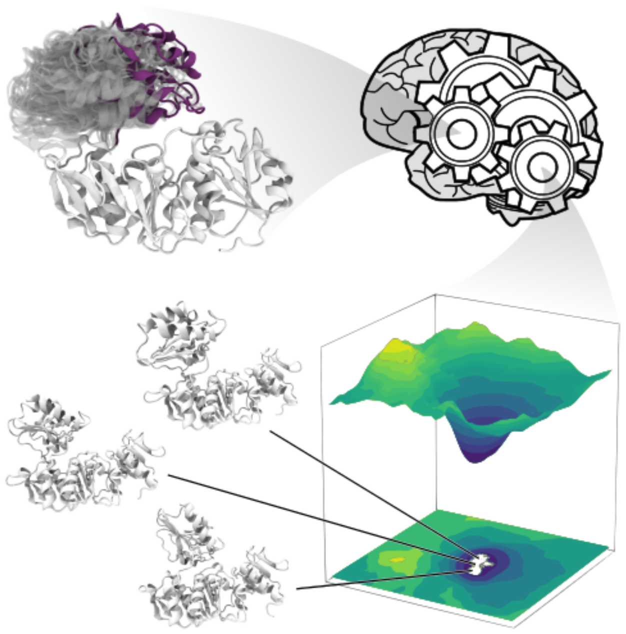


Molecular Dynamics and Generative Neural Networks to Explore Protein Conformational Space

PhD Thesis

Author:
Samuel Musson

Supervisors:
Matteo Degiacomi
Mark Miller

Saturday 31st August, 2024

Contents

Abstract

Determining the different conformational states of a protein and the transition paths between them is key to fully understanding the relationship between biomolecular structure and function. This can be accomplished by sampling protein conformational space with molecular simulation methodologies. Despite advances in computing hardware and sampling techniques, simulations always yield a discretized representation of this space, with transition states undersampled proportionally to their associated energy barrier.

We present neural networks that learn a continuous conformational space representation from example structures. We define a physics-based loss function that, applied to network generated structures, discourages physically implausible conformations. We show that this network, trained with simulations of distinct protein states, can correctly predict a biologically relevant transition path, without any example on the path provided. We also show we can transfer features learnt from one protein to others, which results in superior performances, and requires a surprisingly small number of training examples. Our methods are compiled into the software package *molearn* that aims to lower the barrier to entry for training and analysing these networks while simultaneously providing a platform for rapid development within our framework. In particular we present a version of our loss function using OpenMM to efficiently calculate energies and forces. Using *molearn* we profile our networks showing that they are able handle different atom selections and multimers. Additionally, we demonstrate the ability of this loss function to effectively 'deblur' Sinkhorn trained networks.

Overall, this work demonstrates how generative models can be effectively exploited to characterise proteins conformational spaces, thereby providing precious insight into their biological function.

Acknowledgements

By bowing to the tyranny of tradition in writing an acknowledgment section, I must finally admit that, despite years of effort to the contrary, others have indeed contributed to my successes. I must thank my parents, who, despite my numerous and prominent flaws, have always supported and ultimately enabled me. I would like to thank most especially my mother, who sacrificed more than people will ever know. She has raised five successful children on a budget not out of place in a third world country, through dedication, hard work, and sheer devotion. I want to thank my four older siblings for setting high benchmarks for what constitutes success in our family. When I eventually achieve my goal of being the most successful, powerful, and all round best looking sibling I will know I have made it in life. Over the years playing music has provided all the escape and social interaction I could need. I will take this moment to thank all the hundreds of members of Stamford Brass, LYWO, NYWE, GMCWB, DUBB, DUCB, Bearpark & Esh colliery band, $\mathbb{N}-$ piece, and finally but not least Durham Miners Association Brass Band. From victories to last place finishes, your tolerance of my presence has made my life entirely more bearable. My friends past and present in the Physics department, Lucy, Ruth, Abhinav, Michael, Emma, Cameron, and Asal I would like to thank for making me feel welcome.

There are many collaborators who have made this work possible. I would like to thank Venkata, whose work I continued; Chris, who has repeatedly provided his expertise and guidance; the Master's students Lucy, Louis, Cameron, and Mateusz; and collaborators Ryan, Asma, and Marco, who have all endured the early stages of *molearn* and provided valuable feedback. Mateusz in particular has provided results that have been used in Section 5.2.3 Finally, I would like to thank Matteo who took me on as a Masters student and made the mistake of encouraging me to apply for SOFI2 and eventually become his PhD student. You have had to put up with so much from me, yet you have still provided endless guidance and support. I have had multiple other students confess their envy of me for having you as a supervisor. I hope you find greater success in the future in getting your students to follow your advice, rather than indulging in the excessive 'productive procrastination' that I did.

Publications

Publications presented in this work:

- V. K. Ramaswamy*, **S. C. Musson***, C. G. Willcocks, and M. T. Degiacomi. ”Deep Learning Protein Conformational Space with Convolutions and Latent Interpolations”. *Phys. Rev. X*, 11, 011052 (2021). DOI: 10.1103/PhysRevX.11.011052
  This work is presented in Chapter 3. Co-first authors are annotated with ”*”.
- **S. C. Musson** and M. T. Degiacomi. ”Molearn: a Python package streamlining the design of generative models of biomolecular dynamics” *Journal of Open Source Software*, 8(89), 5523, (2023). DOI: 10.21105/joss.05523
  This work is presented in Chapter 4.

Publications not presented in this work:

- L. S. P. Rudden, **S. C. Musson**, J. L. P. Benesch, M. T. Degiacomi. ”Biobox: a toolbox for biomolecular modelling”. *Bioinformatics*, Volume 38, Issue 4, 1149–1151 (2022). DOI: 10.1093/bioinformatics/btab785
- J. Bruggisser, I. Iacovache, **S. C. Musson**, M. T. Degiacomi, H. Posthaus, B. Zuber. ”Cryo-EM structure of the octameric pore of Clostridium perfringens β-toxin” *EMBO Reports*, 23:e54856 Volume 23, Issue 12, (2022). DOI: 10.15252/embr.202254856

Co-authored open-source software associated with this work:

- molearn: github.com/Degiacomi-Lab/molearn
- biobox: github.com/Degiacomi-Lab/biobox

Chapter 1

Introduction

1.1 Proteins

Proteins are a class of large biomolecules that play an essential role in the biochemistry of all life. This diverse group of macromolecules exhibits an array of structural, chemical, and dynamic properties. These properties equip proteins to participate in a wide variety of biological processes. Primarily due to this versatility, proteins are the most abundant intracellular organic biomacromolecules, and their synthesis consumes a large portion of the resources available to an organism. Using proteins provides natural selection with a powerful functional search space, which has enabled life as we know it to be possible.

A protein is ultimately made up of chains of amino acids, the identity and sequence of which promote the adoption of a specific 3-dimensional structure. The functional diversity of proteins arises from vast number of possible proteins given all possible permutations of the 20 common amino acids and protein sizes of up to 34,000 amino acids. In the traditional view, this 3D structure, referred to as the native state, then determines the function of the protein. Unfortunately, while this view satisfactorily explains the function of some proteins, it is challenged by the existence of intrinsically disordered proteins, which do not form a well-defined three-dimensional structure but are still functional. To understand the function of many proteins, it is important to recognise that they are dynamic systems subject to random thermal fluctuations that explore a continuous space of structures, or conformations, which we refer to as its conformational space.

1.1.1 Amino acids

Amino acids are made up of carbon (named the α-carbon) bonded to an amino (NH_2) group, a carboxyl group, and an additional functional group. Four different groups around a carbon atom gives rise to chiral stereoisomerism, wherein mirror image arrangement of atoms are not superimposable. The mirror-image isomers are called enantiomers. The only exception is the amino acid Glycine, where the additional functional group is a second hydrogen. Amino acids in biological systems are typically exclusively Levo (L) or left-handed enantiomers. Dextro (D) or right-handed amino acids occur rarely in contexts such as some cell wall components of bacteria, the neurotransmitter D-serine, and some peptide toxins.

Amino acids in proteins are covalently connected by peptide bonds between the amino and carboxyl oxygen group to form a linear chain called a polypeptide. We refer to the chain of α-carbons connected by peptide bonds as the backbone, and the additional functional groups are referred to as the side chains. Polypeptides interact with themselves and their environment primarily by non-covalent means. Notable exceptions are disulphide bond crosslinking and post-translational modifications. To understand how amino acids, and ultimately proteins, interact we need a good understanding of what forces are at play. In the following section (1.1.2) we outline a detailed description of the relevant interatomic forces.

1.1.2 Intermolecular forces

Sterics

The most important interaction experienced by all matter is Pauli Repulsion. This occurs due to the overlap of electron orbitals, which adheres to the Pauli exclusion principle, causing atoms to experience a short-ranged but strong repulsive force as they draw close together. Consequently, all atoms and, by extension, molecules, occupy a space that cannot be reasonably entered by other molecules without being first vacated. While of quantum mechanical origin, Pauli repulsion can be modelled classically as a strong repulsive term, as we will see later in Section 2.1.2. This approximation introduces several limitations, such as an inability to account for orbital-specific effects, including directionality in π-systems, but it is generally considered sufficient for most protein modelling applications.

For proteins, this has a number of important implications. Notably, like all polymer chains, proteins are self-avoiding; two parts of the chain cannot pass through each other without breaking the chain. This has strong implications for determining folding or fold-switching pathways and the accessible conformational space of a protein in general.

Furthermore, in the context of amino acids, the physical size and shape of functional groups play a crucial role in protein structure, function, and dynamics. The influence of the spatial arrangements of atoms within molecules is referred to as steric effects. The steric effects of side chains play an important role in promoting or disrupting local structure by exerting a controlling influence on the ability of the backbone to align correctly for backbone hydrogen bonding.

The size and flexibility of the side chains are important determiners in how tightly residues are packed within the core of a protein. This internal packing significantly impacts the stability of the global structure. In addition to packing, we would expect small residues like glycine to provide flexibility to the chain, while large side chains would restrict rotation about the backbone dihedral bonds. These variations in flexibility and mobility affect the number of accessible conformations, with direct entropic implications for the free energy landscape of protein folding and stability. If, in order to pass from one conformation to another, the residue must go through higher energy conformations, then we have a kinetic barrier. These sorts of kinetic barriers will strongly affect the timescales over which a protein folds or explores its conformational space. Over what timescales we must study proteins to capture all relevant conformations is an important consideration in determining what methods for studying protein dynamics are feasible.

Electrostatics

From a classical perspective, electrostatic forces arise due to the interaction between permanent charges. At a neutral pH, the residues aspartic acid and glutamic acid readily adopt a negative charge, and lysine, arginine, and histidine readily adopt a positive charge. Charged residues will experience strong charge-charge (ionic) forces with other charged moieties. Polar residues such as serine, threonine, tyrosine, asparagine, and glutamine contain electronegative atoms joined to electropositive atoms in their side chains and thus exhibit a smaller but still non-trivial charge called a partial charge. Polar residues can interact electrostatically with charged moieties (charge-dipole) and other polar residues (dipole-dipole). Hydrogen bonds are a special case of dipole-dipole interaction between a hydrogen atom bonded to an electronegative oxygen, nitrogen, or fluorine and a second electronegative atom with a lone pair of electrons. Hydrogen bonds are often classified separately from other dipole-dipole interactions as they are usually much stronger, more directional, and are particularly important in biochemistry. The directionality of hydrogen bonds in particular allows very strong hydrogen bonds to form when the electronegative atoms and hydrogen atom are nearly collinear. In some cases, the hydrogen is thought of as partially shared between the two electronegative atoms. A purely electrostatic interpretation of hydrogen bonding is unable to elucidate all experimental and theoretical observations. While a IUPAC task group has put forward recommendations to remove the reference to hydrogen bonding as an electrostatic interaction[1], viewing hydrogen bonds as being of electrostatic origin is sufficient for this work. A review of the energetic contributions to hydrogen bonding is given by Van Der Lubbe et al. [2]. All amino acids are capable of forming hydrogen bonds between the carbonyl oxygen of one backbone peptide bond and the amide hydrogen of a different peptide bond. Which hydrogen bonds are able to form is crucial for determining what local structure a polypeptide will adopt. The exception to this rule among standard amino acids is proline, which cannot donate a hydrogen as its amide nitrogen is bonded to its side chain to form a cyclic structure instead of a hydrogen. Many amino acids are able to form hydrogen bonds from their side chains. The capability of amino acid side chains to form hydrogen bonds is integral to the properties and functions of many proteins. Many protein-protein, protein-DNA, receptor-ligand, and enzyme-substrate interactions are facilitated and stabilised by the presence of side chain hydrogen bonding. Side chain hydrogen bonding is also responsible for the stabilisation of many local and global structural elements within proteins.

Induced forces

In the presence of a charge or dipole, all molecules, including non-polar molecules, will experience a distortion in their electron clouds resulting in a temporary induced dipole. We can then observe that all matter can electrostatically interact, albeit weakly, with permanent dipoles and charges. Induced forces are not typically classified as electrostatic forces as the mechanism, being of quantum mechanical origin, is not explained by classic electrostatics. How easily a temporary dipole can be induced depends on the number of electrons and how tightly bound those electrons are. The aromatic amino acids, phenylalanine, tyrosine, and tryptophan are then highly polarisable due to their delocalised π-electrons. Additionally, large aliphatic side chains such as those in leucine and isoleucine would be more polarisable than much smaller side chains like glycine and alanine.

A second type of induced force is the London dispersion interactions. Fluctuations in the electron density around an atom or molecule result in instantaneous dipoles that can induce a dipole in neighbouring atoms and molecules. This results in a weak attractive force between all matter which is particularly notable for non-polar and uncharged atoms and molecules which otherwise do not experience any attractive intermolecular forces. London dispersion forces are often modelled as a contribution to van der Waals forces, which also includes Keesom (rotationally time-averaged dipole-dipole) and Debye (dipole-induced dipole) forces.

Some classical models can partially capture the effects of induced forces implicitly through a combination of van der Waals interactions and Coulombic electrostatics. However, standard fixed-charge models struggle in cases where polarisation plays a significant role, such as ion interactions or hydrogen bonding. Explicit polarisation, whether incorporated into classical polarisable force fields or hybrid classical-quantum models, provides a more accurate description of these effects but is significantly more computationally expensive. As a result, fully polarisable models are typically avoided unless strictly necessary.

Disulfide bonds

Disulphide bonds are covalent bonds formed between two thiol groups (-SH) of two cysteine residues. They are formed under oxidising conditions, often with the help of a catalyst, and help stabilise proteins found in the extracellular medium, which is typically an oxidising environment. The disulphide bond has the effect of pinning two regions of a protein together. This can be important entropically in the folding process or in limiting the final folded conformations reasonably accessible to a protein. Disulphide bonds are also commonly found when protein subunits need to be assembled into larger complexes, such as antibodies, collagen, and elastin. Disulphide bonds can be broken in a reducing environment allowing cellular regulation of protein function by changing the redox environment of the protein. In classical models, covalent bonds are fixed and unable to be broken or formed dynamically without introducing a potentially expensive scheme, such as reactive force fields or hybrid quantum-classical methods. In classical simulations, disulphide bonds are usually predefined and remain fixed throughout.

Aromatic interactions

Amino acid residues with aromatic side chains can engage in π-stacking. π-stacking is the attractive force that occurs during the alignment of aromatic rings in an off-centred, parallel displaced, or T-shaped orientation. This attractive force results from a London dispersion interaction arising from instantaneous multipole-induced multipole charge fluctuations.[3] These interactions are often crucial in protein-DNA, protein-RNA, protein-carbohydrate interactions.[4] The presence of delocalised π orbitals allows aromatic residues to also interact with charged molecules, sulphur atoms, and with aliphatic C-H. Classical models can implicitly capture these interactions through the use of van der Waals and electrostatic terms, but higher accuracy can be achieved with polarisable models. Unless high accuracy is required, such as in cation-π interactions or precise free-energy calculations, polarisable models are usually avoided due to their high computational cost.

Solvent interactions

The hydrophobic effect is an entropy-driven concept characterised by the aversion of nonpolar solutes to water or the affinity of nonpolar solutes for each other in aqueous solutions. The Gibbs energy of transferring a small nonpolar solute, such as methane or ethane, from a polar solvent such as water into a non-polar organic solvent, such as heptane, is large and negative. Water is able to form up to four hydrogen bonds with its neighbours, resulting in a more structured and highly cohesive liquid. In the presence of a small hydrophobic solute, water will reorganise to maintain as many hydrogen bonds as possible. The resulting hydrogen bonding network is more constrained, leading to a reduction in water's configurational entropy and influencing the overall free energy of the system. Half of the naturally occurring amino acid side chains are nonpolar and are thus considered similarly hydrophobic. However, it has been found that burying hydrophobic side chains cannot be viewed as equivalent to the partition of a hydrophobic solute, as the hydrophobic group attached to the protein backbone is surrounded by water molecules perturbed by the backbone.[5] Additionally, macroscopic solutes have a larger enthalpic contribution to their solvation energies as hydration waters are simply unable to form all four hydrogen bonds, similar to a water-vapour interface.[6] While hydrophobic interactions in amino acid side chains undoubtedly exist, their strength and prominence are still debated.[5] Although hydrophobic residues are highly concentrated in the interior of the protein, exposed hydrophobic residues are often associated with protein-protein interaction sites and have been identified as the driving force behind protein-membrane interactions.[7] Experimental protein mutant studies on a range of proteins have suggested hydrophobic interactions being the largest contributor to protein stability followed by hydrogen bonds.[8]

While hydrophobic interactions have been described as the primary driving force behind protein folding for many decades, this is contested by other works with the identification of solvent-induced hydrophilic effects involving bridging waters hydrogen bonded to multiple residues that are longer range and stronger. [5] These effects obviously disappear once water has been entirely excluded to be replaced with hydrogen bonds between residues.

In summary, hydrophobic interactions are not explicit interactions, but rather an emergent property arising from the exclusion of water. In many classical models, water is treated as a rigid molecule, and these effects emerge through a combination of van der Waals and electrostatic interactions. A rigid treatment of water is fundamentally flawed. Without the ability to polarise dynamically, water cannot fully reorganise around hydrophobic solutes or participate correctly in bridging interactions. As a result, the hydrogen bonding network is often misrepresented, and water's behaviour is only an approximate fit to experimental data rather than a truly emergent property. In practice, the rigid water approximation is widely accepted because it allows for a larger time step (see Section 2.1.3), avoids the computational cost of a polarisable model, and still captures bulk properties and energetic trends sufficiently well, despite failing to reflect the true behaviour of water.

1.1.3 Structure

While there are many ways of classifying proteins, including by their biochemical function, cellular location, and organism of origin, we primarily classify proteins by their sequence and structure. To do this, we look at four hierarchical aspects of a protein: primary, secondary, tertiary, and quaternary structure.

Primary structure

The lowest level of structure is the primary structure, the sequence of different amino acid monomers that are synthesised together to form linear polypeptide chains. This sequence is ultimately determined by its corresponding DNA sequence. Many proteins are capable of spontaneously adopting their native fold, while others need the assistance of chaperones. Either way, this means that the primary structure is the ultimate determinant for higher levels of structure. Even single point mutations to the primary structure can often have extreme effects such as causing misfolding and inactivation of the final protein, both of which contribute to a number of genetic diseases.

Secondary structure

The secondary structure of a polypeptide describes the local structural arrangements of its amino acids. These structures are maintained primarily by intrabackbone hydrogen bonding, but side chain interactions have an important role in promoting or disrupting particular types of secondary structure and then stabilising these structures in their final folded configurations. The two main types of secondary structure are α-helices and β-sheets. α-helices are a helical configuration that allows for intrabackbone hydrogen bonds between every fourth amino acid. β-sheets are the parallel or anti-parallel alignment of polypeptide chains, with two intrabackbone hydrogen bonds per residue between adjacent strands. Other structures such as 3_{10}-helices and π-helices, helical configurations characterised by hydrogen bonds between every third and fifth residue respectively, are sometimes classified alongside α-helices as simply helices. β-bridges are isolated residues that fulfil the same criteria as β-sheets except that they do not involve the minimal sheet residue count of two residues per partner. Similarly, we assign the term ‘turn’ to a structure containing a single helix hydrogen bond.[10] Our ability to predict the secondary structure from the sequence of amino acids has dramatically improved in recent years.[11, 12]

Tertiary structure

The folding of a polypeptide chain in an aqueous environment is associated with the burying of the majority of hydrophobic residues in its core and exposing hydrophilic groups at its surface. A single poly-peptide chain can form just one or many hydrophobic cores, each associated with a distinct region of the protein referred to as a domains. Domains can often fold independently and then function either independently or in cooperation with neighbouring domains. In many cases, different arrangements of domains can result in different protein functions. Domains are often classified by the evolutionary conservation of sequential and structural characteristics across related protein families. We expect many small proteins to be single-domained, while larger, more complex proteins may contain dozens of domains and are referred to as multi-domain proteins. Regardless of whether a polypeptide chain is single-domain or multi-domain, the resulting three-dimensional structure is referred to as its tertiary structure. Many proteins are fully functional at this stage, with the tertiary structure representing the highest level of structure that some proteins obtain.

Many proteins contain regions that do not form a well-defined three-dimensional structure. These regions are often associated with an insufficient number of hydrophobic amino acids to form the hydrophobic core, that usually forms the basis of a well structured domain. These regions are termed intrinsically disordered regions. Additionally, a protein that lacks any significant tertiary structure is referred to as an intrinsically disordered protein. Despite the lack of structure, these intrinsically disordered proteins and regions still exhibit functionality. The classification of intrinsically disordered proteins is extensively covered by van der Lee et al.[13]

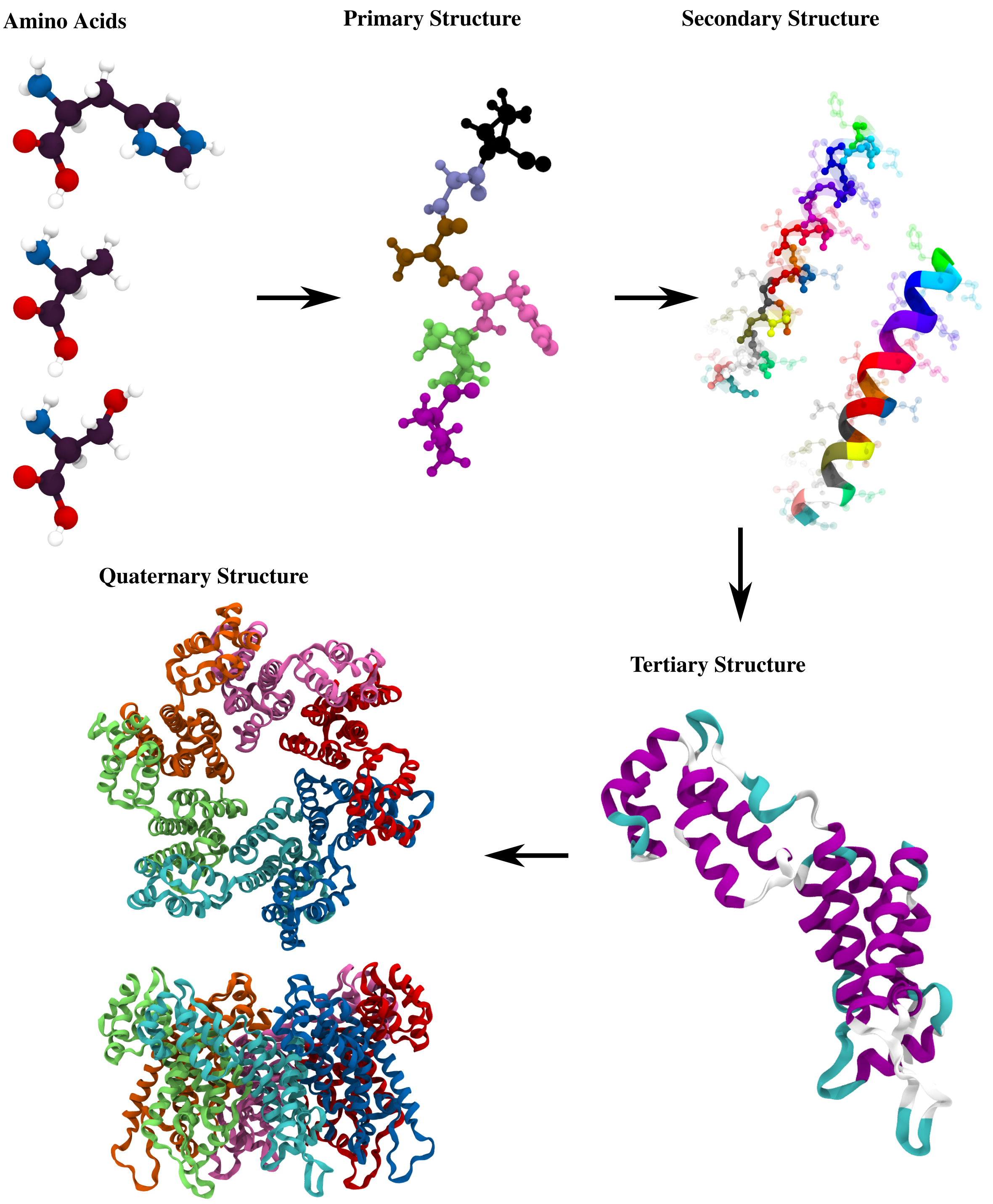


Figure 1.1: The hierarchical nature of protein structures exemplified here by the HIV p24 capsid (PDB ID: 3MGE)[9].

Quaternary structure

Many proteins are formed by oligomerisation of two or more polypeptide chain monomers into a larger oligomeric or multimeric complex. The monomers that make up a protein can be identical, giving us a homo-oligomer, or different, giving us a hetero-oligomer. Quaternary structure allows to access larger, more complex proteins. These larger more complex proteins often exhibit multiple functions that, when co-located and coordinated, can enable new higher-order functions or otherwise improve the efficiency of their existing functions. This is important in enzyme, signalling, and transport mechanisms.

Adoption of quaternary structure can also dramatically increase the stability of a protein. For example, the pore-forming toxin Aerolysin is significantly more resistant to chaotropic agents in its heptomeric form than as a momomer.[14] Aerolysin, like many pore-forming toxins, is secreted as a water-soluble monomer that recognises and binds to membranes before assembling into the active pre-pore, that is then able to insert and form a transmembrane pore. The ability to form quaternary structure in this case allows the protein to be produced and transported safely in its smaller, inactive, and soluble monomeric form.

Protein-protein interactions can be transient, and within the crowded cellular environment, the vast number of possible heteromeric structures makes defining, identifying, and describing quaternary structures particularly difficult. There is a general bias towards studying Homomeric complexes, likely due to the technical challenges, experimental complexity, and high potential for confounding variables associated with studying heteromeric structures. Both homomeric and heteromeric quaternary structure are reviewed by Marsh et al. [15]

1.1.4 The origin of function in dynamics

The traditional view of proteins is that the three-dimensional shape of a folded protein determines its function. A folded protein is thought of as adopting a single structure or state, the native state at the minimum of its free energy landscape. This view can be immediately challenged by the existence of intrinsically disordered proteins, which do not have a well-defined native state but are still functional.[16–18] Proteins are not so massive, or their intermolecular forces so strong, that they are unaffected by the intrinsic thermal motions of a system at physiological temperatures. Thermal perturbation will cause proteins to explore a space of conformations, as illustrated in Figure 1.2. This exploration underpins the conformational selection mechanism of protein binding, in which the bound-like state of a protein is a transient conformation that can be adopted with some probability to allow binding of a substrate.[19, 20] The main alternative to conformational selection is the induced fit mechanism of protein binding, in which a substrate is initially only loosely bound in a random conformation. This change in the environment of the protein, due to the presence of the substrate, results in a change in the free energy landscape of the protein that allows a conformational change to the bound state.[20, 21] The relative contributions of these two mechanisms are measurable and the factors that affect them can be studied,[22] but to understand either mechanism requires looking beyond viewing proteins as static structures.

As well as protein binding, a protein's dynamics play a role in the function of complexes themselves, for example, the gating mechanism that regulates protein degradation by the 20S proteasome[23] or the extrusion mechanism of the ClpP protease.[24] Protein allostery, the change in protein function at an active site due to the binding of an effector at a distal site, requires looking at the conformational dynamics to elucidate the various mechanisms of action.[20] In the extreme case are intrinsically disordered complexes containing disordered regions or entirely disordered peptides, whose lack of structure means their function almost entirely results from their dynamics.[17, 18] In many cases, the degree of disorder in a protein is sensitive to its environment, providing a mechanism by which some function can arise. We can also observe dynamics within quaternary structures, such as small heat shock proteins, which transition from a dodecomer to an interconverting ensemble of quaternary structures at heat shock temperatures.[25]

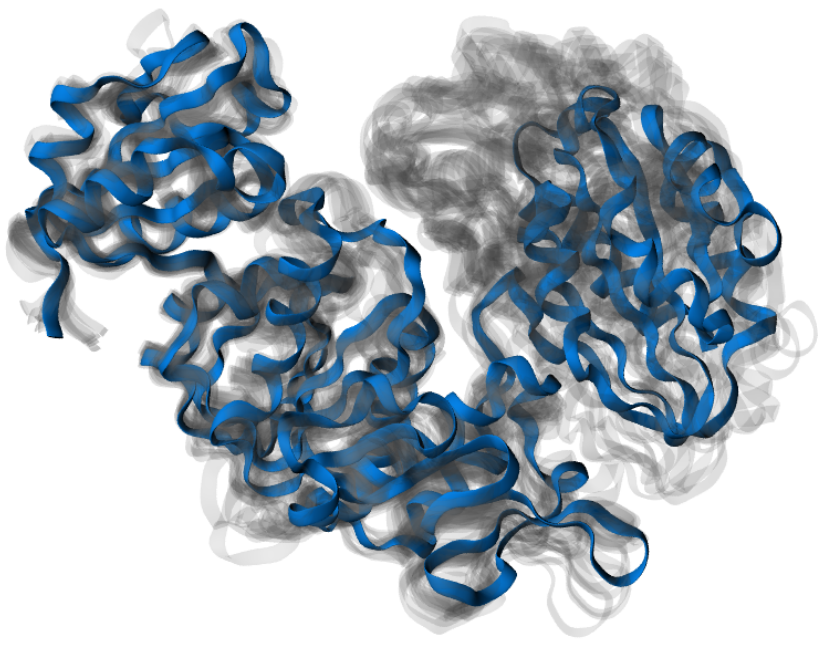


Figure 1.2: Rendering of the MurD protein in a ribbon representation (blue) overlayed with multiple other possible conformations indicating the range of different conformations adopted by MurD.

1.2 Characterising protein structure and dynamics

1.2.1 Experimental methods

It is possible to obtain high-resolution three-dimensional atomic density maps with X-ray crystallography and cryogenic electron microscopy (cryo-EM). From these density maps, we can model the positions of atoms to obtain reasonable estimates of static structures. X-ray crystallography requires crystallisation of the protein, which inherently limits its use to proteins that form well-ordered crystals. Large complexes, proteins with disordered or flexible domains, and proteins interacting with ligands, cofactors, and membranes can all present challenges. The information we do obtain is also of a static structure in a crystal, which does not reflect the environment of a protein in solution and provides no dynamic information. For cryo-EM, samples are plunge-frozen in the hope that the protein will not have time to substantially relax as it cools, and hydration is maintained in a frozen state as amorphous ice. In single-particle cryo-EM, many individual proteins are imaged, and the results are combined to overcome the poor signal-to-noise ratio. This technique allows imaging of many systems inaccessible to X-ray crystallography. The specific details are reviewed in detail elsewhere[26, 27] and not relevant here. The resulting density map from cryo-EM is averaged over many individual samples of the conformational space, which can cause difficulties when fitting structures to this map if, as is common, each individual image is assumed to be identical once aligned. There has been some success in retrieving dynamic information by classifying and separating different conformations in single-molecule images.[27]

Nuclear magnetic resonance (NMR) is another common method for structural determination of proteins and protein complexes at atomic resolution. Unlike cryo-EM or X-ray crystallography, NMR can be performed in solution, physiological mimicking environments or even, in some circumstances, within living cells. NMR provides structural information in the form of distance restraints and was previously limited to proteins with a molecular weight less than 50kDa. With the introduction of various techniques, including methyl transverse relaxation optimised spectroscopy (TROSY) and the use of deuteration, this limit has been pushed up substantially.[28] Standing out from cryo-EM and X-ray crystallography, NMR is able to offer a toolbox of methods to study dynamics at a range of timescales. Of particular note is the ability for NMR to detect transient rare conformations in addition to characterising dynamic properties like conformation correlation times and exchange rates.[29] However, as with cryo-EM and X-ray crystallography, NMR provides specific measurements about the system that must be modelled in order to be interpreted. The results will be sensitive to any assumptions made in this modelling process and often require validation with another method. In general, NMR struggles to provide detailed structural and dynamic characterisations in situations such as those involving IDPs and large protein complexes, but in these cases, NMR can be used to complement other methods.

1.2.2 Computational Modelling

Molecular dynamics

Through the use of X-ray crystallography, NMR, and cryo-EM, there has been an exponential growth in atomic-resolution three-dimensional structures of proteins made available through the Protein Data Bank.[30] We can use these structures as starting points for molecular dynamic (MD) simulations to computationally explore the dynamics of these systems. From some initial starting point, the time evolution of the positions and velocities of atoms is followed by numerical integration of Newton's equations of motion. By sampling the atomic positions at regular intervals, we generate a trajectory. Given sufficient time to evolve the system, this trajectory can be taken as a representative sample of the conformational space of the system. Any arbitrary system, such as those including protein-protein, protein-ligand, and protein-membrane interactions, can be simulated. Assuming that we can initialise the simulation with a reasonable state, the primary limiting factor for MD simulations is computational power. However, fully characterising the slow motions of many large systems would require simulations over millisecond to second timescales. Highly expensive specialised hardware can achieve on the order of 100 µs/day, while off-the-shelf NVIDIA GPU hardware, that is more likely to be available to the average researcher, can only achieve at least two orders of magnitude less.[31] These speeds can be further improved upon by moving to implicit solvents and coarse-grained force fields, but we lose detailed structural information and specific atomic interactions, such as hydrogen bonding that will have an impact on the accuracy of the results.

The MD trajectory is Boltzmann distributed, with the probability p of observing any particular conformation being inversely proportional to its energy ϵ:

$$p_i \propto e^{-\frac{\epsilon_i}{k_B T}} \tag{1.1}$$

where k_B and T are the Boltzmann constant and temperature, respectively. This is because, in an ergodic and equilibrium simulation, the system explores all accessible conformations over time. Since transitions between states occur with probabilities that depend on their energy differences, lower-energy conformations are visited more frequently as they require less thermal activation. Over long timescales, the fraction of time spent in each conformation approaches its Boltzmann probability, ensuring that the trajectory samples states according to their thermodynamic equilibrium distribution. MD will then be particularly inefficient when the conformational space contains two or more conformations whose interconversion involves passing through energeticallyunfavourable conformations. The free energy landscape of such a system can be described as containing two or more low-energy basins separated by a high-energy barrier. During an MD simulation, the system would cross this barrier only rarely due to the low probability of accessing the high-energy conformations required to transition between the two basins, such as those illustrated in Figure 1.3. It is often difficult to determine beforehand whether additional basins of conformations exist; thus, knowing whether a trajectory is a representative sample of the full conformational space can be challenging without a priori knowledge. Additionally, we are often most interested in rare events, such as the high-energy conformations in a transition pathway and transiently occupied metastable states, which will likely be poorly sampled in the MD trajectory.

