are updated using the preferred optimization method.

This process, called backpropagation [10], is repeated many times for all the instances in the training set, to reach a good approximation of the hidden function (i.e., the unknown function that best correlates input to output). One round of training, where all the training instances are fed into the network, is called epoch.

3.1.4 Optimization methods

The algorithms that try to find the parameters that optimize a given function are called optimizers, the most common being Stochastic Gradient Descent and Adam [58].

Gradient Descent GD is a method to optimize an objective function [58] $f(\theta)$, where θ contains tunable parameters (similar to weights in NNs). This is done by following the negative gradient $\frac{\delta f}{\delta \theta}$ in order to update θ step after step:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \cdot \frac{\delta f}{\delta \theta^{(t)}} \tag{3.9}$$

where η is the *learning rate* hyperparameter, a coefficient controlling how much the weights are modified towards the direction of negative gradient, t is the current optimization step.

Adam Adaptive Moment Estimation (Adam) [58] is a method born to correct the direction of multiple consecutive steps towards the gradient direction. *Momentum* is a method that helps accelerate in the relevant direction and smooths the oscillations of consecutive optimization steps. Adam [59] algorithm updates, at each step t, the exponential moving averages of the gradient m_t and the squared gradient v_t controlled by two decay hyperparameters β_1 and β_2. On top of that it computes the parameters θ_t as:

$$\theta_t = \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{(\sqrt{\hat{v}_t} + \epsilon)} \tag{3.10}$$

where $\hat{m}_t$ and $\hat{v}_t$ are the bias-corrected moving averages, α is the stepsize and ϵ a small positive number (usually 10^-8).

3.1.5 Classification and prediction tasks

Usually, the data set is split in three parts, each with a specific goal:

1. training set: it is used to teach the model how to predict (or classify);
2. validation set: is used to avoid overfitting;
3. testing set: it is used to assess the performance of the trained model.

Regarding the first point, models may be trained on classification or prediction tasks. The former happens when the entire dataset has labels belonging to a finite and discrete set. When doing these kind of tasks, the network usually uses a final layer of softmax activation with n outputs, where each one of them represent a single label class. Instead, if the task is the prediction (or regression), the output has to be a single number: the result of the regression. In the methods presented in this thesis, we will always employ prediction tasks.

The second point represents an important step in the NN training and the next chapter has been specifically dedicated to the overfitting problem.

Finally, in the testing set the performance of a model is evaluated by means of selected **metrics**. For prediction tasks, the most employed metrics are the same function used for the loss function (i.e., RMSE, MSE, MAE). To be noticed that, even if the same equation is used for both loss function and metric, the former guides the training process (point 1) while the latter computes the performance of the trained model, respectively.

3.1.6 Overfitting problem

The usefulness of a model does not come in its ability to discriminate (or predict) training instances, but in its estimation of unseen data. During the training process, the weights applied to the input parameteres will converge to a specific value while the objective function (the loss) is minimized against the training data. If we keep training, the model might move away from the generalization of the underlying hidden function, specializing on the seen instances. We have to consider that the training instances do not encompass the entire spectrum of meaningful data, thus keeping the model general is paramount.

The problem of over-fitting is well-known in the supervised ML world: the model is able to predict with a very low error instances in the training set, but generalize very badly on unseen instances. One way, but not the only one, to spot the overfitting is to plot the loss function on the training set side-by-side to the one of the validation set for each epoch. If we pick a fixed amount of epochs to train the model, we will notice that at first both functions rapidly decrease. This behaviour will slow down up to a moment when the validation function will invert the trend while the training error will keep getting lower and lower.

At this point the model will have began the over-fitting phase, where it overlearns how to reduce loss on training data, slowly losing the ability to generalize. The most general instances we use in our validation set, the more we can realize the model is overfitting by looking at the validation loss over time. That's why the broadness of example in the dataset is the key to avoid (and recognize) overfitting.

Various techniques have been studied including two of the most common being: dropout (used in the EdgePooling layers) and early stopping.

Early stopping The early stopping goal is to understand when the overfitting phase starts and block the training process before the model becomes too specific on the training data. A prediction on the validation set is done after each epoch of training to keep track of the loss on unseen instances. It is helpful to remeber that the validation instances are not used to update the network parameters but only to check whether to stop the training or not.

When the model approaches convergence, it is possible that the validation loss fluctuates around a given value. In early stopping, a *patience* hyperparameter is set so as to define a threshold point at which the training phase will be stopped. In practice, this value represents the number of epochs that are waited without any improvement in the validation loss. If an improvement over the last best validation loss happens, the *patience* counter is reset to 0. Otherwise, the training is stopped

and the testing phase begins.

Dropout Dropout [60] is a technique used to avoid over-fitting which randomly drops NN nodes during training. The choice of which units to drop is random, so that each unit must learn to work with a randomly chosen sample of other units. This should turn the NN more robust towards noise.

3.2 Deep learning

The need of NNs that approximate more complex non-linear functions led to the design of Deep Neural Networks (DNN) [61]. A deep-learning architecture is a stack of learning modules that compute non-linear mappings. The increased number of layers and weights allow to learn representations with multiple layers of abstraction [9]. DL models have a very large number of learnable weights, thus they are able to represent very complex functions, but they also need a huge amount of training data, with the risk of overfitting it. This may happen because the function is composed by thousands of parameters that are able to learn very specific details of the instance data.

3.2.1 Convolutional Neural Networks

Convolutional NNs (CNNs) are designed to process data that comes in the form of a multi-dimensional array [9]. The easiest example of structured data is a grey-scale image, encoded through a matrix of pixel, each containing the grey value to display. While each pixel on its own might not be so useful to understand the meaning of the whole picture, combining neighbours together give us insights about a specific object; the opposite applies for two very distant pixels in the image that are very unlikely to have a common meaning. It is important then to pass, the concept of structure to the network since it helps the NN in reasoning at a higher logic level instead on the plain values only.

CNNs have brought a breakthrough in the image processing and recognition field because of their structural approach. Combining local connections, shared weights, downsampling and the usage of many layers [9] they are able to generally reason over structured data and, given a huge amount of training data, to not overfit on samples.

The key of the success of CNNs is their ability to exploit structure properties such as stationarity, compositionality and locality of natural sciences data [10], while generalizing on the input dimension. **Stationarity** is a data property identifying

similar local patterns shared across the domain. These local features are recognized with localized convolutional filters [62]. The **compositionality** enables the NN to perceive higher-level characteristics, shared across different instances of the same real world object. At the same time, features are statistically more prone to be influenced by other features closer in the structure (**locality**).

CNNs have become the leading solutions for all the task regarding image processing and recognition. They combine three key principles to reason over structured data: local connections, shared weights and aggregation functions [9].

The potential comes with the possibility of definition of multiple feature maps (convolutional filters) so that different local features can be extracted from the same portion of data. As a by-product we greatly reduce the number of parameters of the network, but at the same time it is common to define a large number of filters so that several features may be detected. The k-th output feature map Y_k can be computed as [7]:

$$Y_k = f(W_k * x) \tag{3.11}$$

where x is the input image, W_k is the convolutional learnable filter, $*(\cdot, \cdot)$ is the convolutional operator that computes the inner product between the filter and each location of the image, and $f(\cdot)$ is the non-linear activation function.

Each filter is applied all over the structured data in order to detect common features across the whole domain. Low level features need to be aggregated so that the next layer of convolutional filters can use them to detect higher-level ones. Pooling layers, which aggregates the information contained in a local portion of the structure, are designed for this purpose. Usually the output of pooling is computed through max or average functions, with no learnable parameters, which discard information to focus only on a relevant portion of a feature map. As a by-product of the aggregation, the size of the input is reduced considerably, thus being less computational expensive. The combination of a convolutional and a pooling layer represents the main building block of deep CNNs.

The high-level features extracted, then, have to be translated into the desired output (a single value in case of prediction, a vector otherwise). Usually, the module that takes care of this is composed by a number of **fully-connected layers**, which express the output as a complex combination of the high-level features.

The main skeleton of CNNs can also be applied to irregular structures, like graphs and manifolds. More attention will be given to NNs for graphs in the next chapter with a specific attention to some of the layers that have been considered in this discussion.

Chapter 4

Geometric Deep Learning

Most of the ML techniques that have been presented so far are suited for data that belongs in the Euclidean space such as images. Euclidean spaces are defined by $\mathbb{R}^n$, thus containing data modelled in n-dimensional linear space. The topic of the following work instead will focus on graphs, unordered structures where the conventional approaches could not be applied straightforward. In this chapter a description of the different approaches for graph NNs is given, with a detailed explanation of the current state-of-the-art techniques that have been used in chapter 6.

4.1 Introduction to Graph Neural Networks

In the category of Graph Convolutional Networks belong all the NNs that take into consideration structured data in the shape of graphs. A graph is a pair $G = (V, E)$ where each vertex (or node) $v \in V$ is connected with other vertexes (or nodes) by edges $e \in E$. Each edge is an ordered pair $e_{ij} = (v_i, v_j)$, meaning that vertex v_i is connected to v_j.

An highlight on the notations for graphs [63] is given, in order to understand better what is presented in the upcoming sections:

- $N(i) = \{j \in V | (i, j) \in E\}$ is the set of vertexes that form the neighborhood of a node i.
- A is the adjacency matrix of size $n \times n$ that describes the connections among vertices:

$$A_{ij} = \begin{cases} 1 & \text{if } e_{ij} \in E \\ 0 & \text{if } e_{ij} \notin E \end{cases} \tag{4.1}$$

- D is the degree matrix of size $n \times n$. It's a diagonal matrix where each value d_{ii} represents the degree of the vertex i. For undirected graphs, the d_{ii} is the number of attached edges on i. For directed ones we distinguish between incoming and outcoming degree, counting the respective type of edges.

- $X \in \mathbb{R}^{n \times d}$ is the matrix of the node features, with d number of features per node.

- $X^e \in \mathbb{R}^{m \times c}$ is the matrix of the edge features, with m cardinality of E and c number of features per edge.

On graphs we may be interested in several classification (or prediction) tasks, depending at which level of granularity we want to obtain information. The difference between classification and prediction is given by the very last layer: single-value output for prediction and softmax layer with N values (for N classes), usually; but they should not influence the model hidden representation. Tasks can be divided into 3 main categories:

- Node-level tasks: one simple example could be classifying users in social networks as bots or not. Each node (or most of them) has to be given a classification, an output, on which it's computed the loss function. Information is propagated through convolutions on the edges in order to extract high-level node representations.

- Edge-level tasks: same as for node-level ones but we have to classify edges instead. Usually using two high-level nodes representations, it has to classify the edge that links them [63]. An example could be classifying the relationship between family members (parent, relative, son, etc...)

- Graph-level tasks: one single output (or many, but independent from vertices cardinality) per graph, either a class or a prediction. Here GNNs are used with a combination of convolutions and pooling to reduce, step by step, the whole graph to a single output. Pooling are used in order to decrease the breadth of the graph while increasing the number of high-level features. An example here could be our use case: computing the free energy of a given molecule conformation.

Since the increasing interest in applying deep learning strategies and model to structured data, several approaches have been developed for non-Euclidean data [63] (graphs and manifolds), too. The basic principles have been taken from the deep learning literature for speech recognition, image, video, natural language processing

and so on. The data is usually represented in the Euclidean space, with a well-defined structure: images, for example, are composed by pixels in a 2-dimensional grid, neighbouring pixels can be exploited to capture patterns, while 2 distant pixels have very little to share. Moreover the number of neighbours is fixed and consistent (apart from borders) across all the structure. In graphs, these well-defined rules don't exist: edges represent an abstract relationship, not always about distance, that sometimes need to be labelled (to carry additional information) to specify, often in a qualitative way, the meaning of the connection. It's easy to think about a real example: in Facebook, while we mainly have friends (bidirectional relationship), we also follow pages (unidirectional). Moreover, we can get reactions on our posts, love, like, laughings and so on can be modelled as a qualitative relationship between friends and our posts.

Such data is large and complex and is a natural target for ML, however is a bit more complex to deal with, since the irregularities and its heterogeneous nature. Geometric deep learning [64] is the term that refers to this category of deep learning models that interacts with graphs and manifolds.

Non-Euclidean data doesn't have also a system of coordinates and a vector space implicit structure [64], unlike in images. Graphs can be irregular, with different number of nodes, variable number of neighbors, several qualitative (or quantitative) meaning of edges, making it difficult to make the same assumptions as the ones made for the image domain. For graph-level tasks, it also holds the permutation invariance property [65].

Permutational invariance

Given a graph G where each node have the same number and type of features, if we exchange the labels of two nodes, the regression function should be invariant to it.

More formally, given a function f and a graph G, $G' = f(G) = f(P \star G)$ where P is a permutation matrix and $\star$ is the *reordering operator* that takes care of reordering nodes and edges accordingly; the network should be able to predict the same graph-level value from both G and G'.

The approaches used on Euclidean data then cannot be used for graph as they are. The first used method is the **network (or graph) embedding** [66]: the graph structure can be encoded in low-dimensional vectors, in a learnable way, without having to use some heuristics or exploiting graph properties that may not be suitable for all the use cases. With this technique researchers are approaching the problem of using ML models on graph from the opposite point of view: bringing the graph to a structure that is usable by existing ML frameworks, known in literature. While

this seems a good idea it may not be optimal: why not dealing with graphs in their natural form? Graph Convolutional Networks tries to tackle the convolutional problem directly exploiting graph-related properties/structure in order to learn high-level representations.

4.2 Graph convolutional layers

Graph convolutional layers belong to two main categories: spectral layers and the spatial ones [63].

4.2.1 Spectral-based methods

The spectral-based NNs define convolutions as a filter of graph signal processing. They use the Laplacian $\mathbf{L}$, which is a matrix representation of the graph. L encodes the information coming from the adjacency and the degree matrix of the graph. For simple, undirected graphs it is computed as:

$$L = D - A \tag{4.2}$$

but we can define a normalized one with the following formula:

$$L = I_n - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \tag{4.3}$$

where I_n is the identity matrix of size n (that is the number of nodes in the graph) D is the degree matrix and A the adjacency one.

The benefit of working with the normalized one is that it is a real symmetric positive semidefinite matrix, thus it can be factored in eigenvalues (spectrum) Λ and eigenvectors $\mathbf{U}$, $\mathbf{L} = \mathbf{U}\Lambda\mathbf{U}^T$, therefore exploiting graph signals and graph Fourier transform theory. In doing so, the spectral convolution $\star$ is defined as a filter $g_\theta = diag(\mathbf{U}^T)$ over the graph signal x:

$$x \star_G g_\theta = U g_\theta U^T x \tag{4.4}$$

assuming that the filter is learnable (like every learnable feature map).

Due to the nature of Laplacian, convolutions in spectral-based approach face some limitations [63]:

1. Any changes in the graph, ends in a eigenbasis change
2. Learned filters are dependent on the domain taken into account, that means they cannot be reused with a different graph structure.

3. Eigen-decomposition requires $O(n^3)$ computational complexity.

While there have been lots of improvements on the computational complexity side, the main limitation that moves most of the research on spatial-based approaches comes from the fixed structure that graphs must have in order to use these convolutions. ChebNet [67], and further studies, solve this limitation by applying the filter on a localized portion of the graph, thus being applicable to graph of any size. Even though the improvement of these, they work on a graph signal representation and not directly on nodes and vertices as the spatial methods do.

4.2.2 Spatial-based methods

Similar to what we have in the convolutional layers for images, spatial-based convolutions work in the spatiality (neighborhood and forth) domain of the graph. A grid of pixels can be seen as a graph where each node (pixel) is connected to its neighbors with an edge, the RGB channels are the nodes features. When we apply a filter using conventional convolutions, we use a fixed $p \times p$ learnable matrix applied as a sliding window on top of $p \times p$ pixels; if we consider $p = 3$ it means that we are convolving the information present only the central pixel and its direct neighbors, like in a graph would be only a node and its direct connections. The analogy holds, but in graphs we have two main differences: the variable number of neighbors and the unordered structure. The main idea of spatial convolution in graphs still remains the same: to propagate the information through the edges in a learnable way.

As this will be the category of convolutions that we are going to use to build the desired model, an explanation of some convolutional layers is given in order to have a clear understanding of how they work.

Diffusion Convolutional NN [68] defines the convolutional operator as an operation that takes into account the probability of a message to be spread across neighborhood:

$$H^{(k)} = f(\mathbf{W}^{(k)} \odot P^k X) \tag{4.5}$$

where:

- $f(\cdot)$ is the activation function
- $P \in \mathbb{R}^{n \times n}$, the transition probability matrix is computed by $P = D^{-1}A$.
- $\mathbf{W}^{(k)}$ is the learnable matrix weight

The hidden representation has the same number of nodes of the initial one X and the probability is strictly dependent from the input size of the graph. We need

something that doesn't work with adjacency matrix so that the size of the matrix weights doesn't depend from the number of nodes in the graph.

The Message Passing (MP) protocol is the first to define a general framework that poses the fundamentals for spatial-based convolutions. As before, edges are used to propagate the information, but the main difference with respect to methods that use the adjacency matrix A is that here we work at the single-edge level. The general message passing convolution can be expressed [69] as:

$$h_v^{(k)} = U_k(h_v^{(k-1)}, \sum_{u \in N(v)} M_k(h_v^{(k-1)}, h_u^{(k-1)}, x_{vu}^e)) \tag{4.6}$$

where:

- $h_v^{(k)}$ is the node features of v at the k-th step of message passing, where $h_v^{(0)} = x_v$
- $M_k(\cdot)$ is a learnable function that takes into account a single neighbor u of the node v and the features of the edge linking v with u. The features are taken from the $k-1$-th step.
- $U_k(\cdot)$ is a learnable aggregation function that combines the $k-1$-th step information of the current node v with all the features coming from the neighborhood (a sum of functions $M_k(\cdot)$.

While not specified in the reference, it should be highlighted that the function that aggregates the neighborhood information is not fixed (as represented by the sum in Equation 4.6) but based on the several implementations may differ from formula to formula. Some of the most used are sum, average and max. Overall, this approach is a clear framework that works on the local structure and provide the starting point to build meaningful abstract hidden representation of nodes and graphs.

GraphConv Implementing the idea of MP, the GraphConv layer [70] uses the edges as a channel into which spread the nodes features: after each convolution step, nodes have a higher-order meaning due to the weighted aggregation of neighborhood information. The formula used to compute the hidden representation of the node v at the k-th step of convolution is:

$$h_v^{(k)} = \mathbf{\Theta}_1 h_v^{(k-1)} + \sum_{u \in \mathcal{N}(v)} \mathbf{\Theta}_2 h_u^{(k-1)} \tag{4.7}$$

Weighted sum is chosen as the function U_k in the Equation 4.6 with $\mathbf{\Theta}_1$ and $\mathbf{\Theta}_2$ as parameters, while M_k is a normal sum.

While it exploits better the concept of neighborhood compared to spectral approaches, in this simple definition all the v's neighbors features are weighted (and so learned) the same way: the function $M_k(\cdot)$ doesn't distinguish between nodes, apart from the edge features that might be unique in the link between v and u.

To broaden the range of capabilities of GNNs, an *attention* concept has been delineated, weighting the information of each node based on its importance in the graph. Graph Attention Network (GAT) [71] exploits the concept of attention to learn the weight α_{uv} of a connection between two nodes u and v. Attention in graph is a very powerful mechanism because it enables to specify different weights to different neighbors without requiring costly matrices operations and with the option to having a variable number of neighbors. In GAT the convolutional operator is defined as:

$$h_v^{(k)} = \sigma(\sum_{u \in N(v) \cup v} \alpha_{vu}^{(k)} \mathbf{W}^{(k)} h_u^{(k-1)} \tag{4.8}$$

But how is the attention weight defined? For GAT it comes after a *softmax* operation over all the connections of the given node v:

$$a_{vu}^{(k)} = softmax(g(\mathbf{a}^T[\mathbf{W}^{(k)} h_v^{(k-1)} || \mathbf{W}^{(k)} h_u^{(k-1)}])) \tag{4.9}$$

where $g(\cdot)$ is the activation function, $\mathbf{a}$ is the learnable vector and $||$ is the concatenation operator. Using the *softmax* operation ensure an even distribution of coefficients all over the neighborhood but favours one connection despite the others (being weighted very low). This might be not perfect since many connections can be important, therefore further analysis and evolutions of attention mechanism use multi-head attention where multiple learnable $\mathbf{W}$ matrices are used, allowing multiple subspaces representation at the same time. The multi-head attention concept, in turn, can be extended more by weighting differently the several subspaces created.

After this broad description of most of the available and used techniques in the state-of-the-art, we will go into deeper details for some convolutional and pooling layers that were used for the model proposed in section 6.2.

GATConv Extending the basic functioning of the previous layer, GATConv [71] uses the attention concept in order to build a self-attentional layer, that is able to predict/classify even on unseen graphs since it can compute an attention score based on the neighbors features. The formula used for the convolution is very similar to Equation 4.7:

$$h_v^{(k)} = \alpha_{v,v} \mathbf{\Theta}_1 h_v^{(k-1)} + \sum_{u \in \mathcal{N}(v)} \alpha_{u,v} \mathbf{\Theta}_2 h_u^{(k-1)} \tag{4.10}$$

with the only addition of the attention coefficient α.

4.3 Graph pooling layers

Before going into presenting other layers we have to distinguish between the two main types of graph pooling: fixed and learned. Fixed ones use an algorithm that doesn't learn over time: each input instance i is treated the same way, reasoning only on that instance without learning from already-seen instances $i-1$, $i-2$, $i-3$, ... Instead the learned pooling layers exploits the same basic principles of neural networks by using matrices, arrays of learnable weights and bias, weighted through backpropagation.

As with the conventional CNN for images, pooling layers work well when coupled to convolutions. While increasing the depth of the model (the hidden channels, or nodes features), we decrease the width (number of nodes) by discarding the less important nodes: pooling layers enable Graph Convolutional Networks to reason over groups of nodes with an abstract (hidden) meaning instead of having all the graph information still divided into the same number of nodes. This layers allow also the GNNs to be more computational efficient, by discarding (or aggregating) some of the nodes/edges in the graph, without losing much accuracy and at the same time generalizing the input (helps in avoiding overfitting).

The first and most simple pooling methods have been introduced by looking at the ones used for regular 2d data (images). Like in the pooling layers for images, we have a sliding window of data to take into consideration on which we apply one aggregation. The main difference here is that 2d data are on a grid, it is then easy to decide which data are close, and so to be considered in the same window: this concept is non-existent on unweighted graphs like the one I am working on. In order to build something meaningful, two main approaches have been used, operating at different levels of the graph:

- Graph-level pooling layers: The outcome of the pooling is computed on the whole graph as window of the data. It is easy to apply to any network structure since you don't have to provide any additional information: each node belongs to the same cluster, the whole graph. While it's easy, it may not be efficient: a lot of information can be lost and a change of a single node might impact a lot on the final result, even if the most of the graph is remained the same.

- Cluster-level pooling layers: Nodes are assigned a cluster identifier and then a pooling is applied independently on each cluster. Compared to the graph-level pooling this is more resilient to little changes in the graph structure since the single change in a node it is forwarded only on the respective cluster pooling outcome. On the contrary there is the unresolved problem of how you do

decide, in advance, the belonging of a node to a cluster. To use this pooling effectively you need to study the structure of the graph in advance and see whether you can split it into multiple meaningfully clusters.

On top of this definitions max, average and sum pooling have been built: each of them can be applied at both graph and cluster-level.

An important restriction on the usage of these approaches is that you have to know in advance the number of clusters you want to split the data in, unless you use dynamic algorithms that computes the clustering for you. In that case you may have to pay attention since then the number of output values is variable, meaning that you might have to tweak your model accordingly. Moreover after the pooling you don't have anymore the graph structure that you want because they are not able to reason over the edges meaning that you can use these poolings only as the last layer, where you cannot perform anymore graph convolutions.

While this might seem a good idea, we are still not considering so much of the structure: these poolings focus on the nodes features without considering edges, that are indeed the useful information that has to be exploited in order to understand which nodes have to be considered close each other.

One of the most challenging feature of pooling layers for graphs is that we should keep the structure consistent, with new nodes and edges connected in a meaningful way. It is easier to reason about discarding some nodes based on the features they have, but what about the resulting graph without them? What if the new graph now is not connected anymore? By *connected* we are referring to an undirected graph where among each pair of node u, v there is a path that links u and v. Pooling layers should not only consider nodes features but topology of the graph so that after having pooled it, it still has a structure that has its own meaning because it may be the case where a model combines multiple times convolutions and poolings like they do for images. I will present two pooling layers that are structure-aware to see how they would like to solve this problem.

EdgePooling EdgePooling [72] belongs to the learned category; it works at edge-level and keep consistent the structure: while removing edges from the graph, it collapse the information in a learnable way to a new node. How to decide which edge (and so the nodes connected by this edge) to collapse? It uses the concept of edge contraction that is based on scores computed for each edge on the graph: this is helpful also because it's a good step towards a meaningful modification of the graph structure instead of working only on nodes features.

The EdgePooling algorithm (as explained in [72]) is the following: we have the input graph $G = (V, E)$, each node $v \in V$ has f features that are represented

in the matrix $V \in \mathbb{R}^{v \times f}$. Each edge $e \in E$ are a set of directed pairs of nodes without weights (or features). Edges that has to be contracted are chosen based on a computed score s. For each edge e_{ij} that connects node i to node j, the raw score r is first computed as a linear combination of the concatenation of the nodes features:

$$r(e_{ij}) = W \cdot (n_i \,||\, n_j) + b \tag{4.11}$$

where n_i and n_j are the nodes features, respectively for i and j, W and b are the learnable parameters. Then, it applies a local softmax normalization on all the outgoing edges of a node: the final score s_{ij} is then the value of softmax adjusted such that the average of the score range is close to 1 in order to not have numerical problems during the unpooling procedure.

$$s_{ij} = 0.5 + \text{softmax}_{r_{*j}}(r_{ij}) \tag{4.12}$$

After this phase we have all the edges in the graph with a score: starting from the highest-scored one the algorithm contracts each edge that has not a node involved in another contraction. How does this action work? Starting from two nodes i and j and the edge between them, we want to build a new structure, that doesn't differ much from the previous one by:

- Removing the edge
- Not changing the rest of the graph structure: this means that if i and j have other edges I should keep them in my collapsed new graph
- Deciding which information has to be kept when merging the two nodes

In order to remove the edge, the algorithm has to remove the two nodes i and j and create a new node k that inherits all the other edges of i and j. This doesn't change the whole graph structure. And about the merge of the information of i and j in k they found out that a simple sum of the node features works well, simply multiplied by the edge score:

$$n_k = s_{ij}(n_i + n_j) \tag{4.13}$$

This also is good in terms of computational complexity, compared to other learned pooling methods: it is linear in the number of edges. But the most important key point of EdgePooling is that it is reliable to local changes: since it's functioning is local and sparse, a change in a node will reflect to a local change of the graph structure resulting out from pooling and not in a totally different one. That is meaningful for our problem because a local change in a molecule may introduce a

change in the free energy but it is local to the part of the molecule that changed and may not influence the whole. At least this happens in simple systems; in complex ones we have a tertiary structure that is influenced by distant atoms but for that, as we have already seen, we need additional information.

ASAPooling Most of existing methods fail to capture the graph structure and are not suitable for large graphs [73]. To overcome these problems, ASAPooling has been created. It is a sparse method that uses a self-attention network approach, exploiting the usage of a GNN to compute attention for each node in the given graph. ASAPooling computes, for each node its cluster assignment, then the clusters are scored using a GNN. By using a *ratio* parameter, it chooses only a fraction (the top scoring) of the available cluster and edges are created among the selected neighboring clusters. Vertices in a cluster get merged to a single node using an attention sum (weigthed sum where the attention coefficients are learned based on the feature values). About cluster assignments: they use a local clustering algorithm [74] which considers neighbors up to h hops distant.

4.3.1 Graph readout layers

When doing graph-level tasks, the network has to return a single batch of values (c for c classes or 1 value for prediction), therefore keeping the information distributed in each node, all over the graph, with an unmodified structure, doesn't let the network to have a unified representation of the all graph, which is mandatory to compute the desired output. To answer this problem there is a category of layers (or functions) called *readout*. They are generally defined as learnable (but could be also fixed) functions $R(\cdot)$ where:

$$h_G = R(h_v^{(K)} | v \in G) \tag{4.14}$$

Two main challenges have to be taken into considerations when speaking about readout operations on graphs: they must work for graphs with a variable number of vertices, producing a fixed-size output while being permutation invariant. A readout layer could be considered as a sort of a pooling method: it has to produce a **fixed** representation of the current status of the graph, possibly a condensed one since we want to group information in the most abstract way to reach then the single value prediction (or classification).

Two pooling algorithm that can be easily used as readout layers, working at node-level pooling and so not considering the structure, are the SortPooling [75] and the TopKPooling [76].

The purpose of both the layers is the same: choose the k best (depending on the implementation) nodes, with k fixed, across the whole graph independently from the number of nodes in input. The way of choosing the k nodes represents the difference between the 2 layer implementations. SortPooling is a fixed pooling, it doesn't have learnable parameters; its main goal is to sort all n nodes in the graph and then pick the first k. The sorting operation is done by looking at features values, starting from the last channel descending, thus, in theory, granting the permutational invariance property since the order is computed only on features values. TopKPooling instead is a learnable pooling layer that learns the importance of nodes in the graph (with a global attention mechanism) and then pick the k most important nodes. Both cannot be used as intermediate pooling because they don't preserve the graph structure, therefore making the successive graph convolutions impossible. That is the main reason why I put them in the graph readout layers.

Flatten The most simple and less flexible layer of readout is the *flatten* one: it flattens the structure by concatening all the nodes features in the graph. With this method we do not lose information, since all the nodes are used to build the next layer, but it's not possible to handle graphs of variables sizes (number of nodes), apart from using a very raw zero-padding technique. Moreover, it is not order invariant since the next layer weights will learn the position of the flattened nodes to be able to learn the importances for the prediction. Practically speaking, this function doesn't consider edges at all but only nodes: given a graph, after all the convolutional and pooling layers, with n vertices and c features each, the resulting output from the flatten layer would be a one-dimensional vector long $n \times c$ coming out from concatenating the c features of $v_1, v_2, ..., v_n$ vertices.

$$f_G = (h_1||h_2||...||h_n) \tag{4.15}$$

Others One very simple way to build a good readout function would be to use a global max/average/sum pooling so that one works at feature-level grouping all the nodes to be one. It is permutation invariant and with a fixed-size output, but there is a problem of expressivity: with only one node that sums up all the abstract meaning of the graph you'll lose a lot of potential and maybe you'll need a lot of features to be able to still do a good prediction.