Aside from sampling efficiency, MD has two further main limitations. Firstly, the choice of how we model the physical interactions (quantum, classical, ab initio etc) in the system is a trade off between accuracy and computational efficiency. This issue goes beyond the scope of this work. The second common issue with MD simulations is that the output trajectories are high-dimensional and noisy. Meaningfully interpreting the huge quantities of data is a difficult task. Fortunately, there are a number of methods for producing lower dimensional

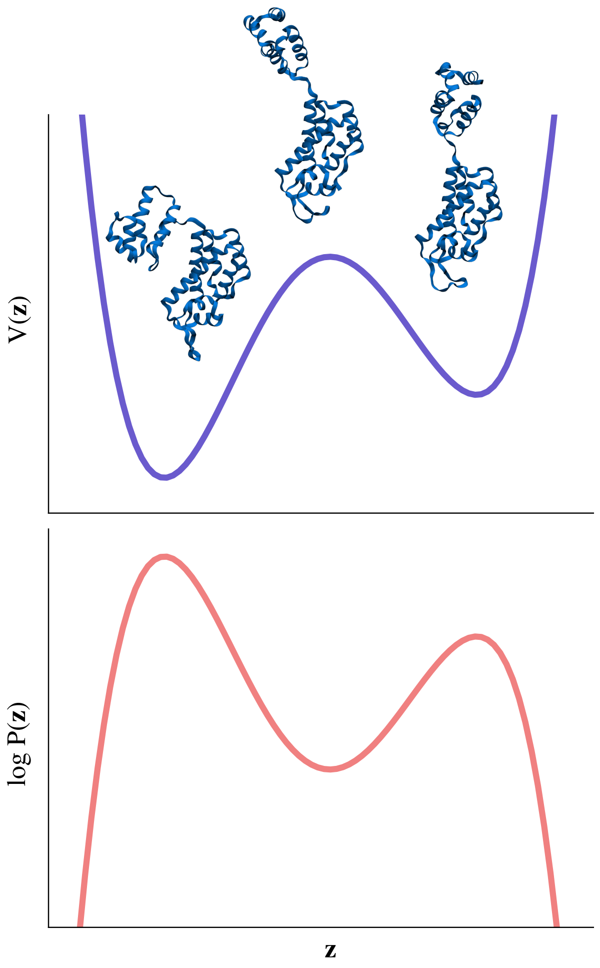


Figure 1.3: Top: Illustration of a 1D projection of the energy landscape of a protein with two states separated by an energy barrier. Bottom: The probability of sampling a conformation along this projection.

representations, many of which can also be used to improve the sampling efficiency, as will be discussed in the next section.

Enhanced sampling

Enhanced sampling refers to a collection of techniques used to improve the efficiency of MD sampling. Efficiency in this context typically refers to either how effectively the configurational space is sampled or how accurately and quickly we can estimate kinetic and/or thermodynamic properties within this space. The former includes discovering new regions of the configurational space associated with free-energy basins, different transition pathways between free-energy basins, and high-energy configurations within a free-energy basin that may be mechanistically relevant. The latter includes estimating equilibrium probability distributions, binding affinities, and transition rates.

Enhanced sampling techniques can be divided into those that make use of Collective Variables (CVs) and those that do not. CVs are low-dimensional representations of a system, defined as a function of atomic coordinates.

They are often curated such as to capture the motions that are of interest in a system. Most CV based enhanced sampling techniques apply external bias within the space defined by the collective variables to drive the simulation towards undersampled regions of the conformational space. The resulting biased trajectory can be used to recover the original unbiased ensemble statistics through the use of reweighting procedures. Popular CV-based techniques include Umbrella Sampling [32], Metadynamics [33] and its subsequently improved variants, such as Well-Tempered Metadynamics [34], and Variationally Enhanced Sampling [35]. Various other non CV-based methods exist such as Replica Exchange MD [36] and Simulated Tempering [37]. Specific details of all these methods are extensively documented within a range of literature reviews [38–41].

Determining Collective Variables

The quality of CVs is crucial for the efficacy of any CV-based enhanced sampling scheme, yet the construction of optimal CVs remains a challenge. Traditionally, CVs were handcrafted to incorporate experimentally determined properties or physical intuition of the system. In the simple case of two interacting atoms, an appropriate collective variable would be the interatomic distance between the atoms. In this way, we have a rotationally and translationally invariant projection of the system that reduces the 6 dimensional Cartesian representation to a single CV. For larger systems with N atoms, we have a 3N dimensional space to represent, and choosing how to represent the rearrangement of a domain in a protein is less trivial. Sometimes, simple global properties can be useful, such as the radius of gyration, which can be used to capture the compactness of a system. Distances between atoms or the centre of mass of groups of atoms can capture specific key interactions that mutant studies, NMR, or physical intuition suggest may be a good probe of a motion. Crafting CVs requires a great deal of time, highly specific expert knowledge, or high quality experimental data. Biochemical systems that are particularly large and complex may require a large number of handcrafted CVs to efficiently represent the motions of interest. Any motions that are not captured by the CVs can result in even more poorly sampled MD trajectories or systematic errors that are not guaranteed to be obvious during later statistical analyses of the system.

The alternative to handcrafting CVs is to programmatically determine them from the system itself. We already have a means of collecting information about the system in the form of MD simulations. Ideally, we want a method of reducing the large, noisy, and high-dimensional coordinate trajectories produced by even relatively short bursts of MD to a low-dimensional representation. Fortunately, dimensionality reduction methods have been widely studied within the field of unsupervised machine learning. In the context of machine learning, we use the term latent variables to describe statistically derived variables that model the underlying factors in the data, while the term Collective Variables is associated exclusively with the field of biomolecular modelling and its roots in handcrafted variables chosen to describe some physical property.

Principal Component Analysis (PCA) is a linear dimensionality reduction technique that decomposes high-dimensional data into orthogonal principal components, which maximise the variance represented in the data. These components are linearly uncorrelated (decorrelated) and ranked in order of the amount of variance they capture. By selecting the first few principal components, one can obtain a lower-dimensional and interpretable projection of the data. The issues with PCA are similarly straightforward. The primary issue with PCA arises when the underlying relationships in the data are non-linear, as PCA will require multiple dimensions to capture this covariance. As protein motions are rarely linearly correlated, this is particularly problematic in this application. Another limitation of this method is its strong dependence on the quality of the sampled trajectory. If the trajectory does not sufficiently explore all relevant conformational states, the constructed PCs may fail to capture meaningful modes, leading to inefficient sampling. This creates a classic chicken-and-egg problem: Good conformational space sampling is needed to construct effective CVs, yet good CVs are required to efficiently enhance sampling and explore the full conformational space. This problem is common to all the methods discussed here; however, PCA is particularly vulnerable because it assumes linearity and ranks motions based solely on variance.

Non-linear dimensionality reduction methods, such as t-distributed stochastic neighbour embedding [42], diffusion maps [43], and autoencoders, [44–48] have all been applied to producing latent variables that can be used as CVs on protein systems. By introducing non-linearities, we start to trade off the ability to interpret and physically rationalise what these CVs represent with general improvements in capturing relevant degrees of freedom.

The field of machine learning has, for the last decade, been dominated by the development of Artificial Neural Networks (ANNs). Among the many existing ANNs, autoencoders are conceptually some of the simplest. These neural networks incorporate dimensionality reduction into their training procedure, making them ideal for generating CVs. Autoencoders attempt to minimise the difference between their input and output, typically by incorporating a low-dimensional bottleneck, to learn a non-linear representation of the data that, similar to PCA, maximises the explained variance. ANNs, and autoencoders in particular, are of specific interest here for a number of reasons. ANNs give us an explicit and differentiable mapping from atomic coordinates to the CVs, allowing us to perform biasing directly in the discovered CV space. More importantly, the training procedure and architectural design of the network can be modified to implicitly and explicitly incorporate prior knowledge, constrain the resulting CVs to take on particular desirable properties or functional forms, and perform additional tasks, such as directly generating the starting conformations for new simulations.

The adaptation of Variational Autoencoders (VAEs) for use in CV extraction are a good example of this flexibility. VAEs constrain the probability density of the latent space to match a prior distribution chosen a priori. This has the benefits of enabling a simple pathway for generating new samples from the probability distribution and allows for the estimation of a variational lower bound on the probability density of samples. Ribeiro et al. [49] used the constraining of the learnt latent space in VAEs to produce latent variables with a more rational physical interpretation.

If instead of training the network to reproduce its input, we train it to produce its state as it looks some set time in the future, then we constrain the network to incorporate temporal information into its collective variables.[47, 49–52] This temporal information assists the ANN in discriminating between high-variance and slowly decorrelating variables. This time-lagged approach is not unique to ANN methods, being the basis for time-lagged independent component analysis.[53] A different approach from Wang et al.,[54] aimed at improving the interpretability of the network, was time-lagged autoencoders with a linear encoder and a stochastic deep decoder. The linear encoder allows for the greater interpretability of methods like PCA while incorporating temporal information.

The time-lagged approach tends to produce latent variables that are comprised of a mixture of high-variance and slow modes. Neural networks incorporating a variational approach for Markov processes (VAMPnets),[52] state-free reversible VAMPnets (SRVs), [51, 55], and variational committor-based networks (VCNs) [51] are all targeted at capturing the descriptors of slow modes.

ANNs can be used with a variety of input types, such asorking with manually defined input features[53], pairwise distances[56, 57], internal coordinates [45], or explicit Cartesian coordinates. [45, 46, 58, 59] There are a variety of architectures that have been used, including shallow fully connected models [44, 45, 57], Siamese networks [47], deep fully connected VAEs [59], deep convolutional models [58], and graph neural networks [60].

1.3 Overview of this work

In previous work by our group a fully connected feedforward autoencoder was adopted to learn a latent representation of the dataset [46]. The network was then used to generate new conformations by decoding latent vectors corresponding to points outside of the known sample space. The paper demonstrated one application in which these conformations were used as candidate conformations for a protein docking procedure. The network was found to perform poorly when attempting to extrapolate protein conformations but could discover new geometrically plausible conformations while interpolating. There are several drawbacks to the methods that will be addressed in this work. Firstly, fully connected feedforward networks scale poorly with the dimensionality of the input dataset. This is the same problem that was run into by those working on image ML tasks. This spurred the development of 2D convolutions, which take advantage of 2D spatial locality of images. We can note that proteins, as long linear polymers, are typically represented as a 1D list of 3D coordinates, and thus it should not be unreasonable to employ 1D convolutions. Convolutional networks offer several benefits over fully connected networks, including reduced computational complexity due to shared weights, better handling of spatial hierarchies through local receptive fields, and improved generalisation by capturing local patterns and structures in the data.

The second issue discovered in previous work was poor performance of the network in regions not local to an example in the training dataset, such as when extrapolating or interpolating between widely separated points. Trivially increasing the size of the training set by running more or longer simulations is not an acceptable solution, as this defeats the purpose of training the network. To encourage the network to produce more physically plausible structures, we introduce a physics-based loss function that penalises extremely high-energy structures. We study the effect this has on the produced latent space and on interpolated and extrapolated conformations. In our initial work, we produced a PyTorch implementation of an MD force field, with which we calculate energies for our loss function. We document subsequent work to integrate OpenMM[61] as the engine behind our physics-based loss function, opening potential avenues for future development of this work. The methods covered in this work have been compiled into *molearn*, an open-source software package.[62]

In the following, Chapter 2 provides a comprehensive description of the theory and methods underpinning this work. Chapter 3 details our published work on the training and analysis of a neural network applied to protein conformational spaces.[58] Chapter 4 details the development and delivery of *molearn*, along with its integration with OpenMM.[62] Chapter 5 details unpublished results, where we leverage on *molearn* to address the limitations of the machine learning model presented in Chapter 3. Finally, Chapter 6 concludes this thesis.

Chapter 2

Theory

2.1 Molecular Dynamics

2.1.1 Basics

Molecular dynamics (MD) is an approach to molecular simulation that follows the motions of atoms by numerically solving Newton's equations. For a system of N atoms, the positions $\mathbf{R}(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N)$ of the atoms at a finite difference in time can be approximated by taking the first three terms in the Taylor expansion of $\mathbf{r}_i(t + \mathrm{d}t)$:

$$\begin{aligned}\mathbf{r}_i(t+\mathrm{d}t) &\approx \mathbf{r}_i(t) + \dot{\mathbf{r}}_i(t)\mathrm{d}t + \ddot{\mathbf{r}}_i(t)\mathrm{d}t^2,\ i = 1 \ldots N \\ &\approx \mathbf{r}_i(t) + \mathbf{v}_i(t)\mathrm{d}t + \tfrac{1}{2}\mathbf{a}_i(t)\mathrm{d}t^2\end{aligned} \tag{2.1}$$

where $\mathbf{a}_i$ and $\mathbf{v}_i$ are the acceleration and velocity of atom i respectively. Equation 2.1 can be repeatedly solved to generate a series of coordinates in time that form a trajectory. The velocities for each atom are calculated from the acceleration, such as in the *velocity Verlet* algorithm, which uses different formulations of the following equation:

$$\mathbf{v}_i(t+\mathrm{d}t) = \mathbf{v}_i(t) + \tfrac{1}{2}[\mathbf{a}_i(t) + \mathbf{a}_i(t+\mathrm{d}t)]\mathrm{d}t \tag{2.2}$$

However, this work uses the *leap-frog* algorithm, which uses the following update rules:

$$\mathbf{v}_i(t+\tfrac{1}{2}\mathrm{d}t) = \mathbf{v}_i(t-\tfrac{1}{2}\mathrm{d}t) + \mathbf{a}_i(t)\mathrm{d}t$$

$$\mathbf{r}_i(t+\mathrm{d}t) = \mathbf{r}_i(t) + \mathbf{v}_i(t+\tfrac{1}{2}\mathrm{d}t)\mathrm{d}t.$$

By interleaving position and velocity updates, we prevent phase lag, ensuring more accurate long-term energy conservation and time reversibility. The *velocity Verlet* algorithm achieves the same properties by using the average acceleration of the current and next time step.

The relationship between acceleration $\mathbf{a}$ of an object and the force $\mathbf{F}$ acting on it is famously given by Newton's Second Law ($\mathbf{F} = m\mathbf{a}$). By writing the force as the negative differentials of a potential function $V(\mathbf{R})$, we can express acceleration on an atom as follows:

$$\mathbf{a}_i = \frac{\mathbf{F}_i}{m} = -\frac{1}{m}\frac{\partial V(\mathbf{R})}{\partial \mathbf{r}_i} \tag{2.3}$$

2.1.2 Force Fields

The potential energy function, referred to in molecular modelling as the *force field*, approximates the potential energy V_i on an atom as the sum of contributions from the interactions with its neighbours.

$$V_i = V_i^{\text{bonds}} + V_i^{\text{angles}} + V_i^{\text{torsions}} + V_i^{\text{non-bonded}} \tag{2.4}$$

Equation 2.4 gives the contributions typically included in a force field, although other contributions can be defined and added. Importantly, the choice of the terms representing these interactions fundamentally defines the force field and can have a profound effect on the validity of a model. With the exception of non-bonded interactions, each interatomic potential is determined by the connectivity of the system defined before MD is started. The connectivity is kept fixed for the duration of an MD simulation, preventing the simulation of chemical reactions. Schemes to modify the connectivity go beyond the scope of this work.

The potential due to the covalency between two atoms can be simply represented by a harmonic potential:

$$V^{\text{bond}}(r_{ij}) = \tfrac{1}{2}k_b(r_{ij} - r_0)^2 \tag{2.5}$$

where r_{ij}, k_b, and r_0 are the distance between atoms i and j, the force constant, and equilibrium distance parameters respectively. The contribution to V_i from the bonds term can thus be written as the sum of Equation 2.5 over all atoms j covalently bonded to atom i:

$$V_i^{\text{bonds}} = \sum_{ij} \tfrac{1}{2}k_b(r_{ij} - r_0)^2 \tag{2.6}$$

The force on an atom due to its bonds can then be written as:

$$\mathbf{F}_i^{\text{bond}} = -\frac{\partial V_i^{\text{bonds}}}{\partial \mathbf{r}_i} = \sum_{ij} k_b(r_0 - r_{ij})\frac{\mathbf{r}_{ij}}{r_{ij}} \tag{2.7}$$

The harmonic potential is reasonably accurate for small deviations from r_0 and at a similar temperature as for that which both r_0 and k_b were parameterised. If a better representation is required, additional higher order corrections can be added, or more representative terms, like the Morse potential, can be used.

As V_i^{bonds} represents bonded two-body interactions, similarly bonded three-body interactions (due to repulsion between bond pairs) can also be represented by a harmonic potential. This potential, termed the angular potential V^{angle}, can be simply constructed by replacing r_0 and k_b with equivalent angle parameters θ_0 and k_θ:

$$V^{\text{angle}}(\theta_{ijk}) = \tfrac{1}{2}k_\theta(\theta_{ijk} - \theta_0)^2, \tag{2.8}$$

$$V_i^{\text{angle}} = \sum_{ijk} \tfrac{1}{2}k_\theta(\theta_{ijk} - \theta_0)^2, \tag{2.9}$$

$$\mathbf{F}_i^{\text{angle}} = \sum_{ijk} k_\theta(\theta_0 - \theta_{ijk})\frac{\mathbf{r}_{ijk}}{r_{ijk}} \tag{2.10}$$

where ijk refers to any set of consecutively bonded atoms: i, j, and k. Like V^{bond}, this potential is only valid for small displacements from θ_0. However, better representations are not usually required, as incurred errors are typically dwarfed by other interaction terms in the majority of cases.[63] In some force fields, like the CHARMM force field[64], 1-3 motions are restrained by applying a two-body potential termed the Urey-Bradley potential.

Four-body interatomic interactions between four consecutive atoms are represented by dihedral or torsional potential terms. A torsional potential introduces a periodic barrier to rotation about the bond between the second and third atoms. This is illustrated for the simple case of butane in Figure 2.1. In the Amber ff14SB [63] and other widely used force fields, these are expressed as:

$$V^{\text{torsion}}(\phi_{ijkl}) = \sum_{n} \frac{V_n}{2}[1 + \cos(n\phi - \gamma)]. \tag{2.11}$$

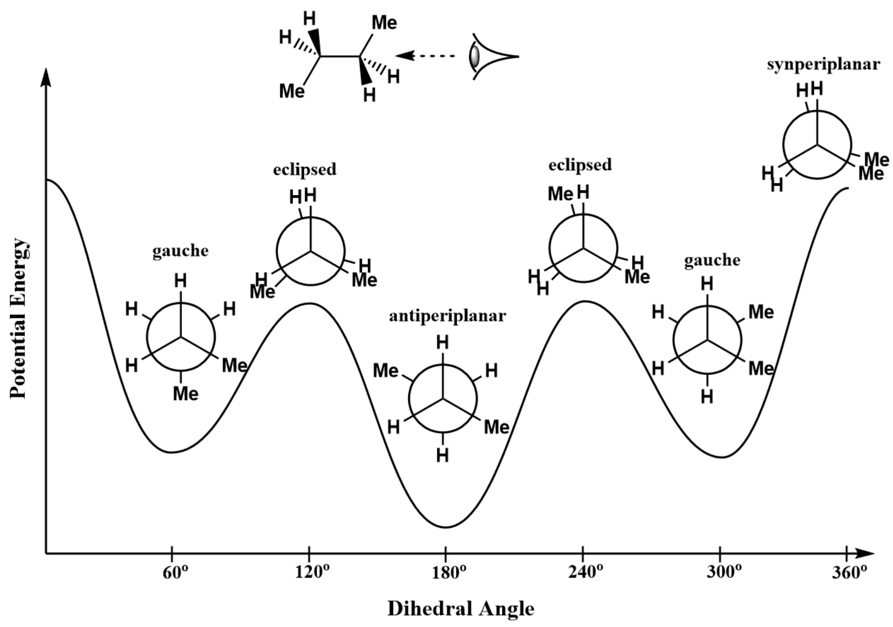


Figure 2.1: Torsional potential energy profile of butane as a function of the dihedral angle. The plot illustrates the energy variations associated with different conformations of butane due to rotation around the central C–C bond. The energy minima correspond to the *gauche* ($\pm 60°$) and *anti* ($180°$) conformations, while the maxima correspond to the *eclipsed* ($0°, 120°, 240°$) and *synperiplanar* ($360°$) conformations. The periodic nature of the torsional potential can be represented using a Fourier series, capturing the threefold symmetry of the potential energy surface.

This is a Fourier series where V_n are the weights, γ are the equilibrium phase values, n is the number of terms in the series, and ϕ is the angle between the planes described by ijk and jkl. Any number of terms in the Fourier series can be used and are supported within most software packages. A minimalist approach, like that used by AMBER force fields, usually only requires three terms in Equation 2.11. In the ff14SB force field, these three terms are parameterised based only on the identity of the centre two atoms but then require careful treatment of the 1-4 non-bonded interactions. There are a number of exceptions to this, such as when atoms 1 and 4 are sufficiently electronegative to incur a significant *'gauche'* effect (hyperconjugational stabilisation of the gauche conformation), allowing just two terms to be sufficient to represent this interaction. Improving the parameterisation of the torsional terms has been the main focus in the development of the original Amber ff94 force field [65] into its subsequent forms (ff99, ff99SB[66], and ff14SB[63]), such that the original functional form and many of the bond and angle parameters remain the same. Improper torsional potentials represent interactions between four atoms not consecutively bonded and are typically used to enforce planarity in ring structures when dihedrals are not sufficient. Terms to represent these interactions are usually formulated identically to proper torsional terms, applying Equation 2.11 to any four bonded atoms.

Non-bonded interactions are described in three forms in a force field: repulsion, dispersion, and electrostatic. A two-body potential term, $V^{\text{non-bonded}}$, can be represented as:

$$V^{\text{non-bonded}}(r_{ij}) = \frac{A_{ij}}{r_{ij}^{12}} - \frac{B_{ij}}{r_{ij}^{6}} + \frac{q_i q_j}{\epsilon_{ij} r_{ij}}, \tag{2.12}$$

$$V_i^{\text{non-bonded}} = \sum_{i \neq j} \left[\frac{A_{ij}}{r_{ij}^{12}} - \frac{B_{ij}}{r_{ij}^{6}} + \frac{q_i q_j}{\epsilon_{ij} r_{ij}} \right], \tag{2.13}$$

$$\mathbf{F}_i^{\text{non-bonded}} = -\sum_{i \neq j} \left[12\frac{A_{ij}}{r_{ij}^{13}} - 6\frac{B_{ij}}{r_{ij}^{7}} + \frac{q_i q_j}{\epsilon_{ij} r_{ij}^2} \right] \frac{\mathbf{r}_{ij}}{r_{ij}} \tag{2.14}$$

where q_i and q_j are the charges of the respective atoms. This term is a combination of a 12-6 Lennard-Jones term, representing repulsion and dispersion interactions, and a Coulombic term, representing electrostatic interactions. A_{ij}, B_{ij}, and ϵ_{ij} are parameters dictating the strength of these interactions. This term is calculated only for intermolecular atoms or intramolecular atoms separated by at least three bonds. Intramolecular 1-4 interactions (atoms separated by three bonds) can be either calculated with an additional scaling factor to reduce their effects or neglected completely, with the Amber14SB force field opting for the former.

Through all the terms described above, a number of parameters have been mentioned: k_b, r_0, k_θ, θ_0, V_n, γ, A, B, q, and ϵ. The values of these parameters are typically determined by the *atom types*, a term normally referring to each atom's element and geometry but can also infer information about bonded atoms. Different atom types are required for the same element when the local atomic environment differs, as this affects bonding, charge distribution, and interaction potentials. What parameters are required and how these are calculated varies substantially between different force fields. The Amber14SB force field parameter set, for example, has come about through the cumulative effort of two and a half decades of active development. The force field has been subjected to extensive benchmarking, followed by careful revision of the parameter sets to better reproduce observables while respecting physics. The exact parameterisation of Amber14SB is discussed at length in the Amber documentation [63, 65].

We use the Amber14SB force field for all MD data collection in this work to remain consistent with previous works by our group[46], from which the trajectories are also reused here. Additionally, we use a subset of the Amber14SB parameters in our custom physics-based loss functions, presented in Chapters 3 and 4, to ensure that the physical model used to generate the training set remains as consistent as possible with the physical model used to constrain network training.

2.1.3 Simulation Time Step

The choice of time step ($\mathrm{d}t$) in an MD simulation has strong implications on the obtained results. A large time step is desirable to efficiently sample the phase space and run simulations of any great length. A key assumption in MD, resulting from the use of the Taylor expansion (Equation 2.1), is that the velocities and accelerations are constant through a time step. If the time step is large relative to the vibrational period of a particle's motion, the validity of this assumption must be called into question. Thus, the highest frequency motions (bond stretching and bending) in a system are the limiting factor in the size of the time step. In practice, the larger the time step, the greater the fluctuations in the obtained instantaneous thermodynamic quantities, and the more likely a system is to experience a significant systematic drift in total energy.[67]

The highest frequency motion in an MD simulation is typically the C-H bond stretch, whose period is on the order of 10^{-14} s. Typically, the time step is set to be roughly 10 times smaller than this motion, giving a time step of 1 fs. This time step can be increased by a factor of 2 by applying constraints in order to remove the vibrational (bond and angle) motions.[68] In the LINCS algorithm, this is done by resetting covalently linked atoms to their equilibrium distances and angles after each time step.[67] It must be noted that the flexibility of bond angles is essential for realistic sampling of configurational space, and so, constraints should not be universally applied to angles.[69] Using LINCS, a 2 fs time step is achievable for the majority of systems. A 2 fs time step means that fully sampling the conformational space of even moderately sized systems is expensive and time consuming. Larger systems often have to be observed on second time scale to properly sample their conformational space, which makes simulating these systems with atomistic MD infeasible from a time and cost perspective.

2.2 Artificial intelligence

Intelligence typically refers to the capacity to understand, learn, and apply knowledge. When man-made systems are designed to imitate these characteristics, we term this artificial intelligence (AI). AI systems, which are often but not exclusively software-based, are capable of performing tasks that would otherwise require a natural intelligence. The primary focus of the AI field for some time has been the study of machine learning (ML) to the extent that the terms AI and ML are used synonymously. We must be careful with this presumption. Expert systems, which mimic human-like reasoning and knowledge representation through a system of predefined rules and a knowledge base manually curated by an expert, are not capable of "learning" but, because they mimic some part of human cognition, are still firmly classified as within the field of AI. There is, however, still a strong expectation of adaptability and learning within AI research.

2.3 Machine learning

A machine learning (ML) algorithm is an algorithm that can experience a data distribution and utilise this to improve its performance on a task with respect to a well-defined performance measure. ML algorithms have been widely successful at a variety of tasks that natural learning systems struggle with. In recent years, the advent of deep neural networks has allowed the ML field to approach, and in some cases has exceeded, human-level performance in a number of tasks previously exclusively in the human domain. Prominent and well-publicised examples include language[70], audio[71], and vision synthesis tasks[72].

2.3.1 Overview

The ML landscape was initially dominated by the fields of statistics; however, many recent advances have been strongly influenced by psychology, biology, and natural systems in general.[73] Early algorithms such as linear regression, logistic regression, and naive Bayes classifier are highly efficient, interpretable, and well-defined. These are attractive properties that justify the continued use of these methods. We obtain these properties by making broad assumptions, such as task linearity, which dramatically restricts the expressive power of these methods. This is acceptable if these assumptions prove sufficiently valid; however, we would correctly expect these methods to perform poorly on tasks involving complex patterns, non-linear dependencies, and high-level abstractions.[74]

Restricting the hypothesis space to the set of all linear functions is a poor assumption for many of the more difficult tasks we are interested in.[74] Linear models can be extended to allow non-linearities with strategies such as the kernel trick, the enhancement that forms the basis of the category of algorithms known as Kernel Machines. The kernel trick is an implicit mapping into a high-dimensional feature space that enables the linear separation of data points that are non-linear in the original space A predominant example is the Support Vector Machines, which, enhanced by the kernel trick, dominated the field of classification through the 1990s and early 2000s.[75] With the introduction of non-linearities, we are trading interpretability and efficiency for a greater expressive power.

How well these machine learning algorithms perform is highly dependent on how the data is presented. Data sourced from the real world is often very noisy and high-dimensional. For example, individual pixels in an image are negligibly correlated with what a human would perceive that image to contain. Given an appropriate set of hand-crafted features from an image, such as the number of legs, eyes, and presence of feathers, scales, or fur, we could easily expect a simple logistic regression to correctly discern between a number of different types of animals. However, the logistic regression would almost certainly fail if given the raw pixel data as features.[74, 76]

The form or format of the data presented to the ML method is called its representation. A good representation would enable even simple ML methods to perform well on a given task.[76, 77] The primary problem faced by any ML engineer is to determine how to transform a raw dataset into a good representation. Representation learning tackles this problem by proposing that producing a good representation itself can be viewed as a task solvable by ML methods.

2.3.2 Supervised and unsupervised learning

To understand how we can learn a representation, we must introduce the concept of unsupervised learning as opposed to supervised learning. The methods described previously are all examples of supervised ML algorithms. With supervised learning, the correct output, the ground truth, must be provided in order for the algorithm to attempt to learn a mapping from the input representation to this output. For an example x with an associated ground truth y that we wish to estimate, we create a mapping of the form $y' = f(x)$, typically by estimating the conditional probability of y given x, $p(y|x)$. In this context, y must be known or decided on beforehand, allowing us to adjust f such that the value of y' approaches y.

In unsupervised learning, we attempt to extract meaningful features by identifying patterns or intrinsic structures in the data. These features form a representation that ideally is lower dimensional, more interpretable, and captures only the most salient information needed for downstream tasks. Distinct from supervised learning, we do not know the optimal representation beforehand. Generally, unsupervised learning involves learning the probability distribution that generated the dataset. We aim to implicitly or explicitly learn $p(x)$ from examples x drawn from the distribution p. Two broad classes of traditional unsupervised algorithms are clustering, e.g., k-means clustering, hierarchical, and Gaussian Mixture Models; and dimensionality reduction techniques: e.g., principal component analysis (PCA), Autoencoders, and t-Distributed Stochastic Neighbour Embedding (t-SNE). These algorithms are adept at extracting simple features from data that can be used as the input representation for supervised ML algorithms. However, extracting abstract and high-level features remains a challenge.

2.3.3 Deep learning

Presently, the field of machine learning is dominated by hierarchical methods that stack many, often very simple, ML algorithms in layers, each depending on the output of the previous layer(s). As the defining characteristic of these methods is the depth of the stack, this subset of the ML field is termed deep learning. Each layer takes the simpler representation produced by previous layers and builds a more complex representation. This allows abstract and high-level features to be extracted from raw data through the composition of many simple transformations. Deep learning methods are therefore far more versatile, applicable to a wider range of tasks, and require less specialist expert knowledge about the task.

2.3.4 Neural Networks

Many natural learning systems are highly capable of performing a wide range of tasks, many of which are complex and abstract, with limited or even no prior experience. The example most familiar to the reader is the human brain. The human brain is composed of neurons, specialised cells that transmit electrochemical stimuli to adjacent neurons when stimulated above a certain threshold by their own neighbours. This network allows for complex processing and reaction to stimuli without prior direct experience. Artificial neural networks (ANNs) are models inspired by the biological brain, although most modern models would be considered a poor imitation of actual biological brains. Attempting to mimic the exact behaviour of the biological brain is both an extremely difficult and expensive task. More importantly, it is largely unnecessary for producing ANNs capable of performing well on any given task.

A typical ANN is a composition of n functions arranged into layers:

$$f(\boldsymbol{x}) = f^n(f^{n-1}(\dots f^2(f^1(\boldsymbol{x}))\dots)).$$

The value of n in this example of a feedforward network is the depth of the model. The final layer f^n is called the output layer. If we want to allow the output to be any real value, we typically use an affine transformation:

$$f^n(\boldsymbol{h}^{n-1}; \boldsymbol{W}, \boldsymbol{b}) = \boldsymbol{W}\boldsymbol{h}^{n-1} + \boldsymbol{b}$$

where $\boldsymbol{W}$ is a learnable weight matrix and $\boldsymbol{b}$ is a learnable bias vector. The parameters transform the hidden representation vector produced by the previous layer, $\boldsymbol{h}^{n-1}$. To approximate a discrete output, we could modify this to:

$$f^n(\boldsymbol{h}^{n-1}; \boldsymbol{W}, \boldsymbol{b}) = \text{softmax}\left(\boldsymbol{W}\boldsymbol{h}^{n-1} + \boldsymbol{b}\right)$$

where the softmax function is applied element-wise to the unnormalised output of an affine transformation, $\boldsymbol{z}$:

$$\text{softmax}(\boldsymbol{z})_i = \frac{\exp(z_i)}{\sum_{j=1}^{C} \exp(z_j)}, \quad \forall i \in \{1, \ldots, C\}$$

where C is the number of discrete outputs.

All layers but the output layer in an ANN are termed hidden layers. This naming convention arises because the training data does not directly specify the outputs $\boldsymbol{h}$; instead, the network learns to determine these features through the training process. Additionally, from the perspective of the external environment, the functionality of the hidden layers is ‘hidden’ within the network’s mechanics. Data is provided to the input neurons and results are extracted from the output neurons. The hidden units are not directly interacted with. A popular hidden layer configuration consists of an affine transformation followed by a nonlinear activation function, as described by the equation:

$$\boldsymbol{h} = g(\boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{c}) \tag{2.15}$$

where $\boldsymbol{W}$ and $\boldsymbol{c}$ are learnable parameters that we aim to optimise such as to minimise some objective function, and $\boldsymbol{x}$ could be the input data to the network or a hidden representation produced by a previous layer. Learnable parameters in a network are often referred to as network weights due to their role in weighting the inputs to a neuron.