Chapter 5

Dataset

In this chapter I will give a detailed description of all the steps that were accomplished to produce the dataset used for ML models presented in the upcoming chapters. The dataset has been built starting from data produced by an *in-vacuum* MetaD simulation of alanine aipeptide. The frames extracted from the simulation were enriched with information found in the force fields and in the topology, both provided by computational software tools. The JSON format was chosen as file format for the dataset because of its flexibility and adaptability to different languages.

5.1 Pipeline description

A dataset for a supervised model is composed by a list of instances with the respective label (target). In the case of the free-energy calculations problem, each instance is represented by a conformation of the given molecule and the label is the free-energy value that the conformation has.

The aim is to build a general procedure that starting from a simulation is able to create a labelled instances, used to form the dataset for the prediction task. While this procedure is general, thus applicable on any simulation with the same force field, in this work it has been followed only for our reference molecule, Alanine Dipeptide.

The pipeline is composed by the following, sequential, actions:

1. Creation of the initial Alanine Dipeptide representation
2. *in-vacuum* Well-tempered MetaDynamics simulation
3. Checks for simulation convergence
4. FES estimation

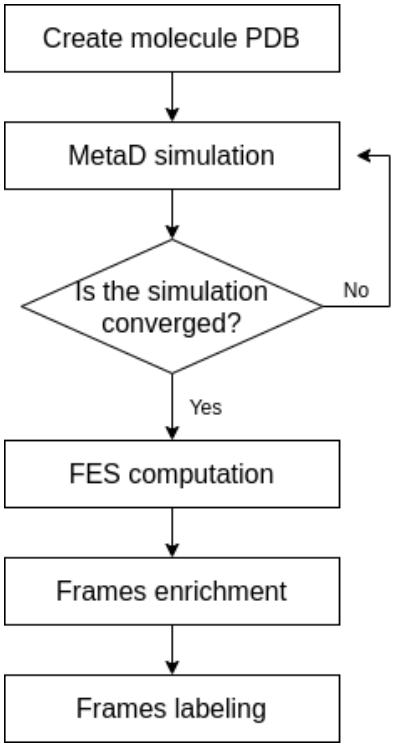


Figure 5.1: Dataset creation pipeline schema

5. Extraction of frames from the trajectory as a list of PDB files
6. Processing of PDB and topology files in order to create a unique JSON file per frame with all the relevant information
7. Dataset assembling through the association of each frame to a single free-energy value

Let's discuss more in detail how this process works.

5.2 Creating Alanine Dipeptide structure

In order to carry on the MetaD simulation, we need to obtain a digital representation of the Alanine Dipeptide. The most used file format to store molecule information in the proteins and ligands world is the Protein Data Bank (PDB): a textual format representing the molecule in its three-dimensional structure.

The PDB file has been built from scratch, having in mind the atoms and bonds that compose the alanine dipeptide, using the Chimera software. Chimera [77] offers the ability to create, observe, modify and store (as PDB in this case) molecules in their three-dimensional shapes.

5.2.1 Protein Data Bank file format

```
CRYST1    0.000    0.000    0.000  90.00  90.00  90.00 P 1          1
ATOM      1 HH31 ACE X   1      -0.991  -3.323  -2.514  0.00  0.00
ATOM      2  CH3 ACE X   1      -1.608  -2.424  -2.530  0.00  0.00
ATOM      3 HH32 ACE X   1      -2.635  -2.731  -2.729  0.00  0.00
ATOM      4 HH33 ACE X   1      -1.254  -1.674  -3.237  0.00  0.00
ATOM      5  C   ACE X   1      -1.734  -1.800  -1.208  0.00  0.00
ATOM      6  O   ACE X   1      -2.836  -1.472  -0.797  0.00  0.00
ATOM      7  N   ALA X   2      -0.616  -1.460  -0.573  0.00  0.00
ATOM      8  H   ALA X   2       0.287  -1.485  -1.025  0.00  0.00
...
```

Figure 5.2: Extract of the hand-crafted Alanine Dipeptide PDB file

The PDB file format is fixed-column delimited: each field has a fixed, maximum number of characters that can occupy. Based on the first keyword, for each line we have several different next values.

For the purpose of the work, the useful lines are the ones starting with the keyword `ATOM`. Each of them contains the information of a single atom of the conformation such as:

- Atom serial number
- Atom name
- Residue name
- Chain identifier
- Residue sequence number
- x, y and z relative coordinates in Angstroms
- Occupancy and temperature factor

I am not covering the additional information that can be present in each of the `ATOM` records because it would be out of the scope of the present work.

The file created in Chimera is very simple and does not contain many PDB sections which, instead, you may find when visiting `rcsb.org` looking for complex proteins. For a detailed documentation of all the sections, one can refer to the PDB specification website [78].

5.3 Well-tempered MetaDynamics simulation

Starting from the `PDB` of the Alanine Dipeptide, several tools have been used to run the simulation and then build the dataset. These are the most used in the Molecular Dynamics field:

- GROMACS [79] version 2019.4 [80] is a free tool that allows to do Molecular Dynamics simulations in a high-performant way. It is designed for biochemical molecules like proteins that have a lot of bonded interactions. It has an excellent support for CUDA devices, helping in the speed up of the process.
- Amber [81] provided libraries and topology files. There we found useful information about force field parameters, atomic properties and interactions coefficients such as atom mass, radius, partial charge, Van der Waals coefficients and so on.
- PLUMED [82] is an open-source, community-developed library that provides a range of different methods such as enhanced-sampling algorithms, free-energy methods, tools to analyze the amounts of data produced by MD simulations. It has been used to modify with additional potential (according to WT method) the Alanine Dipeptide *in vacuum* MD simulation.

Since the goal is to estimate the FES of Alanine Dipeptide, we chose a Well-tempered MetaDynamics as the best method to achieve it because of its ability to converge to a finite value in the long run (subsection 2.3.1). WT is carried on by PLUMED while the GROMACS MD simulation is running: every x time steps, PLUMED will put external bias potential, in the form of Gaussians, which modify the forces driving the simulation.

5.3.1 Alanine Dipeptide in vacuum

Before going into detail about the important parameters for the WT, the configuration used for the MD simulation is presented.

```
Steepest Descents converged to Fmax < 1000 in 7 steps
Potential Energy  = -6.5387970e+01
Maximum force     =  9.7676282e+02 on atom 15
Norm of force     =  4.0457389e+02
```

Figure 5.3: Output of the minimization step

The minimization step has been done using the steep descent method, it stopped after a few steps because in vacuum the minimization is very quick (no solvent atoms).

The MD simulation has been carried on for $200ns$ with a timestep $\delta t = 1fs$, using the leapfrog integration algorithm [83]. The temperature of the system has been set to $300°K$ (room temperature) using the Berendsen thermostat [84]. Electrostatic forces and Van der Waals interactions, given n atoms in the system, have a computational complexity of $O(n^2)$. For larger systems there is the need of introducing cut-off schemes which avoid computing interactions between distant atoms, allowing to bring the complexity down to $O(n\log(n))$, like particle mesh Ewald summation method [85]. In our case, no cut-off for the atom-to-atom interactions has been used because, being an in vacuum simulation, the system is composed only from the main molecule atoms (no solvent), thus computing all the paired interactions it is still fast in absolute timing. The choice of a good timestep δt is important because by its value depends the accuracy of the integration of the forces. The smaller δt is, the more precise (with less risk of computing too strong forces) and the slower (more frequent computations) the simulation is.

As we have seen in subsection 2.3.1, the WT, using a reference temperature, adds external bias in form of Gaussians. The height of the wells decrease over time and based on the frequently visited CVs values. How do we select the most useful CVs? Which could be a good value for the initial height? and what about σ_i, the width per CV? After how many steps a new Gaussian should be put?
For the Alanine Dipeptide, as papers in literature suggest [50], we considered the two central dihedrals, phi (ϕ) and psi (ψ), as the CVs. The selection of the others hyperparameters influences the data we will obtain from the simulation. If a tall Gaussian is chosen, the FES wells will be filled very quickly but not in an uniform way: there would be a lot of spikes and new small imperfections on the surface. In the case of small initial Gaussian instead, the additional bias may not cover the whole FES, thus reporting converged, but not complete simulation data. In the case of the width's choice, a large Gaussian will not capture all the minima characteristics because it will cover them straight away; on the contrary with a narrow one we are

very precise but also slow. It is worth noting also, how a low *biasfactor*, may not give enough power to overcome barriers, thus our simulation might converge, but too early. The term *biasfactor* refers to what it's used in PLUMED as:

$$\gamma = \frac{T + \Delta T}{T} \tag{5.1}$$

which is the ratio between the temperature of the CVs $T + \Delta T$ and the system temperature T.

A good estimate of the σ_i widths has been chosen as the standard deviation (divided by 2) of the 2 unbiased CVs in a very short MD run ($100ps$).

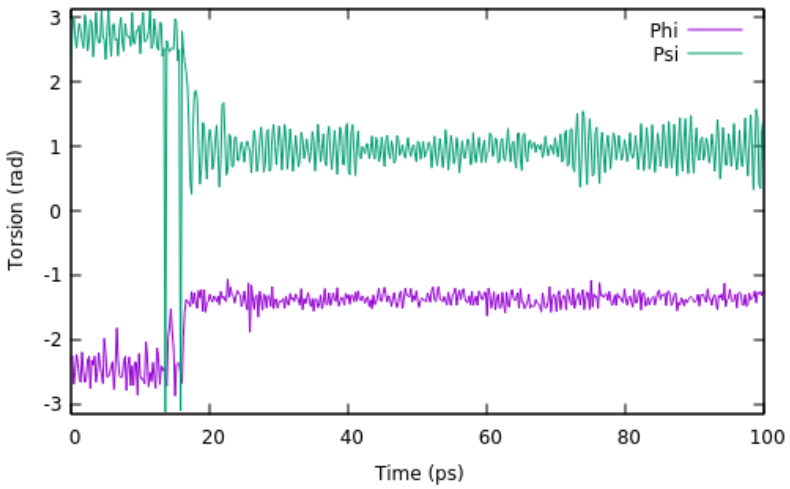


Figure 5.4: Phi and Psi oscillation in the micro MD run

The standard deviation on CVs has been computed after the first $20ps$, so as to consider measurings only when the molecule was stable in the minimum (as shown in Figure 5.4).

While the simulation is running PLUMED keeps appending information to the log file. This allow an early analysis of the CVs selected to make decisions during the run without waiting the end. One could launch a simulation with a very high time limit and then stop it after a while if he sees that it reached **convergence**. How does one check for convergence? What does it mean that the simulation converge? The concept of real convergence is broad [86]: one can only do some checks that suggest the system has reached it. When using Well-Tempered MetaDynamics, only the regions of the FES where the temperature of the CV reach at most $T + \Delta T$ are visited. This is a consequence of the formula that computes the dynamic height of

Force field	Amber99sb-ildn
δt	1 fs
Time limit	200 ns
W	1.0 kJ/mol
σ_ϕ	0.05
σ_ψ	0.12
γ **(biasfactor)**	10
pace	500

Table 5.1: Summary of the parameters used both for MD and WT simulation

the Gaussian. At convergence, we have a situation where all the feasible regions are covered and the additional bias plus the FES forms a flat surface. A non-converged simulation produces FES values that are not reliable, therefore several checks have been employed:

- Difference of energy between two minima (Figure 5.5) needs to remain stable, converging to a finite value. This because, at convergence, adding a Gaussian on a minimum creates a small imbalance which is compensated after a short time by another Gaussian in the other minimum.

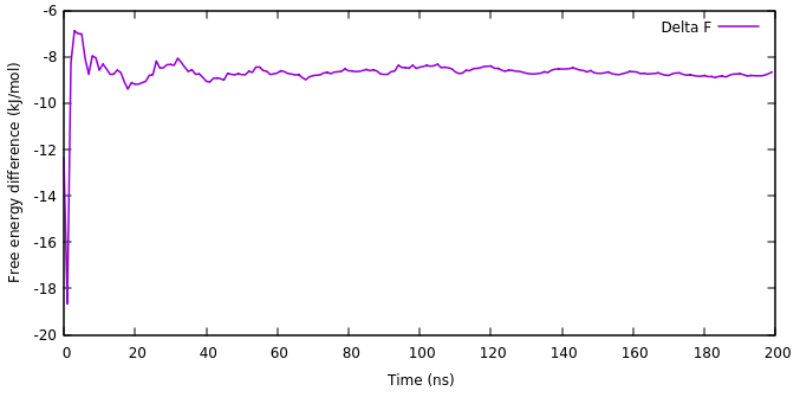


Figure 5.5: Free energy difference overtime between the two basins

- The system should sample in a semi-diffusive way the CVs phase space, keeping

in mind that a converged simulation explores most of the CVs space apart from the unfeasible conformations.

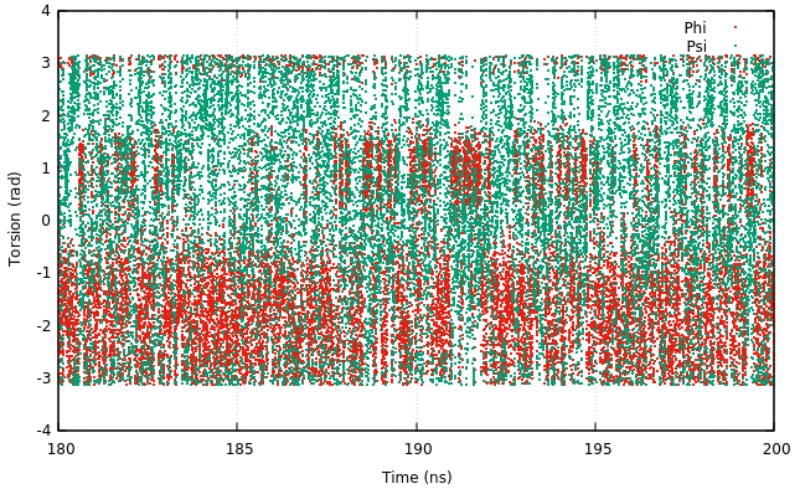


Figure 5.6: ϕ, ψ values in the last 20ns of the simulation. Each point belongs to a sampled conformation over time. Towards the end the CVs fluctuate quicker.

- FES reconstruction (Figure 5.7) over time helps in visualizing how much the free-energy is changing during the simulation time: the more we go towards convergence, the more the FES doesn't change apart from a constant offset because we are adding a small (given the small Gaussian height) layer all over the flat surface.
- Gaussians height goes towards 0 at convergence.

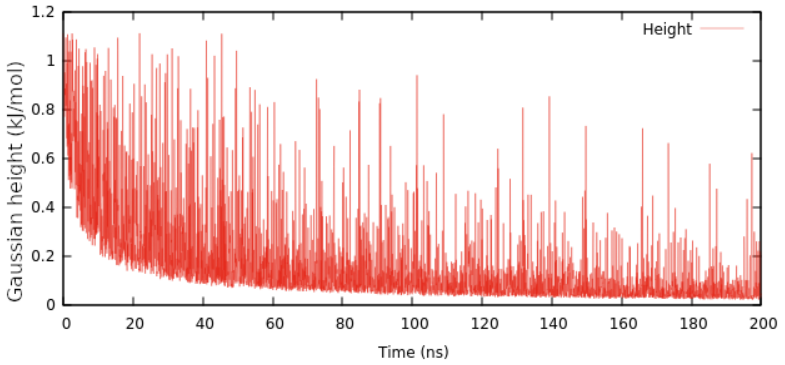


Figure 5.8: Gaussian dynamic heights over time

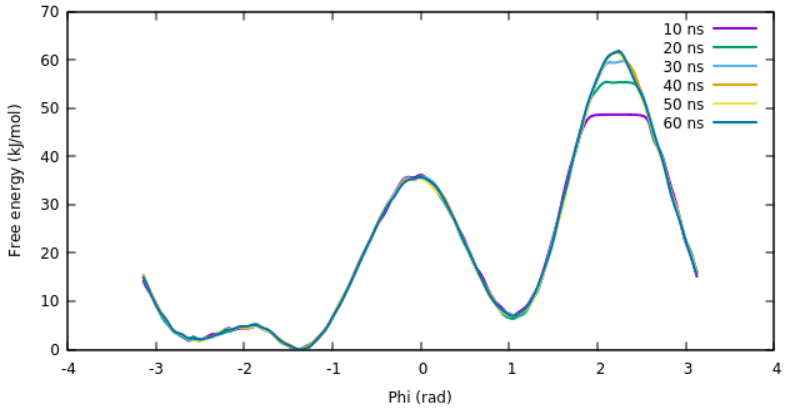


Figure 5.7: Free energy reconstruction overtime

Now that we are sure that this simulation was at convergence, we know that FES estimation in function of the two selected CVs will be reliable. While doing the simulation, PLUMED wrote a `HILLS` file containing all the Gaussians deposited over time with the corresponding CVs values where they have been added to. We know that the FES is the complement of the sum of the external potential so we can compute it very easily (PLUMED comes in help with the `sum_hills` command).