Activation functions, g in Equation 2.15, introduce non-linearity into neural networks, enabling them to model complex functions beyond what linear transformations allow. Without activation functions, a deep neural network would be mathematically equivalent to a single-layer perceptron, regardless of depth. The default nonlinear function is element-wise application of the rectified linear unit (ReLU), where $g(z) = \max\{0, z\}$. ReLU units behave linearly everywhere they are active, ensuring that they are relatively simple to optimise. Since the second derivative of ReLU is zero across its domain, except at zero where it is undefined, there are no complex second-order effects to complicate the optimisation process within the parameter space. ReLU units have a first derivative equal to zero for half of their domain when $z_i = \boldsymbol{w}_i\boldsymbol{x} + b_i < 0$. In this state, the unit is inactive and gradient-based optimisation methods will not modify $\boldsymbol{w}_i$ and b_i. Typically, all elements of $\boldsymbol{b}$ are set to a small positive value to make it likely that the ReLU units will be initially active for some inputs. This allows the optimisation process to selectively deactivate only units that are unnecessary. If too many units become inactive during training, this can lead to poor performance due to insufficient functional capacity of the network. Additionally, if no input to the neural network results in a positive value, then the unit cannot learn to become active again even if this would be beneficial, and the unit is considered dead. The Leaky ReLU activation function addresses this problem by formulating the rectification as $g(z) = \max\{0, z\} + a\min\{0, z\}$, where a is fixed to a small value such as $1e^{-2}$. This results in the linear units receiving nonzero gradient everywhere.

Older neural networks often used logistic sigmoid activation functions, but these are generally avoided because they have a tendency to saturate across most of their domain.[78] There are many other variants of ReLU, as well as other distinct classes of activation functions that are not covered here. The types, rationale, and theoretical foundations of various classes of activation functions are covered extensively elsewhere[76, 79]. There are also alternatives to the affine transformation used above, such as convolutions, which can be used to introduce different properties to the layers. Convolutional networks will be discussed in detail later in Section 2.7.2.

A Multilayer Perceptron (MLP) with a linear output layer and at least one hidden layer with an appropriate activation function can approximate any continuous function to any arbitrary amount of error with a finite number of neurons. This capability is established by the universal approximation theorem. The number of neurons, while finite, may still be inpractically large, and the optimisation algorithm might be inefficient at searching the hypothesis space. Deeper networks offer a potential solution that has been empirically shown to perform better with fewer required neurons on a wider variety of tasks and also satisfy the universal approximation theorem[80]. This is likely because deeper networks implicitly encode the prior that complex, abstract features can be extracted through a hierarchical composition of many simpler features.

2.3.5 Back-propagation

Neural networks are trained using gradient-based optimisation techniques (see Section 2.4). These approaches require the availability of gradients throughout the network. Forward propagation describes the process by which the input $\boldsymbol{x}$ provides information to the first layer, which then propagates through the hidden units to give us the final output $\hat{\boldsymbol{y}} = f(\boldsymbol{x}, \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is the learnable parameters of the network. A loss function such as mean squared error:

$$J(\boldsymbol{\theta}) = \mathcal{L}_{\text{MSE}}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{m}\sum_i (\hat{\boldsymbol{y}} - \boldsymbol{y})_i^2 \tag{2.16}$$

gives us a scalar cost $J(\boldsymbol{\theta})$ of the output with respect to some ground truth $\boldsymbol{y}$. Minimising Equation 2.16 is equivalent to maximising the maximum likelihood estimator if the output is assumed to be normally distributed as $p(y|\boldsymbol{x}) = \mathcal{N}(y; f(\boldsymbol{x}, \boldsymbol{\theta}), \sigma^2 = 1)$. The back-propagation algorithm enables the propagation of information about the cost backwards through the network, facilitating the computation of the gradient of the cost function with respect to the parameters $\nabla_\theta J(\theta)$.

The back-propagation algorithm is the recursive application of the chain rule of calculus. For functions $u : \mathbb{R}^p \to \mathbb{R}$ and $v : \mathbb{R}^q \to \mathbb{R}^p$ we can write $m = u(\boldsymbol{n})$ and $\boldsymbol{n} = v(\boldsymbol{o})$, with $m \in \mathbb{R}$, $\boldsymbol{n} \in \mathbb{R}^p$, $\boldsymbol{o} \in \mathbb{R}^q$. Then the chain rule of calculus would be:

$$\nabla_{\boldsymbol{o}} m = \left(\frac{\partial \boldsymbol{n}}{\partial \boldsymbol{o}}\right)^\top \nabla_{\boldsymbol{n}} m$$

where $\frac{\partial \boldsymbol{n}}{\partial \boldsymbol{o}}$ is the $p \times q$ Jacobian matrix of v. For a hidden layer k in an MLP defined by Equation 2.15, we can split the forward propagation into:

$$\boldsymbol{a}^k = \boldsymbol{b}^k + \boldsymbol{W}^k \boldsymbol{h}^{k-1}$$

$$\boldsymbol{h}^k = g(\boldsymbol{a}^k)$$

where $\boldsymbol{h}^{k-1}$ is the output of the previous hidden layer, $\boldsymbol{h}^k$ will subsequently be utilised as the input to the next layer. During backpropagation, we will receive from layer $k+1$ the gradients with respect to $\boldsymbol{h}^k$, $\nabla_{\boldsymbol{h}^k} J(\boldsymbol{\theta})$. Assuming g is applied through element-wise multiplication, we can define the gradients with respect to the previous layer and the gradients with respect to the parameters:

$$\nabla_{\boldsymbol{a}^k} J(\boldsymbol{\theta}) = \nabla_{\boldsymbol{h}^k} J(\boldsymbol{\theta}) \odot g'(\boldsymbol{a}^k)$$

$$\nabla_{\boldsymbol{b}^k} J(\boldsymbol{\theta}) = \nabla_{\boldsymbol{a}^k} J(\boldsymbol{\theta})$$

$$\nabla_{\boldsymbol{W}^k} J(\boldsymbol{\theta}) = \nabla_{\boldsymbol{a}^k} J(\boldsymbol{\theta}) \boldsymbol{h}^{k-1\top}$$

$$\nabla_{\boldsymbol{h}^{k-1}} J(\boldsymbol{\theta}) = \boldsymbol{W}^{k\top} \nabla_{\boldsymbol{a}^k} J(\boldsymbol{\theta}).$$

2.4 Optimisation

Artificial Neural Networks are used to solve tasks that traditional machine learning techniques often struggle with. These tasks typically involve high-dimensional inputs and complex relationships, necessitating large models with many learnable parameters. Optimising the performance of these networks with respect to a pre-defined loss function requires navigating a vast and intricate parameter space. This optimiszation process is inherently non-convex, characterised by an extremely rough loss surface with high Lipschitz constants, meaning that small changes in input can lead to large changes in output. Consequently, the loss landscape is steep and abrupt, with numerous pronounced local minima that make finding a global optimum challenging. The following sections outline some of the most popular approaches for effectively searching this parameter space.

2.4.1 Stochastic gradient descent

We are typically working with a set of training data that represents an empirical data distribution $\hat{p}_{\text{data}}(\boldsymbol{x}, y)$ which is drawn from some underlying data-generating distribution $p_{\text{data}}(\boldsymbol{x}, y)$. This scenario is for the supervised case but can be generalised to the unsupervised case. We can compute the gradient of the loss with respect to the parameters with:

$$\hat{\boldsymbol{g}} = \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_{i} \mathcal{L}_{\text{MSE}}(f(\boldsymbol{x}^i, \boldsymbol{\theta}), y^i) \tag{2.17}$$

and update $\boldsymbol{\theta}$ in the direction of $\hat{\boldsymbol{g}}$ to minimise the loss using the update rule:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \hat{\boldsymbol{g}} \tag{2.18}$$

where ϵ is the learning rate. If we go through multiple iterations of this process, with an appropriate value of ϵ, this algorithm will follow the gradients downhill towards a minimum of the loss function. This algorithm is known as batch gradient descent. Calculating $\hat{\boldsymbol{g}}$ with respect to the whole of our training data can be computationally expensive, especially with large datasets. We could instead perform Stochastic Gradient Descent where $\boldsymbol{\theta}$ is updated with an estimate of the gradient calculated from a single randomly sampled example in each iteration. However, it is common practice to use stochastic minibatch gradient descent. This approach uses a random set of examples, often between 32 to 256 in number, to calculate an estimate of the gradients each iteration. This approach represents a compromise between the accuracy of the estimate of the gradient and the efficiency with which we can explore the parameter space. In fact, there is empirical evidence showing that when using batch sizes over 512, there is a degradation in the generalisability of the model[81]. The noise introduced by the small batch size is held to have a regularising effect on the model. We discuss the concepts of generalisation and regularisation later in Sections 2.5 to 2.7. The performance of any particular batch size, with respect to the use of computation resources and the ML task itself, must be empirically determined. The choice of batch size is highly problem-dependent, to my knowledge there is no way to figure out a priori what batch size to use. Given the ubiquity of minibatch gradient descent, the term 'SGD' frequently denotes minibatch gradient descent rather than true single-example stochastic gradient descent; this convention will be adhered to throughout this work.

Even using the optimal batchsize, SGD can still exhibit slow convergence. To accelerate training, we can introduce the concept of velocity v, which represents both the direction and speed at which parameters are moving through parameter space.[82] If we assume unit 'mass' of the parameters, the velocity becomes equivalent to the momentum, from which we derive the name of this technique. The update rule in Equation 2.18 can be modified to:

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \hat{\boldsymbol{g}}$$
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}.$$

Here, the hyperparameter α ensures the contributions of previous gradients to v exponentially decay when $\alpha < 1$, and, along with ϵ, defines the maximum velocity for a given magnitude of $\boldsymbol{g}$:

$$\frac{\epsilon \|\boldsymbol{g}\|}{1 - \alpha}$$

If the gradient is instead evaluated after applying the velocity, we get the Nesterov momentum which converges quicker for batch gradient descent but has little impact on SGD.[83]

SGD with momentum is especially effective in navigating regions of parameter space that are particularly flat, noisy, as well as landscapes that are particularly steep and thus require a very small ϵ. Additionally, momentum also makes escaping local minima more efficient in the case of the non-convex optimisation typical to NN tasks.

2.4.2 Adaptive learning rates

The choice of learning rate strongly influences the performance of a model. A learning rate that is too small will inefficiently sample the parameter space and may be unable to overcome barriers around local minima on a reasonable timescale. Conversely, a large learning rate increases the risk of exploding gradients, especially in cases where the parameter landscape contains steep regions. Furthermore, the random sampling of m training examples introduces noise into the gradient estimation that does not vanish as we approach a minimum. Therefore, it is necessary to use a sufficiently small learning rate so that this noise does not lead to divergence from the desired stable solution. Moreover, we need to decay the learning rate as we approach the minimum in order to converge.

We can adjust the learning rate automatically for each parameter by scaling the learning rate based on historical gradients. The AdaGrad optimisation algorithm does this with the inverse of the sum of the square of previous gradients:

$$\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$$

where $\odot$ represents the Hadamard product. However, in practice, this often results in a premature decrease in the effective learning rate, and AdaGrad tends to perform particularly poorly in nonconvex scenarios.[84] RMSProp attempts to overcome AdaGrad's limitations by exponentially decaying the influence of historical gradient contributions.

$$\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1-\rho)\boldsymbol{g} \odot \boldsymbol{g}$$

This introduces a new hyperparameter ρ that controls the rate at which historical contributions to $\boldsymbol{r}$ decay. Adam[85] can be viewed as an extension of RMSProp that incorporates momentum by tracking an estimate of the first-order momentum (the mean), $\boldsymbol{s}_t$, in addition to the second-order moment (the uncentered variance), $\boldsymbol{r}_t$, of the gradient. These estimates are calculated for time step t as:

$$\boldsymbol{s}_t \leftarrow \rho_1 \boldsymbol{s}_{t-1} + (1-\rho_1)\boldsymbol{g}_t$$

$$\boldsymbol{r}_t \leftarrow \rho_2 \boldsymbol{r}_{t-1} + (1-\rho_2)\boldsymbol{g}_t \odot \boldsymbol{g}_t.$$

These moving averages are biased towards zero at the beginning of training, but we can produce an unbiased estimate with:

$$\hat{\boldsymbol{s}}_t = \frac{\boldsymbol{s}_t}{1-\rho_1^t}$$

$$\hat{\boldsymbol{r}}_t = \frac{\boldsymbol{r}_t}{1-\rho_2^t}$$

that can then be used to update the parameters:

$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \epsilon \frac{\hat{\boldsymbol{s}}_t}{\delta + \sqrt{\hat{\boldsymbol{r}}_t}}.$$

Adam is noted for its robustness to the choice of hyperparameters and is highly effective on a wide range of tasks.[85] There are many variants based on Adam and SGD that have a variety of different properties. Benchmarks of different optimisers on a variety of tasks find that trying a different optimiser with default hyperparameters can sometimes be as effective as attempting to tune the hyperparameters of any one optimiser.[86] Ultimately, the 'best' optimiser for a particular job is dependent on both the nature of the task and how much computational budget we are willing to make available to search the hyperparameter space and the parameter space.

2.4.3 Second order methods

All of the gradient descent-based algorithms discussed previously can struggle to converge when the optimisation objective exhibits pathological curvature. Second-order gradient methods offer a potential solution in these cases, as they can account for local curvature.[87] Second-order gradient methods, such as Newton's method, require the calculation of the Hessian of the cost function J with respect to the parameters $\boldsymbol{\theta}$. If the model contains k parameters, the Hessian would be composed of $k \times k$ elements. The computational complexity of calculating the inverse of the Hessian is $O(k^3)$. This is not feasible for all but the smallest neural networks.

There are alternatives to Newton's method that make various trade-offs between memory, compute, and accuracy. The Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm is a quasi-Newton method that approximates the inverse Hessian using iterative updates derived from historical gradients. Limited Memory BFGS avoids even having to store the full inverse Hessian ($O(k^2)$ memory) by storing only a few vectors that implicitly approximate the Hessian.

Empirically, it has been found that choosing a model family that is easy to optimise is more effective than using a powerful optimisation algorithm. Adam and SGD-based algorithms still account for the majority of optimisers used to train state-of-the-art neural networks.[86]

2.5 Testing

We have so far described training ML models with an empirical data distribution $\hat{p}_{\text{data}}$, which is drawn from the true data distribution p_{data}. However, we are not usually interested in the performance of the model with respect to $\hat{p}_{\text{data}}$, but with respect to p_{data}. In other words, we primarily care about the performance of the algorithm on previously unseen data that is independent and identically distributed to the training data. The model's performance on this unseen data is referred to as the generalisation error. We can evaluate a measure of the generalisation error by withholding a set of data from the training process and then evaluating the performance on this withheld data. This withheld data is known as the test set. We can compare the generalisation error to the model's error with respect to the training set. There are two scenarios in which a model may exhibit a poor generalisation error: underfitting and overfitting. Underfitting describes the scenario in which the performance on both the training data and the test set is poor. This can occur when the hypothesis space — the set of all functions that a given learning algorithm can select from within a network's parameter space — is insufficient to approximate the true function we are trying to learn. Alternatively, we often find that the hypothesis space contains a good solution but the optimisation procedure is unable to discover it. We refer to the combination of a model's hypothesis space and the ability of the learning algorithm to search this space as the model's capacity. We can straightforwardly increase the capacity of a neural network by increasing the number and size of hidden layers. While we might expect increasing the capacity of the model to improve the performance with respect to the training data, there is no guarantee that the generalisation error will similarly improve. In fact, we often find that simply increasing the capacity can make the generalisation error worse. This scenario, good performance with respect to the training data but poor performance with respect to the test set, is referred to as overfitting. Overfitting arises when a model learns to recognise noise and random fluctuations specific to the training data, rather than discerning just the structure and underlying patterns of the true data distribution. Consequently, the model is often described as having memorised the training data rather than learning to generalise from it.

2.6 No free lunch

The No Free Lunch Theorem (NFLT)[88] tells us that no ML algorithm can generalise well from a finite number of training examples when averaged over all possible data-generating distributions. This means, in effect, no single algorithm is guaranteed to generalise any better than simply outputting the same answer or random noise for every input. Counter-intuitively, a hill-climbing algorithm is just as effective as a hill-descending algorithm for finding a minimum when averaged over all possible tasks.

We overcome NFLT by recognising that we are not attempting to find a single method that will solve every problem simultaneously. Firstly, we only want to solve 'useful problems'. The majority of the problem space that we must average over to arrive at NFLT involves mapping what would appear to us as noise to noise, problems that cannot be described as useful. The vast majority of real-world problems possess predictable structures, and we expect large groups of problems to share these structures. As a result, these problems should be efficiently solved by a common algorithm.

Secondly, deep learning encompasses a collection of methods, each tailored to different classes of problems, often with the only meaningful common feature being a high-dimensional parameter space. The NFLT simply

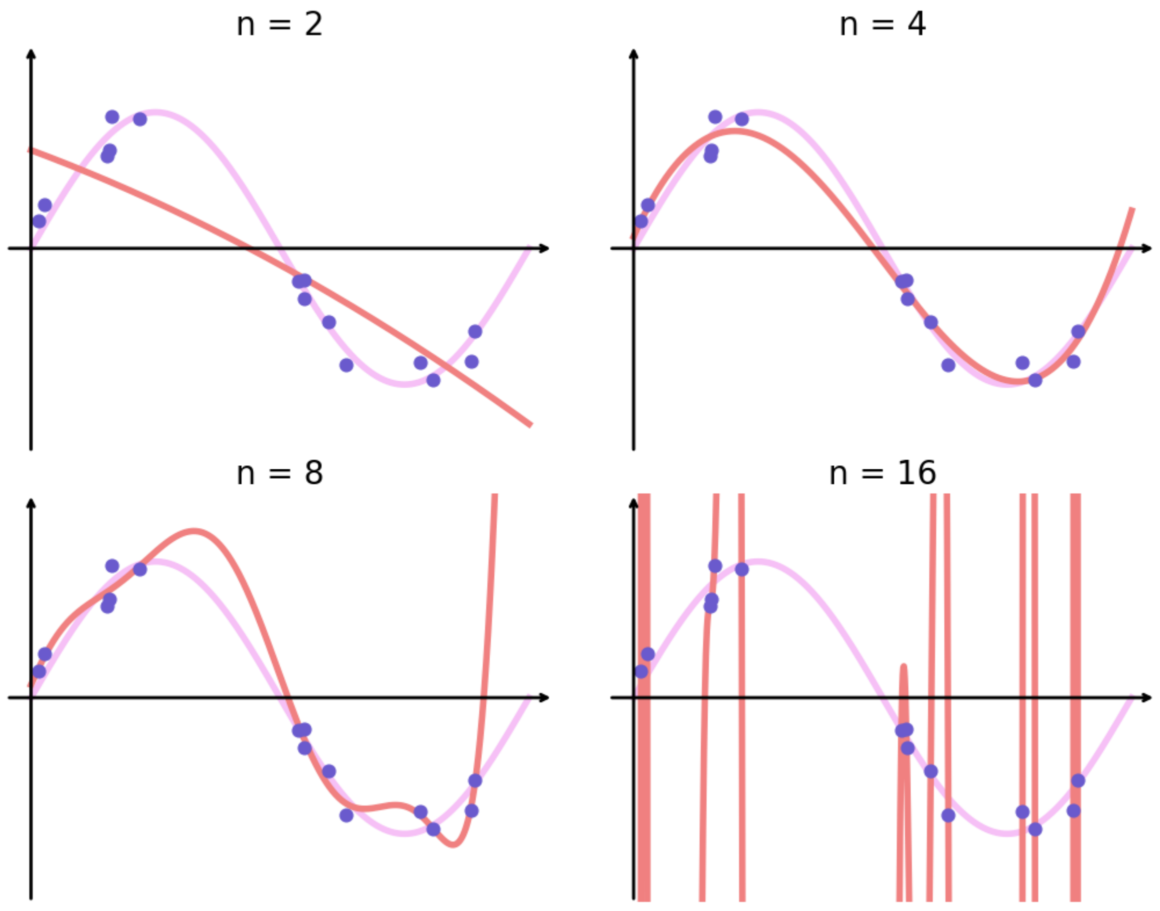


Figure 2.2: Fitting order n = 2,4,8,16 polynomials (red) to noisy samples (blue) drawn from some underlying function (pink). Low order polynomials have insufficient capacity to fit to the data, a sufficiently high order polynomial will fit the data perfectly but by also memorising the noise will fail to learn an approximation of the underlying function.

highlights that each algorithm's strengths and weaknesses depend on the particular problem. Therefore, we should use our knowledge of the structure and nature of a problem to select the algorithm most suited to the task.

2.7 Regularisation

Regularisation describes a number of strategies that modify a learning algorithm with the intention to reduce generalisation error, often even at the expense of increased training error. These strategies allow us to encode specific kinds of prior knowledge and control the capacity of a model. We also know from the NFLT that the effectiveness of any particular regularisation strategy is task-dependent. Most regularisation strategies are controlled by their own set of parameters, which govern the learning behaviour of the model and typically cannot be learned through gradient-based methods. To prevent overfitting the hyperparameters with respect to the test set, we optimise these hyperparameters using an additional independent set of data, referred to as the validation set. We evaluate the model's performance with respect to the test set only after determining fixed values for the hyperparameters.

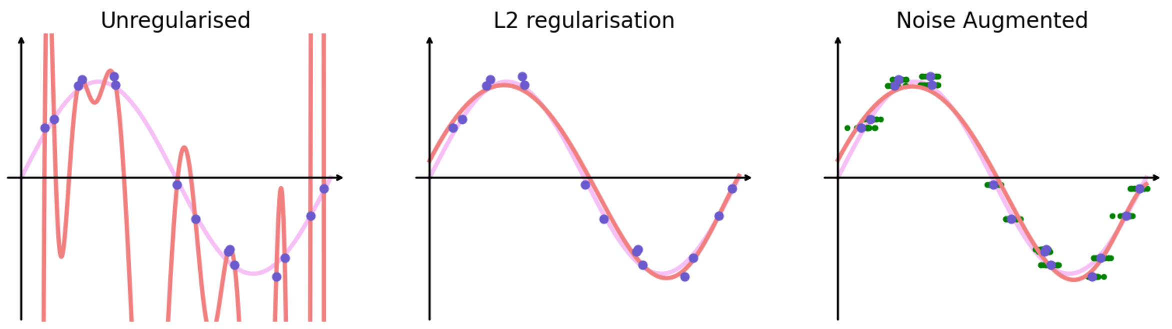


Figure 2.3: Fitting order 16 polynomials (red) to noisy random samples (Blue) drawn from some underlying function (pink). Left: Unregularised Least squares regression resulting in a strongly overfit polynomial. Middle: Least squares regression conditioned with a small L2 penalty on the exact same data. Right: Least squares regression on multiple duplicates (green) of the samples each with the addition of a small amount of Gaussian noise to the input.

Penalties

One of the most common ways of regularising a model is to modify the loss function $L(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y})$ by adding additional terms Ω that have a regularising effect:

$$L' = L(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \lambda\Omega(\boldsymbol{\theta})$$

where λ is a new hyperparameter controlling the strength of the regularising effect. We can then train the model with respect to the regularised loss L'. If the regularising term is the L^2 parameter norm, $\Omega(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2$, this has the effect of expressing a preference for solutions with small weights.[89] Small weights have a number of desirable properties. Firstly, small weights ensure that the resulting model is smooth. If a model is overparameterised, there will be an infinitely large set of solutions that achieve a very low training error, most of which will be overfitted. Occam's razor gives us advice on how to select a solution from this space: choose the simplest solution, which in the context of deep learning translates to the smoothest function. In particular, smooth decision boundaries are less sensitive to small variations in input data. Secondly, small weights ensure that no single weight can propagate a signal independently from one layer to the next. The signal is more likely to be distributed among many neurons. This results in neurons that are far more robust to adverse signals in any particular neuron, and this should ensure the model generalises more successfully.[89] Thirdly, networks with large weights are more prone to issues like exploding gradients. Encouraging a network to only explore solutions with small weights will improve the stability of the training procedure.[89] An example of the effect of L^2 regularisation can be found in Figure 2.3.

Other regularising terms can have different effects. The L^1 norm, where $\Omega(\boldsymbol{w}) = \|\boldsymbol{w}\|_1$, not only drives weights to be small but also to zero. This results in a scenario where all but the most important weights are inactive, and we get a solution that is sparse. This can be used, for example, to simplify a machine learning problem by selecting a subset of input features. This is referred to as feature selection.

Reprojection

An alternative to using a penalty term is to explicitly constrain the weights of the network by projecting the parameters $\boldsymbol{\theta}$ to the nearest point that satisfies a specific constraint after every gradient descent update. For example, spectral normalization [90] normalises each layer of a network to have a Lipschitz constant equal to a predetermined hyperparameter, typically set to 1.0. This has proven to be highly effective in stabilising the training of certain classes of generative networks.

2.7.1 Dataset Augmentation

The simplest way to improve the generalisation of a neural network is to use a larger set of data for training. Dataset augmentation is an approach that attempts to mimic this effect by generating synthetic data. One way to achieve this is by recognising that many types of data have inherent symmetries. For instance, a picture of a dog remains recognisably as such, regardless of a number of possible transformations that we could apply to it, including rotation, translation (assuming the dog remains within frame), mirroring, and scaling. We could take a single picture of a dog and sample the space of images that these transformations provide. A neural network trained to associate all appropriate transformations of a picture with the same class or same representation would learn to be invariant to these transformations. By doing this, we are injecting a priori knowledge of invariance to these specific transformations. This a priori information must be correct, for example, the digit '6' rotated 180 degrees does not preserve the identity of its class but instead becomes a '9'. Many biological systems, such as proteins, exhibit enantiomerism and are therefore not invariant to mirroring.

We can attempt to reduce a model's sensitivity to small variations in features by injecting small amounts of noise into the training set. Continuing the analogy above, we know that a slightly noisy image of a dog is still recognisably an image of a dog. Memorising noise in the datasets is a major cause of poor generalisation. If we can mimic the variability real-world sources of noise introduce, we can encourage invariance to this type of noise. This requires having a prior knowledge of the type and magnitude of noise present in the dataset. We have shown the effect of this form of regularisation in Figure 2.3.

Dropout

We previously discussed how L^2 parameter norm regularisation indirectly promotes signals being distributed across multiple neurons. If we add noise to the hidden representations $\boldsymbol{h}$ of each layer during training, we can achieve a similar effect. This can be viewed as dataset augmentation at every layer of the network and additionally promotes discovering hidden features that are more robust. The most successful form of this noise is dropout.[91] Dropout applies a random binary mask to all inputs and hidden layers, randomly setting a fraction, defined by a hyperparameter, of the output units to zero. This means the model cannot depend on the result of any single input. Much like the L^2 parameter norm, dropout forces decision making to be distributed across multiple neurons. Dropout is inexpensive and broadly applicable. It is, however, less effective with small networks for which the deactivation of a significant proportion of neurons would push their capacities into an underfitting regime. Dropout is also less effective on the input layer where the number of features is small andon small datasets where the integrity of the whole input is important. As not all the weights are exposed to gradients at every step, using dropout will require more iterations of training in order to converge.[76, 91]

2.7.2 Convolutional Networks

We can use the architecture of the network itself to impose a priori knowledge. A discrete convolutional operation, denoted by $*$, is the sum of the Hadamard product between a sliding window over the input $\boldsymbol{x}$ and a kernel $\boldsymbol{w}$. A 1-dimensional discrete convolution $\boldsymbol{x} * \boldsymbol{w}$ is given by:

$$(\boldsymbol{x} * \boldsymbol{w})_i = \sum_m \boldsymbol{x}_m \boldsymbol{w}_{i-m}$$

for values of m within the bounds of $\boldsymbol{x}$ and $\boldsymbol{w}$. However, the convolution is usually implemented as the cross-correlation:

$$(\boldsymbol{x} * \boldsymbol{w})_i = \sum_m \boldsymbol{x}_{i+m} \boldsymbol{w}_m.$$

Kernel sizes are kept small, typically between 3-7 elements long. The output of the kernel operation is referred to as a feature map. We often stack many kernels in parallel to give us many feature maps in a multichannel output. This allows us to learn multiple features in parallel. We can also extend this to allow multichannel inputs by defining a kernel for each input channel and then combining the results to capture interactions between channels. We refer to this as a multichannel convolution. There are similar formulations for any arbitrary dimensional data; however, large and high-dimensional dense tensors quickly become infeasible from a memory perspective. 1-D, 2-D, and 3-D multichannel convolutions are defined in most machine learning libraries.

$$\begin{bmatrix}1&0&0&0&0&0&0&0\\2&3&0&0&0&0&0&0\\4&5&6&0&0&0&0&0\\0&7&8&9&0&0&0&0\\0&0&10&11&12&0&0&0\\0&0&0&13&14&15&0&0\\0&0&0&0&16&17&18&0\\0&0&0&0&0&19&20&21\\0&0&0&0&0&0&22&23\\0&0&0&0&0&0&0&24\end{bmatrix}$$

(a) A locally connected fully connected weight matrix

$$\begin{bmatrix}1&2&3&4&5&6&7&8\\1&2&3&4&5&6&7&8\\1&2&3&4&5&6&7&8\\1&2&3&4&5&6&7&8\\1&2&3&4&5&6&7&8\\1&2&3&4&5&6&7&8\\1&2&3&4&5&6&7&8\\1&2&3&4&5&6&7&8\\1&2&3&4&5&6&7&8\\1&2&3&4&5&6&7&8\end{bmatrix}$$

(b) A fully connected weight matrix with row weight sharing

$$\begin{bmatrix}3&0&0&0&0&0&0&0\\2&3&0&0&0&0&0&0\\1&2&3&0&0&0&0&0\\0&1&2&3&0&0&0&0\\0&0&1&2&3&0&0&0\\0&0&0&1&2&3&0&0\\0&0&0&0&1&2&3&0\\0&0&0&0&0&1&2&3\\0&0&0&0&0&0&1&2\\0&0&0&0&0&0&0&1\end{bmatrix}$$

(c) Translational equivariance arising by off-setting weight sharing.

Figure 2.4: The Convolutional concepts of Local connectivity 2.4a, weight sharing 2.4b, and translational equivariance 2.4c written out as their fully connected analogues.

Convolutional layers take advantage of three concepts: Local connectivity, weight sharing, and translational equivariance.

Local connectivity is the idea that features have strong spatial dependence, in that spatially adjacent features are highly correlated. In an image, adjacent pixels are highly likely to belong to the same object or otherwise determine the boundary between two objects. For this reason, we can use small kernel sizes with respect to the size of the input. This can be imagined as a very sparse fully connected layer with all weights but those close to the diagonal being zero. Convolutions, therefore, require far fewer parameters and computing the output requires far fewer operations.

The weights of an edge detector looking at any 3×3 region of an image will work just as well as an edge detector in any other 3×3 region of the image. Therefore, sharing weights between edge detectors operating in different regions is practical and efficient. The reuse of the same parameters for more than one function in a model is called parameter sharing. Convolutions take parameter sharing to an extreme by applying the same kernel weights $\boldsymbol{k}$ to calculate every feature in a particular feature map. This, in addition to sparse connectivity mentioned above, makes convolutions highly parameter and statistically efficient relative to fully connected layers.

The translational nature of the parameter sharing in convolutions leads to translational equivariance. Translationally transforming the input to a convolution equivalently translates the output. The output of a convolution, therefore, gives a map of the location of features, hence why it is referred to as a feature map. Additionally, convolutions can handle arbitrarily sized inputs, in contrast to fully connected networks where the dimension of the input must be compatible with the dimension of the weights.

With respect to regularisation, convolutions can be viewed as an infinitely strong prior on local connectivity. A single layer would then be unable to capture dependencies among features that are spatially separated by a distance greater than the kernel size. However, as we stack many layers of convolutions, the receptive field increases. A deep convolutional network then not only allows more abstract feature maps, but allows these features to depend on a larger field of inputs. This implies that the network should be sufficiently deep as to ensure the receptive field encompasses the entirety of the input and therefore enable the discovery of global dependencies.

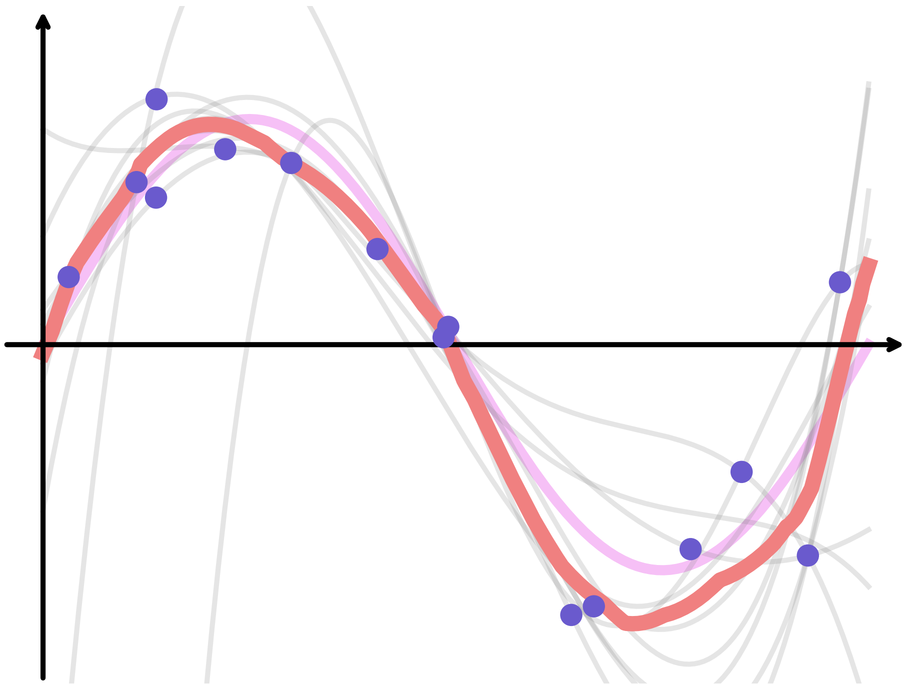


Figure 2.5: Over fitting multiple polynomials (grey) to random subsets of noisy samples (blue) drawn from some underlying function (pink). The ensemble average (median) of multiple polynomial model producing an estimate closer to the ground truth than any single polynomial model due to the cancelling out of errors.

2.8 Ensemble methods

Ensemble regularisation methods are a form of regularisation in which we combine the outputs from a diverse set of models. To generate diverse models, we could design models with different architectures, train models in different ways, or train with different data. Even the random initialisation of model parameters before training can often result in a set of parameters discovered by the optimiser that is sufficiently diverse to have a regularising effect. The regularising effect arises because the errors on each example are often not perfectly correlated. Therefore, if we average the results among an ensemble, these errors can cancel out, revealing an estimate closer to the ground truth. From another perspective, we can describe a single model as capturing a single mode in the solution space. Each mode in the solution space discovers a different set of patterns and relationships in the data. Ensemble methods enable us to capture insights from multiple modes in a single output. Ensembles typically perform at least as well as any of their members, and when errors are truly independent, the ensemble can perform better than any of its individual members. Dropout, as discussed above, can be viewed as an implicit ensemble method where each model is defined by a subset of features remaining after masking. The sharing of parameters between models means that this method is far more parameter-efficient than training separate models, though the models are not fully independent.

Ensemble methods are so successful and reliable at improving generalisation error that their use is discouraged in benchmarking any particular method in scientific contexts.[76] When implemented as standalone models, the contribution and effects of new methods are more clearly isolated, making the evaluation of their performance more straightforward.