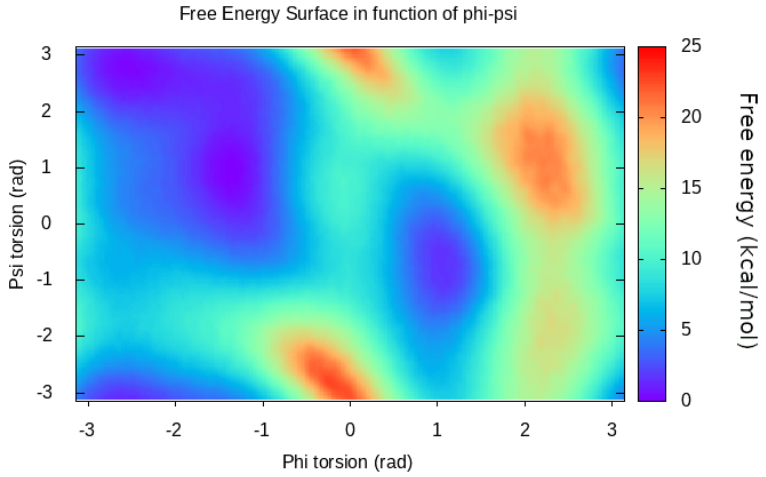


Figure 5.9: Final FES of the converged WT simulation of in-vacuum Alanine Dipeptide as function of ϕ and ψ

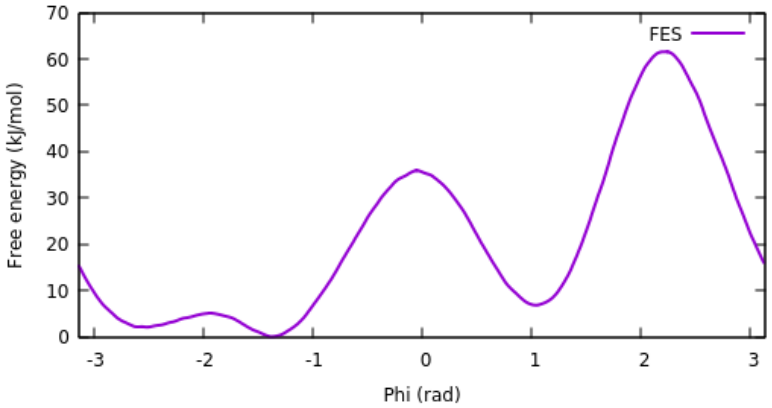
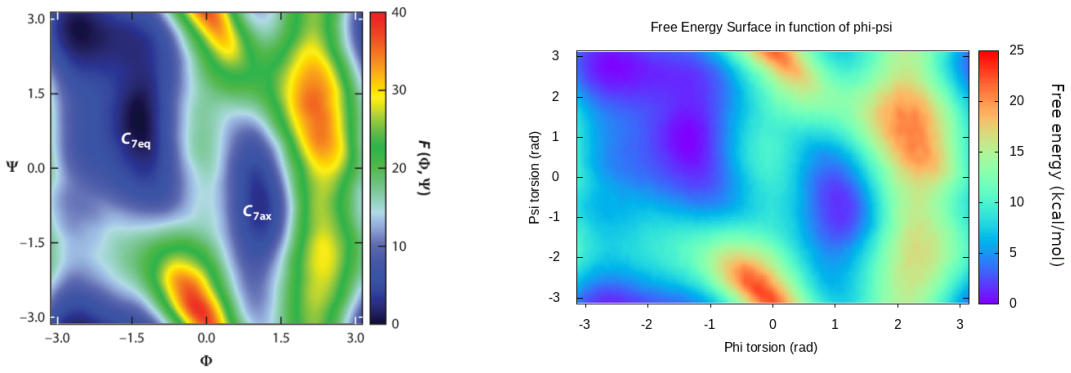


Figure 5.10: Final FES of the converged WT simulation of in-vacuum Alanine Dipeptide only represented as function of one CV: ϕ

The FES is stored in a columnar file with 3 fields: ϕ, ψ and the corresponding free energy. This is used to assign a value to each of the trajectory's frames.

A graphical FES, computed from the MetaD simulation, is shown in Figure 5.9 and Figure 5.10, respectively in 2 and 1 dimension.



(a) FES in literature [87]

(b) Our simulation FES

Figure 5.11: Comparison between literature and computed FES

5.4 Dataset creation

Having in mind the goal of training a NN to predict the FES of the Alanine Dipeptide, we have to build our dataset accordingly. Each dataset's instance is represented by a snapshot (frame) of the system at a certain time, labelled with the corresponding FES value. Since the FES obtained from the simulation is represented as a discrete function of the two dihedrals, to assign to each frame a value we have to choose the closest pair of ϕ and ψ in the FES domain. It means that two frames with similar (but different) value of ϕ and ψ might have the same closest point, and so, the same free energy label.

The simulation produced 400′000 snapshots of the system (in the PDB format), one every 5 ps. We decided to select 50′000 among the total because the process of converting them to a better format, readable by the ML algorithm, and the training itself of the model would waste a lot of time and computational resources.

The goal is to include every meaningful detail of the conformation in a JSON file (the instance) to be able to have a 1-to-1 representation which can be converted again to a PDB file in case. The algorithm take as input a PDB file, the conformation, producing a JSON file with the enriched data, ready to use.

The easiest way to deal with PDB files was by using the BioPandas [88] library that offers out-of-the-box parsing, error support and conversion to Pandas [89] dataframe. Based on the atoms name, I enriched their properties with atom radius, mass and partial charge found with the help of supplementary libraries file from Amber (force field, topology).

The detailed content of each frame's JSON can be so summarized for a given a molecule with n atoms:

- For each atom a_i, $i \in [0, n)$: mass, radius, partial charge and x, y, z coordinates.
- A $n \times n$ bonds matrix:

$$b_{i,j} = \begin{cases} 1 & \text{if } a_i \text{ has a covalent bond with } a_j \\ 0 & \text{otherwise} \end{cases} \tag{5.2}$$

  The matrix is symmetric by construction.

- For each planar angle p_i: the tuple of indexes of the a atoms array, named j, k, l, and the angle value.
- For each dihedral d_i: the tuple of indexes of the a atoms array, named j, k, l, m, and the dihedral value.
- A $n \times n$ non-bonded coulomb electrostatic matrix:

$$c_{i,j} = \begin{cases} 0 & \text{if } a_i \text{ in range 1-2 and 1-3 with } a_j \\ \dfrac{1}{cm_{i,j}} \cdot \dfrac{ch_i \cdot ch_j}{d(i,j)} & \text{if } a_i \text{ in range 1-4 with } a_j \\ \dfrac{ch_i \cdot ch_j}{d(i,j)} & \text{otherwise} \end{cases} \tag{5.3}$$

  where:

  - $cm_{i,j}$ is the multiplier used in the force field by Amber
  - ch_i and ch_j are the partial charges of a_i and a_j
  - $d(i,j)$ is the physical distance between the atoms a_i and a_j

  The matrix is symmetric by construction.

- A $n \times n$ non-bonded Van der Waals interactions matrix:

$$w_{i,j} = \begin{cases} 0 & \text{if } a_i \text{ in range 1-2 and 1-3 with } a_j \\ \dfrac{1}{wm_{i,j}} \cdot \left(\dfrac{C_{i,j}}{d(i,j)^{12}} - \dfrac{C_{i,j}}{d(i,j)^6} \right) & \text{if } a_i \text{ in range 1-4 with } a_j \\ \dfrac{C_{i,j}}{d(i,j)^{12}} - \dfrac{C_{i,j}}{d(i,j)^6} & \text{otherwise} \end{cases} \tag{5.4}$$

  where:

- $wm_{i,j}$ is the multiplier used in the force field by Amber
- $C_{i,j}$ is a coefficient used to compute Lennard Jones potential and depends on pairs of atom types (A full description can be find at [25], equation 5.117).
- $d(i, j)$ is the physical distance between the atoms a_i and a_j

The matrix is symmetric by construction.

Each file is named with a number, which will be used to index the free energy value in the target file.

The complete list of JSON files is then saved in an appropriate folder. For each frame, the closest, in terms of ϕ and ψ, free energy value is chosen and saved in the target file. This file is just a list of values, referenced by the frame number (contained in the filename).

Chapter 6

Methods

In this chapter a description of the reasoning behind the modeling choices is given with a focus on how several representation and implementations may be suited for the free-energy prediction task.

The goal of this work is to build a NN that is able to predict the free energy of alanine dipeptide conformations, with a reusable and extensible approach. This means that the model should rely on the structure of the molecular representation and not on only its finite number of parameters. Exploiting the structure could help in the training and open the way towards more complex approaches in which different molecules are fed into the NN. Future extensions could bring the model to the prediction of molecules not present in the training set.

In order to reach such goal we have to make some initial considerations and define the problem statement.

Problem statement Given a conformation, the model has to predict its free-energy value coming from a converged simulation. The free-energy value of a conformation c is nothing but the difference between it and c_m, where c_m is the conformation with minimum free-energy value of the system:

$$\Delta G = G(c) - G(c_m) \tag{6.1}$$

where $G(c_m) = 0$. $\Delta G \in \mathbb{R}$, thus we are facing a regression task, for which a NN will be used.

In the previous chapter we tackled the problem of creating a dataset starting from a simulation trajectory but we didn't put the focus on the important questions that will make us understanding better the problem of estimating the free-energy:

- How do we represent a molecule?

- Which is the important information contained in each frame c that describes $G(c)$?
- What should be used as input of the NN?

While answering the questions, I will outline also how the choice of the molecule representation heavily influenced the input features for the NN and vice versa.

6.1 Molecular abstract representation

Being able to build a network that exploits the structure of the molecule to find insights about the FES is a key point because it leads to some important progress towards the generalization of the model:

- A general model doesn't rely on a fixed number of inputs, therefore enables training on different molecules.
- Testing the model on a new, unseen molecule, chemically similar to one seen by the model in the training set, could give us preliminar insights without doing the full MetaDynamics simulation.

Given the diversity of molecular information to be modelled, we identified several candidate design choices:

- Use a single, large, graph:
  - Atoms, with their properties, are modelled as vertices;
  - Two category of different edges model all the atom-to-atom relationships: unweighted for bonds (or weighted 1s and 0s), weighted for Van der Waals and electrostatic interactions;
  - How do we model the information that is in angles and dihedrals? The only possible way to include this into the single graph is by adding additional set of vertices. These, as opposed to atoms, have only one feature, the angle value, and they will be connected to the atoms they are in relationship with.
- Using a cluster of graphs. We may use one graph for each purpose, in order to have a molecule that is described in its entirety by several sub structures:
  1. One graph for each kind of atomic interactions:

- Bonds
- Van der Waals
- Electrostatic

2. For dihedrals and angles, the single atom cannot be modelled as vertex. Modeling tuples of atoms instead could be a solution, the angle then is the relationship between two tuples of atoms (e.g. an angle ABC is the edge that connect AB to BC).

- Using hypergraphs (subsection 6.1.1). This design choice seems to be most suitable: hypergraphs let you define sets of relationships between a number of nodes that is ≥ 2. Perfect for our use case (with angles and dihedrals).

- Discarding less important categories of data in order to have a single graph. The molecule is described by all these different concepts, but do we need them all to do the FES prediction? We might understand first what is, for the selected dataset instances, the important information that enable us to predict, with a certain accuracy, the free energy value.

The single graph might include everything, but we have a very complex structure: different types of vertices and edges, both with or without features.

The cluster of graphs might be optimal if we plan to handle each single graph in a different way (given their differences in terms of features). What might be difficult is that we have to converge and return a single value from the whole cluster.

Given these limitations on the former approaches, we decided to focus on hypergraphs and custom alternative solutions (which model relevant information only).

6.1.1 Hypergraph

An hypergraph [90] is a pair $G = (V, E)$ with $|V| = N$ vertices and $|E| = M$ hyperedges. Each hyperedge $e \in E$ is a set that has its own cardinality ≥ 2. Hyperedges represent the important difference between graph and hypergraphs: while in a graph an edge has fixed cardinality of 2, an hyperedge goes beyond that.

For the single conformation, defining what are the main components of the hypergraph it's straightforward:

- Vertices denotes atoms and all their properties

- Each atom-to-atom interaction (bonds, Coulomb, Van der Waals) is represented by a set of weighted hyperedges of cardinality 2

- Planar angles are modelled as a set of weighted hyperedges of cardinality 3
- Dihedrals are modelled as a set of weighted hyperedges of cardinality 4

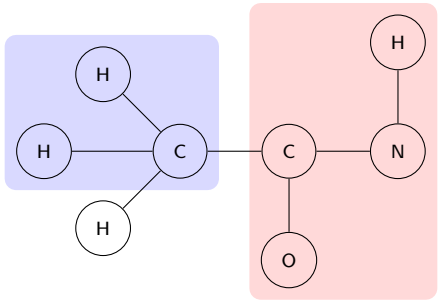


Figure 6.1: Simplified hypergraph representation of a portion of Alanine Dipeptide. In purple a planar angle, hyperedge of cardinality 3. In red a torsional angle, hyperedge of cardinality 4.

Even if this representation seems the most suitable, it is difficult to work with because of the lack of NN convolutions and pooling layers implementation (such as [90]). The main limitation is that the information needed to predict the alanine dipeptide relies on dihedrals values, which are modelled as hyperedge features. All the implementations of hypergraph convolutions consider edges only as a way of spreading the information (through the Message Passing protocol seen in chapter 4), thus making it impossible for the hyperedge features to be used as key information for the regression task.

6.1.2 Dihedrals overlap graph

All the previous solutions take into account every single molecular property, but are all of them needed to predict the FES? Since, by construction, the simulation estimated the FES in function of ϕ and ψ, we should build a representation around the dihedrals concept. Modeling them as vertices of the graph, is important if we want to exploit existing convolutional layers, since they work by propagating the node information with the neighborhood, with very little attention instead to edges weights (or features).

How can we use this assumption to build a graph structure with dihedrals in mind? The intuition comes from the overlap graphs, very common in bioinformatics [91, 92].

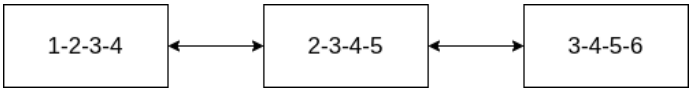


Figure 6.2: Resulting overlap dihedrals graph of 6 connected atoms, in a linear structure.