2.9 Autoencoders

Autoencoders [92, 93] consist of a combination of two models, an encoder $g : \mathbb{R}^m \rightarrow \mathbb{R}^n$ and a decoder $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$, such that $f(g(\boldsymbol{x})) \approx \boldsymbol{x}$. The encoder defines a mapping from the input space to some latent representation space $\boldsymbol{h} = g(\boldsymbol{x})$ and the decoder attempts to reconstruct something resembling the input $\boldsymbol{x}' = f(\boldsymbol{h})$. If the dimensionality of the latent space is the same as the input space ($n = m$), the autoencoder could trivially learn two identity functions such that $\boldsymbol{x} = \boldsymbol{h} = \boldsymbol{x}'$. By learning the identity function, an autoencoder is unlikely to have learnt anything useful from the training data, so we typically employ some strategy that forces the autoencoder to learn something about the underlying structure of the data. Autoencoders have a wide range of use cases. The combined encoder and decoder can be utilised for dimensionality reduction and lossy data compression. The projection of input data into the latent space can provide insights into the relationships between data points. The encoder can also be leveraged in downstream tasks, serving as part of unsupervised pre-training or in semi-supervised classification. Lastly, new points can be sampled in the latent space, with the decoder used for generative modeling.

2.9.1 Undercomplete Autoencoders

The oldest and simplest strategy to force an autoencoder to learn salient information from the training data is to constrain the dimensionality of the latent space (n) to be less than that of the input space (m). An autoencoder where $n < m$ is referred to as undercomplete. We commonly train autoencoders by minimising the mean squared error L_{MSE} between the input and output of the network for continuous data:

$$L_{\text{MSE}} = \mathbb{E}_{\boldsymbol{x} \sim p_d(\boldsymbol{x})} \left[\|(f(g(\boldsymbol{x})) - \boldsymbol{x}\|^2 \right] \tag{2.19}$$

The mean squared error criterion is equivalent to the negative log-likelihood when we use linear output units to parameterise the mean of a Gaussian distribution. If the hypothesis space of the encoder g and decoder f includes nonlinear functions, we can think of the autoencoder as learning a nonlinear generalisation of PCA. By optimising with respect to MSE, we implicitly specify that a feature is only considered salient if it causes significant changes to the data. The network will learn a latent space that maximises the explained variance of the dataset. If the capacities of the encoder and decoder are too high, the autoencoder could still theoretically learn to copy the training data while failing to learn a useful representations. An undercomplete autoencoder trained with MSE on conformations of a protein sampled from an MD simulation will tend to average out small, high frequency motions involving small numbers of atoms in favour of encoding large scale, highly correlated motions of entire domains.

2.9.2 Regularised Autoencoders

Instead of relying purely on the size of the latent space to force the autoencoder to learn salient features, we can instead employ regularisation strategies that guide the learning process towards solutions with desirable properties. Regularisation can allow the autoencoder to learn salient features even in the overcomplete case, where the latent space has a larger dimensionality than the data.

If we add an L^1 norm on the latent space features $\sum_i |g(\boldsymbol{x})|_i$ to the loss function or any other form of regularisation that encourages sparsity in the latent representation, we get sparse autoencoders. While undercomplete autoencoders have an explicit architectural bottleneck, sparsity can be viewed as an implicit bottleneck. The network is encouraged to learn efficient latent features. The degree of sparsity is controlled by how strongly we apply the regularisation and can be fine tuned throughout training.

Denoising Autoencoders, a type of regularised autoencoder, do not attempt to reconstruct their input but instead attempt to denoise their input. This is directly useful for data processing tasks where we want to clean noisy data, such as images, audio, and video. To do this, we are forcing the encoder and decoder to implicitly capture the underlying structure of the training data and therefore learn a robust representation of the underlying clean data. Models that learn to denoise are prolific in the ML field, and denoising autoencoders refer specifically to models where we are intending to learn a good internal representation as a side effect of learning to denoise. We can construct a denoising autoencoder by modifying the loss function $L(f(g(\boldsymbol{x})), \boldsymbol{x})$ to $L(f(g(\tilde{\boldsymbol{x}})), \boldsymbol{x})$, where $\tilde{\boldsymbol{x}}$ is generated by corrupting $\boldsymbol{x}$ in some noising process $C(\tilde{\boldsymbol{x}}|\boldsymbol{x})$. Common types of noising processes include the addition of Gaussian noise $\epsilon \sim \mathcal{N}(0, \sigma^2 I)$ or multiplying by a random mask, similar to dropout. The latter strategy is often done in the context of an infilling task where we are attempting to learn to reconstruct missing parts of an input. The addition of Gaussian noise has the effect of imparting a resistance to small perturbations of the input. Contractive autoencoders, where we explicitly penalise the derivative of the latent space with respect to the input, have a similar effect but applied only to the encoder. This penalty takes the form $\sum_i \|\nabla_{\boldsymbol{x}} g(\boldsymbol{x})\|_i^2$ and is typically too computationally expensive for anything other than very small networks. A more detailed look at the theory, implementation, and applications of autoencoders can be found elsewhere. [76, 94]

2.10 Sinkhorn Divergence

In an autoencoder, the latent space is simply an encoding designed to be decoded by the decoder. The structure of samples projected into this space will not always be meaningful. Additionally, generating samples from the latent space that are not associated with the training data can often produce incoherent samples, as the latent space in an autoencoder is not designed to facilitate sampling. If we choose to define a prior, an easily sampled latent space $z \sim p_z(z)$, we want to train a generator network $G : \mathbb{R}^m \to \mathbb{R}^n$ that captures the data distribution p_d. If the generated distribution p_G matches the data distribution p_d, then we will be able to generate new structures by sampling p_z. To train this network, we need to minimise $D(p_G, p_d)$, where D is some measure of the difference between the two distributions.

The Wasserstein-1 distance W is a potential candidate for (D) drawn from the optimal transport (OT) field. Wasserstein-1 is formally defined as:

$$W(p_d, p_G) = \min_{\pi \in \Pi(p_d, p_G)} \int_{\mathcal{X} \times \mathcal{Y}} \|\boldsymbol{x} - \boldsymbol{y}\| \mathrm{d}\pi(\boldsymbol{x}, \boldsymbol{y}) \tag{2.20}$$

where $\Pi(p_d, p_G)$ denotes the space of all joint distributions $\pi(\boldsymbol{x}, \boldsymbol{y})$ with marginals $p_d(\boldsymbol{x})$ and $p_G(\boldsymbol{y})$, and $\mathcal{X}$ and $\mathcal{Y}$ are the feature spaces associated with $p_d(\boldsymbol{x})$ and $p_G(\boldsymbol{y})$, respectively. The joint distribution $\pi(\boldsymbol{x}, \boldsymbol{y})$ over $\mathcal{X} \times \mathcal{Y}$ defines a transport plan, specifying how probability mass is moved from p_d to match p_G while minimising the transport cost.

Calculating W is challenging.[95] If we add an entropic regularising term:

$$W_\epsilon(p_d, p_G) = \min_{\pi \in \Pi(p_d, p_G)} \int_{\mathcal{X} \times \mathcal{Y}} \|\boldsymbol{x} - \boldsymbol{y}\| \mathrm{d}\pi(\boldsymbol{x}, \boldsymbol{y}) + \epsilon D_{\mathrm{KL}}(\pi \| p_d \otimes p_G) \tag{2.21}$$

where:

$$D_{\mathrm{KL}}(\pi \| p_d \otimes p_G) = \int_{\mathcal{X} \times \mathcal{Y}} \log \left(\frac{\pi(\boldsymbol{x}, \boldsymbol{y})}{p_d(\boldsymbol{x}) p_G(\boldsymbol{y})} \right) \mathrm{d}\pi(\boldsymbol{x}, \boldsymbol{y}) \tag{2.22}$$

we can make an approximate solution to the OT problem that is differentiable and of a lower computational complexity ($O(n^2)$ rather than $O(n^3)$.[95]. Here, ϵ is a parameter that controls the strength of the regularising term D_{KL}, the Kullback-Leibler divergence. However, D_{KL} introduces an entropic barrier that blurs the transport plan and introduces a bias that results in a number of undesirable properties, such as not being able to satisfy the triangle inequality and thus not being a distance. The Sinkhorn divergence reformulates W_ϵ to address this bias:

$$S_\epsilon(p_d, p_G) = W_\epsilon(p_d, p_G) - \frac{1}{2} W_\epsilon(p_d, p_d) - \frac{1}{2} W_\epsilon(p_G, p_G) \tag{2.23}$$

Feydy et al. provide an efficient GPU based Sinkhorn solver for PyTorch with their GeomLoss library. [95] They also provide more detailed discussion, including derivations and various proofs.

2.11 Conclusions

In this chapter, we have introduced the key concepts of MD simulations and ML underpinning this work. MD is a mature field that has evolved over decades to obtain its status as a cornerstone in the study of biomolecular systems. Although significant advances in enhanced sampling have been made in the last 20 years, the surge in accessible simulation sizes and timescales over the last decade has been primarily due to improvements in computational hardware rather than developments in the methods themselves. The methods outlined here will likely remain relevant for some time. On the other hand, the ML field has progressed rapidly in the last decade, driven not only by increases in available computational power but also by advances in algorithmic design and greater data availability. The progression of ML is such that many of the novel technologies used in this work have already been surpassed by new methods, while this document was being written.

Chapter 3

Deep learning protein conformational spaces with convolutions and latent interpolations

The content of this chapter has been published in: V. K. Ramaswamy, S. C. Musson, C. G. Willcocks, and M. T. Degiacomi. "Deep Learning Protein Conformational Space with Convolutions and Latent Interpolations". *Phys. Rev. X*, 11, 011052 (2021). DOI: 10.1103/PhysRevX.11.011052

3.1 Introduction

Deep neural networks are able to learn continuous representations that capture the structure of a dataset. In particular, generative models (such as variational autoencoders [96] (see Section 2.9), generative adversarial networks [97] or Boltzmann generators [98] have been showing a remarkable ability to synthesise complex and sparse datasets. Generative neural networks create an internal model recapitulating example data, a model that can then be interrogated to generate new, plausible data samples. While most generative models produce new samples from an assumed prior distribution, recent architectures improve interpolations through additional adversarial components [99]. Many successful generative architectures utilise convolutional neural networks (CNNs) [74]. See Section 2.7.2 for a detailed discussion on convolutions. Though a surfeit of applications of CNNs are in the fields of image, video, audio, and speech recognition [74], variants have also been applied to bioinformatics ranging from gene expression regulation [100–102], anomaly classification [103–105], to prediction of protein secondary structure [106, 107], and protein folds [108–111].

We present a 1D CNN architecture (Figure 3.1A), directly trainable with protein structures to build a model of their underlying conformational space. To improve the generalisation capability of our network, we design a new loss function that leverages on, as a prior, knowledge of physical laws dictating atomic interactions. We enforce this physics-based loss function on the manifold between two protein conformations but outside of the known sampled conformational space. Our architecture and training approach lead to several significant advantages over conventional networks taking atomic coordinates [46] or molecular features as input. First, being fully convolutional, our architecture features a small number of parameters and is therefore easy to train. Second, it can handle input molecules with arbitrary numbers of atoms, enabling network training with different molecules, either simultaneously or via transfer learning. Third, it does not make assumptions on the distribution of data around observations used for training. We show that these features can enable the identification of biologically relevant intermediate protein configurations along plausible transition paths between known low-energy states.

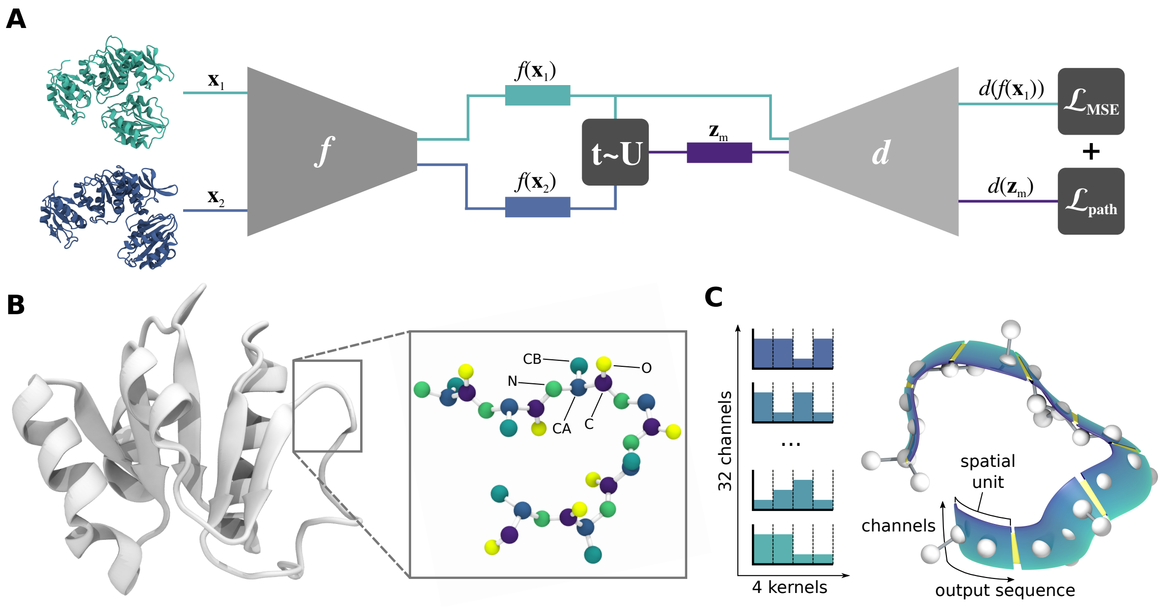


Figure 3.1: Neural network design. (A) The generative architecture is composed of an encoder f and a decoder d, and is trained with a collection of protein conformations. The loss function couples a geometric term $\mathcal{L}_{\text{MSE}}$ to ensure original and encoded-decoded structure are similar, and physics-based terms $\mathcal{L}_{\text{path}}$ to ensure that latent space interpolations z_m between any pair of conformations produce protein structures of low energy. (B) Protein atoms can be sorted in a list so that atoms that are adjacent in the list are also adjacent in the Cartesian space. The convolutional neural network operates on this list, that can be of arbitrary size. (C) The first 1D convolution layer learns $32\times$ feature detectors, each with a kernel size of 4. The stride is set to 2, therefore the output sequence is half the input size for any input length. Each subsequent layer further reduces the spatial length of the molecule, warping the input such that it becomes progressively deeper and thicker (more ribbon-like) as well as more abstract.

3.2 Methods

3.2.1 Network architecture

Proteins are defined by their amino acid sequence, and each sequence maps onto an ensemble of possible three-dimensional atomic arrangements (conformations). The space of possible conformations associated with a specific sequence may be extremely reduced for a protein taking a single well-defined state, or broad for a flexible protein capable of interconverting between multiple states. As the proteome is vast and many proteins are resistant to most forms of experimental interrogation, only a relatively subset of proteins have had at least one of their possible conformations revealed at atomic resolution. Furthermore, as the techniques characterising molecular structures typically report on low energy conformations, transition states are undersampled proportionally to their associated energy barrier.

From a machine learning perspective, we can define the entire proteome as a distribution $p_d(\mathbf{x})$, where $\mathbf{x} \in \mathbb{R}^{3\times n}$ is a protein, and $m_d(\mathbf{x})$ a collection of conformations of a specific protein experimentally (e.g. nuclear magnetic resonance spectroscopy, x-ray crystallography) or computationally (e.g. Monte Carlo or MD simulations) determined. Here, the vector $\mathbf{x}$ comprises all the coordinates ($\mathbb{R}$) of a protein made of n atoms. We wish to learn a low m-dimensional embedding $f : \mathbb{R}^{3\times n} \to \mathbb{R}^m$ that maps proteins onto the latent space, where sampling any point $\mathbf{z} \in \mathbb{R}^m$ and taking the inverse f^{-1} yields a continuous space of physically plausible molecular structures. However, as the expected observations $\mathbf{x} \sim p_d(\mathbf{x})$ and $\mathbf{x} \sim m_d(\mathbf{x})$ relax into a subset of conformations [112], the behaviour at the valley regions on the manifold (maxima in the energy landscape) is difficult to capture explicitly from the observations.

Let $f(\mathbf{z}|\mathbf{x};\theta)$ be an encoder function with parameters θ, and $d(\hat{\mathbf{x}}|\mathbf{z};\theta)$ be a decoder function that approximates the inverse f^{-1} accordingly. The conventional approach as described in Chapter 2.9.1 is a simple reconstructive autoencoder that minimises the mean squared error loss $\mathcal{L}_{\text{MSE}}$, defined by Equation 2.19. This follows the geometry and probability distribution from which the dataset was collected and therefore fails to generalise, especially at undersampled regions associated with transition states.

Proteins undergo conformational changes following lower energy paths in their energy landscape, where transition states are expected to be saddle points. The protein's expected energy, as determined by its atomic interactions, can be expressed as a loss function defined as:

$$\mathcal{L}_{\text{phys}} = \mathbb{E}_{\mathbf{x}\sim p_d(\mathbf{x})}\left[\Psi(d(f(\mathbf{x})))\right] \tag{3.1}$$

where Ψ evaluates the energy of the decoded structure. However, as with the naïve autoencoder, this also fails to generalise at regions far from conformations provided as example. In principle, it is possible to enforce physics to be respected in regions outside the known conformational space, for example with a Gaussian prior in the latent encoding. However, this makes an assumption on the distribution of the latent space.

Any midpoint along the geodesic (i.e. shortest path on the learned manifold) between any two protein conformations $(\mathbf{x_1},\mathbf{x_2}) \sim m_d(\mathbf{x})$ will also be a protein of same connectivity and composition. Assuming a degree of convexity of the latent space, we can enforce our physics-based loss function at random midpoints between $\mathbf{x}_1$ and $\mathbf{x}_2$, sampled linearly from a uniform distribution:

$$\mathcal{L}_{\text{path}} = \mathbb{E}_{(\mathbf{x_1},\mathbf{x_2})\sim m_d(\mathbf{x}),\; t\sim U}\left[\Psi(d(\mathbf{z}_m))\right] \tag{3.2}$$

where $\mathbf{z}_m = (1-t)f(\mathbf{x}_1) + tf(\mathbf{x}_2)$ are midpoints in the latent space between two protein conformations. This enables physical characteristics to be enforced on the manifold between two points, but outside of the known sampled conformational space.

We note that, in principle, this approach allows for the training of a neural network with m_d simulations of multiple proteins, whereby pairs (to interpolate between) are picked from the same simulation.

Putting this together, we can assemble the final loss as a weighted sum of the geometric term and the path term (which itself is comprised of various individually weighted physics terms):

$$\mathcal{L} = \alpha\mathcal{L}_{\text{MSE}} + \beta\mathcal{L}_{\text{path}} \tag{3.3}$$

3.2.2 Masked bonded loss

We have described a basic Molecular Mechanics force field in Chapter 2.1.2 and how it is broken down into a sum of individual energy terms (see Equation 2.4). We define our physics-based loss to be functionally identical to the Amber force field, thus compelling the potential energy of intermediate structures to be minimised. This would enforce physics to be respected at all points along the geodesic between two protein conformations and constrain the intermediates to follow a locally minimum energy path. We have developed a PyTorch implementation of the Amber force field allowing us to both utilise GPU acceleration and enable backpropagation through the physics loss.

In Equation 2.5, the Amber force field represented bonds as a harmonic potential about the equilibrium bond distance r_0: with a force constant k_b dependent on the specific type of bonded atoms. Here we take the force constants and equilibrium distances directly from the Amber ff14SB parameter sets.

Atoms in a protein structure file are listed residue-by-residue. So, the positions of atoms throughout the list are spatially coherent, whereby atoms involved in a common bond or angle are most likely adjacent in the list. We can efficiently utilise GPUs by calculating $\mathbf{r}^d$ such that $r_i^d = \left\|\mathbf{u}_i^d\right\|_2 = \|\mathbf{x}_i - \mathbf{x}_{i+d}\|_2$, then we can calculate the bond potential with:

$$V_{bonds} = \sum_{d} \mathbf{k}_b(\mathbf{r}^d - \mathbf{r}^{eq})^2 \tag{3.4}$$

for all relevant d. We can mask any values in $\mathbf{r}^d$ that do not correspond to bonded atoms by setting the corresponding element in $\mathbf{k}_b^d$ to zero.

The Amber force field represents angles similarly as a harmonic potential about θ, the angle between two bond vectors $\mathbf{u}^{d_1}$ and $\mathbf{u}^{d_2}$. We calculate the angle potential as:

$$V_{angles} = \sum_{d_1,d_2} \mathbf{k}_\theta(\boldsymbol{\theta}^{d_1,d_2} - \boldsymbol{\theta}_0)^2 \tag{3.5}$$

where

$$\theta_i^{d_1,d_2} = \arccos\left(\frac{\mathbf{u}_i^{d_1} \cdot s^{d_2}\mathbf{u}_{i+d_1}^{d_2}}{\left\|\mathbf{u}_i^{d_1}\right\| \left\|\mathbf{u}_i^{d_2}\right\|}\right) \tag{3.6}$$

where for the angle between atom indices i,j, and k we can determine d_1 and d_2 as $j - i$ and $j - k$. The factor s^{d_2} is 1 if $j < k$ or -1 if $j > k$.

Dihedral angles are represented in the Amber force field as:

$$V_{dihedrals} = \sum_{dihedrals} V_n[1 + cos(n\phi - \gamma)] \tag{3.7}$$

where V_n, n, ϕ and γ represent the barrier, periodicity, torsion angle and phase, respectively. We implement this as:

$$V_{dihedrals} = \sum_{n} \sum_{d_1,d_2,d_3} V_n \left[1 + \cos\left(n\boldsymbol{\phi}^{d_1,d_2,d_3} - \gamma\right)\right] \tag{3.8}$$

where

$$\begin{aligned}\phi_i^{d_1,d_2,d_3} = \text{atan2}\big(&\mathbf{u}_2 \cdot ((\mathbf{u}_1 \times \mathbf{u}_2) \times (\mathbf{u}_2 \times \mathbf{u}_3), \\ &\|\mathbf{u}_2\| \, (\mathbf{u}_1 \times \mathbf{u}_2) \cdot (\mathbf{u}_2 \times \mathbf{u}_3))\big)\end{aligned} \tag{3.9}$$

$$\mathbf{u}_x = s^{d_x}\mathbf{u}_i^{d_x} \tag{3.10}$$

The use of the atan2 function eliminates the possibility of a division by zero, resulting in infinite gradients.

3.2.3 Warped non-bonded loss

The non-bonded potential $V_{\text{nonbonded}}$, given in Equation 2.12, consists of a sum of two terms, describing van der Waals and electrostatic interactions. $V_{\text{nonbonded}}$ is evaluated for any pair of atoms not involved in a mutual bond, angle, or dihedral. Given $\mathbf{r}$, the distance between two atoms, the van der Waals interactions are described as in the Amber force field by a 12-6 Lennard-Jones potential V_{LJ}:

$$V_{\text{LJ}} = \sum_{i}^{N-1} \sum_{j=i+1}^{N} \left[\left(\frac{A_{ij}}{r_{ij}^{12}}\right) - \left(\frac{B_{ij}}{r_{ij}^{6}}\right)\right] \tag{3.11}$$

$$A_{ij} = \epsilon_{ij} R_{iJ}^{12} \quad \text{and} \quad B_{ij} = 2\epsilon_{ij} R_{ij}^{6} \tag{3.12}$$

$$R_{ij} = 0.5\left(R_{\text{min},i} + R_{\text{min},j}\right) \tag{3.13}$$

$$\epsilon_{ij} = \sqrt{\epsilon_{ii} \cdot \epsilon_{jj}} \tag{3.14}$$

where $R_{\text{min},x}$ and ϵ_{xx} are the van der Waals radius and well depth of the potential for an atom type x, as defined within the Amber parameter set. Equations (3.13) and (3.14) show the Lorentz-Berthelot mixing rules used to obtain cross terms between different i and j. Since side-chain atoms (except Cβ) are not used in the network training, only the repulsive term of the Lennard-Jones is applied. This is to avoid the protein structure packing excessively, filling the voids left by missing side-chain atoms.

Thus, here we only retain the repulsive component of the Lennard-Jones potential:

$$V_{\text{LJ}} = \sum_{i}^{N-1} \sum_{j=i+1}^{N} \left[\left(\frac{A_{ij}}{r_{ij}^{12}} \right) \right] \tag{3.15}$$

Electrostatic interactions are described by a Coulombic potential V_{C}:

$$V_{\text{C}} = \sum_{i}^{N-1} \sum_{j=i+1}^{N} \left[k_e \frac{q_i q_j}{\epsilon_0 r_{ij}} \right] \tag{3.16}$$

where q_x is the charge on atom x, ϵ is the dielectric constant, and k is the Coulomb force constant. As $\lim_{\text{r}\to 0} V_{\text{NB}}(r) = \infty$, the gradient descent becomes unstable at short inter-atomic distances. Clamping $\mathbf{r}$ causes the gradients to get stuck in the corners of the hypercube. Therefore, we approximate the non-bonded potentials by warping the input space $\mathbf{r}' \approx \mathbf{r}$ piecewise with an exponential. This is achieved equivalently and efficiently with the ELU [113] intrinsic activation function for some offset constant k, where:

$$\mathbf{r}' = \text{elu}(\mathbf{r} - k, \alpha = 1) + k \tag{3.17}$$

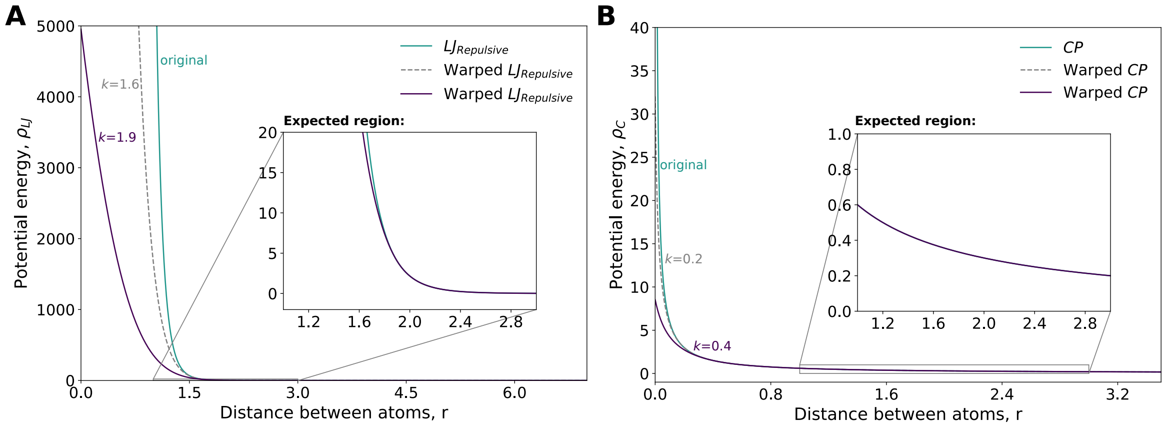


Figure 3.2: Comparison of original and warped non-bonded potentials. (A) Lennard-Jones and (B) Coulomb (with charge $q = 0.6$)

such that the potentials now tend to high positive or negative values, without significantly altering the profile of the potential well (Figure 3.2). We choose $k = 1.9$ for V_{LJ} and $k = 0.4$ for V_{C}, giving large positive or negative y-intercepts for the expected upper and lower bounds for A_{ij} and $q_i q_j$ accordingly.

3.2.4 Implementation

Atoms (points) of a molecular structure (such as in a PDB file, the standard representation for molecular 3D structure data, Figure 3.1B) can be sorted in a list so that covalently connected atoms appear close to each other. In the case of protein PDB files, sorting is typically achieved by grouping atoms by residues, and listing residues from N-terminal to C-terminal. Atoms within residues are ordered with the backbone atoms first, followed by the side-chain atoms, progressing outward from the backbone.

The 1D CNN architecture was implemented in PyTorch with 12 layers (6 in the encoder and 6 in the decoder components). The first layer has 3 input channels (for each atom's 3D coordinates) and 32 output channels, where subsequent layers of depth t have $\lfloor 32 \cdot 1.5^t \rfloor$ input channels with batch normalisation and ReLU activations. Convolutions have size 4 kernels with strides of 2 and padding of 1, halving the spatial dimension each layer. This means the molecule becomes like a progressively thicker but shorter 'ribbon' whose activations correspond to more deep and abstract characteristics of the molecule, Figure 3.1C. The latent space was fixed to two dimensions using a 1D adaptive mean pooling layer to handle arbitrary length input sequences.

We found that increasing the dimension of the latent space, adding an adversarial discriminating component, or adding residual layers led to only negligible generalisation improvements. The total number of parameters is 11,892,767 optimised using Adam with a learning rate of $1e^{-4}$ and weight decay of $1e^{-5}$. We used a batch size of 5 for 200 epochs and 1000 optimisation steps per epoch for the training of MurD. Training the model takes $\sim$7 hours on an NVIDIA TITAN Xp graphics card. A larger batch size allows a higher utilisation of the GPU so for transfer learning MurD was retrained with a batch of 16 for 200 epochs and 500 optimisation steps per epoch, and then retrained on the other structure sets at 100 optimisation steps per epoch. The decreased number of optimisation steps has no impact on training other than to increase the rate at which the network performance was tested and decrease the total allowed training time.

For the results given for MurD, we repeated the training of the neural network 10 independent times with random initial weights; all values reported (including the mean and standard deviation of the loss function, as well as the DOPE score of the resulting interpolated structures) are the average of these 10 repeats. For transfer learning to the other structure sets, we repeated training of the neural network 10 independent times, 5 with random initial weights and 5 retaining the weights from networks pretrained on MurD. The weights α and β in equation (3.3) were controlled at each step to keep the magnitude of $\mathcal{L}_{\mathrm{MSE}}$ 10 times greater than $\mathcal{L}_{\mathrm{path}}$, thus putting an order of magnitude greater emphasis on accurately reconstructing known structures as generating low energy intermediate structures.

3.2.5 Datasets

MurD has been crystallised in its open (PDB: 1E0D [114]) and closed (PDB: 3UAG [115]) states, as well as in intermediates between the two (PDB: 5A5E and 5A5F [116] with a backbone RMSD of secondary structure elements equal to 1.12 Å between them). The difference between these states comes from a large-scale conformational change of one of its three globular domains (residues 299 to 437) caused by substrate binding. Conformations from MD simulations of the closed and open states performed by some of the authors [46] were used here as the training dataset (4420 conformations in total comprising of 2507 and 1913 from closed and open simulations, respectively). In order to evaluate the predictive performance of our network in interpolating between the diverse conformational states, a third set of MD simulations were performed with the ligand removed from the closed state (closed-apo) [46]. We performed two additional repeats of this simulation using the same simulation protocol as [46]. This produced a total of 1513 conformations representing a transition from the closed-to-open state unseen in either the closed or open state simulations.

To test the transfer learning capabilities of the neural network, we selected simulations of p24 and TBE-sE from [117], HSP from [46] and SurA from [118]. For each of these simulation datasets, we created five training sets featuring 1000 conformations and five featuring 100 conformations via random selection.

All our neural networks were trained on the C, Cα, N, O (backbone), and Cβ (side chain) atoms, as representatives of protein structure. These are sufficient to describe the global conformation (fold) of a protein and the orientation of each side chain. The two extreme conformations in all cases (MurD, p24, TBE-sE, HSP and SurA) were selected by calculating an RMSD (backbone) matrix considering all the conformations of the training set and picking the pair with the highest RMSD. In all cases, we asked the networks to generate 20 conformations interpolating between these extrema. The RMSD between these extreme conformations, an indicator of the protein's range of dynamics, was 10.2 Å for MurD, 16.9 Å for p24, 4.9 Å for TBE-sE, 2.8 Å for HSP and 38.3 Å for SurA.

3.2.6 Percentage error calculation

We used the equilibrium values of bonds and angles from the Amber ff14SB force field [63] to estimate the percentage error in the corresponding bonded parameters modelled by our network. The error over all the bonds and angles present in a conformation was summed to obtain the $\%_{\text{TotalErr}}$ as a measure of accuracy of the network in generating physically plausible protein structures: $\%_{\text{TotalErr}} = \text{Err}_{\text{bonds}} + \text{Err}_{\text{angles}}$. Similarly, the per-residue errors were calculated as the sum of errors associated with interactions involving any atom within a residue.

3.2.7 Analysis of MurD opening angle

The opening angles (azimuth and elevation) of MurD were calculated by aligning the protein along the stable domain (residues 1 to 298), centring the resulting alignment on the hinge between the mobile and stable domains (centre of mass of residues 230 to 298), and reporting the position of the centre of mass of the mobile domain (residues 299 to 437) in spherical coordinates. A schematic representation of the angles calculated is shown in Figure 3.9A.

Visual inspections of the protein structures and related figures were done with VMD 1.9.2 [119] and PyMOL [120]. Graphs were plotted using matplotlib [121].

3.3 Results

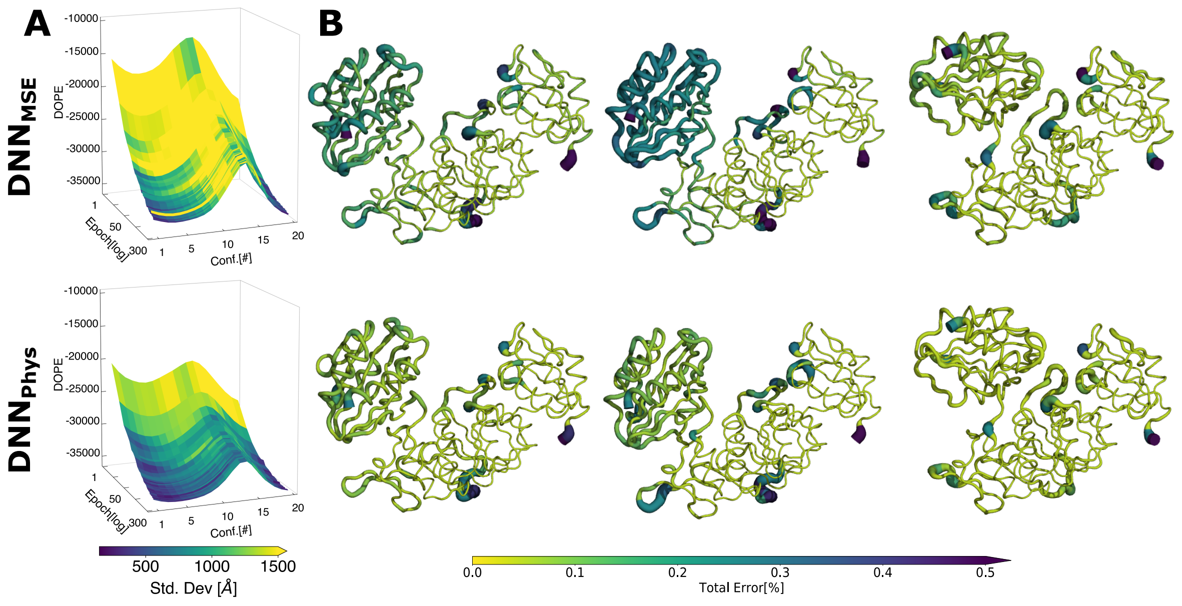


Figure 3.3: Performance of the neural network trained to minimise MSE alone or in conjunction with physical terms (labelled on the left). At every epoch, each network was asked to generate 20 conformations interpolating from MurD closed to open state. (A) At each epoch, we calculate the DOPE score of each conformation and report the mean values from the 10 repeats on the vertical axes, with colour corresponding to the standard deviation. Physics-based loss functions lead to interpolations with better structural quality, reflected by lower DOPE scores. (B) The network-predicted protein conformations of open (left), intermediate (centre), and closed (right) states at the last epoch, shown in sausage representation with the thickness and colour corresponding to the percentage error at the residue level.

MurD (UDP-N-acetylmuramoyl-L-alanine:D-glutamate ligase) [122] plays a key role in the peptidoglycan biosynthesis of almost all bacterial species by catalysing the addition of D-glutamic acid to UDP-N-acetylmuramoyl-L-alanine. The protein consists of three globular domains, one of which undergoes a large-scale rearrangement (from open to closed state) triggered by substrate binding to activate the catalytic site. The open (PDB: 1E0D [114]) and closed (PDB: 3UAG [115]) states, as well as a few intermediates (PDB: 5A5E and 5A5F [116]), have been crystallised, providing key experimental evidence about the possible protein's mode of action. MD simulations of the open and closed states, and of the transition between them have been previously carried out [46], providing a useful dataset for our network training and its performance evaluation. Such extensive data and the importance of MurD as a potential antibacterial drug target [123, 124] make this protein a particularly interesting test case for this study.

Here, we train our neural network with conformations of MurD open and closed states generated by MD (*training set*) and assess the network's capacity for predicting a possible transition path between the two. We assess the quality of the predicted path in terms of its structural quality, as well as matching with the closed-to-open MD simulation and available intermediate crystal structures, not provided for training. We then evaluate the capacity of our network trained on MurD to adapt to other proteins in a transfer learning scenario.

3.3.1 Force field-based loss functions improve network accuracy

We first assessed whether training the neural network using information on the physical properties of proteins is beneficial. To this end, we trained it with five different loss functions featuring an increasing number of physics-based constraints (Figures 3.3 and 3.4). Each network was trained with conformations produced by the MD simulations of MurD closed and open states. To track the network training progress, we divided it

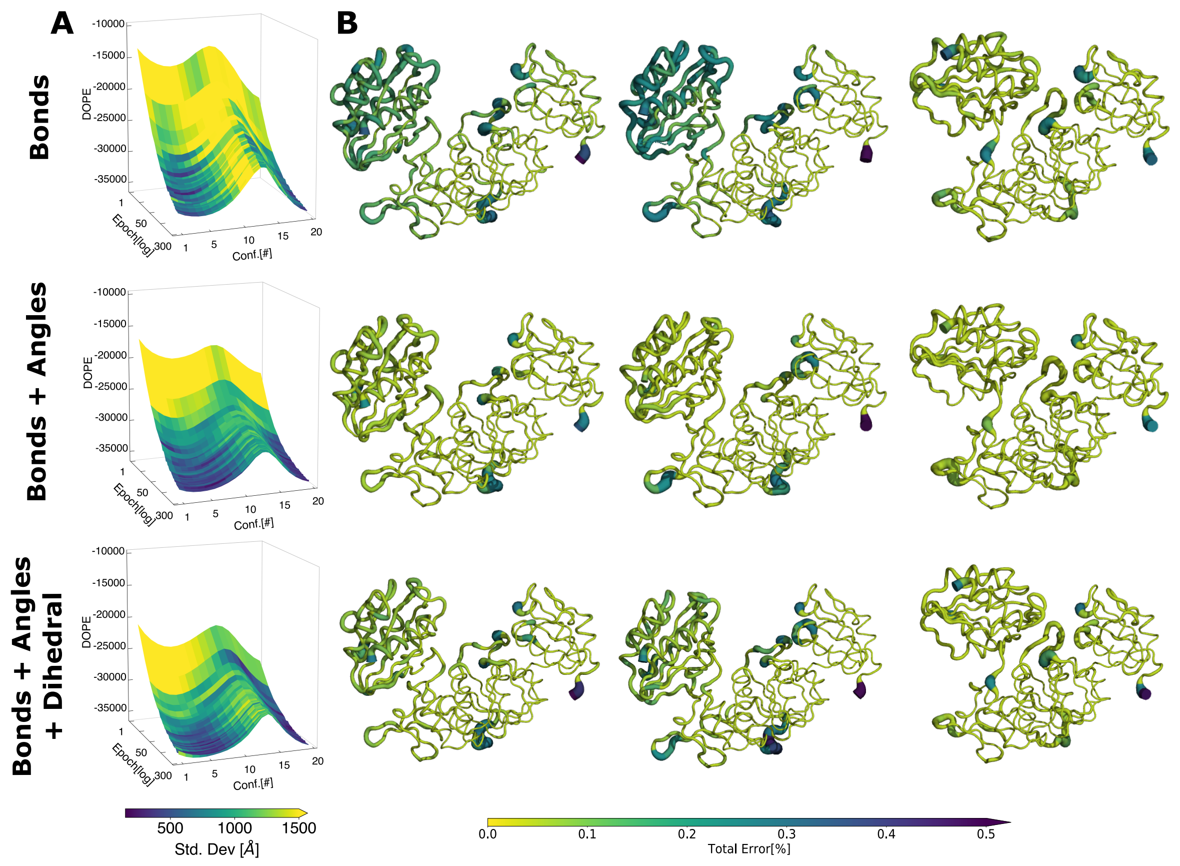


Figure 3.4: Performance of the neural network trained with different loss functions featuring an increasing number of physical terms (labelled on the left). At every epoch, each network was asked to generate 20 conformations interpolating from MurD closed to open state. (A) At each epoch, we calculate the DOPE score of each conformation and report the mean values from the 10 repeats on the vertical axes, with colour corresponding to the standard deviation. (B) The network-predicted protein conformations of open (left), intermediate (centre), and closed (right) states at the last epoch, shown in sausage representation, with the thickness and colour corresponding to the percentage error at the residue level. Addition of physics-based terms in the loss function improves interpolated structures quality.

into 200 parts termed epochs. At the end of every epoch, each network was challenged to predict a transition path by producing 20 intermediate structures corresponding to evenly spaced points along the straight line connecting the two states in the networks' latent space. To assess the quality of intermediate conformations throughout the training, we measured three different quantities: average distance of all bonds and angles from their expected equilibrium value (as per the Amber ff14SB force field [63]), percentage of residues in favourable and outlier Ramachandran plot regions (assessment of dihedral angles distributions), and Discrete Optimized Protein Energy (DOPE, an atomic distance-dependent statistical potential commonly used to assess protein structures) score [125].

MurD structures generated by the network trained without any physics-based constraints (using just the Mean Square Error, $\mathrm{DNN}_{\mathrm{MSE}}$) could only poorly reproduce expected bond and angle equilibrium values. The average error of intermediate structures generated at training completion was equal to $\sim$22.9% $\pm$ 2.3, with an average DOPE score of -33017 ($\pm$ 1005). Notably, the interpolation quality degraded the further the intermediate structure was from examples within the training set (errors of up to 26.6% and DOPE score reaching $\sim$-27642). Regions of the protein featuring the highest error were concentrated in loops and on the mobile domain of the protein (Figure 3.3B: $\mathrm{DNN}_{\mathrm{MSE}}$). Coupling information on bonded (bonds, angles, dihedrals) and non-bonded (steric hindrance and electrostatics) potentials with MSE in the loss function improved the quality of the protein structures generated (Figure 3.3B: $\mathrm{DNN}_{\mathrm{Phys}}$). Particularly, the average DOPE score of intermediate structures generated at training completion was -34810 ($\pm$ 752), and the transition state conformations showed

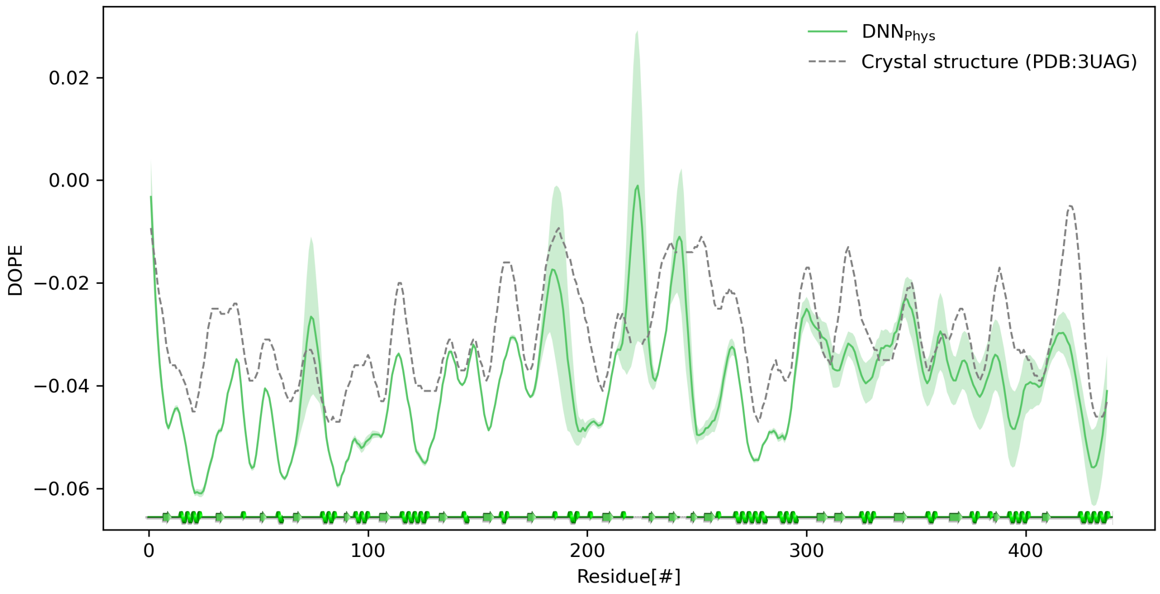


Figure 3.5: DOPE score profile of the conformations generated by the best network. The mean DOPE score is plotted in bold and the corresponding standard deviations (calculated using the 20 transition state conformations) in transparent. The dashed grey line represents the DOPE profile of the MurD crystal structure in its closed state (PDB: 3UAG). The secondary structure of MurD is also shown in green at the bottom for reference (helix: α-helix; arrow: β-strand; line: coil)

a maximum of -31601, lower than what was measured on $\mathrm{DNN_{MSE}}$. Overall, the residue-level DOPE score of $\mathrm{DNN_{Phys}}$ models was comparable with the profile of the MurD crystal structure (PDB: 3UAG; see Figure 3.5). The network showed an increasingly better performance in terms of the protein structural quality with an increasing number of physics-based terms (Figure 3.4). However, the addition of the non-bonded component showed only a statistically insignificant improvement. Evaluating Ramachandran distributions of amino acids in our models further highlighted how using physics-based restraints in $\mathrm{DNN_{Phys}}$ allow it to produce models of higher structural quality than $\mathrm{DNN_{MSE}}$ (Figure 3.6). Models produced by $\mathrm{DNN_{MSE}}$ featured 87.6% $\pm$2.3 of residues in Ramachandran favoured regions, and 4.3% $\pm$1.3 outliers. Models produced by $\mathrm{DNN_{Phys}}$ featured a higher percentage of residues in favoured regions (93.4% $\pm$ 1.3) and only 1.8% $\pm$ 0.7 outliers, all located close to the acceptable ranges.

To obtain a comprehensive view of the protein conformational space encoded in our neural networks, we calculated the DOPE score of protein models generated by regularly sampling the networks' latent spaces (Figure 3.7). The latent space of the network trained solely with MSE featured two distinct regions, associated with MurD open and closed states, separated by a region of unacceptable quality (high DOPE score). In the network trained with our physics-based loss function, the whole conformational space appears near-convex, and all structures located between the closed and open basins had a low DOPE score. This was possible because the latter network is also trained to minimise the energy of random midpoints between training examples. Overall, training our 1D CNN with a loss function featuring a combination of MSE, bonded, and non-bonded terms yielded interpolations of good structural quality through conformations not represented within the training set.

3.3.2 Neural network predicts a possible state transition path

The switch of MurD between closed and open conformations involves the rigid-body rearrangement of one domain (residues 299 to 437) with respect to the rest of the protein (residues 1 to 298). We can readily characterise this movement by tracking the position of the centre of mass of this domain with respect to its connection to the rest of the protein and reporting it in spherical coordinates (Figure 3.9A). Describing the closed and open MurD MD simulations according to this metric reveals that the conformations of these two states are clearly

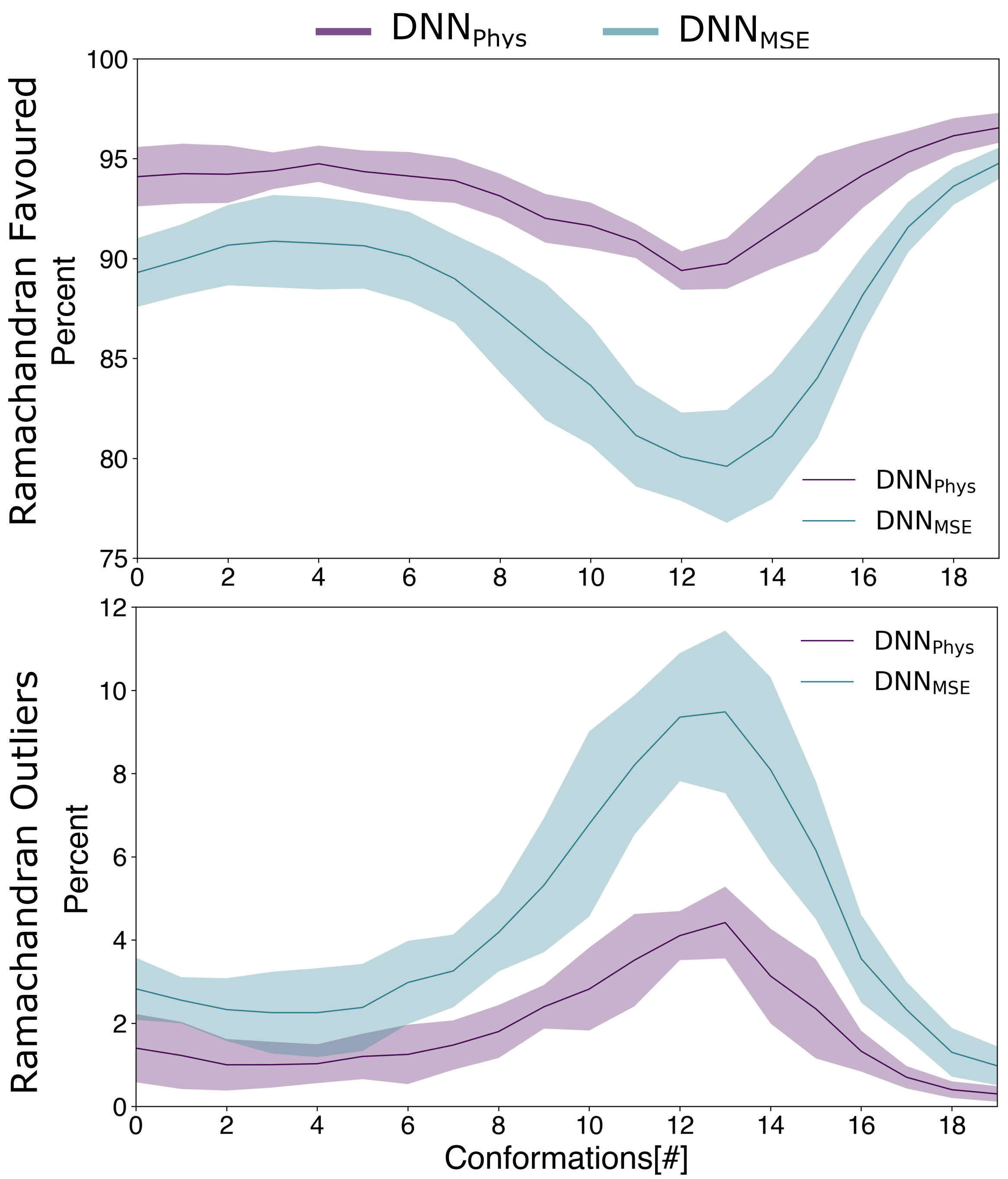


Figure 3.6: Comparison of the Ramachandran distributions for the conformations generated by $\mathrm{DNN_{MSE}}$ and $\mathrm{DNN_{Phys}}$. High quality models are expected to have > 98 % favoured and $< 0.2\%$ outliers. Ramachandran outliers increase as a function of resolution [126].

distinct (two light green regions in Figure 3.9B). The MD simulation of MurD switching from closed-to-open state (hereon *test set*) follows an irregular path: first, a concerted increase in elevation and azimuth opens the domain, leading to conformations closely resembling the crystal structure of the intermediates (RMSD of secondary structure elements equal to 1.16 and 1.12 Å versus PDBs 5A5E and 5A5F, respectively), then an increase in azimuth and radius leads the domain to its final equilibrium position. This two-step movement in the Azimuth-Elevation space is associated with a straight line within the latent space described by our network. As the neural network is trained to minimise the energy of any midpoint between any two states, we consider con-

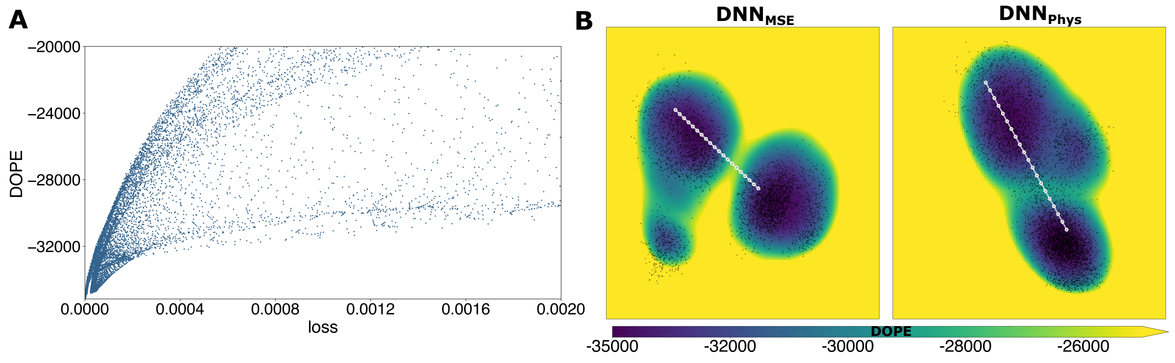


Figure 3.7: Analysis of latent spaces of networks trained on conformations of MurD open and closed states. (A) The physics-based loss function used to train our neural network correlates with the DOPE score, making it a reasonable estimator of protein structural quality. (B) DOPE scores corresponding to the latent spaces of our neural network trained with two different loss functions. On the left, the network is trained to only minimise mean square error (MSE) between input and output structure, on the right the loss function combines MSE and a physics-based loss function (DDN). Yellow regions indicate structures of poor quality, black points report on the projection in the latent space of all training examples, including the generated midpoints $\mathbf{z}_m$ used to train DDN. The transition paths predicted by the networks are projected as connected white dots onto their respective latent space. The latent space of the network based solely on MSE feature two basins associated with closed and open states, separated by a region of poor quality models. In the network trained with both MSE and physics ($\mathrm{DNN_{Phys}}$), the two states are connected by acceptable protein models. Also see Ffigure 3.8 comparing the structures from the identified basins.

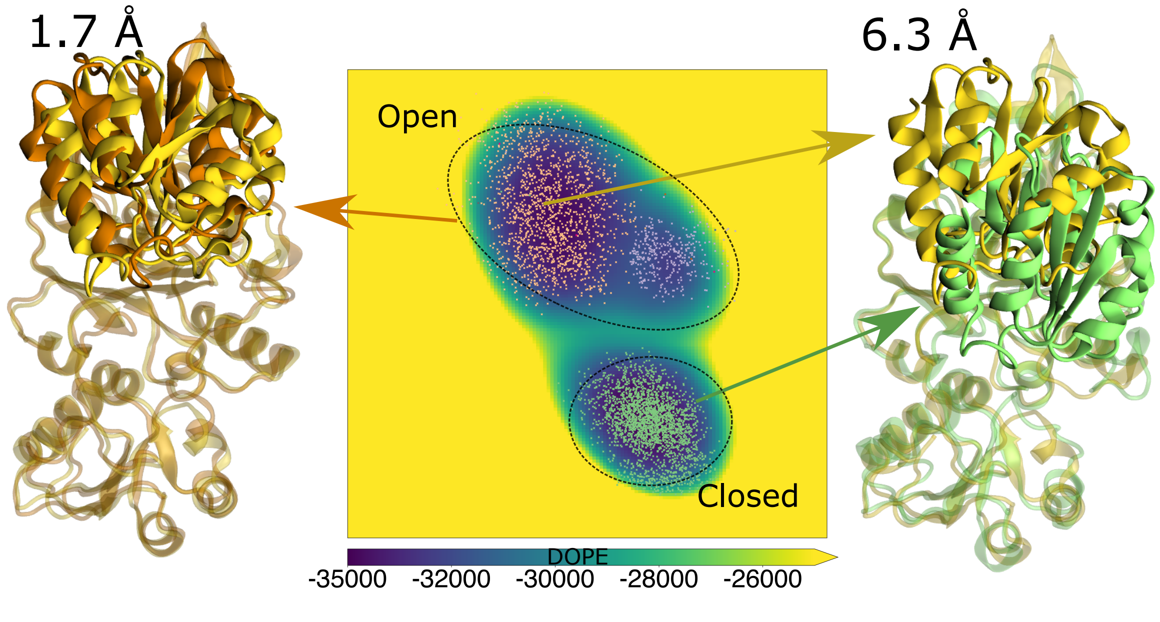


Figure 3.8: Latent space landscape coloured according to the DOPE score showing two basins corresponding to the closed and open states of MurD. The points (coloured according to their cluster) indicate the projection in the latent space of all training examples. Example structures (open structures in yellow and brown cartoon representations while closed in green) corresponding to the identified basins are superposed to show their structural differences. Backbone RMSD of structured regions of the mobile domain are indicated on top. The stable and mobile domains are shown in transparent and solid, respectively.

formations generated by a straight line in the latent space as an approximation of a possible transition path. To illustrate that methods relying on purely geometric interpolations would be unable to predict such a transition path, we generated an interpolation exploiting a principal components analysis (PCA) of MurD conformations.

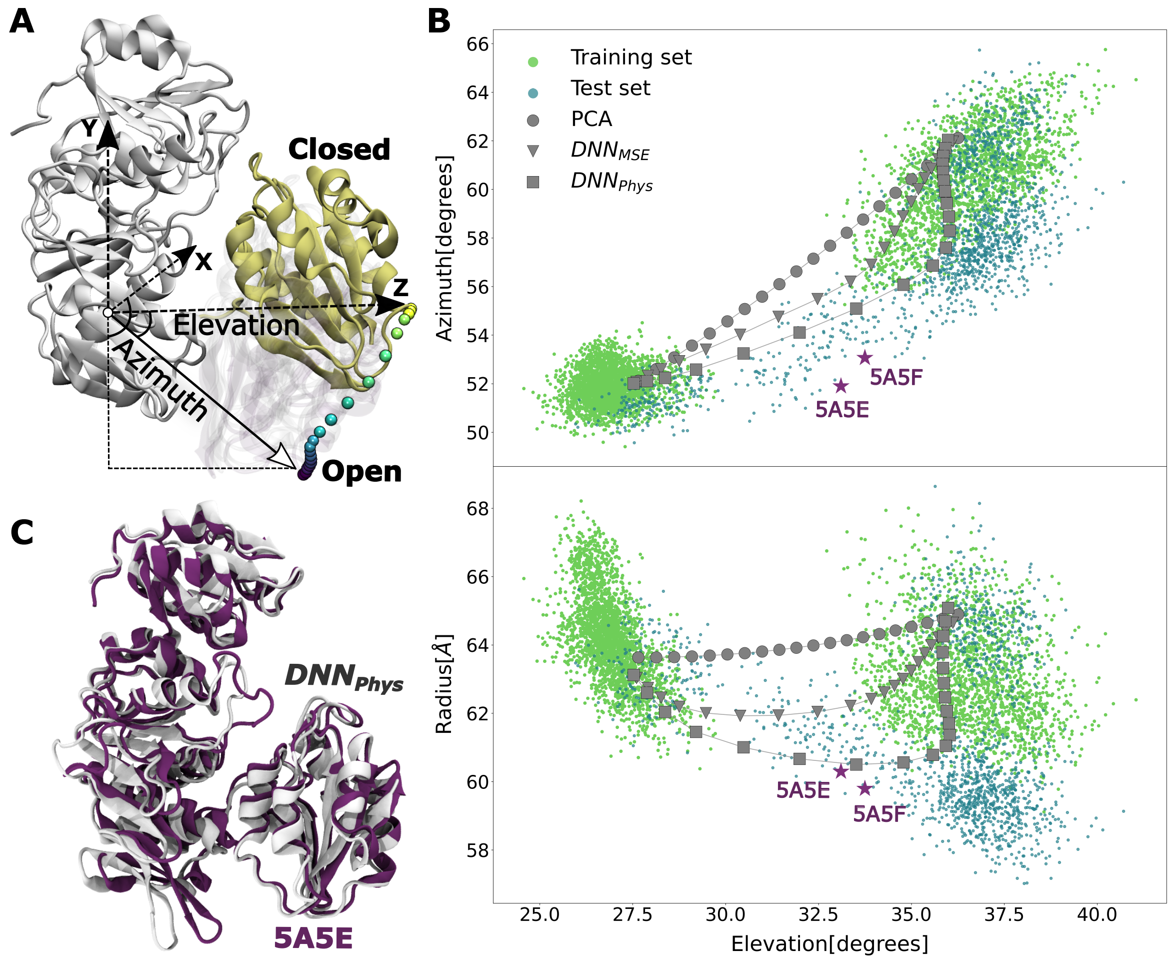
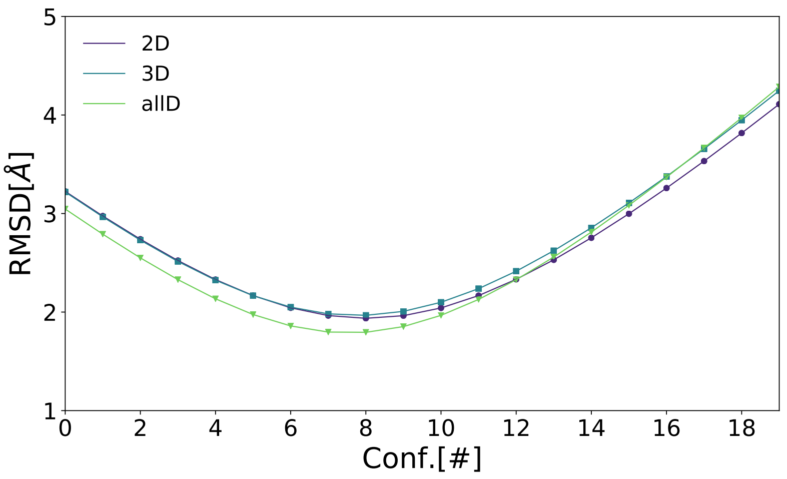


Figure 3.9: State transition path prediction by the neural network. (A) Side view of MurD showing the transition of its mobile domain from the closed to open state (with the time evolution marked as beads coloured from yellow to violet). (B) The conformational change of MurD can be described in terms of spherical coordinates between its three domains (see Section 3.2.7). We report the opening angle of each conformation in the training set (light green), test set (dark green), intermediate crystal structures (palatinate stars) as well as interpolations generated by PCA (gray circles), a purely geometry-based neural network ($\mathrm{DNN}_{\mathrm{MSE}}$, grey triangles) and a neural network combining geometry and physics ($\mathrm{DNN}_{\mathrm{Phys}}$, grey squares). The interpolation produced by $\mathrm{DNN}_{\mathrm{Phys}}$ best reproduces the path described by the test set and transits in the vicinity of intermediate crystal structures. (C) Superimposition of intermediate MurD conformation 5A5E (in palatinate) and an intermediate conformation generated by the neural network (in white), with an RMSD of 1.3 Å.

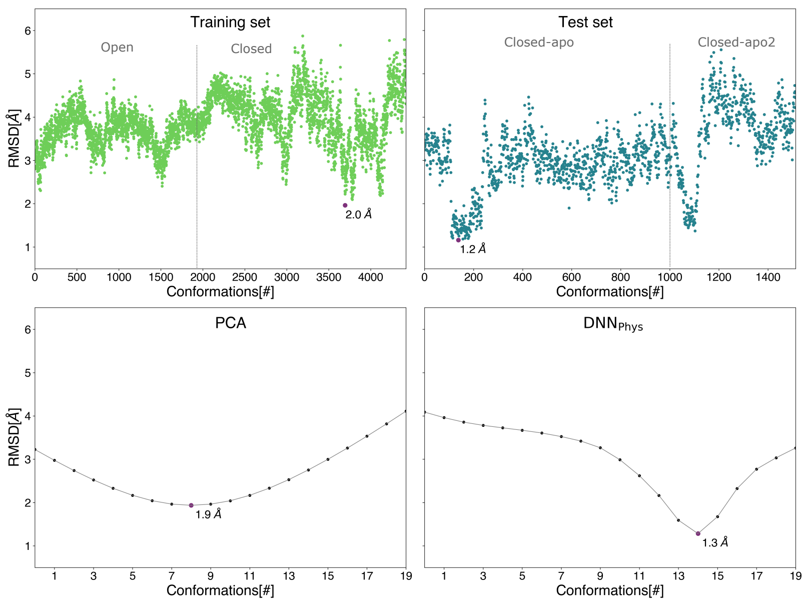
Conformations generated as regularly spaced linear combinations of the two main eigenvectors trace a uniform circular trajectory, far from what was observed in the test set. Considering more than two eigenvectors also does not provide a significant improvement in the interpolations generated by PCA (Figure 3.10a).

The transition path predicted by the neural network trained solely based on MSE is less uniform, but yet only poorly recapitulates the test set. The interpolation produced by the network trained considering both MSE and physics-based terms traces instead a path closely resembling the test set. The additional physics-based terms in the loss function not only helped in generating structures with correct bond lengths, angles, and dihedrals, but also prevented an interpolation that would have required the protein domains to slightly compenetrate. Remarkably, one of the interpolated conformations had a backbone RMSD of 1.28 Å from the intermediate crystal structure (PDB: 5A5E), a quantity comparable to that of the test set (Figure 3.10b).

Thus, our results show that in this application, our 1D CNN trained with a combination of MSE and physics-



(a) Comparison of the backbone RMSD of secondary structure elements of the interpolated structures generated by PCA considering 2 dimensions (2D), 3 dimensions (3D) and all dimensions (allD) with respect to the intermediate state crystal structure (PDB: 5A5E). Increasing the number of eigenvectors describing protein motion only marginally improves the quality of the interpolated structures.



(b) Backbone RMSD of MurD structured regions with respect to the intermediate state crystal structure (PDB: 5A5E). The top panels show the RMSD of MD simulations used for training (left; open and closed states) and as test (right; closed-apo and closed-apo2). The bottom left panel reports the RMSD of structures generated via a PCA-based interpolation. The bottom right panel reports the RMSD of structures generated by interpolation of our neural network ($\mathrm{DNN_{Phys}}$, i.e. with a loss function featuring both MSE and physics-based terms). In all images, the structure having the smallest RMSD from the intermediate MurD crystal structure 5A5E is marked with a palatinate dot. The $\mathrm{DNN_{Phys}}$ interpolation produces the model best recapitulating the intermediate MurD conformation.

Figure 3.10

based terms was capable of correctly identifying a possible transition path between two distinct states.

3.3.3 Transfer learning improves convergence

Training a neural network on a small dataset can be facilitated if the network is pre-trained on a similar, larger dataset, a process called transfer learning. The convolutional nature of our network enables this training approach, as its architecture (and thus the number of its parameters) is independent of the number of atoms in the dataset under study. To assess the transferability of our trained network, we leveraged the MD simulations of four different proteins: the HIV-1 capsid protein (monomeric p24, PDB: 1E6J [127]), the tick-borne encephalitis virus envelope protein E (TBE-sE, PDB: 1SVB [128]), the small heat shock protein αB crystallin (HSP, PDB: 2WJ7 [129]), and the periplasmic chaperone SurA (SurA, PDB: 1M5Y [130]). These proteins differ from MurD in mass (24-kDa, 43-kDa, 10-kDa and 45-kDa vs 47-kDa, respectively) as well as fold and dynamics (Figure 3.13A). p24 consists of two rigid domains connected by a flexible linker, TBE-sE is highly elongated and features several long loop regions (e.g., residues 73 to 89 and 146 to 160), HSP is small and features only local fluctuations, whereas SurA is a multidomain protein featuring broad and highly complex dynamics.

We randomly sampled each simulation dataset to produce multiple training sets featuring 1000 and 100 representative structures (see Methods). For each resulting training set, we then trained our neural network in two different ways: from scratch, and by initialising its parameters with those of the best network trained on the MurD dataset using our physics-based loss function. Each neural network was assessed according to its capacity to interpolate between the two most different conformations (i.e., largest RMSD) in its training set.

In all cases, the networks having transferred parameters featured a mean loss starting from smaller values, and dropping faster. Networks trained with 1000 examples featured only a marginally lower mean loss than those trained with 100 examples (Figure 3.13B). Furthermore, transfer learning yielded latent spaces with overall lower DOPE scores. In all cases but SurA, negative DOPE scores were observed in regions extending slightly beyond the front defined by the training set (Figure 3.11). SurA was the only protein featuring DOPE landscapes without low and well-defined minima. As their associated loss functions were larger than in all other cases and had not converged yet, we trained four neural networks (initialised from scratch and via transfer learning, trained with 100 and 1000 examples) 10 times longer. Their loss function and DOPE profiles kept dropping, indicating that learning complex dynamics requires longer training (Figure 3.12).