In graph theory, an overlap - or De Bruijn [93] - graph is a structure representing overlaps between sequences. Any node in the graph represents a fixed-length (n) subsequence and it is connected to another if they share $n-1$ consecutive characters.

Even though graphs are not modellable as plain sequences, we might exploit this representation to build an unweighted overlap graph (Figure 6.4) where:

- Dihedrals, written as an ordered sequence of atoms identifiers, are the vertices of the graph
- Two dihedrals are linked by an edge if they overlap 3 atoms.

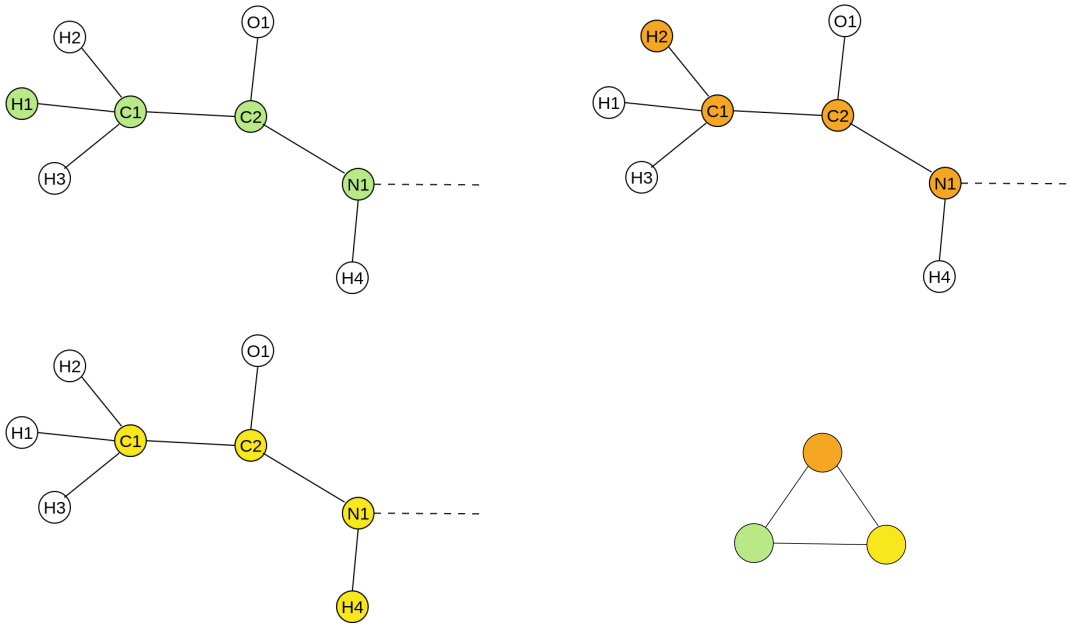
One may want to connect dihedrals more "distant" (with only 2 overlapping atoms), but to do that we should introduce edge weights, that for now are avoided. The reason behind this design choice is that we want to keep the model as simple as possible. However, the expressivity of the unweighted graph is the same as the weighted one. Indeed, we are still able to represent the connection between dihedrals with 2 overlapping atoms with a 2-hops path.

One undesired effect of this representation is that two dihedrals on the same bond are 1-hop distant, the same happens with two adjacent dihedrals (on different bonds). To explain better, in Figure 6.3 orange and green represent the information of the same dihedral (the torsion is on the bond between `C1 and C2`); the yellow instead is a different one. In the final overlap graph, though, you cannot tell the difference since they are 3 different nodes, each of them 1-hop distant.

The dihedrals overlap graph is not extensible to other type of molecular information. For a simple system as alanine dipeptide, the FES is well described by the backbone dihedrals. However, one should note that if more complex systems are investigated, additional data should be provided and modelled (i.e., the interatomic interactions).

6.1.3 Simplified dihedrals overlap graph

The previous representation design was driven by the fact that each tuple of 4 atoms is considered an independent dihedral.



(d) Resulting dihedrals overlap graph

Figure 6.3: Example of 3 dihedrals overlap graph representation

With a different point of view, we might observe that when the torsion on a bond change, all of the 4-atoms dihedrals on that bond change accordingly by the same shift. Therefore, starting from one torsional value we can derive others by adding, or subtracting the shift occurred. This means that the information from having multiple 4-atoms dihedrals representing the same torsion could be discarded in advance. Therefore each bond, on which a torsion occur, is represented by just one value instead of multiple. While simplifying the graph, by removing vertices and edges, we are also solving the inconsistency outlined in Figure 6.3. So, for each tuple of 4 atoms, we consider only the 2 central ones as representative of the torsional.

The analogous graph now becomes made up of:

- Central pair of atoms that represents the dihedrals as vertices.
- If two vertexes share one atom, we draw an edge.

In this case we scaled down the complexity of the alanine dipeptide to a very simple graph composed by 7 nodes and 7 edges (Figure 6.5).

While the structure is very simple and plain for this molecule, it may expand a lot for more complex ones.

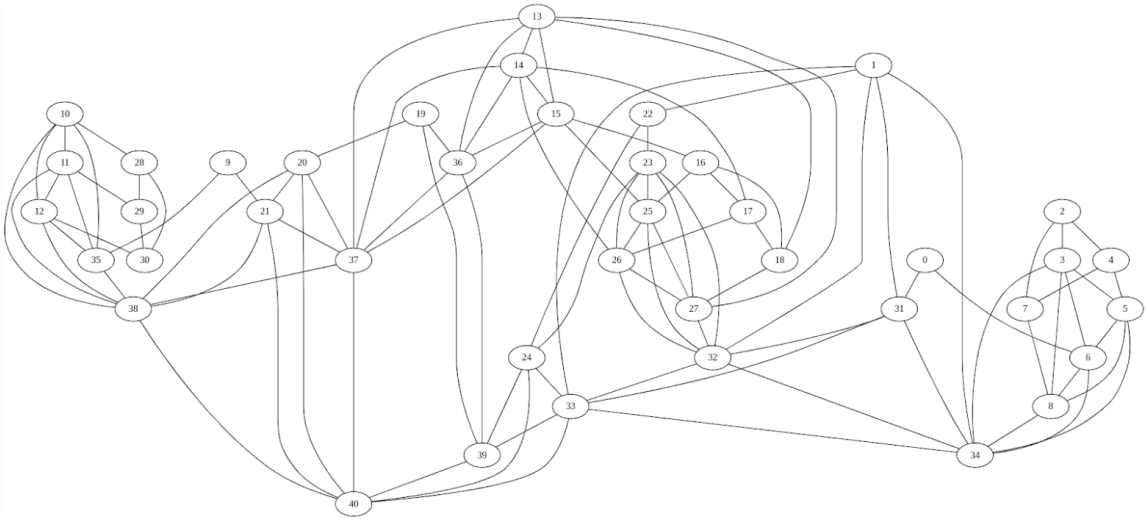


Figure 6.4: The Alanine Dipeptide dihedrals overlap resulting graph

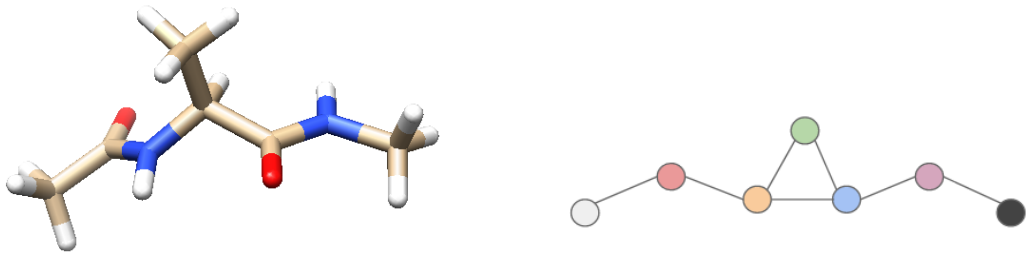
6.1.4 Angular value encoding

Angles (and dihedrals) computed from the MD simulation's data are expressed in the plain angle's domain, from -180 to 180. What is worth noting is that the neural network doesn't know the concept of periodicity and we would like to find a representation where a small shift in the angle value is reflected by a small shift in the represented values. This is offered by the sin and cos angle decomposition. For each angle, we use its sin and cos components to encode it, these two values will change smoothly in case there is a shift.

To understand the difference in a more clear way, an example is provided. Given 2 angles, α and β, measuring, in degrees, -175 and 175 respectively; if we encode their values with the plain degrees number they will be at the opposite ends of their domain (i.e. $[-180; 180)$). If we encode them using sin/cos decomposition, $\alpha = (-1^-, 0^-)$ and $\beta = (-1^-, 0^+)$ which are very close in their domain (i.e. the circonference of radius 1 with the center in $(0, 0)$). The example is shown also in Figure 6.6.

6.2 Model architectures

When using ML to solve molecular problems, we know in advance most of the several physical laws that govern them, thus we know what constraints the predictions have to respect. In the case of predicting distances between atoms, we know that



(a) Input (b) Output

Figure 6.5: Simplified representation take into account the uniqueness of dihedrals

the prediction should be a positive number; if we shuffle the two atoms, the prediction should be the same since the distance is a symmetric function. When dealing with these kind of properties, either we build them into the model, or we use data augmentation. The former is the optimal approach and it is what we tried to pursue.

We are looking for a model that is able to ingest data coming from different molecules, which means different number of atoms, angles, dihedrals, etc. Abstracting the high-level features of the conformations in order to predict the FES is the key to build a performant model. That's why our focus moved to GNN, which offers the powerful convolutional filters while dealing with non-Euclidean data.

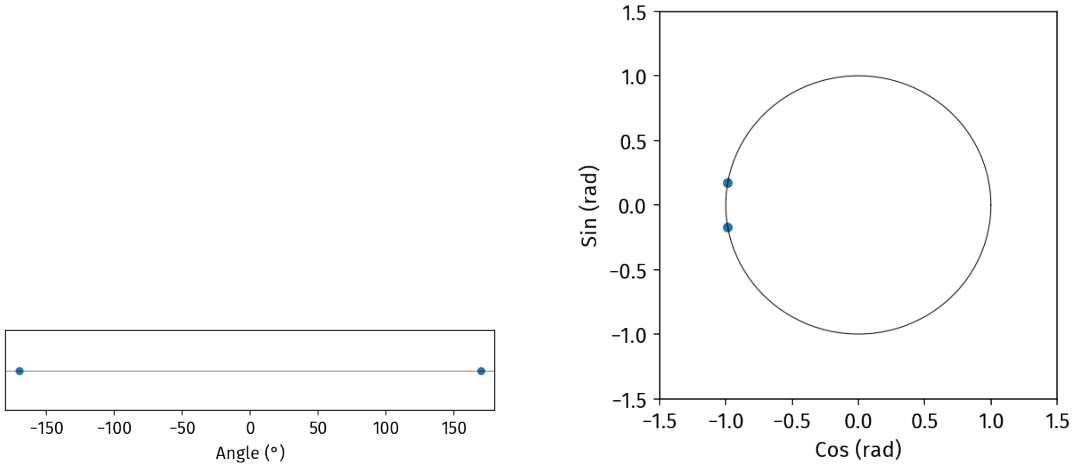
Designing a GNN has to consider the properties that should hold for graphs:

- They are unordered structures, thus the network should not learn the order of nodes given as input but should be permutation invariant.
- They may vary in sizes: the network should be general enough to accomplish the task if graphs with different number of nodes are fed.
- The computational complexity should not be very high: our end goal is to use this on proteins with hundreds (or thousands) of values.

The upcoming architectures consider the GNN layers that have been explained in chapter 4.

The general architecture of the network is the following:

- An initial module which stacks spatial convolutions to propagate the information in a learnable way through the structure;
- An intermediate pooling to focus on a relevant portion of the graph;



(a) Plain angle

(b) Sin and cos decomposition

Figure 6.6: Distance between α and β angles, on the 2 different domains.

- A readout function/layer that removes the graph structure;
- A final block that group the abstract features and outputs a single value.

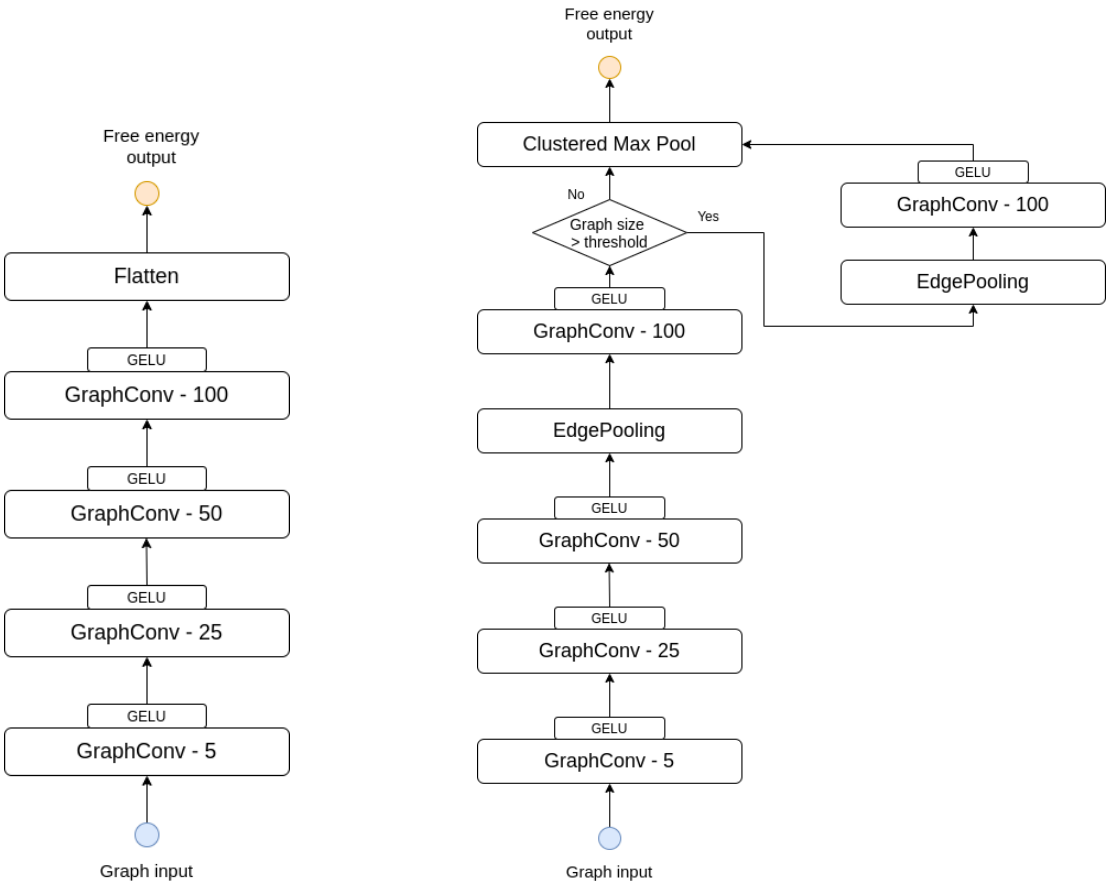
On top of this, several models have been built and analyzed. The general structure has been designed empirically and by having the graph properties in mind, while the hyperparameters such as learning rate, number and channels depth of layers have been selected by using an hyperparameter optimization framework, namely Optuna for PyTorch [94].