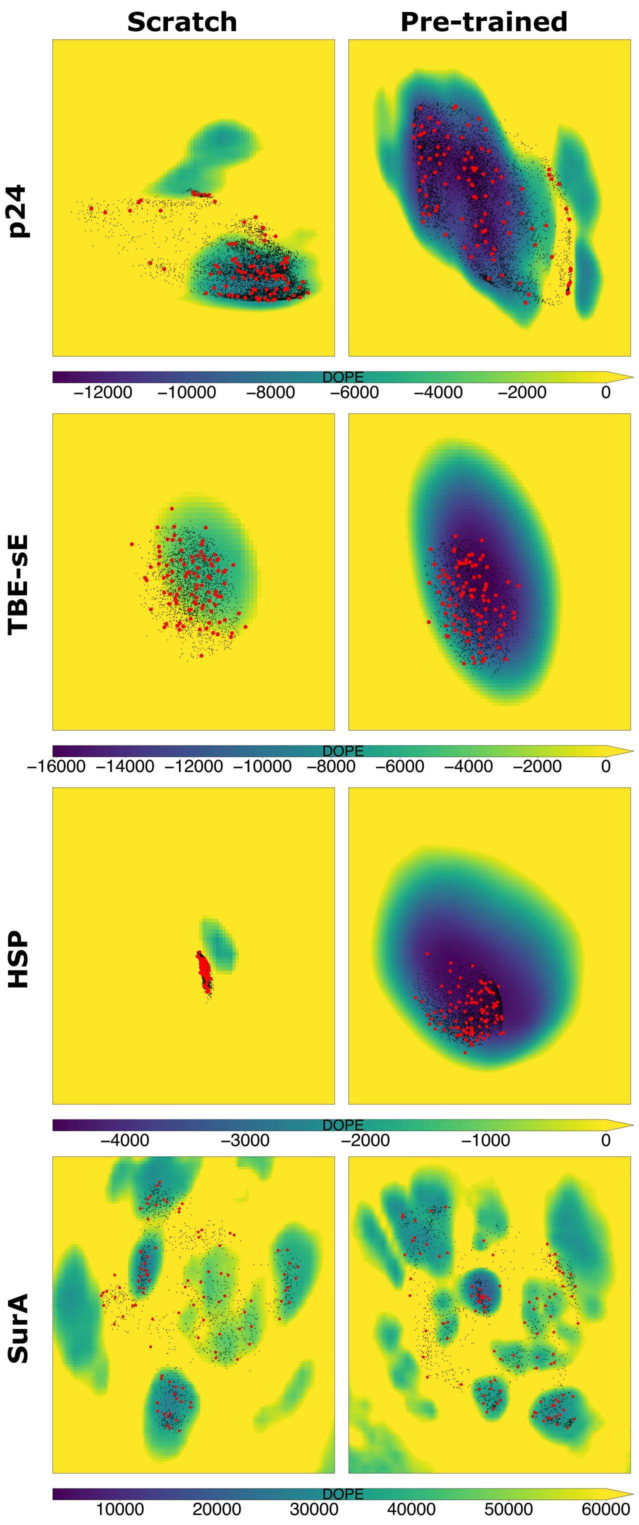


Figure 3.11: Comparison of latent space DOPE scores for four proteins (p24, TBE-sE, HSP, and SurA), comparing networks trained from scratch or pre-trained with MurD. Black points indicate the position of all conformations obtained via MD simulation, while red points indicate the subset of 100 conformations randomly selected to be part of the training set.

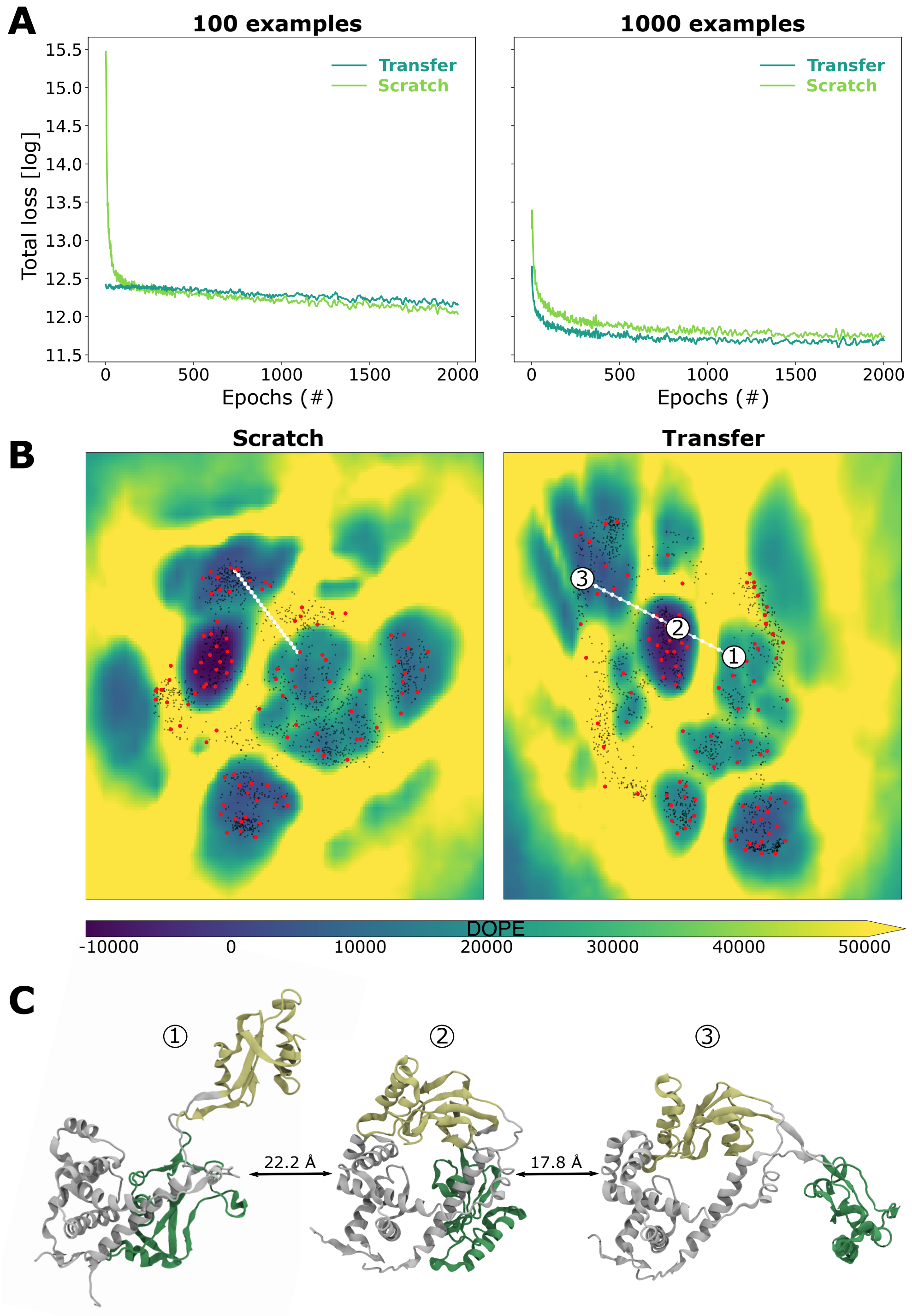


Figure 3.12: Long training of neural networks with SurA conformations. (A) Evolution of the loss function when training with 100 and 1000 examples. Transfer learning with 1000 examples leads to lower loss throughout training. (B) DOPE landscapes for networks trained 10 times longer from scratch and via transfer learning with 100 examples. Red and black points are defined as per Figure 3.11. White lines indicate the interpolation between the most extreme conformations, shown in panel (C) along with an intermediate and their respective RMSDs.

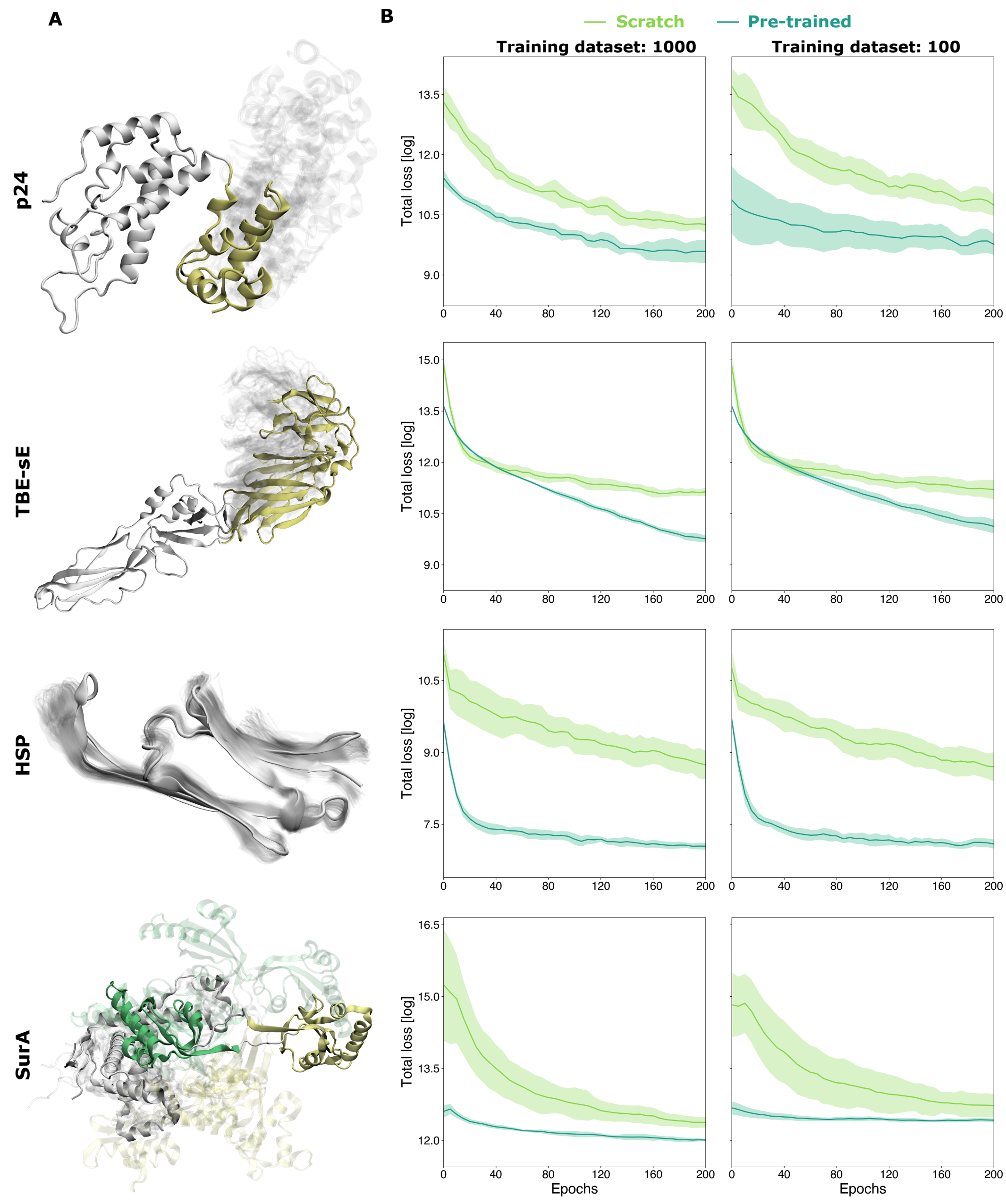


Figure 3.13: Transfer learning of our best network trained on MurD to four different proteins. (A) Side view of p24, TBE-sE, HSP, and SurA showing example alternative conformations with transparency and the highly mobile domains of p24, TBE-sE, and SurA in colour (yellow and green). (B) Moving average (window of 3) mean (solid line) and standard deviation (shaded region) of the total loss (log scale), comparing the performance of the pre-trained network with its counterpart trained from scratch using a training dataset of size 1000 and 100, all using the same loss function featuring both MSE and physics-based terms.

To characterise the ability of our neural networks to generate structures unseen during training, we examined those exposed to examples of the protein p24. As demonstrated by an available crystal structure (PDB: 3MGE), this protein assembles into a circular homo-hexamer via a large-scale rearrangement of its two globular domains. The structure in the p24 unbound simulation most closely resembling the bound state has an RMSD of 3.7 Å from it. We asked each trained network to encode and decode the bound state structure (unseen during training) and compared the RMSD of the resulting generated model to the input.

Neural networks trained with 100 examples performed equivalently (RMSD equal to 8.23 ± 1.72 Å for those trained from scratch, and 8.32 ± 1.65 Å for those pre-trained). This poor performance indicates that training with only 100 examples yields neural networks that, although already capable of generating low-energy protein conformations, have limited generalisation capabilities. Using 1000 training examples led to substantially better results, with pre-trained networks outperforming those trained from scratch (4.69 ± 0.15 Å the former, 5.22 ± 0.57 Å the latter). Since these RMSDs are still higher than that of the best available training example, we systematically sampled the networks' latent spaces. We found regions where generated structures featured RMSDs from the bound state marginally lower than any available example (∼3.6 Å when training with 1000 examples). Remarkably, these regions were located beyond the front of training structures, meaning that most suitable bound state conformations could be found via local extrapolation (Figure 3.14).

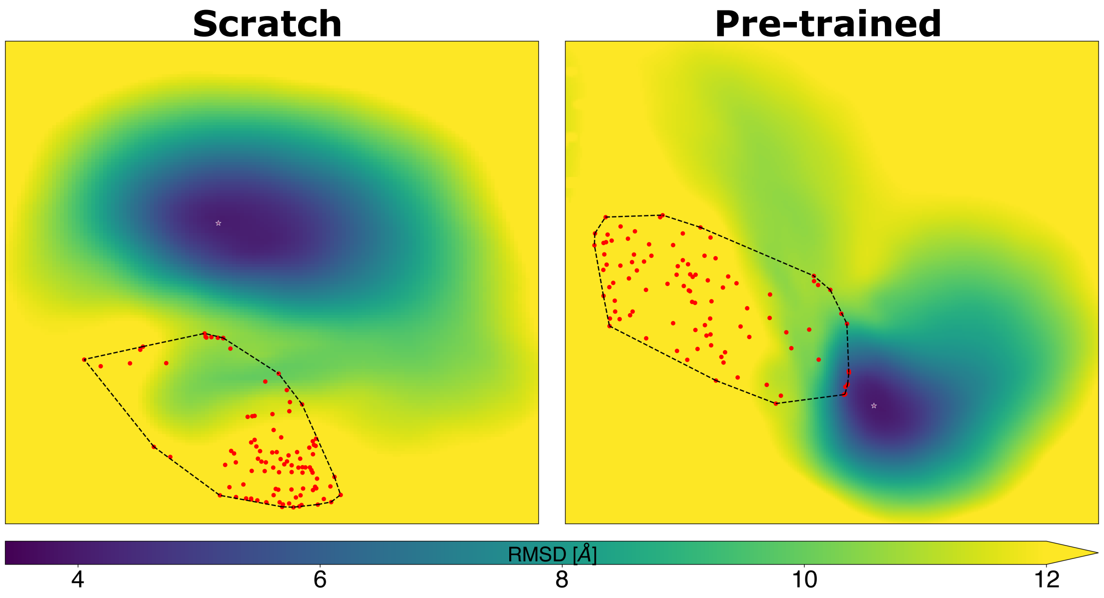


Figure 3.14: Latent spaces of neural networks trained from scratch (left) and via transfer learning (right) with p24 conformations. Red points indicate the position of the 100 training set conformations, and the dashed black line shows their convex hull. The background colour indicates the local RMSD between the decoded structure and the known bound state (PDB: 3MGE), where the position of lowest value is indicated with a palatinate star. In both trained networks, the conformation with lowest RMSD from the known bound state falls outside of the training set's convex hull. In the training set, the lowest RMSD from the bound state is 3.8 Å, in the network trained from scratch it is 4.1 Å and in the network having parameters initialised by transfer learning it is 3.9 Å.

Overall, these results demonstrate that our network can be trained with proteins of arbitrary size, and indicate that transfer learning enables faster training convergence, even when training data is limited.

3.4 Discussion & Conclusion

Conformational dynamics are an intrinsic property of any protein, their magnitude depending on the required biological function. Several computational methods have been designed to sample the protein conformational landscape and predict existing states. Methods leveraging MD coupled with enhanced sampling methods, while helpful, are still limited by the timescales at which the transitions occur and the associated high demand for computational power. We have designed and trained a generative neural network with collections of protein conformations as a means to predict plausible intermediates between any of them. While here the network has been trained with structures from MD simulations, any source of structural information can in principle be used. Since as little as 100 structures were sufficient to produce physically correct models with our trained network, this opens the door to directly leveraging collections of experimentally determined atomic structures.

Generative networks [96, 97, 99] are effective when interpolating between existing data but, unless additional information is provided, their extrapolation capabilities are typically insufficient to generate plausible molecular structures. The additional difficulty when training a neural network with protein atomic coordinates is associated with the very high number of degrees of freedom to be handled (often >1000 individual x, y, and z coordinates), making the training process arduous. To overcome these limitations, we have designed a 1D CNN architecture and introduced physics-based terms in the loss function. The convolutional architecture, already commonplace for image and text processing as well as bioinformatics tasks, is associated with a smaller number of trainable parameters and is independent of the number of degrees of freedom (and thus atoms) within the training set. Our physics-based terms in the loss function directly constrain the learning of protein structures obeying essential force field potentials. To further facilitate the training process, we have introduced a warping on the non-bonded potential terms (see Section 3.2.3), leading to better gradients encouraging convergence in otherwise high Lipschitz constant conditions. To test our network we have purposely designed a set of hard tasks, by selecting the most different conformations from protein simulations featuring different ranges of conformational changes, and asking the neural network to generate suitable interpolations by leveraging only a 2-dimensional latent space, as well as a limited number of training examples.

Challenged with the flexible protein MurD, our network revealed a surprising transition path between two distinct states, closed and open. Remarkably, conformations along this path were both physically plausible and compatible with an existing MD simulation featuring a closed-to-open transition as well as with two crystal structures of MurD locked in intermediate conformations. While the MSE term in the loss function was key for the network to learn the global protein shape, the addition of physics-based terms was crucial to generating low-energy conformations, associated with a low DOPE score. The bonded terms (bonds, angles, dihedrals) helped fine-tune local atomic arrangements, whereas the non-bonded ones (electrostatics [131] and Lennard-Jones [132]) were a significant player in identifying a suitable transition path within the conformational space. The structures produced by our new network were found to be both of lower energy and greater biological relevance than those obtained by PCA-based interpolations or by a neural network trained solely according to an MSE-based loss function (Figures 3.9B and C). By analysing the loss function values within the 2-dimensional latent space of our physics-based neural network, we observed a clearly defined and near-convex minimum, despite the network having been trained with distinct conformational states (Figure 3.7B).

An attractive feature of our convolutional network is that it can be trained with conformations of proteins of arbitrary amino acid sequence. This also enables transfer learning, whereby a pre-trained network is repurposed to tackle a new, though related task, expected to lead to improved generalisation and faster network optimisation. In this context, we noted that different proteins still share common features (e.g., typical bond lengths, and angles, as well as the same sequence of atoms in the backbone), that should not be re-learned. We transferred the network trained on MurD to four different proteins (HIV-1 capsomer protein p24, tick-borne encephalitis virus envelope protein E, αB crystallin small heat shock protein, and periplasmic chaperone SurA), for which we provided only a limited number of training examples (as low as 100). Remarkably, after a few epochs the total error associated with structures generated by the pre-trained network dropped lower than that of structures produced by the network trained from scratch.

A detailed analysis of generated protein structures indicated that our neural network may produce suboptimal loop regions when these are highly flexible and the protein movement is dominated by larger domain-level conformational changes. Thus, we expect our neural network to perform best with folded proteins, with atoms featuring correlated movements. The intermediate structures generated by our neural network will feature overall low energy according to a subset of typical terms associated with molecular structures, namely their bonds, angles, dihedrals, van der Waals, and electrostatic interactions. Still, the energy of predicted intermediates will not be an accurate estimation of the energy barriers a conformational change is associated with. Thus, while capable of local extrapolation, the network should not be expected to predict new, completely unseen states. Nevertheless, knowledge of possible transition paths can provide guidance for the definition of appropriate reaction coordinates/collective variables in MD-based enhanced sampling schemes. Furthermore, as the neural network effectively transforms a discrete collection of protein conformations into a "conformational continuum", it can find applications in flexible protein-protein docking scenarios where, under the conformational selection paradigm, the ability to fine-tune protein conformations to maximise their compatibility with a binding partner is desirable [46].

Due to the use of MSE in the loss function, our networks will primarily capture global, high-variance motions. If local motions in a particular region are more desirable, then the network can be biased toward these regions by reweighting the MSE contributions of atoms in that region or, alternatively, by training networks on a cropped version of the protein that contains the regions of interest. In Section 1.2.2, we discussed methods that attempt to capture slow motions rather than high-variance motions. Our physics-based loss function could be incorporated into the loss functions of these networks to prioritise slow motions over high-variance motions.

In summary, we have designed a new architecture with physics-based loss functions that significantly improve the synthesis of protein atomic structures in different conformational states. We have shown that biologically relevant transition paths can be predicted when synthesised interpolations must respect physical laws. The estimated transition path retains a surprising resemblance to the ground truth, and even more surprisingly, this is achievable with only a small number of training examples. Further, our findings show that, through our fully-convolutional architecture, we can transfer features learnt from one protein to another, indicating the possibility of simultaneously training with conformations of multiple proteins, towards a network trained with the whole proteome.

Chapter 4

The *molearn* software package

The software presented in this chapter was reported in: S. C. Musson and M. T. Degiacomi. ”Molearn: a Python package streamlining the design of generative models of biomolecular dynamics” *Journal of Open Source Software*, 8(89), 5523, (2023). DOI: 10.21105/joss.05523

4.1 Introduction

In Chapter 2, we have described how MD simulations yield atomistic information on the conformational landscape of proteins through numerical integration of the laws of motion. While MD enables obtaining key insight into biomolecular function, it is not a silver bullet: exhaustive sampling of processes such as folding or ligand binding usually lay beyond what can be routinely observed. Generative Neural Networks have been shown to be effective predictors of a protein's 3D structure from its sequence [111, 133]. Several efforts have also demonstrated that a neural network trained with MD conformers can learn a meaningful dimensionality reduction of the data, usable for reaction coordinate definition [44, 134], or driving conformational space sampling[98, 135, 136]. In this context, we have presented, in Chapter 3, generative neural networks capable of producing protein conformations based on small pools of examples produced by MD. The issue is that developing a machine learning model to study biomolecular dynamics is a lengthy process. This requires setting up means of transforming conformational space data into tensor data submittable to a model, as well as assessing a model's quality (e.g., in terms of their energy or according to structural descriptors).

In this chapter, we present *molearn*, a Python framework facilitating the implementation of generative neural networks learning protein conformational spaces from examples obtained via experiments or MD simulations. This framework is broken down into a number of submodules that provide support for each step in a typical workflow. Firstly, for dataloading, we provide methods to parse molecular conformation files and transform them into tensors suitable for training. We also provide a number of pre-implemented models ready to be trained or to be subclassed to create custom models. We have also produced an improved implementation of the physics loss function discussed in Chapter 3, that integrates OpenMM [61] as the core physics engine. We provide a number of customisable classes that abstract away much of the boilerplate code required for training, validation, logging, and checkpointing models using PyTorch. Finally, once a model is trained, we have provided a number of rapid methods for gathering metrics defining the quality of generated protein structures. Class diagrams for molearn are given in Figures 7.1 and 7.3

Molearn is fully documented and comes with a series of examples, usable as-is, to train and analyse a neural network. Tutorials on neural network analysis are also available, including a Graphical User Interface (GUI) to directly interact with a trained neural network (see Figure 4.2).

4.2 Package architecture and features

In *molearn*, we have subdivided the process of creating, training, and analysing a machine learning model of protein dynamics into a series of Python submodules. These are schematically represented in Figure 4.1 and described in the following subsections.

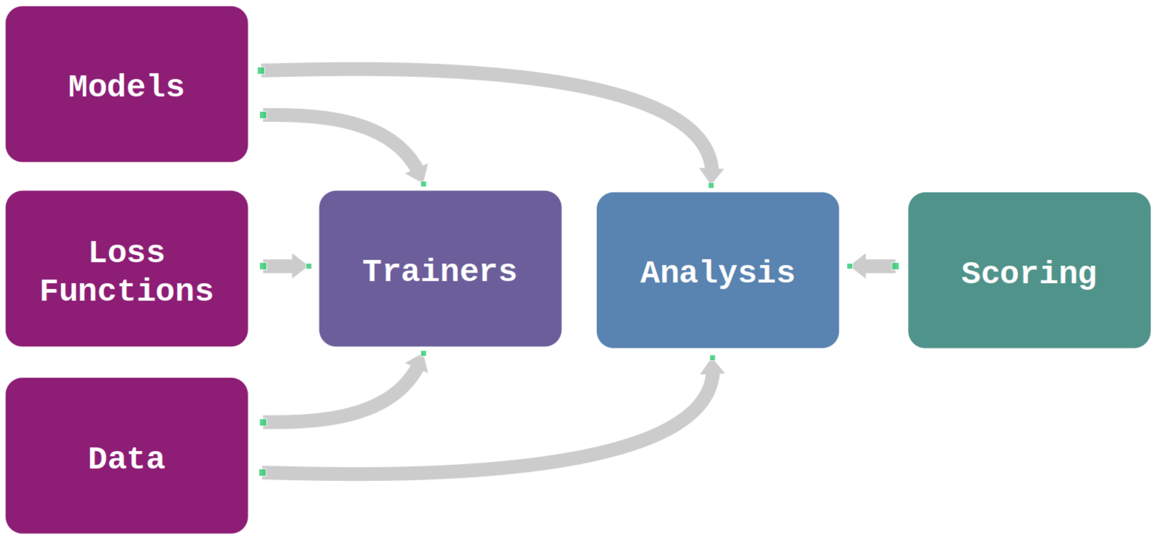


Figure 4.1: Overview of the *molearn* submodules with a typical workflow.

4.2.1 Data Loading

Molearn contains a `data` submodule for the loading and manipulation of input data. We use biobox[137], a Python package we co-authored, to parse molecular conformers in `PDB` format and extract the desired subset of atom coordinates into an array. We provide methods that convert this array to a PyTorch `Dataset` or `Dataloader` after performing a number of transformations, including normalising, centring, and axis permutation. In addition, we provide simple methods for extracting information about the inputs and for other tasks, such as splitting a dataset into training and validation sets. In total, we have abstracted away this otherwise boilerplate code to as little as a couple of lines of Python.

4.2.2 Models

Assuming not all biomolecular researchers have an intimate knowledge of ML architectural design with PyTorch, we thought it prudent to provide a range of pre-implemented models. These models can be subclassed to be customised or used as they are. Otherwise, entirely custom models can be incorporated into our framework by subclassing `torch.nn.Module` and implementing the `encode` and `decode` methods, responsible for taking input data and returning an output with correctly shaped and sized tensors.

4.2.3 Physics Loss Function

Our implementation of a physics-based loss function presented in Chapter 3 is written only in the high-level PyTorch Python API. This implementation was designed to be used with only the non-hydrogen backbone atoms and the β-carbon. With only this subset of atoms, the indexes of bonded atoms are always either adjacent or separated by one other atom. We took advantage of this in the PyTorch API by simply calculating all bond

Protein Size	Torch Loss			OpenMM Loss		
	Ψ_B	$\Psi_{B+RepNB}$	Ψ_{B+NB}	Ψ_B	Ψ_{B+NB}	$\Psi_{B+SoftNB}$
all 437 residues:						
Backbone	6.71	14.06	18.19	0.90	1.09	1.09
Heavy	50.19	63.20	72.65	0.92	1.20	1.19
217 residue crop:						
Backbone	4.37	5.45	6.28	0.78	0.78	0.84
Heavy	40.83	42.74	44.84	0.80	0.88	0.9
110 residue crop:						
Backbone	3.89	4.10	4.23	0.49	0.56	0.57
Heavy	46.22	45.98	46.42	0.66	0.67	0.68

Table 4.1: Average Execution time for the calculation of the energies and forces of all 4420 frames of MurD in batches of 16. Proteins of sizes other than the full 437 residues were simulated by cropping the dataset to the first 217 and 110 residues, respectfully. Execution time for atom selections of both Backbone atoms (atoms = CA, C, N, CB, O) and Heavy atoms (All non-hydrogen atoms) were tested at multiple levels of included physics. For the Torch loss function, the execution time for bonded interactions Ψ_B, bonded and repulsive non-bonded interactions $\Psi_{B+RepNB}$, and bonded and all non-bonded interactions Ψ_{B+NB} are recorded. For the OpenMM loss function, the execution time for bonded interactions Ψ_B, bonded and non-bonded interactions Ψ_{B+NB}, and bonded and a soft repulsive non-bonded interaction $\Psi_{B+SoftNB}$ are recorded.

distances between atom i and atoms $i+1$ and $i+2$, and then masking off non-bonded atoms. Angle and torsion energies can be calculated similarly, calculating all possible angles and torsions between locally indexed atoms and masking off those that are unnecessary. The forces associated with these are calculated using the PyTorch autograd differentiation engine. While this approach was sufficiently efficient for the purpose of demonstrating our method, it scaled poorly as additional side-chain atoms are added. Our crude implementation of non-bonded potentials was also very inefficient in terms of GPU memory and compute. To facilitate the maintenance and further development of our methods, we opted to explore integrating OpenMM as the engine for calculating the forces and energies.

Unfortunately, OpenMM does not implement any methods that would grant direct access to the calculated forces and energies held in GPU memory. In the normal operation of the OpenMM package, it is not expected that anyone would want to directly access any quantities on the GPU, and therefore, there is no interface that directly exposes them. The interface for accessing OpenMM system energies and forces performs a GPU-CPU memory copy, which in our use case would be immediately followed by performing a CPU-GPU memory copy to a Torch tensor. To perform a GPU-GPU memory copy of the forces, we chose to implement an OpenMM plugin consisting of a non-functional OpenMM integrator class that performs a GPU-GPU memory copy from the OpenMM internal force representation into an array associated with a PyTorch tensor. We can then create a custom PyTorch function where the forward pass returns the OpenMM energies, and the backward pass injects the OpenMM calculated forces as the gradient.

Similarly to OpenMM itself, our plugin consists of a C++ backend using SWIG[138] to generate a wrapper for a python interface. Within *molearn*, we use `TorchExposedIntegrator` from the plugin along with standard OpenMM tools to create `ForceField` and `Simulation` instances for our desired system. To inject OpenMM forces and energies into PyTorch, we create a custom `torch.autograd.Function`, where the forward pass returns the energy and the backward pass returns the force. This is all wrapped into a single `torch.nn.Module` subclass for ease of use. In benchmarks (see Table 4.1) this OpenMM-based loss function provided a $10\times$ speed-up when using only the heavy backbone atoms and a $50\times$ speed-up when all heavy atoms are included, compared to our original implementation.

4.2.4 Trainers

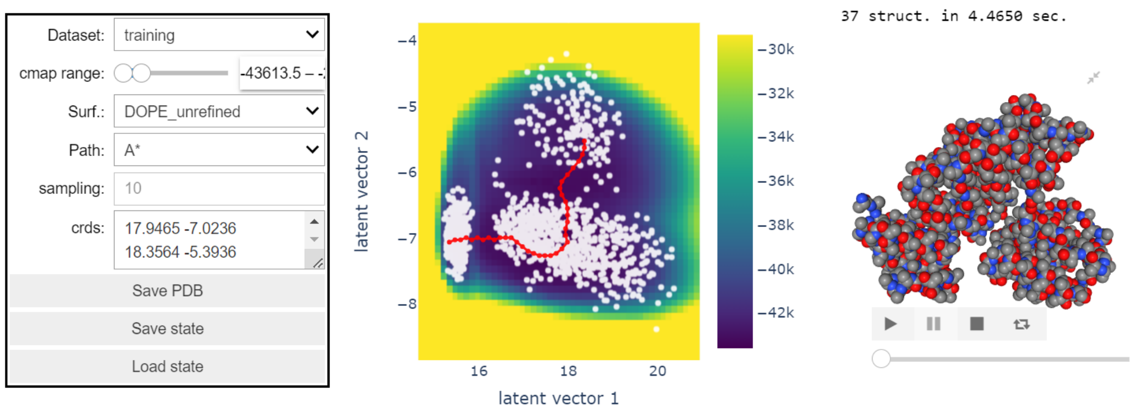
The process of training a PyTorch model typically involves a substantial amount of boilerplate code, a relatively thorough understanding of ML, and a good grasp of PyTorch itself in order to do basic tasks. We envisaged a situation in which using *molearn* might be a prospective user's first foray into using PyTorch specifically or ML generally. With this in mind, we have implemented a number of trainer classes that abstract away almost the entire training process. The trainers handle setting up the optimisation protocol, integrating the physics loss function, the training and validation of the model, checkpointing the model, and collecting and saving statistics about the training process. It is usually advisable for ML beginners to manually implement the boilerplate code so as to encourage a thorough understanding of the underlying workflow. However, *molearn* is likely to attract users simply wishing to apply the methods presented in Chapter 3 to their own work without a desire to use *molearn* as a platform for method development. We aimed to fashion *molearn* as both a development platform for efficiently developing our own methods and also as a toolset to cater to those wishing to use our tools. In order to enable rapid development of methods, we have followed a similar framework philosophy as PyTorch Lightning[139], in which we take advantage of object oriented programming. The addition of new functionality can be achieved by simply subclassing one of the provided trainers and modifying individual relevant methods to inject or modify their behaviour.

4.2.5 Scoring

There are a variety of packages that provide methods for judging the quality of protein structures. In particular, in Chapter 3.3 we have previously used the Discrete Optimised Protein Energy (DOPE) from Modeller [125] and the `ramalyze` submodule from cctbx [140] for calculating the Ramachandran statistics for a protein model. The typical workflow for using these packages is to either start with a PDB file or to have produced the model within the package,s ecosystem. Our neural network models simply produce tensors that correspond to the Cartesian coordinates of the protein's atoms, which are often incomplete. Saving our structures to PDB files only to reload them in a separate package is inefficient at best. Within the *molearn* `scoring` submodule, we provide methods to significantly improve the speed of these calculations. We take advantage of the fact that we are typically calculating statistics for many different conformations of the same protein, in which case atom types and positions in a tensor remain the same for all conformations. We can therefore create a single instance of the modules from Modeller and cctbx and simply inject different coordinates for atoms as required. In the case of Moddeler's DOPE score, this process is complicated by the fact that Modeller reorders atoms on import, and DOPE score operates on structures with all heavy atoms present. This required us to create a map from the indexes of atoms in our tensor to the Modeller internal representation and to use Modeller's methods to rebuild missing atoms before calculating the DOPE score. One final tactic to speed up these methods that we have employed, is to wrap them in a Python `multiprocessing` module to parallelise the calculation of these statistics.

4.2.6 Analysis

Once a model is trained, it is important to gather metrics defining the quality of the protein structures it can generate. Within *molearn*, we provide an `analysis` submodule that, in addition to producing the statistics from the scoring tools discussed in Section 4.2.5, implements methods to quickly quantify structure quality in terms of root mean square deviation between the coordinates of atoms in the input and output, as well as user-defined functions. Analysis data is returned in the form of NumPy arrays [141] for ease of manipulation and plotting. We also provide a graphical user interface (GUI), enabling the visualisation of neural network latent space, and generation of interpolations between states, visualised in an interactive 3D panel by a combination of MDAnalysis[142] and NGLview [143]. This GUI, which when used within a Jupyter notebook [144], would produce something akin to Figure 4.2. It should improve the accessibility of this work to non-specialists, as well as providing a visual mechanism with which to directly interact with our neural networks.


Dataset: training
cmap range: -43613.5 – -
Surf.: DOPE_unrefined
Path: A*
sampling: 10
crds: 17.9465 -7.0236
18.3564 -5.3936

```
37 struct. in 4.4650 sec.
```

Figure 4.2: *molearn* analysis tools include a graphical user interface, enabling the on-demand generation of protein conformations. The panel on the left controls how the neural network latent space is presented, the central panel is a Plotly interactive graph displaying the latent space, and the panel on the right is a 3D representation of an interpolation through the latent space supported by NGLview.

4.3 Delivery

Molearn has been fully open-sourced under a GNU General Public Licence. The source code is hosted on GitHub and can be accessed at `https://github.com/Degiacomi-Lab/molearn`. In addition, *molearn* is available via conda-forge for easy installation and integration into existing workflows using conda. Our OpenMM plugin, discussed in Section 4.2.3, has been distributed by the name openmmtorchplugin on conda-forge and placed as an upstream dependency of *molearn*. We have produced a full documentation of the methods in *molearn*, which can be found hosted at `https://molearn.readthedocs.io`. *Molearn* comes with a series of examples, usable as-is, to train and analyse a neural network. This is in addition to the examples mentioned in Section 4.2.4, which demonstrate how provided methods may be modified and extended. These examples can be found within the main GitHub repository. We have also produced Jupyter notebook tutorials detailing how to interact with a trained neural network and analyse its performance, and showcasing via a GUI how the latent space can be interrogated to generate protein conformers. These notebooks are hosted on a separate GitHub repository that can be found at `https://github.com/Degiacomi-Lab/molearn_notebook`

4.4 Conclusions

Molearn is an open-source, modular, and extensible software aimed at minimising the barrier to entry for using the methods described in Chapter 3, while also providing a more future-proof platform for further development of these methods. *Molearn* is designed to be accessible to users with minimal experience with ML and PyTorch, yet it has also been designed with extensibility in mind to cater to advanced users. The integration of OpenMM into *molearn* has provided an immediate performance gain, while also avoiding the burdensome task of developing an entire molecular mechanics engine in PyTorch. Currently, OpenMM is only used in the training of neural networks; however, further development of the plugin could open up avenues for our neural networks to also interact with an OpenMM Simulation as a modified integrator or as a custom force field, similar to other plugins such as OpenMM-Torch[145]. This could open up the possibility of performing online training, wherein the network is trained in parallel to an MD simulation.

In Chapter 5, we detail a number of examples of further method development that were enabled and expedited by the introduction of *molearn*.

Chapter 5

Model development using the *molearn* framework

5.1 Introduction

The primary limitations of the work in Chapter 3 were speed and stability of the network. Although we could overcome stability issues by using a very low learning rate, this came at the cost of long training times and poor sampling of the network's functional space. Training speed was further affected by the calculation of the physics potential, which was done crudely within PyTorch itself in a manner that was inefficient, difficult to maintain, and complicated to extend. Our methods were therefore an unsuitable platform on which to build future work. Our methods worked well on the protein MurD, a relatively rigid protein whose conformational space is dominated by movements around a single hinge. However, tests on four additional proteins yielded mixed results in terms of performance. In particular, we were unable to generate satisfactory interpolations or landscapes with the protein SurA, which, while a similar size to MurD, exhibits significantly more complex dynamics.

In this Chapter we present unpublished work that tackles these issues. We make use of the new physics-based loss function leveraging on the third-party molecular dynamics engine OpenMM [61] that, as described in Chapter 4.2.3, is highly efficient and extendable. In Section 5.2 we benchmark our current CNN architecture in a number of new scenarios including; incorporating different variants of the new loss function (see Section 5.2.1), testing how our networks would perform on multimers (see Section 5.2.3), and testing how well the network can handle the inclusion of side chain atoms (see Section 5.2.2).

In Section 5.3, we profile modified and entirely new network architectures on the more difficult case of the SurA protein. In Testing different machine learning model architectures, namely a modified FoldingNet [146] (see Section 5.3.2), we discover that the stability issues observed in our original CNN were specific to our implementation and not a feature of all networks. By updating our design choices accordingly, we produce a new CNN model enabling the use of much larger learning rates, dramatically improving the performance of the network (see Section 5.3.1). We then systematically investigate the effect of latent space size, one of the primary limiting factors in performance, on our new neural network architecture (see Section 5.3.1).

5.2 Benchmarking *molearn* models

5.2.1 Modified OpenMM Physics

In Chapter 3.3.1, the effect of the addition of force field terms was discussed. The addition of bonded force field terms (Ψ_{Bonded}), including bond, angle, and dihedral constraints, improved the quality of the generated structures and interpolations. The addition of warped non-bonded constraints, softened by an ELU activation as discussed in Section 3.2.3, had no statistically significant effect on network performance.

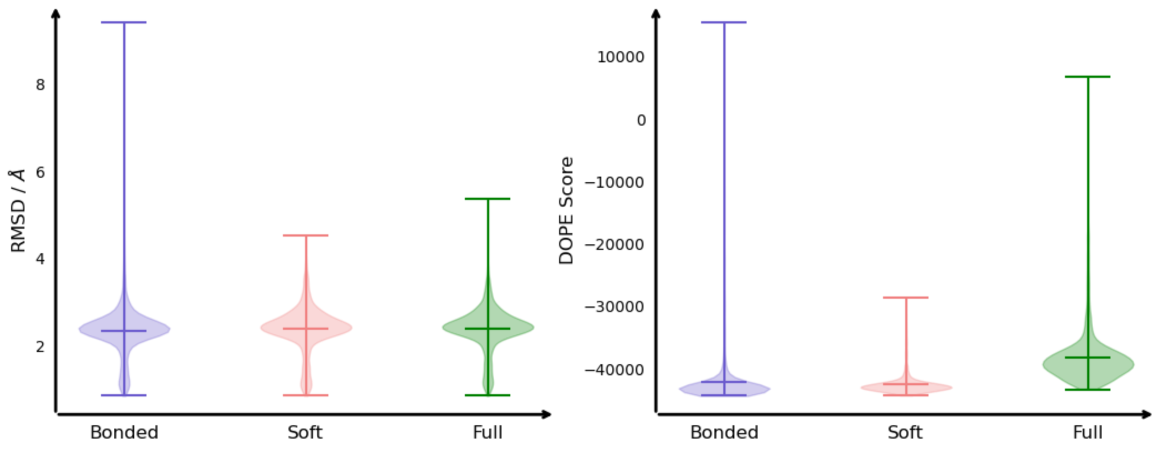


Figure 5.1: The performance of CNN networks, trained with Open and Closed state conformations of MurD, on a test set of structures featuring a transition between the two states. Bonded networks were trained with only Ψ_{Bonded} in the loss function. Soft networks were trained with $\Psi_{\text{Bonded}} + \Psi_{\text{Soft}}$. Full networks were trained with unsoftened $\Psi_{\text{Bonded}} + \Psi_{\text{Non-Bonded}}$. Left: The distribution of RMSD values between the inputs and output of the network. Right: The distribution of DOPE scores for reconstructed test set structures.

There are a variety of reasons why using an unsoftened non-bonded potential ($\Psi_{\text{Non-Bonded}}$) would be unsuitable. Firstly, the network directly generates new structures rather than using an iterative approach such as MD, where a situation involving overlapping atoms can be avoided by taking small enough steps. Secondly, while the training of the network is iterative, the use of high learning rates and a stochastic optimiser means avoiding overlapping atoms is not feasible. Using $\Psi_{\text{Bonded}} + \Psi_{\text{Non-Bonded}}$ to train a network results in worse performance than simply using Ψ_{Bonded} alone (see Figure 5.1). A similar problem occurs when constructing the input to an MD simulation starting from an incomplete structure with missing atoms. The missing atoms can be added by inserting atoms based on templates or placing them at the most likely position based on geometry from existing atoms. At this stage, there is a non-zero probability of one or more inserted atoms being placed so close to others that there is insufficient numerical precision to perform energy minimisation incorporating a Lennard-Jones potential. Programs such as pdbfixer[147] get around this problem by using a soft repulsive non-bonded potential for the energy minimisation:

$$\Psi_{\text{Soft}} = \frac{1}{kr^4 + 1} \tag{5.1}$$

where k controls the strength of the potential.

Using $\Psi_{\text{Bonded}} + \Psi_{\text{Soft}}$ to train the networks instead of Ψ_{Bonded} seems, similar to our previous attempts of softening the non-bonded function, to have no statistically significant effect on the performance of the networks. However, as it does not seem to be negatively affecting the performance of the network, we use this softened potential for all further work.

5.2.2 All Heavy Atoms

In Chapter 3, we opted to train the neural networks on C, Cα, N, and O atoms of the backbone and a single atom, Cβ, of the side chain. We argued this was sufficient to describe the fold of the protein and the orientation of each side chain. Additionally, our PyTorch implementation of bonded physics restraints relied on bonded atoms being co-located as close as possible for maximum efficiency. Programs like Modeller [148] are capable of constructing and packing the side chains to a physically plausible state and so different selections of atoms were not trialled in our published work.

Ultimately, the more missing atoms we have when training our network, the less meaningful our physics-based loss function is. Additionally, there are scenarios in which networks that can handle different sets of atoms may be wanted. There are legitimate questions about whether a CNN using kernels with size 4 will perform equally well when bonded atoms are greater than 4 indices apart, as would be the case with most side chains. The new implementation of our physics-based loss function using OpenMM is vastly more computationally efficient when including the side chain atoms (see Table 4.1), allowing us to incorporate more atoms without sacrificing significantly on training time.

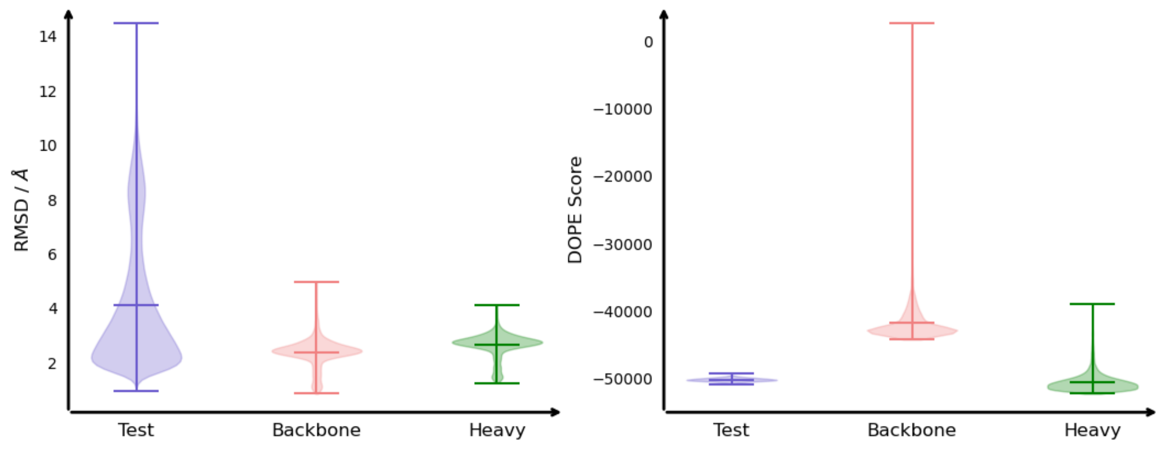


Figure 5.2: RMSD (Left) and DOPE scores (Right) for networks trained on different atoms selections of MurD. The Backbone atom selection (red) includes the non-hydrogen backbone atoms and just Cβ from the side chain. The Heavy atom selection (green) includes all non-hydrogen backbone and side chain atoms. As a reference (blue), we have provided the distribution of RMSDs between every test set structure and every other test set structure for the left plot, and DOPE scores of the test set for the right plot.

When training the network with all heavy atoms, we find a small but statistically significant increase in the average test set RMSD between input and output, from 2.4 Å to 2.6 Å, likely explained by some combination of the greater flexibility of side chain atoms and an increase in the number of atoms from 2145 to 3286. However, there was a dramatic improvement in DOPE scores of the output of the network when including heavy atoms, with the network producing structures with $\sim$-50300 compared to $\sim$-41800 for backbone only (see Figure 5.2). The average DOPE score for structures produced by the all-heavy-atom networks was comparable to those in the test set itself, albeit with a larger standard deviation (1500 compared to 230 for test set). The improvement in DOPE scores seen between training networks with backbone and heavy atoms is likely due to the lack of need to reconstruct the missing atoms in order to calculate DOPE. If we take the test set, remove the side-chain atoms, and then reconstruct them to calculate DOPE, the average DOPE score is -41200, comparable to the structures produced by the backbone networks.

5.2.3 Multimers

All of our work to date has been on monomeric protein structures; however, it is well established that the majority of proteins carry out their biological tasks as part of a protein complex.[149] Adapting our methods to multimers could present a number of challenges. Firstly, our use of a 1D CNN architecture relied on the assumption of adjacent input coordinates being spatially adjacent, also. In the case of multimeric systems, there will be a large spatial interruption between the C-terminal of one chain and the N-terminal of the next chain. We can test the effect of these spatial interruptions by randomly introducing spatial interruption into the 1D representation of a monomeric protein. If we take MurD and randomly cut it into subunits, then reorder these subunits, we end up with an identical point cloud but with its 1D representation containing interruptions. When training networks on this new representation of the data, we found no statistically significant difference in the RMSD (see Figure 5.3). It is then likely that our network is relatively unaffected by the presence of coordinate interruptions.

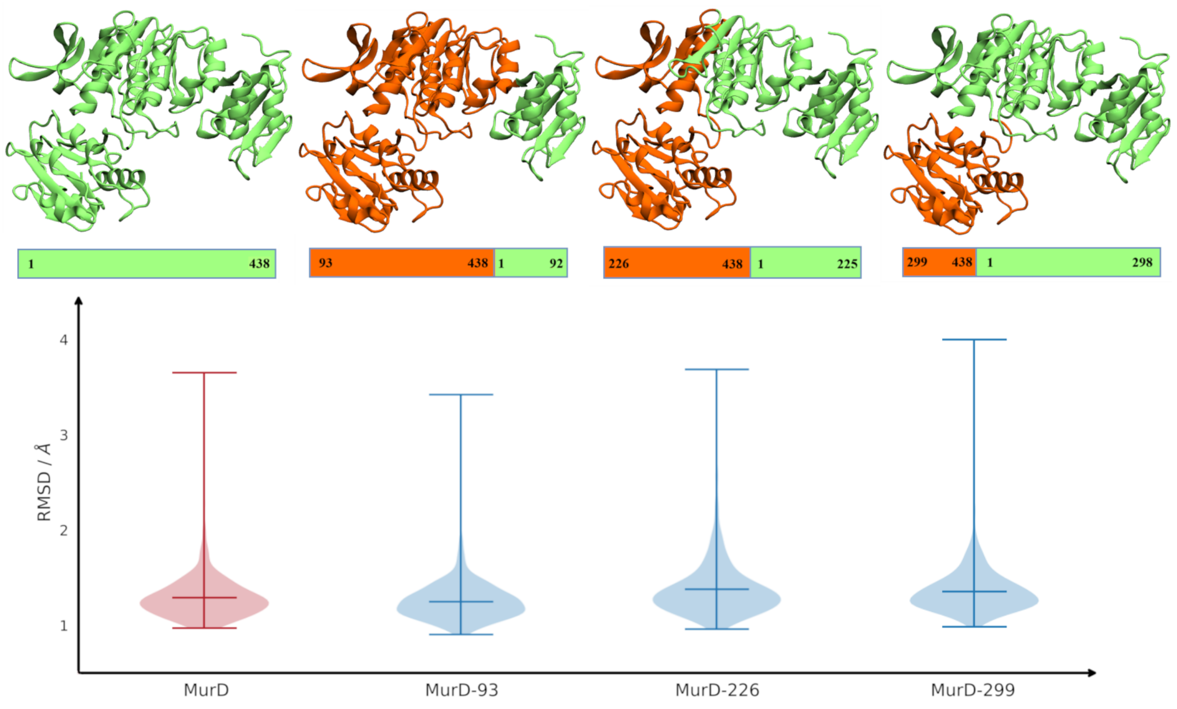


Figure 5.3: The introduction of coordinate interruptions to MurD by cyclic permutation of the 1D point-cloud representation. Top: We depict the original MurD representation alongside a cyclic shift of 93, 226, and 299 residues. Bottom: Corresponding RMSD performance of CNN networks trained on these representations.

Homo-multimers present a uniquely curious situation for dimensionality reduction because of their inherent symmetry. We would therefore expect more redundancy in a homo-multimer than in an equivalent hetero-multimer, which, aside from coordinate interruptions, would be nearly indistinguishable from a large monomeric protein to a neural network. We have simulations of the hexameric form of the p24 capsid protein, which is depicted in Figure 1.1. This hexamer can be cropped to simulate a monomer, dimer, through to the full hexamer. When training our network on these systems, we find that the RMSD increases linearly with the number of subunits, with a gradient of 0.447. Doubling the number of atoms does not result in an RMSD performance that is twice as bad. The DOPE scores of structures produced by these networks were all comparable to the input structures, indicating the networks were well trained. By training with additional subunits, the number of atoms is increasing. Due to the size of the latent dimension being held constant for each network, the data compression factor increases with increasing number of subunits. There is potential for the latent dimension bottleneck to be a constraining factor for performance here, so we investigate the effect of latent space size in Section 5.3.1.

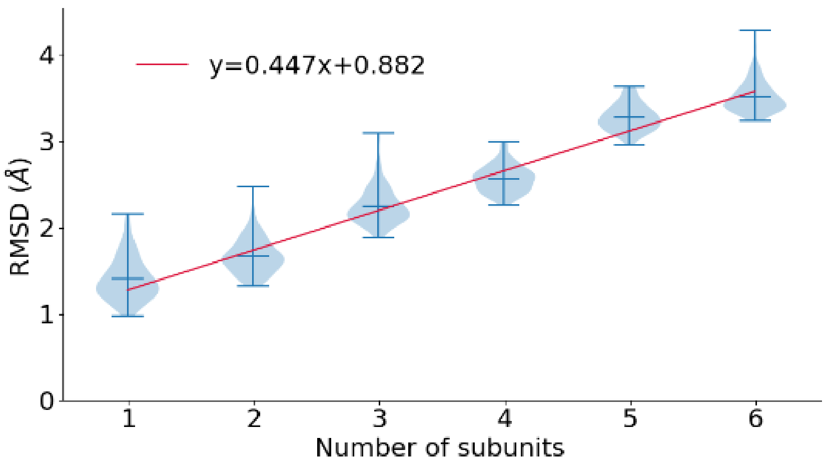


Figure 5.4: RMSD of networks trained with increasing numbers of subunits of the p24 hexamer that make up the HIV-1 capsid.

5.3 Profiling ML Model architecture

5.3.1 CNN architecture

Latent Sigmoid

The network and training protocol used in our work published in Ramaswamy et al. [58] and covered in Chapter 3 was found to be highly susceptible to getting stuck in a poorly optimised state early on in training. When analysing the projected latent spaces of these poorly optimised networks, it was found that one or more latent dimensions had collapsed to a unitary value. There are a number of potential causes, including over-regularisation, vanishing gradients, poor architecture design, or a poorly conditioned loss function. We overcame the problem by using very low learning rates (10^{-5} to 10^{-6}), careful control of the contribution of the physics loss to the final loss function, softening the non-bonded contributions to the physics loss function, and simply rerunning with a new initialisation any networks that failed to optimise well. The root cause of these degenerate latent dimensions was not explored further at that time. Using *molearn*, we later experimented with alternative network architectures (such as FoldingNet, discussed in Section 5.3.2) which did not suffer the same instability. We were led to the origin of the problem by noticing a reduced propensity for latent collapse with our original network when training networks with more than 2 latent dimensions (see Section 5.3.1). Additional dimensions were acting as redundancy should one dimension collapse, suggesting that gradient flow through the penultimate layer of the encoder was not an issue. The primary difference between the final layer of the encoder in our original network and the other networks we had tested was the use of a sigmoid function to map the latent space to between 0 and 1. The sigmoid function‘s primary purpose in the original network was to create a bounded latent space that was easier to visualise with grid sampling. Thus, this could be removed without too much issue, as it is quite trivial to produce adequate grid sampling bounds from the projection of known structures. On removing the latent sigmoids in the CNN, the latent collapsed disappeared entirely. For very large positive or negative values, the sigmoid function approaches its asymptotes (0 and 1), which would mean the gradient would become very small. If at the initialisation of the network or at any point during training we were in this regime, we would run into a form of the vanishing gradient problem, and the network would not optimise.

Our results show that using a standalone sigmoid to bound the space introduces instabilities in the training process. For this reason, in our following work, we will not use it. Assuming our speculation of the root cause is correct, we can suggest potential solutions, such as increasing the L2 regularisation of the final layers of the encoder or incorporating some form of normalisation immediately before applying the sigmoid function.

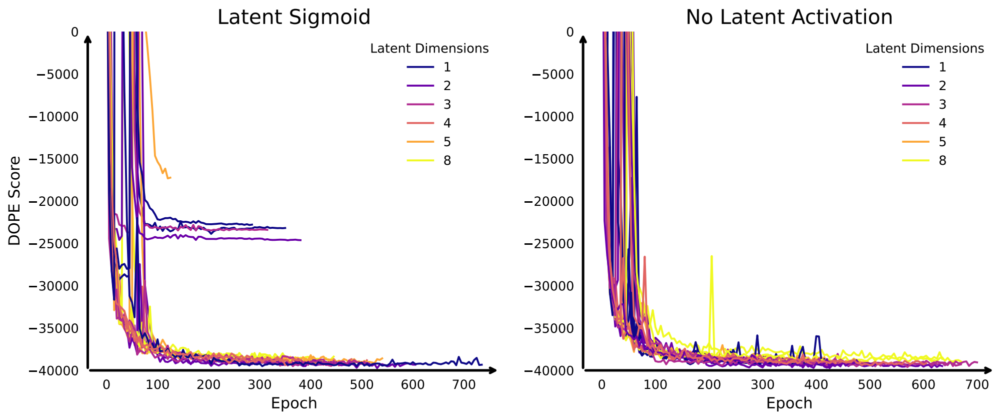


Figure 5.5: DOPE scores during training of networks on MurD for a variety of latent dimension sizes. Left: Autoencoders including a sigmoid activation function on the final layer result in occasional early plateauing of the loss function. Right: No activation function results in more consistent training.

Latent Space Size

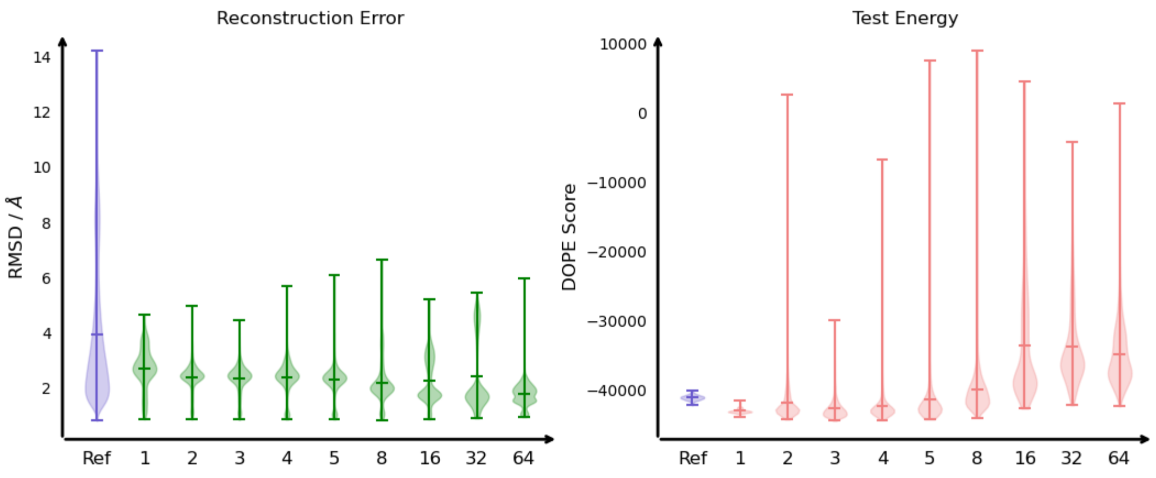


Figure 5.6: The performance of networks trained on MurD with different latent space sizes. Left: The reconstruction error of the test set structures (which contain a transition between the open and closed state) measured by the RMSD between the input and output to the network (green). Right: Test energy is measured as the DOPE scores of the output of the network where the input was the test set structures. As a reference (blue), we have provided the distribution of RMSDs between every test set structure and every other test set structure for the left plot, and DOPE scores of the input test set for the right plot.

The bottleneck of our autoencoder, the latent size, has so far been fixed at two dimensions. Two dimensions are desirable, as it is trivial to visualise the full latent space landscape by simply performing a grid search to generate a 2D image, such as in Figure 3.7. For proteins such as MurD, it seems that even as few as one dimension is sufficient to capture the majority of the global motions, including low energy estimates of transition structures of the test set between the open and closed states. Our results, given in Figure 5.6, show that there are meagre improvements to be gained from increasing the number of dimensions. On the contrary, at larger numbers of dimensions (> 8), we see a drop in the quality, as measured by DOPE, of structures associated with transitions. We speculate that this is due to the sampling of interpolated structures during the training process, to which we apply our physics-based loss function, becomes increasingly sparse with an increasing number of dimensions. The constraint of interpolations being of minimal energy effectively becomes weaker with an increasing number of dimensions.

MurD represents a relatively simple system, being particularly rigid and its global motions dominated by movements around a single hinge region. However, we were unable to generate satisfactory interpolations or landscapes with the protein SurA, which, while a similar size to MurD, exhibits significantly more complex dynamics. When looking into the effect of latent size on network performance with SurA, we find a significant improvement in the networks with a larger number of dimensions (see Figure 5.7). In particular, it seems the network requires at least 3 latent dimensions before being able to consistently capture the majority of dominant conformations of SurA. After 3 latent dimensions, there ceases to be any significant improvements in reconstruction error, with the RMSD between inputs and outputs of the network only reducing by an average of 0.020 ± 0.004Å for every additional dimension. The energy of the generated transition structures improves by a modest -180 ± 70 to the DOPE score per additional latent dimension above 3 dimensions.

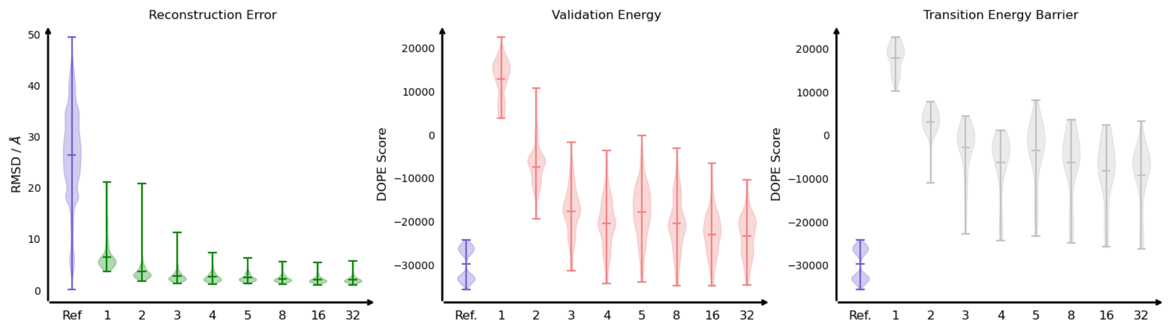


Figure 5.7: The performance of networks trained on SurA with respect to different latent space sizes. Left: The reconstruction error of the validation set structures (green), measured by RMSD between the input and output of the network. Middle: Validation energy (red) is measured as the DOPE scores of the output of the network where the input was the validation set structures. Right: Transition structure energy (silver), as measured by the DOPE of the transition structures between 6 of the largest energy basins of SurA. For each of the 15 unique pairs of basins, we select structures associated with these basins and generate structures corresponding to linear interpolations in the network's latent space. Taking these interpolations as transition pathways, we calculate the DOPE score for each structure in the pathway and report the transition structure energy as the maximum DOPE score in each pathway. In each graph, we provide a reference distribution (Ref, shown in blue), which represents RMSDs between every validation structure and every other validation structure for the left plot, and DOPE scores of the input validation set for the middle and right plots.

Our results show that a more complex protein, or at least a protein exhibiting more complex protein dynamics, requires a larger bottleneck to capture the dataset. We expect this relationship between the complexity of the dataset and the number of latent dimensions required to achieve an arbitrary level of performance to hold for other proteins. These results are obtained with a $10\times$ emphasis on minimising MSE rather than physics. It would be interesting for future work to examine the interplay between these two training criteria. For example, would a stronger emphasis on minimising physics lead to a proportional decrease in reconstruction error, and where would the optimal balance between the two lie?

As the dimensionality of the latent space increases beyond two dimensions, visualising the structure of this space becomes increasingly challenging. While this limitation may not be critical for all applications, particularly those focused on performance metrics rather than interpretability, it does present significant difficulties in understanding, sampling, and analysing the latent space. These challenges can lead to issues such as reduced interpretability, potential overfitting, and difficulties in ensuring meaningful latent representations.[150]

By choosing regularised autoencoders instead of probabilistic models like VAEs, we lack the structured latent space that facilitates direct random sampling. As a result, we have relied on grid sampling or interpolation methods to explore the latent space. However, grid sampling becomes increasingly impractical as the number of latent dimensions grows, due to the exponential increase in the number of samples required to adequately cover the space. This scalability issue highlights the need for alternative strategies that can effectively explore and utilise high-dimensional latent representations without relying solely on grid sampling.

5.3.2 FoldingNet

FoldingNet is a type of neural network architecture specifically designed for learning representations of 3D point clouds, introduced by Yang et al. [146]. FoldingNet was developed to address the challenge of effectively learning compact and informative representations of 3D shapes represented as point clouds, which are unordered sets of 3D points commonly used in computer vision and graphics. FoldingNet uses a graph-based encoder that enables it to handle unordered point sets. The decoder learns a folding operation that maps a 2D grid into a 3D point cloud. This approach is based on the idea that the surface of any 3D object can be transformed to a 2D plane and vice versa.

As a protein is effectively a 1D sequence folded into 3D space, we have created a modified FoldingNet that maps a 1D grid into a 3D point cloud. In this modified architecture, the decoder comprises a series of folding layers. Each layer takes an input code, replicates it m times, and concatenates it with a size m linear spatial vector, effectively serving as a form of positional encoding. This approach should allow the network to learn to transform a 1D sequence into a structured 3D representation, making it suitable for modelling complex structures such as proteins.

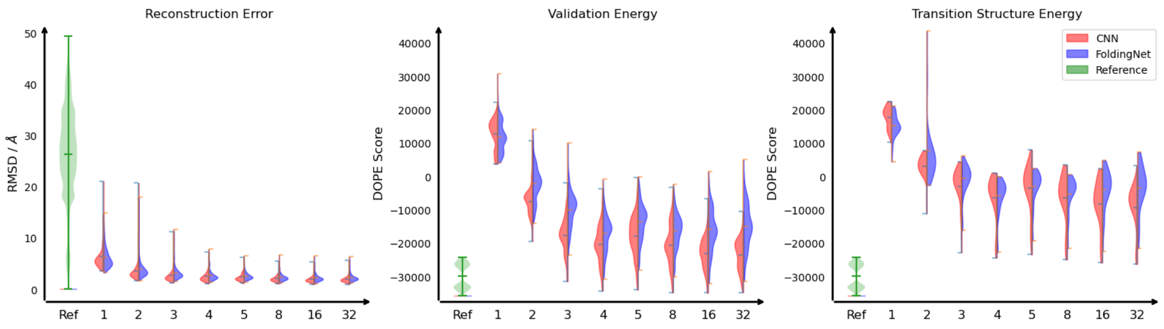


Figure 5.8: The performance of CNN and FoldingNet compared on the protein SurA for different latent space sizes. For each violin, excluding the reference (green), the left side (red) is the new CNN, while the right hand side (blue) shows the FoldingNet results. The reconstruction error (left), validation energy (middle), and transition structure energy (right) as presented are determined in the same manner as Figure 5.7. Reference values are represented in the colour green.

Our first tests with our FoldingNet exceeded the performance of the original CNN presented in Chapter 3. However, as discussed in Section 5.3.1, the poor performance and instability of the original CNN have been fixed by removing the latent sigmoid. Figure 5.8 shows the performance against the new CNN, in which the FoldingNet consistently performs slightly worse. Despite these early results, we expect that graph-based networks will scale better as the number of atoms grows and the complexity of interactions increases. This particular implementation will need further investigation to bring its performance in line with the CNN.

5.4 Sinkhorn Generators

So far in this work, we have exclusively used autoencoders regularised with physical constraints. For applications that require direct modelling of the data distribution and seek to avoid the degenerate distributions often produced by regularised autoencoders, Sinkhorn generators offer a compelling alternative. By leveraging Sinkhorn divergence, these generators facilitate efficient and reliable sampling of the latent space.

The definition and theory behind the Sinkhorn divergence (S_ϵ) was introduced in Chapter 2.10. We define a generator $G : \mathbb{R}^m \to \mathbb{R}^{n,3}$, which, given a prior latent distribution $z \sim p_z(z)$, capture the protein data space p_d by matching the output of the network to p_d. We choose the prior latent distribution here to be $\mathcal{N}(0, I)$. Using S_ϵ can be associated with blurred outputs in some contexts, arising due to the entropic regularisation applied in the Sinkhorn algorithm. We hope to counteract this blurring by adding our physics- based constraints Φ_{Soft} to enforce only low-energy structures. To train the generator we optimise the value function $V(G)$:

$$\min_G V(G) = \mathbb{E}_{z \sim p_z, \boldsymbol{x} \sim p_d} \left[S_\epsilon(\boldsymbol{x}, G(z)) \right] + \gamma \mathbb{E}_{z \sim p_z} \left[\Phi_{\text{Soft}}(G(z)) \right] \tag{5.2}$$

where γ represents the strength of the applied constraints. To more easily project samples from p_d into p_z space, we also define an encoder $E : \mathbb{R}^{n,3} \to \mathbb{R}^{(}m)$, which we optimise using $V(E)$:

$$\min_E V(E) = \mathbb{E}_{z \sim p_z} \left[\| z - E(G(z)) \|_2^2 \right] \tag{5.3}$$

We produce two sets of models: Sinkhorn-Φ_{Soft} and Sinkhorn. The latter is trained without physics by setting $\gamma = 0$. For the networks G and E, we simply used the decoder and encoder, respectively, from our CNN autoencoder networks implemented in *molearn*. We used a larger batch size of 256, the maximum that the memory of a 12GB NVIDIA TITAN Xp could handle. All other hyperparameters and training procedures remained the same as for our autoencoders.