The initial module is shared among all the models and it is pretty simple. It is composed by a number of convolutional layers with an incremental number of hidden channels layer after layer. While there are a lot of different convolutional operators that we could have taken into consideration, for a simple and almost linear structure, we chose GraphConv and GATConv for comparison. The only difference between them is that GATConv implements an attention mechanism, but for the problem we are considering it has not been useful, worsening the prediction compared to the GraphConv one. The choice of stacking several convolutional layers comes from the fact that after each convolution, the information is propagated to a distance of one hop only. In conventional feature maps for Euclidean data instead, usually you can select the desired size of the window, making it a bit more flexible.

6.2.1 Flatten model

The most simple model we realized represents the first meaningful milestone: a number of convolutional layers, of different size, followed by a readout layer that **flattens** out the graph structure, concatenating nodes features in the input order.

Optuna found, as the best configuration, the combination of 4 GraphConv layers with the GELU activation function. This means that the attention concept, provided instead by GATConv, is not helpful in this context.



(a) Flatten (b) C&P

Figure 6.7: NN model architectures. After the hyphen, the number of hidden channels for convolutions

Flatten limitations

Given as output the one-dimensional vector (section 4.3.1) of size $n \times c$, with n number of vertices and c number of nodes features after the convolutions, the output of the network is directly dependent on the graph's size. This means it can only process graphs of size n. This model is not permutation-invariant since the readout is just placing, side by side, node after node. The final block will learn the order of flattened vertices (therefore learning the weights based on the node ordering, not only on the features).

It is a very constrained model that works only for the selected molecule and it has been built and tested for comparison with baseline models.

6.2.2 Convolutions and pooling model

In order to generalize the model to ingest several molecules of different size, we needed to abstract more the simple flatten model by introducing some pooling after the convolutions and changing the readout layer to something that is more flexible.

This model has been thought with large graphs in mind: I built a block of convolutions and pooling in order to downsample big graphs to a manageable size (computationally speaking) that can be used, if the graph is still too large, in a iterative way. Then convolute again to distribute knowledge between pooled nodes and finally a global pooling layer to return a fixed-size one-dimensional vector that is able to generalize over different sizes of pooled graphs.

By using the EdgePooling layer (section 4.3) iteratively, after each pooling we'll have $\sim$50% of the nodes pooled while maintaining the ideal graph structure, an efficient way to trim down the graph size. In addition, we can on-the-fly choose if using a combination of EdgePooling and GraphConv when the graph size is above a certain threshold before the last pooling/readout.

The global pooling methods/readouts that have been considered in the design of this are: SortPooling, Max/Average cluster pool and TopKPooling.

6.3 Baseline models

In order to assess the performance of a model and compare it, we need to find a baseline. Since doing the regression task of the FES of molecules is a new topic, we built a baseline model that has in mind the goal of providing the best result possible without taking into consideration the molecular structure. In simple words,

we would like to train a model by passing all the information we have, side by side, without telling the model that this information is organized in a structure.

This is done because, when building a representation that in our opinion is reasonable, we'd have to test it against the "no-structure" model to see if passing the structure to the model is improving our results, so that we can draw some conclusions from it.

While not considering the structure, we identified 2 subsets of "baseline" models composed of fully-connected layers:

- Flat models, involving up to 1 hidden layer, very simple and straightforward. The one used for the evaluation comparison has the hidden layer composed by 512 nodes.

- Deep models, involving a number of hidden layers with increased width and depth. The one used for the evaluation comparison has 4 hidden layers with size 65, 340, 441 and 197 respectively. The size of the hidden layers has been chosen through the usage of Optuna.

Both of these groups of models, useful for comparison with more complex ones, have one critical limitation: the input size is fixed. It is not possible, unless of weird padding, to provide several molecules as training because the number of input parameters is not variable. That means, in very simple words, that if we add/remove an atom from the current alanine dipeptide, these models are not able to provide an answer because of size mismatch (an error will be generated).

Chapter 7

Results and discussions

In this chapter I will present results on various prediction tasks, comparing them among different models, representations and datasets.

All the results are presented as an average of 5 independent and randomized runs, with the respective standard deviation. While 5 is not a number that is statistically relevant, it allows to qualitatively discern if the obtained results are consistent or not. Doing more than 5 runs would have been very computationally expensive, considering all the tasks that have been undertaken. Each of the run has been stopped using the **early stopping** mechanism with patience set as 30 epochs.

If not specified differently, the runs were organized using 5000 random frames from the full dataset as training, 5000 frames as validation and the rest 40000 frames were used as test. The input data, target included, have been normalized with $\mu = 0$ mean and $\sigma = 1$ standard deviation. The limited number of training frames has been chosen to speed up the training while not losing much performance. In a following section, a comparison between prediction error with different number of frames is given.

The metric I used to assess the performance is the Root Mean Squared Error (RMSE), based on its unit measure. Maintaining the unit of the computed property (i.e., kJ/mol since we are dealing with free energy), it is possible to assess the accuracy of the method and compare it with the state-of-the-art techniques employed to estimate the free-energy difference of processes and with the intrinsic precision of current force fields, estimated in the range of around 1 kcal/mol (1cal = 4.184J). Thus, models that predict in the 1kcal error range are considered performant.

Moreover, the outliers predictions weigh more with RMSE: if we have a big prediction error, the square will consider it more than a simple average (like in MAE). The required model should be able to predict all of the states (high or low-energetic)

with an acceptable confidence, thus emphasizing bad predictions will make the overall prediction worse.

7.1 Alanine dipeptide FES prediction

The main goal of the aforementioned methods is the accurate prediction of free-energy values starting from MD simulation poses. As previously mentioned, we selected the alanine dipeptide as a benchmark system and we employed a Graph Convolutional Network. Given that the peptide is expressed in function of 2 dihedrals, it is the perfect starting point for the development of the model (as outlined from the baseline models results). Various tasks have been performed, with different models, in order to compare the properties and the prediction accuracies.

A short list of the employed architectures, with respective references to the sections where they are described, is reported below:

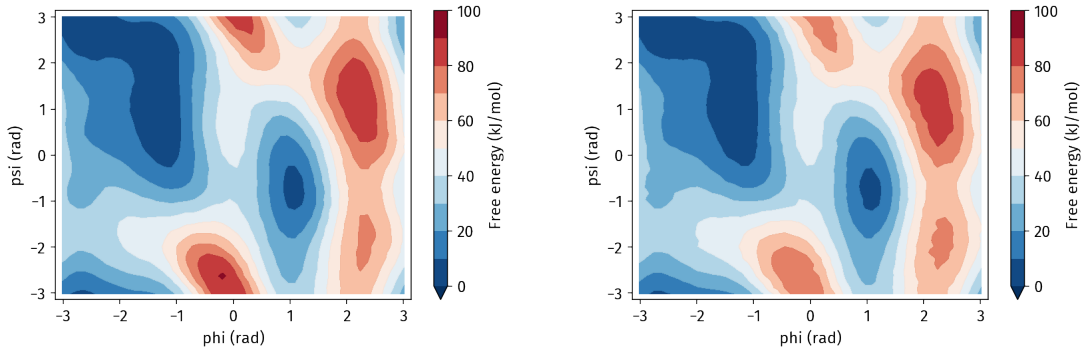
- Flat and deep baseline models (section 6.3)
- Flatten model (subsection 6.2.1)
- Convolutions and Pooling (C&P) (subsection 6.2.2) models with different final pooling layer: SortPooling, TopKPooling, Cluster/Global Max/Average.

Whole FES prediction As the first evaluation task, the models performances have been compared on the prediction of the free-energy value of the conformations belonging to the test set. Only the best configuration for each category of the model have been considered. In the case of C&P model, the one who has clustered max pooling with 3 final clusters has been chosen; instead, for the others the configurations are the ones presented in their respective sections.

	RMSE	**# epochs**
Flat baseline	2.25 pm 0.33	66.6 pm 17.1
Deep baseline	0.94 pm 0.14	76.8 pm 14.5
Flatten	1.04 pm 0.05	183.8 pm 22.0
C&P - Max	1.30 pm 0.11	162.4 pm 32.7

Table 7.1: Comparison among baseline and graph convolutional models predictions

Given the test set predictions, it is possible to interpolate the values to obtain a reconstruction of the FES from the trained model. In Figure 7.1 the predicted FES follows the shape of the original one. The errors have been plotted (Figure 7.2) in function of ϕ and ψ to highlight which conformations have the worst prediction. Apart from the high-energetic conformations where $\phi \sim= 0$ and $\psi \sim= -\pi$ where the error is higher, the other conformations were predicted remarkably well, with an error < 2kJ/mol.



(a) Original FES from the MD simulation (integration of 50k frames)

(b) Reconstructed FES from Flatten model predictions on test set (interpolation of 40k frames)

Figure 7.1: Comparison between the FES computed by MetaD (a) and FES interpolated from the model (b).

As shown in Figure 7.1 and Figure 7.2, most of the errors are located in high-energetic conformations. This finding is confirmed by looking at the correlation between target and prediction, on a free-energy value scale (Figure 7.3). A possible explanation is the different density of input frames along the CVs domain. High energy conformations are not easily populated by the molecule, thus bigger uncertainties are to be expected for states not well-represented in the training set.

7.1.1 C&P final poolings

Since the flatten model has limited applicability (only to the same molecule, same number of dihedrals), as outlined in section 6.2.1, the C&P model have been built and tested with different final poolings in order to understand which is the most suitable for the task at hand. Among TopKPooling, SortPooling and the clustered Max/Average, the most succesful ones are the clustered poolings (Table 7.2).

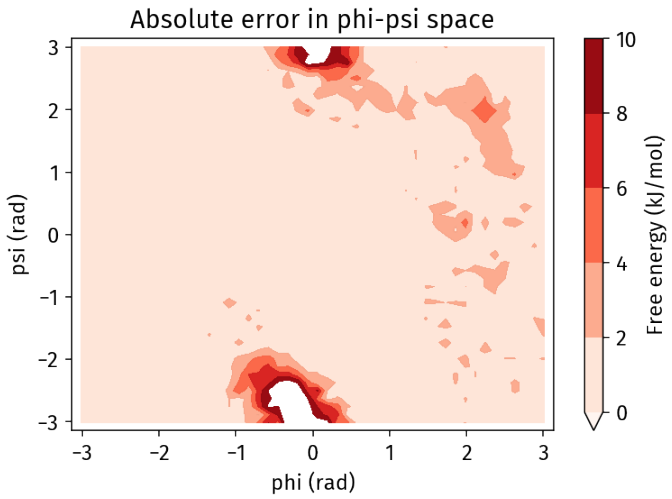


Figure 7.2: Prediction error, of Flatten model, in function of ϕ and ψ

	RMSE	**# epochs**
TopKPooling	10.44 ± 0.09	76.0 ± 15.6
SortPooling	10.54 ± 0.08	105.0 ± 27.0
Cluster Max	**1.30 ± 0.11**	162.4 ± 32.7
Cluster Avg	1.53 ± 0.13	132.4 ± 35.5

Table 7.2: Comparison among final pooling layers in the C&P model

Among all the tried global poolings, no permutational invariant one has been able to give good performance. This probably happens because either they are not suited for the problem or they trim down too much the information needed to give a good estimate (in case of single-cluster pooling).

7.1.2 Different number of training frames

The number of ideal training frames, set to 5000, has been chosen as a compromise between training speed and accuracy. In Figure 7.4 the reader has an overview on prediction RMSE of trained models with several number. Moreover, it highlights how different representations behave based on the number of frames they have been trained with.

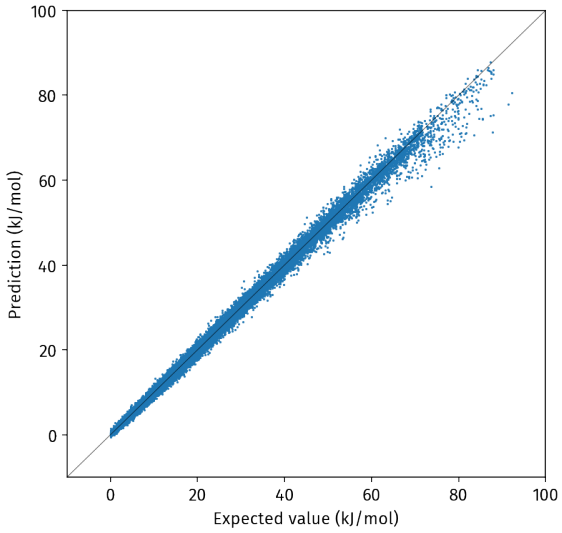


Figure 7.3: Correlation between prediction and target on the free-energy scale

7.2 Comparison among different angular representations

As outlined in subsection 6.1.4, various calculations have been performed in order to understand which dihedral representation, between sin/cos and plain angle value, is better suited for the NN in terms of speed of convergence and prediction error.

The model used as reference of comparison is the Flatten one (described in section 4.3.1) and the results achieved are summarized in the table Table 7.3.

	RMSE	**# epochs**
sin/cos	**1.04 ± 0.05**	**183.8 ± 22.0**
plain	1.26 ± 0.12	288.6 ± 64.6

Table 7.3: Prediction on alanine dipeptide with different angular representation

Results speak for themselves: we have a clear improvement in the convergence speed and a slightly better and more consistent prediction over the 5 runs for the sin/cos representation. This suggests that explicitly decomposing the angle as a pair of sin/cos values improves both the process of training and the overall performance.

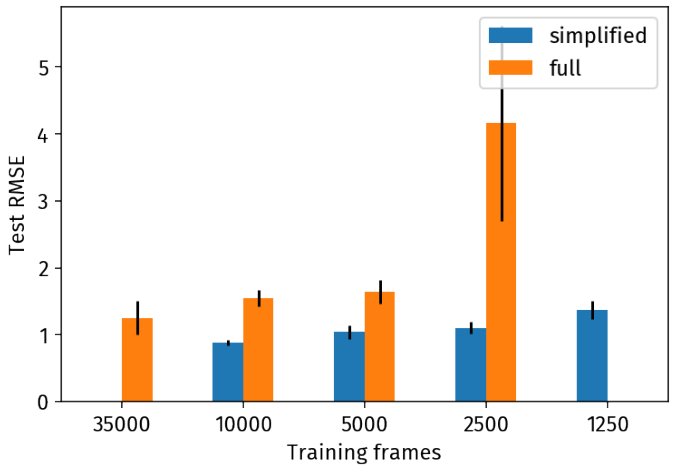


Figure 7.4: Comparison among different representations and different number of frames. Black lines represents the standard deviation interval

7.3 Comparison among different molecule representations

Representing the molecule as a dihedrals overlap graph leads to two different structures: the full one, using all possible 4 atoms combination to define all existent dihedrals in the molecule, and the simplified version, in which only a representative of a group of dihedral existing on the same bond is selected. The runs have been using the sin/cos as angular representation. A comparison has been done on the speed of convergence and on the prediction difference, leading to these results:

	RMSE	**# epochs**
full	1.32 ± 0.17	231.0 ± 31.2
simplified	**1.04 ± 0.05**	**183.8 ± 22.0**

Table 7.4: Prediction on alanine dipeptide with different overlap graphs representations

As it is possible to observe, these results are useful to understand that with the simplified version we earn in terms of convergence speed and slightly even in predictions. Therefore, the combination between sin/cos and simplified dihedrals

representation is working better than the full one.

7.4 Shuffling the nodes

One important property that should distinguish graph neural networks from images ones is the permutational invariance.

The end goal is to build a model that learns only the graph features and topology, not the order of the nodes in the nodes features matrix. To assess if the built models respect the permutation invariance property, it should be checked that every layer is invariant. Moreover we can experimentally check this by shuffling the rows of the nodes features matrix before training (updating the edges accordingly).

The only models that hold the permutation invariance are the ones who have the global max/average pooling (i.e., with only one cluster) and the ones who use TopKPooling or SortPooling as final/readout layer. The predictions of these models unfortunately are not performant (Table 7.2) as we would like to, but this could be due to the specificity of the dataset describing only a single simulation, thus not providing enough data samples to train from.

7.5 Prediction on unseen minima

Given that MD simulations take a very long time to explore the entire combination of states promoted by slow degrees of freedom, it is possible that scientists explore only a selection of states, instead of the entire picture. Hysteresis problem might cause incomplete sampling and thus non-converged results. One interesting direction that could be explored is to train the model in order to predict unseen FES regions Figure 7.5.

In order to tackle this different problem, the dataset have been split (as in Figure 7.6) by removing from the training set all poses belonging to the minimum at $\phi \sim 1$ and $\psi \sim -1$. The removed frames were successively used as test set Figure 7.7.

Interestingly, the model predicted the presence of a minimum in 3 cases out 5 replicas (with $\sim$ 4 kJ/mol RMSE). Having only one simulation in the dataset is surely not enough for the model to generalize and predict unseen regions, thus the achieved results might be the product of a good interpolation.

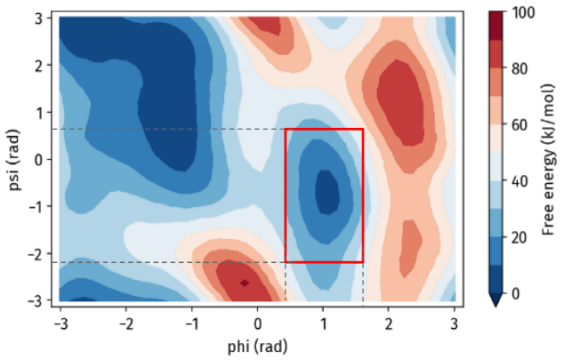
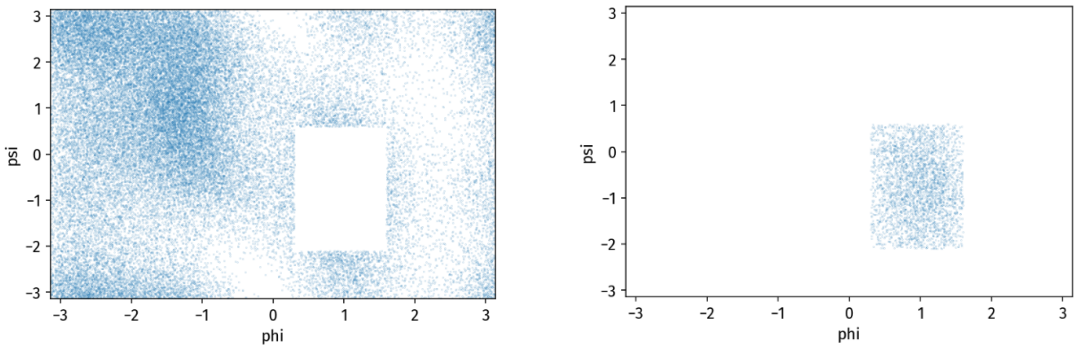


Figure 7.5: In red a portion of FES representing a local minimum that we would like to predict without training



(a) Training set

(b) Test set

Figure 7.6: Unseen region dataset

7.6 Prediction on both the representations

To assess the model's ability to train on different graph sizes, it was tested on both the overlap graph representations described in subsection 6.1.2 and subsection 6.1.3, respectively.

	RMSE	# epochs
C&P - Max	**1.44 ± 0.01**	195.0 ± 10.4
C&P - Avg	1.62 ± 0.03	150.3 ± 16.4

Table 7.5: Both representation training

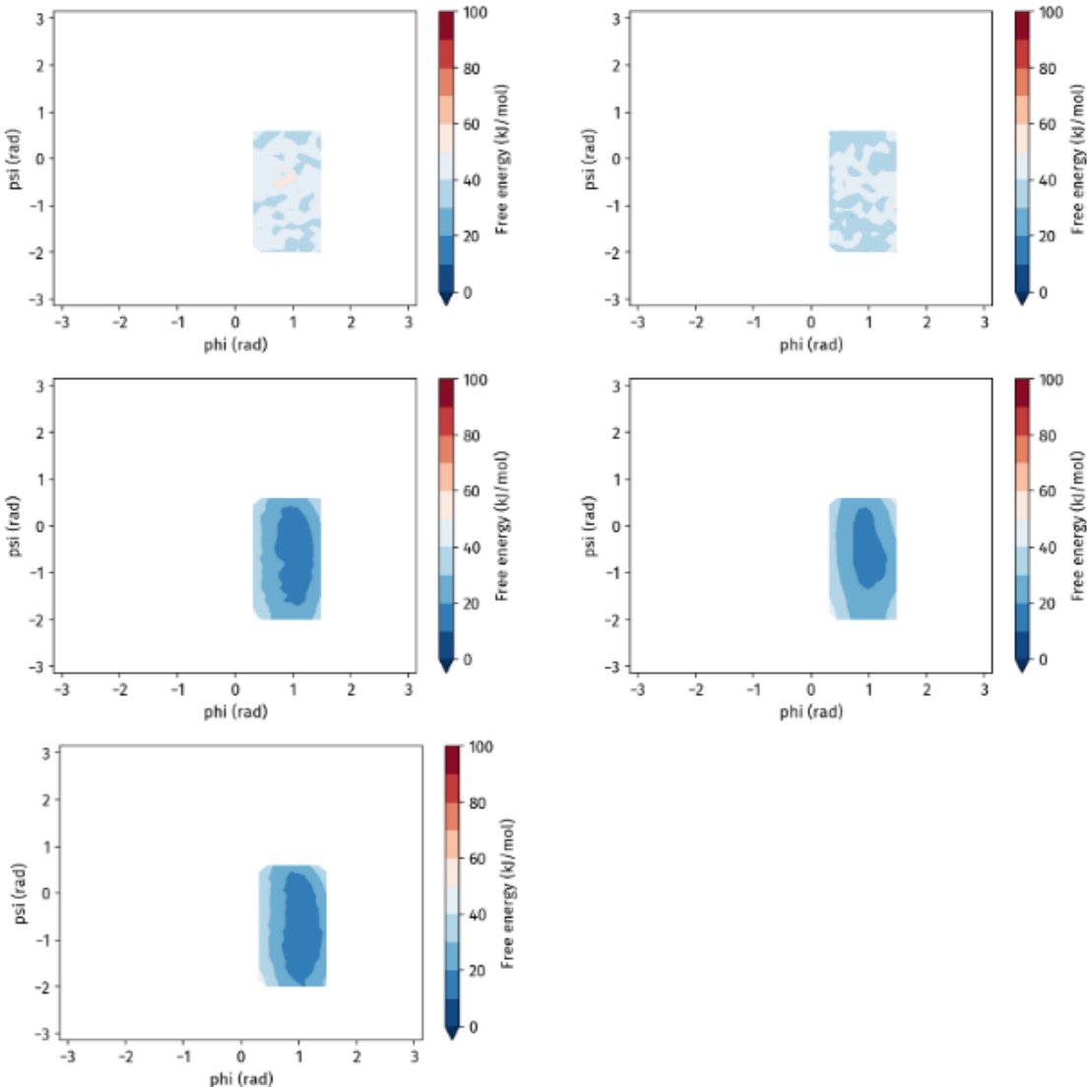


Figure 7.7: 5-run results of predictions on unseen minima

Chapter 8

Conclusions and future work

In this chapter, I will first discuss about the main outcomes of the my thesis work with the methods used and analyzed in the previous chapters. Then, an highlight about the directions that the project should follow in future is given.

As outlined in the "Results" chapter, we have been able to build a model which:

- Predicts with a very good grade of accuracy (< 0.5 kcal/mol) the alanine dipeptide FES starting from a limited number of trajectory's frames
- Provides a successful graph molecular representation to produce predictions, focusing only on some specific, relevant information;
- Is able to train on graph of different sizes (as shown in section 7.6), thus ready to be expanded towards more complex systems;
- Keeps complexity as low as possible. EdgePooling (section 4.3) is linear in the number of edges, GraphConv too;
- Is able to predict - albeit in all the cases - the presence of a minima without being trained on conformations belonging to that minima.

We proved also that choosing the right representation, both for the molecule and the angles value, gives relevant benefits:

- With the simplified dihedrals overlap graph. The needed number of training conformations in order to have a good prediction is reduced significantly (Figure 7.4);
- By using the sine and cosine decomposition. There is a consistent improvement in the training speed (Table 7.3).

The results obtained with the baseline models outline that the prediction of the alanine dipeptide's FES is a simple task once you have clear what is the important information to model. However, it is worth noting that the "deep baseline model", the best performing one, is more prone to overfitting due to the very large number of parameters ($> 200'000$).

Despite the promising results, the C&P model has some limitations that open the door to further investigations to improve the model. Among these:

- The final readout layer is not permutation-invariant, thus if we shuffle the order of the nodes, the NN is not able to predict with a good accuracy. All the trials with permutational invariant readout layers haven't reported good performance;
- It models only molecule's dihedrals. Other information (i.e. interatomic interactions) is needed when studying more complex systems. With the current dihedrals overlap graph, this cannot be added in a straightforward way.

We point out that the importance of this work doesn't come solely in the prediction accuracy, but also in the approach towards the generalization of the free-energy calculations which represents a stepping stone towards more complex and different problems. The model could be reused for other task, such as transfer learning [95], where the knowledge gained in the free-energy prediction of a molecule could be stored and used for other molecules. Or the prediction of unseen regions of the FES could be studied close up and could be improved with additional data. In the next section, more detailed extensions that should be addressed to improve the model and to explore more the free-energy calculations field are reported.

8.1 Future work

Permutation invariant readout The missing block of the current best model is that the final layer learns the order of the abstract high-level representations. While this does not bias the calculation (the model is still able to train and learn), it represents a limitation since graphs should not have an intrinsic order and therefore the NN should not predict based on it. There have been many trials towards the building and the selection of a readout layer of this kind, with no good performance. There is room for improvement, also considering the attention dedicated to this topic in literature [96]. A review on deep learning for graphs [97] suggests also the possibility to use stacked fully-connected layers as final output module. While this could help since various weights can be learned for a single node, it doesn't guarantee

the order invariance. The same has been proposed as a universal readout function for graphs [96].

Hypergraph neural network Modeling the molecule through the hypergraph solution is needed in order to consider different kind of molecular information in the same, single structure. I personally think that this could be the most useful approach, but also the hardest one. While building a properly functioning hypergraph for biomolecules could be very challenging - especially considering the novelty of this topic and the scarse literature [90] - going beyond the dihedrals-only representation is **mandatory** when dealing with larger molecules like proteins. In hypergraphs we have the important information in the hyperedges (interactions and dihedrals) so it is needed to find a convolutional layer that uses the hyperedges features as a discriminant to predict the output, not only as a weight to propagate the information between nodes.

ΔG prediction If we would like to consider a further extension of the applicability of the model we might rethink the problem statement. The current NN goal is finding the relative amount of free energy of a molecule's conformation. Expanding and generalizing the prediction on several molecules, by adding new simulations, will not probably help in achieving meaningful results. This because each free energy value is computed as a difference with the respective molecule's global minimum. And even though two global minimum of two different systems have the same amount of free energy (i.e. 0), it doesn't mean that their value comes from similar structural properties. Because the single free energy value is coupled to the system it is computed based on its own structural features.

Moving the problem to a more general statement instead, could bring a different perspective with additional outcomes. What if we try to predict the ΔG between two same system's conformations? The idea is to learn which are the conformational changes that drives up or down the free energy value. Therefore what the network may learn from multiple simulations is that similar changes in molecules bring similar changes in the free energy. In this paradigm, the problem statement would become computing the ΔG between 2 conformations c_1 and c_2 of the same system:

$$\Delta G = G(c_1) - G(c_2) \tag{8.1}$$

Focusing on the ΔG, the NN will be able to predict the lower and higher states of the system and reveal if passing from one conformation (c_1) to another (c_2) is energetically favoured. In addition, its general applicability and ability to model

different systems (transfer learning) - molecules composed of a different number of atoms - might be possible.

The implementation of the methods is open-source and available on GitHub at `https://github.com/limresgrp/free-energy-gnn`

Bibliography

[1] N.K. Broomhead and Soliman M.E. Can we rely on computational predictions to correctly identify ligand binding sites on novel protein drug targets? assessment of binding site prediction methods and a protocol for validation of predicted binding sites. *Cell. Biochem. Biophys.*, (75):15–23, 2017.

[2] Seonwoo Min, Byunghan Lee, and Sungroh Yoon. Deep learning in bioinformatics. *Briefings in Bioinformatics*, (5), 2017.

[3] T.D. Pollard. A guide to simple and informative binding assays. *Mol. Biol. Cell.*, (21):4061–4067, 2010.

[4] Vittorio Limongelli. Ligand binding free energy and kinetics calculations in 2020. *WIREs Comp. Mol. Science*, (10), 2020.

[5] Frank Noe, Gianni De Fabritiis, and Cecilia Clementi. Machine learning for protein folding and dynamics. *Current Opinion in Structural Biology*, 60, 2020.

[6] Hao Ding, Ichigaku Takigawa, Hiroshi Mamitsuka, and Shanfeng Zhu. Similarity-based machine learning methods for predicting drug–target interactions: a brief review. *Briefings in bioinformatics*, 15(5):734–747, 2014.

[7] Waseem Rawat and Zenghui Wang. Deep convolutional neural networks for image classification: A comprehensive review. *Neural computation*, 29(9):2352–2449, 2017.

[8] George E Dahl, Dong Yu, Li Deng, and Alex Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on audio, speech, and language processing*, 20(1):30–42, 2011.

[9] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521 (7553):436–444, 2015.

[10] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[11] Krzysztof J Geras, Stacey Wolfson, Yiqiu Shen, Nan Wu, S Kim, Eric Kim, Laura Heacock, Ujas Parikh, Linda Moy, and Kyunghyun Cho. High-resolution breast cancer screening with multi-view deep convolutional neural networks. *arXiv preprint arXiv:1703.07047*, 2017.

[12] T Ciodaro, D Deva, JM De Seixas, and D Damazio. Online particle detection with neural networks based on topological calorimetry information. In *Journal of physics: conference series*, volume 368, page 012030. IOP Publishing, 2012.

[13] Vedran Dunjko and Hans J. Briegel. Machine learning & artificial intelligence in the quantum domain, 2017.

[14] Andrea Grisafi, David M. Wilkins, Gábor Csányi, and Michele Ceriotti. Symmetry-adapted machine learning for tensorial properties of atomistic systems. *Physical Review Letters*, 120(3), Jan 2018. ISSN 1079-7114. doi: 10.1103/physrevlett.120.036002. URL `http://dx.doi.org/10.1103/PhysRevLett.120.036002`.

[15] Louis-Fran çois Arsenault, Alejandro Lopez-Bezanilla, O. Anatole von Lilienfeld, and Andrew J. Millis. Machine learning for many-body physics: The case of the anderson impurity model. *Phys. Rev. B*, 90:155136, Oct 2014. doi: 10.1103/PhysRevB.90.155136. URL `https://link.aps.org/doi/10.1103/PhysRevB.90.155136`.

[16] Jörg Behler and Michele Parrinello. Generalized neural-network representation of high-dimensional potential-energy surfaces. *Phys. Rev. Lett.*, 98:146401, Apr 2007. doi: 10.1103/PhysRevLett.98.146401. URL `https://link.aps.org/doi/10.1103/PhysRevLett.98.146401`.

[17] Stefan Chmiela, Huziel E. Sauceda, Klaus-Robert Müller, and Alexandre Tkatchenko. Towards exact molecular dynamics simulations with machine-learned force fields. *Nature Communications*, 9(1), Sep 2018. ISSN 2041-1723. doi: 10.1038/s41467-018-06169-2. URL `http://dx.doi.org/10.1038/s41467-018-06169-2`.

[18] James McCarty and Michele Parrinello. A variational conformational dynamics approach to the selection of collective variables in metadynamics. *The Journal*

of Chemical Physics, 147(20):204109, Nov 2017. ISSN 1089-7690. doi: 10.1063/1.4998598. URL http://dx.doi.org/10.1063/1.4998598.

[19] Andreas Mardt, Luca Pasquali, Hao Wu, and Frank Noé. Vampnets: Deep learning of molecular kinetics. *Nature Communications*, 9, 10 2017. doi: 10.1038/s41467-017-02388-1.

[20] Luigi Bonati, Yue-Yu Zhang, and Michele Parrinello. Neural networks-based variationally enhanced sampling. *Proceedings of the National Academy of Sciences*, 116(36):17641–17647, 2019.

[21] Raimondas Galvelis and Yuji Sugita. Neural network and nearest neighbor algorithms for enhancing sampling of molecular dynamics. *Journal of Chemical Theory and Computation*, 13(6):2489–2500, 2017. doi: 10.1021/acs.jctc.7b00188. URL https://doi.org/10.1021/acs.jctc.7b00188. PMID: 28437616.

[22] Protein structure. https://www.khanacademy.org/science/biology/macromolecules/proteins-and-amino-acids/a/orders-of-protein-structure. Accessed: 2020-06-14.

[23] Ron O. Dror Scott A. Hollingsworth. Molecular dynamics simulation for all. *Neuron*, 2018. URL https://doi.org/10.1016/j.neuron.2018.08.011.

[24] Hiroshi Nakatsuji, Hiroyuki Nakashima, and Yusaku I. Kurokawa. Solving the schrödinger equation of atoms and molecules: Chemical-formula theory, free-complement chemical-formula theory, and intermediate variational theory. *The Journal of Chemical Physics*, 149(11):114105, 2018. doi: 10.1063/1.5040376. URL https://doi.org/10.1063/1.5040376.

[25] Lindahl, Abraham, Hess, and van der Spoel. Gromacs 2020.2 manual, April 2020. URL https://doi.org/10.5281/zenodo.3773799.

[26] Stefano Forli and Arthur J Olson. A force field with discrete displaceable waters and desolvation entropy for hydrated ligand docking. *Journal of medicinal chemistry*, 55(2):623–638, 2012.

[27] Wolfgang Damm, Antonio Frontera, Julian Tirado-Rives, and William L Jorgensen. Opls all-atom force field for carbohydrates. *Journal of Computational Chemistry*, 18(16):1955–1970, 1997.

[28] Jing Huang and Alexander D MacKerell Jr. Charmm36 all-atom additive protein force field: Validation based on comparison to nmr data. *Journal of computational chemistry*, 34(25):2135–2145, 2013.

[29] W.D. Cornell, P. Cieplak, C.I. Bayly, I.R. Gould, K.M.Jr Merz, D.M. Ferguson, D.C. Spellmeyer, T. Fox, J.W. Caldwell, and P.A. Kollman. A second generation force field for the simulation of proteins, nucleic acids, and organic molecules. *J. Am. Chem. Soc.*, 117(19):5179–5197, 1995.

[30] Jacob N. Israelachvili. Van der waals forces in biological systems. *Quarterly Reviews of Biophysics*, 6(4):341–387, 1973. doi: 10.1017/S0033583500001566.

[31] Van der waals interaction energy-distance. `https://en.wikibooks.org/wiki/Structural_Biochemistry/Chemical_Bonding/Van_der_Waals_interaction`. Accessed: 2020-06-11.

[32] Helmut Grubmüller, Helmut Heller, Andreas Windemuth, and Klaus Schulten. Generalized verlet algorithm for efficient molecular dynamics simulations with long-range interactions. *Molecular Simulation*, 6(1-3):121–142, 1991.

[33] Marco De Vivo, Matteo Masetti, Giovanni Bottegoni, and Andrea Cavalli. Role of molecular dynamics and related methods in drug discovery. *Journal of Medicinal Chemistry*, 59(9):4035–4061, 2016. doi: 10.1021/acs.jmedchem.5b01684. URL `https://doi.org/10.1021/acs.jmedchem.5b01684`. PMID: 26807648.

[34] Mikhail V Vener, AN Egorova, AV Churakov, and VG Tsirelson. Intermolecular hydrogen bond energies in crystals evaluated using electron density properties: Dft computations with periodic boundary conditions. *Journal of Computational Chemistry*, 33(29):2303–2309, 2012.

[35] Jay W Ponder and Frederic M Richards. An efficient newton-like method for molecular mechanics energy minimization of large molecules. *Journal of Computational Chemistry*, 8(7):1016–1024, 1987.

[36] Basic principles of molecular dynamics (md) theory. `http://web.archive.org/web/20190712155444/http://www.archie-west.ac.uk/wp-content/uploads/2012/06/MD_theory.pdf`. Accessed: 2020-06-04.

[37] Josiah Willard Gibbs. *Elementary principles in statistical mechanics: developed with especial reference to the rational foundations of thermodynamics*. C. Scribner's sons, 1902.

[38] Molecular dynamics theory. https://embnet.vital-it.ch/MD_tutorial/pages/MD.Part3.html. Accessed: 2020-07-05.

[39] David E Shaw, Martin M Deneroff, Ron O Dror, Jeffrey S Kuskin, Richard H Larson, John K Salmon, Cliff Young, Brannon Batson, Kevin J Bowers, Jack C Chao, et al. Anton, a special-purpose machine for molecular dynamics simulation. *ACM SIGARCH Computer Architecture News*, 35(2):1–12, 2007.

[40] Chapter 6 - the second law of thermodynamics and entropy. In Bruce Fegley, editor, *Practical Chemical Thermodynamics for Geoscientists*, pages 173 – 224. Academic Press, Boston, 2013. ISBN 978-0-12-251100-4. doi: https://doi.org/10.1016/B978-0-12-251100-4.00006-7. URL http://www.sciencedirect.com/science/article/pii/B9780122511004000067.

[41] Daniel Vallero. Chapter 17 - air pollutant kinetics and transformation. In Daniel Vallero, editor, *Fundamentals of Air Pollution (Fifth Edition)*, pages 413 – 435. Academic Press, Boston, fifth edition edition, 2014. ISBN 978-0-12-401733-7. doi: https://doi.org/10.1016/B978-0-12-401733-7.00017-7. URL http://www.sciencedirect.com/science/article/pii/B9780124017337000177.

[42] Kevin Leung, Susan B Rempe, and O Anatole von Lilienfeld. Ab initio molecular dynamics calculations of ion hydration free energies. *The Journal of chemical physics*, 130(20):204507, 2009.

[43] Vittorio Limongelli. Ligand binding free energy and kinetics calculation in 2020. *WIREs Computational Molecular Science*, 10(4):e1455, 2020. doi: 10.1002/wcms.1455. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/wcms.1455.

[44] Jorge Kurchan, Giorgio Parisi, and Miguel Angel Virasoro. Barriers and metastable states as saddle points in the replica approach. *Journal de Physique I*, 3(8):1819–1838, 1993.

[45] Alessandro Laio and Michele Parrinello. Escaping free-energy minima. *Proceedings of the National Academy of Sciences*, 99(20):12562–12566, 2002.

[46] Mark Ebden. Gaussian processes: A quick introduction, 2015.

[47] DJ Tildesley and MP Allen. Computer simulation of liquids. clarendon, 1987.

[48] Alessandro Barducci, Giovanni Bussi, and Michele Parrinello. Well-tempered metadynamics: a smoothly converging and tunable free-energy method. *Physical review letters*, 100(2):020603, 2008.

[49] James F Dama, Michele Parrinello, and Gregory A Voth. Well-tempered metadynamics converges asymptotically. *Physical review letters*, 112(24):240602, 2014.

[50] Joannis Apostolakis, Philippe Ferrara, and Amedeo Caflisch. Calculation of conformational transitions and barriers in solvated systems: Application to the alanine dipeptide in water. *The Journal of chemical physics*, 110(4):2099–2108, 1999.

[51] Pratyush Tiwary and Michele Parrinello. A time-independent free energy estimator for metadynamics. *The Journal of Physical Chemistry B*, 119(3):736–742, 2015. doi: 10.1021/jp504920s. URL `https://doi.org/10.1021/jp504920s`. PMID: 25046020.

[52] Pratyush Tiwary and Michele Parrinello. From metadynamics to dynamics. *Physical review letters*, 111(23):230602, 2013.

[53] Sho Sonoda and Noboru Murata. Neural network with unbounded activation functions is universal approximator. *Applied and Computational Harmonic Analysis*, 43(2):233 – 268, 2017. ISSN 1063-5203. doi: https://doi.org/10.1016/j.acha.2015.12.005. URL `http://www.sciencedirect.com/science/article/pii/S1063520315001748`.

[54] Xin Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9): 1423–1447, 1999.

[55] A. Hoffmann. Artificial and natural computation. In Neil J. Smelser and Paul B. Baltes, editors, *International Encyclopedia of the Social Behavioral Sciences*, pages 777 – 783. Pergamon, Oxford, 2001. ISBN 978-0-08-043076-8. doi: https://doi.org/10.1016/B0-08-043076-7/00551-9. URL `http://www.sciencedirect.com/science/article/pii/B0080430767005519`.

[56] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[57] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[58] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[59] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[60] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL `http://jmlr.org/papers/v15/srivastava14a.html`.

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

[62] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pages 3844–3852, 2016.

[63] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[64] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[65] Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks, 2018.

[66] William L. Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications, 2017.

[67] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pages 3844–3852, 2016.

[68] James Atwood and Don Towsley. Diffusion-convolutional neural networks. In *Advances in neural information processing systems*, pages 1993–2001, 2016.

[69] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry, 2017.

[70] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks, 2018.

[71] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[72] Frederik Diehl. Edge contraction pooling for graph neural networks. *arXiv preprint arXiv:1905.10990*, 2019.

[73] Ekagra Ranjan, Soumya Sanyal, and Partha Pratim Talukdar. Asap: Adaptive structure aware pooling for learning hierarchical graph representations, 2019.

[74] Satu Elisa Schaeffer. Graph clustering. *Computer science review*, 1(1):27–64, 2007.

[75] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *AAAI*, 2018.

[76] Hongyang Gao and Shuiwang Ji. Graph u-nets, 2019.

[77] Huang CC Couch GS Greenblatt DM Meng EC Ferrin TE. Pettersen EF, Goddard TD. Ucsf chimera–a visualization system for exploratory research and analysis. *J Comput Chem. 2004 Oct;25(13):1605-12.* doi: 10.1002/jcc.20084.

[78] Pdb file format full documentation. `http://www.wwpdb.org/documentation/file-format-content/format33/v3.3.html`. Accessed: 2020-06-13.

[79] Mark James Abraham, Teemu Murtola, Roland Schulz, Szilárd Páll, Jeremy C. Smith, Berk Hess, and Erik Lindahl. Gromacs: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. *SoftwareX*, 1-2:19 – 25, 2015. ISSN 2352-7110. doi: https://doi.org/10.1016/j.softx.2015.06.001. URL `http://www.sciencedirect.com/science/article/pii/S2352711015000059`.

[80] Lindahl, , Abraham, Hess, and Van Der Spoel. Gromacs 2019.4 source code, 2019. URL `https://zenodo.org/record/3460414`.

[81] Romelia Salomon-Ferrer, David A. Case, and Ross C. Walker. An overview of the amber biomolecular simulation package. *WIREs Computational Molecular Science*, 3(2):198–210, 2013. doi: 10.1002/wcms.1121. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/wcms.1121`.

[82] Gareth A. Tribello, Massimiliano Bonomi, Davide Branduardi, Carlo Camilloni, and Giovanni Bussi. Plumed 2: New feathers for an old bird. *Computer Physics Communications*, 185(2):604 – 613, 2014. ISSN 0010-4655. doi: https://doi.org/10.1016/j.cpc.2013.09.018. URL http://www.sciencedirect.com/science/article/pii/S0010465513003196.

[83] Kimberly C. B. New, Keith Watt, Charles W. Misner, and Joan M. Centrella. Stable 3-level leapfrog integration in numerical relativity. *Physical Review D*, 58(6), Aug 1998. ISSN 1089-4918. doi: 10.1103/physrevd.58.064022. URL http://dx.doi.org/10.1103/PhysRevD.58.064022.

[84] H. J. C. Berendsen, J. P. M. Postma, W. F. van Gunsteren, A. DiNola, and J. R. Haak. Molecular dynamics with coupling to an external bath. *The Journal of Chemical Physics*, 81(8):3684–3690, 1984. doi: 10.1063/1.448118. URL https://doi.org/10.1063/1.448118.

[85] Ulrich Essmann, Lalith Perera, Max L Berkowitz, Tom Darden, Hsing Lee, and Lee G Pedersen. A smooth particle mesh ewald method. *The Journal of chemical physics*, 103(19):8577–8593, 1995.

[86] Lucas Sawle and Kingshuk Ghosh. Convergence of molecular dynamics simulation of protein native states: Feasibility vs self-consistency dilemma. *Journal of Chemical Theory and Computation*, 12(2):861–869, 2016. doi: 10.1021/acs.jctc.5b00999. URL https://doi.org/10.1021/acs.jctc.5b00999. PMID: 26765584.

[87] Omar Valsson, Pratyush Tiwary, and Michele Parrinello. Enhancing important fluctuations: Rare events and metadynamics from a conceptual viewpoint. *Annual review of physical chemistry*, 67:159–84, 2016.

[88] Sebastian Raschka. Biopandas: Working with molecular structures in pandas dataframes. *The Journal of Open Source Software*, 2(14), jun 2017. doi: 10.21105/joss.00279. URL http://dx.doi.org/10.21105/joss.00279.

[89] Wes McKinney. pandas: a foundational python library for data analysis and statistics. *Python for High Performance and Scientific Computing*, 14, 2011.

[90] Song Bai, Feihu Zhang, and Philip H. S. Torr. Hypergraph convolution and hypergraph attention. *CoRR*, abs/1901.08150, 2019. URL http://arxiv.org/abs/1901.08150.

[91] Phillip EC Compeau, Pavel A Pevzner, and Glenn Tesler. How to apply de bruijn graphs to genome assembly. *Nature biotechnology*, 29(11):987–991, 2011.

[92] Daniel R Zerbino and Ewan Birney. Velvet: algorithms for de novo short read assembly using de bruijn graphs. *Genome research*, 18(5):821–829, 2008.

[93] N.G. Bruijn, de. A combinatorial problem. *Proceedings of the Section of Sciences of the Koninklijke Nederlandse Akademie van Wetenschappen te Amsterdam*, 49 (7):758–764, 1946. ISSN 0370-0348.

[94] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, KDD '19, page 2623–2631, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362016. doi: 10.1145/3292500.3330701. URL `https://doi.org/10.1145/3292500.3330701`.

[95] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A survey on deep transfer learning. In *International conference on artificial neural networks*, pages 270–279. Springer, 2018.

[96] Nicolò Navarin, Dinh Van Tran, and Alessandro Sperduti. Universal readout for graph convolutional neural networks. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2019.

[97] Ziwei Zhang, Peng Cui, and Wenwu Zhu. Deep learning on graphs: A survey, 2018.