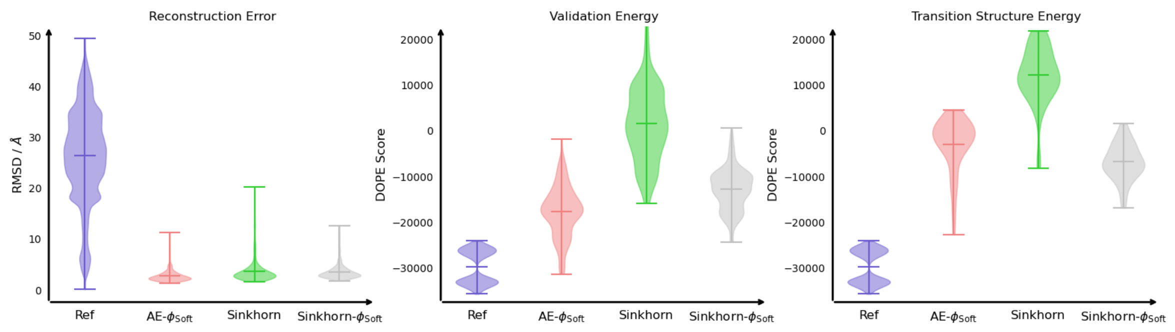


Figure 5.9: A comparison of the performance of traditional autoencoders with physical constraints (AE-Φ_{Soft} in red) with networks trained using Sinkhorn loss with and without physical constraints (Sinkhorn-Φ_{Soft} in silver and Sinkhorn in green, respectively). We measure reconstruction by the RMSD between input and output structures for the validation set, Validation Energy as the DOPE of the reconstructed validation structures, and Transition Structure Energy as the Maximum DOPE along interpolations between SurA energy basins.

We trained these Sinkhorn generators on the challenging SurA dataset, allowing the networks a latent space of three dimensions. Our results, given in Figure 5.9, show that this style of Sinkhorn generator is as capable as a traditional autoencoder at capturing the dataset, as measured by the RMSD between the input and output of the validation dataset. When trained without physics, the Sinkhorn network produces poor DOPE scores for both known and potential transition structures. The Sinkhorn-Φ_{Soft} networks produced protein structures of a similar quality to the traditional autoencoder networks (AE-Φ_{Soft} in Figure 5.9). There also appears to be a slight improvement in the quality of transition structures, although with a T-test p-value of 0.07, this improvement has poor statistical significance.

Our results show that Sinkhorn generators are a promising alternative to classical autoencoder architectures. A drawback of this method, however, is that it requires defining a prior on the latent space distribution. As we have seen in Section 5.3.1, for SurA, the number of latent variables is one of the major limiting factors in network performance, and any further constraints on the latent space would likely affect performance. Alternatives to normally distributed priors could be explored in the future.

5.5 Conclusion

In this chapter, we have worked on improving and profiling our methods presented in Chapter 3 using the *molearn* framework introduced in Chapter 4. Our physics loss function, now built upon OpenMM, was capable of significant improvements in the quality of generated structures. We introduced Φ_{Soft} in Section 5.2.1, which allows non-bonded terms to be included without resulting in large forces that otherwise fail to physically constrain the network generated structures. We investigated the ability of our networks to handle additional side-chain atoms (Section 5.2.2) and multimers (Section 5.2.3). While the networks could handle interruptions in proteins without issues, the larger number of atoms was likely the primary culprit behind a drop in performance when handling increasingly large multimers. By profiling the effect of latent size in Section 5.3.1, we demonstrate that the previous poor performance with the more difficult protein SurA was due to limiting the latent space size to two dimensions.

To go beyond the performance of our CNN autoencoder, we investigated the possibility of changing the machine learning model architecture or optimisation framework. In Section 5.3.2, we tested a new FoldingNet architecture, which ultimately performed worse than our CNN. Testing Sinkhorn generators as a promising alternative framework, we showed, excitingly, that physical constraints were successful at 'deblurring' the generated structures, producing more physically plausible structures and low-energy transition structures.

Chapter 6

Conclusions

The primary focus of this work has been the definition, implementation, and demonstration of physical constraints in the optimisation of neural network models for protein structure generation and latent space analysis. In Chapter 3,, we implemented, in PyTorch, methods for calculating the energies of incomplete protein structures with respect to Amber14 force field based terms. The PyTorch `autograd` machinery then allows the optimisation of neural network models with respect to these energies via backpropagation. We have demonstrated that these physical constraints, applied to interpolations between known structures, result in improved structure quality of generated structures, as measured by Ramachandran based scores and DOPE. In concert with a physics-based loss function, we introduced a convolution-based autoencoder, which we demonstrated to be capable of predicting possible transition paths between known states (Chapter 3.3.2) and transfer learning to multiple other proteins with fewer examples (Chapter 3.3.3). These models struggled to learn the conformational space of SurA, a protein featuring complex conformational dynamics. We found that these networks perform poorly in the extrapolatory rather than interpolatory regime, indicating generalisation only in the local conformational space. Thus, our models are promising for discovering transition paths between states, but are unsuitable, in their current form, for discovering entirely new states.

All our methods have been integrated into our new software package, *molearn* (see Chapter 4). *Molearn* serves two main purposes: Firstly, to lower the barrier to entry for the development and deployment of our advanced methods; and second, to provide a highly efficient and extendable platform for rapid development and innovation of new techniques. As part of *molearn*, we introduced a new OpenMM-based implementation of our physics-based loss function. This gives us the ability to integrate OpenMM tools into our methodology, use the significantly more efficient implementation of energy and force calculations, and shift some of the burden of software maintenance to a well established third-party.

In Chapter 5, we use the *molearn* framework to validate the new physics loss function (Section 5.2.1), determine that our CNN model is capable of handling side chains (Section 5.2.2), apply our methods to multimers (Section 5.2.3), profile and improve the existing CNN architecture (Section 5.3.1 and 5.3.1), introduce a new FoldingNet architecture (Section 5.3.2), and implement Sinkhorn-based training (Section 5.4). We determined that the primary limiting factor for our CNN implementation is the size of the latent space, and demonstrated the ability of physics-based constraints to effectively 'deblur' Sinkhorn-based optimisation. FoldingNet was ultimately less performant than our CNN implementation in the case of SurA, but we expect a simple profiling of more recent architectures on a broader range of test cases would quickly exceed the performance of our CNN.

The field of machine learning and neural network design is incredibly fast moving. The methods developed here, while relevant during their development, are rapidly becoming outdated. From an architectural perspective, we expect that to keep *molearn* up-to-date, we would need to investigate attention-based architectures [151], which have been shown to be highly scalable and effective across a broad range of tasks, including the image [152], audio [153], and textual [154] domains.

From a framework perspective, exploring the data space through a series of small denoising transformations, such as in diffusion models [155], presents an obvious direction for further research. However, integrating physical constraints into diffusion models could be challenging because these models primarily train and operate within a noisy internal space. Only the initial step of the noising process and the final step of the denoising process directly interact with the data space. The state-of-the-art models in most tasks are large multi-task models operating in a 'big data' context. In the field of protein structures, AlphaFold[111] is the quintessential example of such an approach. As demonstrated in Chapter 3.3.3, the effectiveness of transfer learning is evident, and we anticipate that training models on a large and diverse set of proteins will significantly reduce the generalisation error. This improvement is particularly expected in the extrapolatory regime, where our current methods struggle. The choices made when selecting proteins for inclusion in such a training set are likely to be critical for the performance characteristics of any model. There will be inherent biases in the conformations present in the available structural datasets due to the limitations, discussed in Section 1.2, of the methods used to obtain them. The impacts of these biases and potential mitigation strategies should be studied in order to maximise the effectiveness of our methods. For future studies, proteins could be selected based on criteria such as structural diversity, functional classification, or evolutionary relationships to encourage a well-balanced training set. Additionally, datasets incorporating experimentally or computationally derived dynamic ensembles (as we have done here), rather than single static structures, may help improve generalisation and mitigate biases.

Finally, the work presented in this thesis has numerous potential applications, including transition path sampling, identification of cryptic binding sites, and discovery of collective variables. For *molearn* to see wider adoption, it is essential to demonstrate the suitability of this approach for these specific applications and to expand the *molearn* toolset to enable seamless integration into the typical workflows of these fields. While *molearn* has already lowered the barrier to training and analysing neural networks for protein-related tasks, it is equally important to reduce the barriers to integration and deployment.

Chapter 7

Appendix

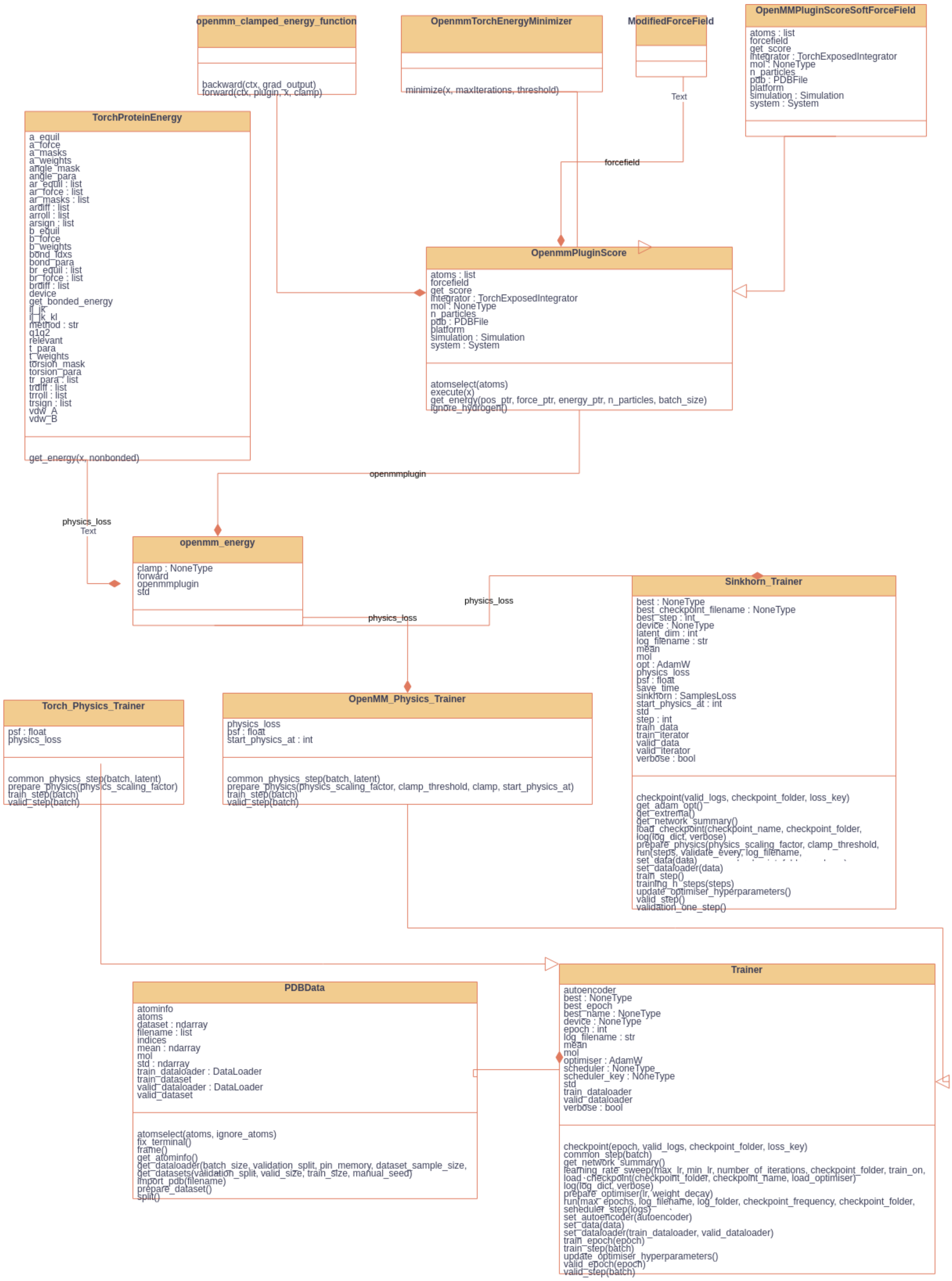


Figure 7.1: Class diagram for the trainers in molearn. This class diagram was automatically generated with pyreverse.[156]

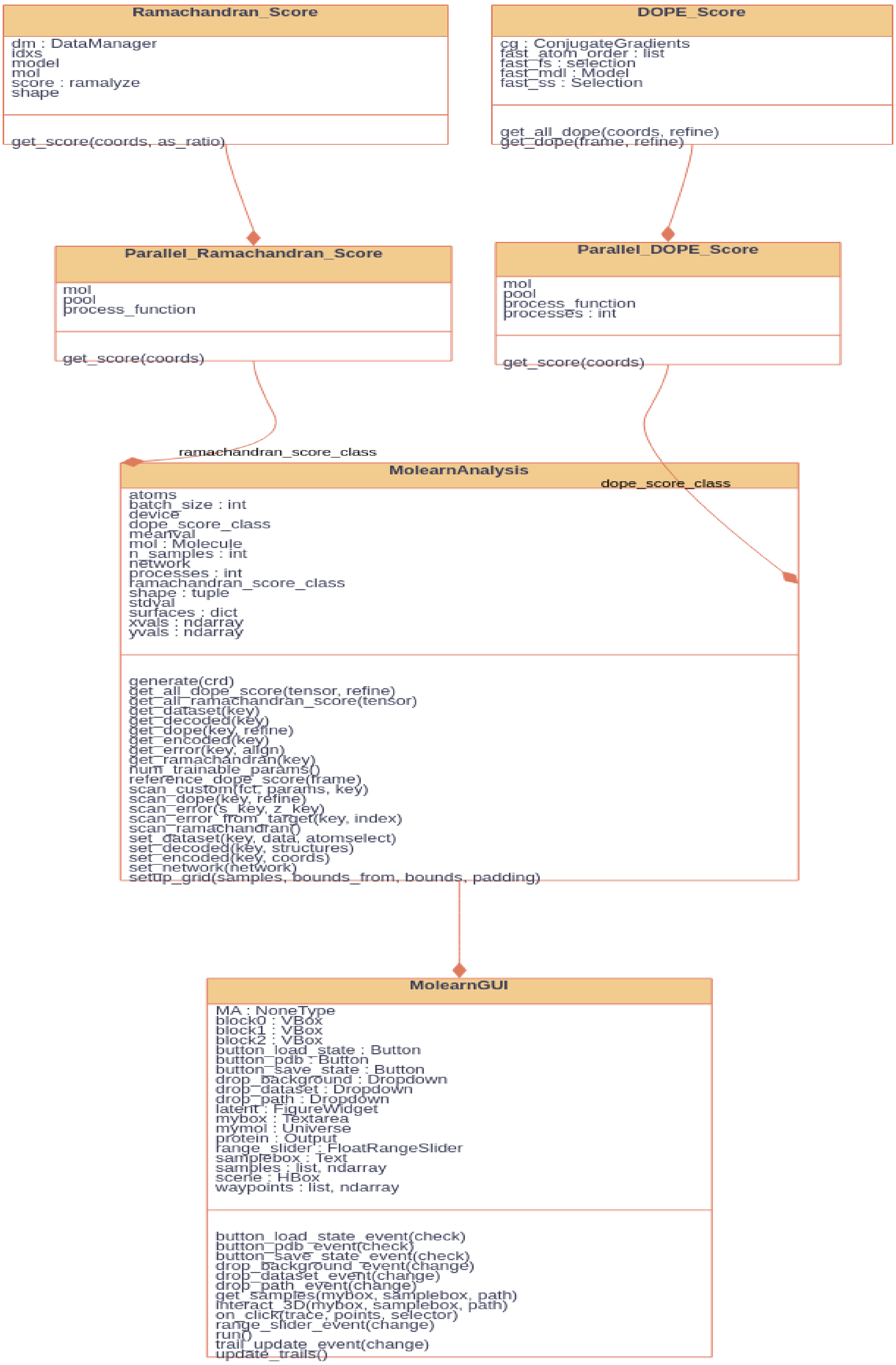


Figure 7.2: Class diagram for the analysis classes in molearn. This class diagram was automatically generated with pyreverse. [156]

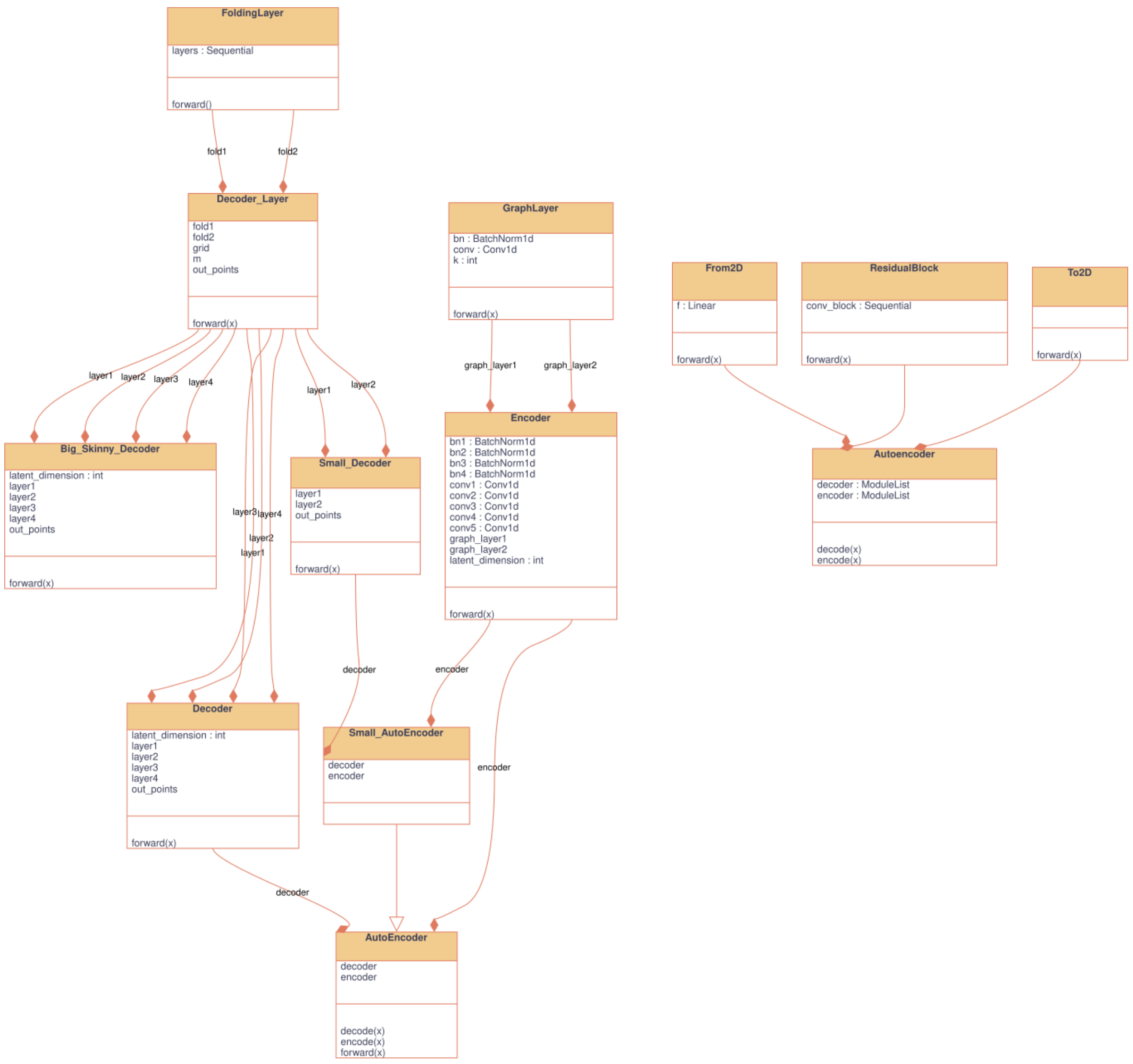


Figure 7.3: Class diagram for Neural networks provided in molearn. This class diagram was automatically generated with pyreverse.[156]

Chapter 8

Bibliography

[1] E. Arunan, G. R. Desiraju, R. A. Klein, J. Sadlej, S. Scheiner, I. Alkorta, D. C. Clary, R. H. Crabtree, J. J. Dannenberg, P. Hobza, H. G. Kjaergaard, A. C. Legon, B. Mennucci, and D. J. Nesbitt, “Definition of the hydrogen bond (IUPAC recommendations 2011)”, Pure and Applied Chemistry **83**, 1637–1641 (2011).

[2] S. C. C. vanderLubbe and C. FonsecaGuerra, “The nature of hydrogen bonds: a delineation of the role of different energy components on hydrogen bond strengths and lengths”, Chemistry – An Asian Journal **14**, 2760–2769 (2019).

[3] M. O. Sinnokrot, E. F. Valeev, and C. D. Sherrill, “Estimates of the ab initio limit for interactions: the benzene dimer”, Journal of the American Chemical Society **124**, 10887–10893 (2002).

[4] K. M. Makwana and R. Mahalakshmi, “Implications of aromatic–aromatic interactions: from protein structures to peptide models”, Protein Science **24**, 1920–1933 (2015).

[5] S. R. Durell and A. Ben-Naim, “Hydrophobic-hydrophilic forces in protein folding”, Biopolymers **107**, e23020 (2017).

[6] N. B. Rego and A. J. Patel, “Understanding hydrophobic effects: insights from water density fluctuations”, Annual Review of Condensed Matter Physics **13**, 303–324 (2022).

[7] A. L. Lomize, I. D. Pogozheva, M. A. Lomize, and H. I. Mosberg, “The role of hydrophobic interactions in positioning of peripheral proteins in membranes”, BMC Structural Biology **7**, 44 (2007).

[8] C. N. Pace, H. Fu, K. L. Fryar, J. Landua, S. R. Trevino, B. A. Shirley, M. M. Hendricks, S. Iimura, K. Gajiwala, J. M. Scholtz, and G. R. Grimsley, “Contribution of hydrophobic interactions to protein stability”, Journal of Molecular Biology **408**, 514–528 (2011).

[9] O. Pornillos, B. K. Ganser-Pornillos, S. Banumathi, Y. Hua, and M. Yeager, “Disulfide bond stabilization of the hexameric capsomer of human immunodeficiency virus”, Journal of Molecular Biology **401**, 985–995 (2010).

[10] C. A. F. Andersen and B. Rost, “Secondary structure assignment”, in *Methods of biochemical analysis*, Vol. 44, edited by P. E. Bourne and H. Weissig, 1st ed. (Wiley, Feb. 10, 2003), pp. 339–363.

[11] T. Smolarczyk, I. Roterman-Konieczna, and K. Stapor, “Protein secondary structure prediction: a review of progress and directions”, Current Bioinformatics **15**, 90–107 (2020).

[12] R. Pearce and Y. Zhang, “Deep learning techniques have significantly impacted protein structure prediction and protein design”, Current Opinion in Structural Biology **68**, 194–207 (2021).

[13] R. Van Der Lee, M. Buljan, B. Lang, R. J. Weatheritt, G. W. Daughdrill, A. K. Dunker, M. Fuxreiter, J. Gough, J. Gsponer, D. T. Jones, P. M. Kim, R. W. Kriwacki, C. J. Oldfield, R. V. Pappu, P. Tompa, V. N. Uversky, P. E. Wright, and M. M. Babu, “Classification of intrinsically disordered regions and proteins”, Chemical Reviews **114**, 6589–6631 (2014).

[14] C. Lesieur, S. Frutiger, G. Hughes, R. Kellner, F. Pattus, and F. G. Van Der Goot, “Increased stability upon heptamerization of the pore-forming toxin aerolysin”, Journal of Biological Chemistry **274**, 36722–36728 (1999).

[15] J. A. Marsh and S. A. Teichmann, “Structure, dynamics, assembly, and evolution of protein complexes”, Annual Review of Biochemistry **84**, 551–575 (2015).

[16] P. E. Wright and H. J. Dyson, “Intrinsically disordered proteins in cellular signalling and regulation”, Nature Reviews Molecular Cell Biology **16**, 18–29 (2015).

[17] M. Wells, H. Tidow, T. J. Rutherford, P. Markwick, M. R. Jensen, E. Mylonas, D. I. Svergun, M. Blackledge, and A. R. Fersht, “Structure of tumor suppressor p53 and its intrinsically disordered n-terminal transactivation domain”, Proceedings of the National Academy of Sciences **105**, 5762–5767 (2008).

[18] P. Tompa and M. Fuxreiter, “Fuzzy complexes: polymorphism and structural disorder in protein–protein interactions”, Trends in Biochemical Sciences **33**, 2–8 (2008).

[19] B. Ma, S. Kumar, C.-J. Tsai, and R. Nussinov, “Folding funnels and binding mechanisms”, Protein Engineering, Design and Selection **12**, 713–720 (1999).

[20] J. Guo and H.-X. Zhou, “Protein allostery and conformational dynamics”, Chemical Reviews **116**, 6503–6515 (2016).

[21] D. E. Koshland, “Application of a theory of enzyme specificity to protein synthesis”, Proceedings of the National Academy of Sciences **44**, 98–104 (1958).

[22] N. Greives and H.-X. Zhou, “Both protein dynamics and ligand concentration can shift the binding mechanism between conformational selection and induced fit”, Proceedings of the National Academy of Sciences **111**, 10197–10202 (2014).

[23] M. P. Latham, A. Sekhar, and L. E. Kay, “Understanding the mechanism of proteasome 20s core particle gating”, Proceedings of the National Academy of Sciences **111**, 5532–5537 (2014).

[24] R. Sprangers, A. Velyvis, and L. E. Kay, “Solution NMR of supramolecular complexes: providing new insights into function”, Nature Methods **4**, 697–703 (2007).

[25] F. Stengel, A. J. Baldwin, A. J. Painter, N. Jaya, E. Basha, L. E. Kay, E. Vierling, C. V. Robinson, and J. L. P. Benesch, “Quaternary dynamics and plasticity underlie small heat shock protein chaperone function”, Proceedings of the National Academy of Sciences **107**, 2007–2012 (2010).

[26] Y. Cheng, “Single-particle cryo-EM—how did it get here and where will it go”, Science **361**, 876–880 (2018).

[27] K. Murata and M. Wolf, “Cryo-electron microscopy for structural analysis of dynamic biological macromolecules”, Biochimica et Biophysica Acta (BBA) - General Subjects **1862**, 324–334 (2018).

[28] S. Schütz and R. Sprangers, “Methyl TROSY spectroscopy: a versatile NMR approach to study challenging biological systems”, Progress in Nuclear Magnetic Resonance Spectroscopy **116**, 56–84 (2020).

[29] Y. Hu, K. Cheng, L. He, X. Zhang, B. Jiang, L. Jiang, C. Li, G. Wang, Y. Yang, and M. Liu, “NMR-based methods for protein analysis”, Analytical Chemistry **93**, 1866–1879 (2021).

[30] wwPDB consortium, S. K. Burley, H. M. Berman, C. Bhikadiya, C. Bi, L. Chen, L. D. Costanzo, C. Christie, J. M. Duarte, S. Dutta, Z. Feng, S. Ghosh, D. S. Goodsell, R. K. Green, V. Guranovic, D. Guzenko, B. P. Hudson, Y. Liang, R. Lowe, E. Peisach, I. Periskova, C. Randle, A. Rose, M. Sekharan, C. Shao, Y.-P. Tao, Y. Valasatava, M. Voigt, J. Westbrook, J. Young, C. Zardecki, M. Zhuravleva, G. Kurisu, H. Nakamura, Y. Kengaku, H. Cho, J. Sato, J. Y. Kim, Y. Ikegawa, A. Nakagawa, R. Yamashita, T. Kudou, G.-J. Bekker, H. Suzuki, T. Iwata, M. Yokochi, N. Kobayashi, T. Fujiwara, S. Velankar, G. J. Kleywegt, S. Anyango, D. R. Armstrong, J. M. Berrisford, M. J. Conroy, J. M. Dana, M. Deshpande, P. Gane, R. Gáborová, D. Gupta, A. Gutmanas, J. Koča, L. Mak, S. Mir, A. Mukhopadhyay, N. Nadzirin, S. Nair, A. Patwardhan, T. Paysan-Lafosse, L. Pravda, O. Salih, D. Sehnal, M. Varadi, R. Vařeková, J. L. Markley, J. C. Hoch, P. R. Romero, K. Baskaran, D. Maziuk, E. L. Ulrich, J. R. Wedell, H. Yao, M. Livny, and Y. E. Ioannidis, “Protein data bank: the single global archive for 3d macromolecular structure data”, Nucleic Acids Research **47**, D520–D528 (2019).

[31] D. E. Shaw, P. J. Adams, A. Azaria, J. A. Bank, B. Batson, A. Bell, M. Bergdorf, J. Bhatt, J. A. Butts, T. Correia, R. M. Dirks, R. O. Dror, M. P. Eastwood, B. Edwards, A. Even, P. Feldmann, M. Fenn, C. H. Fenton, A. Forte, J. Gagliardo, G. Gill, M. Gorlatova, B. Greskamp, J. Grossman, J. Gullingsrud, A. Harper, W. Hasenplaugh, M. Heily, B. C. Heshmat, J. Hunt, D. J. Ierardi, L. Iserovich, B. L. Jackson, N. P. Johnson, M. M. Kirk, J. L. Klepeis, J. S. Kuskin, K. M. Mackenzie, R. J. Mader, R. McGowen, A. McLaughlin, M. A. Moraes, M. H. Nasr, L. J. Nociolo, L. O’Donnell, A. Parker, J. L. Peticolas, G. Pocina, C. Predescu, T. Quan, J. K. Salmon, C. Schwink, K. S. Shim, N. Siddique, J. Spengler, T. Szalay, R. Tabladillo, R. Tartler, A. G. Taube, M. Theobald, B. Towles, W. Vick, S. C. Wang, M. Wazlowski, M. J. Weingarten, J. M. Williams, and K. A. Yuh, “Anton 3: twenty microseconds of molecular dynamics simulation before lunch”, in Proceedings of the international conference for high performance computing, networking, storage and analysis (Nov. 14, 2021), pp. 1–11.

[32] G. Torrie and J. Valleau, “Nonphysical sampling distributions in monte carlo free-energy estimation: umbrella sampling”, Journal of Computational Physics **23**, 187–199 (1977).

[33] A. Laio and M. Parrinello, “Escaping free-energy minima”, Proceedings of the National Academy of Sciences **99**, 12562–12566 (2002).

[34] A. Barducci, G. Bussi, and M. Parrinello, “Well-tempered metadynamics: a smoothly converging and tunable free-energy method”, Physical Review Letters **100**, 020603 (2008).

[35] O. Valsson and M. Parrinello, “Variational approach to enhanced sampling and free energy calculations”, Physical Review Letters **113**, 090601 (2014).

[36] Y. Sugita and Y. Okamoto, “Replica-exchange molecular dynamics method for protein folding”, Chemical Physics Letters **314**, 141–151 (1999).

[37] E Marinari and G Parisi, “Simulated tempering: a new monte carlo scheme”, Europhysics Letters (EPL) **19**, 451–458 (1992).

[38] Y. I. Yang, Q. Shao, J. Zhang, L. Yang, and Y. Q. Gao, “Enhanced sampling in molecular dynamics”, The Journal of Chemical Physics **151**, 070902 (2019).

[39] D. Ray and M. Parrinello, “Kinetics from metadynamics: principles, applications, and outlook”, Journal of Chemical Theory and Computation **19**, 5649–5670 (2023).

[40] M. Chen, “Collective variable-based enhanced sampling and machine learning”, The European Physical Journal B **94**, 211 (2021).

[41] P. Tiwary and A. Van De Walle, “A review of enhanced sampling approaches for accelerated molecular dynamics”, in *Multiscale materials modeling for nanomechanics*, Vol. 245, edited by C. R. Weinberger and G. J. Tucker, Series Title: Springer Series in Materials Science (Springer International Publishing, Cham, 2016), pp. 195–221.

[42] H. Zhou, F. Wang, and P. Tao, “T-distributed stochastic neighbor embedding method with the least information loss for macromolecular simulations”, Journal of Chemical Theory and Computation **14**, 5499–5510 (2018).

[43] E. Chiavazzo, R. Covino, R. R. Coifman, C. W. Gear, A. S. Georgiou, G. Hummer, and I. G. Kevrekidis, “Intrinsic map dynamics exploration for uncharted effective free-energy landscapes”, Proceedings of the National Academy of Sciences **114** (2017).

[44] W. Chen, A. R. Tan, and A. L. Ferguson, “Collective variable discovery and enhanced sampling using autoencoders: innovations in network architecture and error function design”, Journal of Chemical Physics **149**, Publisher: American Institute of Physics Inc. (2018).

[45] W. Chen and A. L. Ferguson, “Molecular enhanced sampling with autoencoders: on-the-fly collective variable discovery and accelerated free energy landscape exploration”, Journal of Computational Chemistry **39**, Publisher: John Wiley and Sons Inc., 2079–2102 (2018).

[46] M. T. Degiacomi, “Coupling molecular dynamics and deep learning to mine protein conformational space”, Structure **27**, Publisher: Cell Press, 1034–1040.e3 (2019).

[47] H. Chen and C. Chipot, “Chasing collective variables using temporal data-driven strategies”, QRB discovery **4**, e2 (2023).

[48] H. Fu, H. Liu, J. Xing, T. Zhao, X. Shao, and W. Cai, “Deep-learning-assisted enhanced sampling for exploring molecular conformational changes”, The Journal of Physical Chemistry B **127**, 9926–9935 (2023).

[49] J. M. L. Ribeiro, P. Bravo, Y. Wang, and P. Tiwary, “Reweighted autoencoded variational bayes for enhanced sampling (RAVE)”, The Journal of Chemical Physics **149**, 072301 (2018).

[50] C. X. Hernández, H. K. Wayment-Steele, M. M. Sultan, B. E. Husic, and V. S. Pande, “Variational encoding of complex dynamics”, Physical Review E **97**, 062412 (2018).

[51] H. Chen, B. Roux, and C. Chipot, “Discovering reaction pathways, slow variables, and committor probabilities with machine learning”, Journal of Chemical Theory and Computation **19**, Publisher: American Chemical Society, 4414–4426 (2023).

[52] A. Mardt, L. Pasquali, H. Wu, and F. Noé, “VAMPnets for deep learning of molecular kinetics”, Nature Communications **9**, 5 (2018).

[53] M. Oh, G. C. A. Da Hora, and J. M. J. Swanson, “tICA-metadynamics for identifying slow dynamics in membrane permeation”, Journal of Chemical Theory and Computation **19**, 8886–8900 (2023).

[54] Y. Wang, J. M. L. Ribeiro, and P. Tiwary, “Past–future information bottleneck for sampling molecular reaction coordinate simultaneously with thermodynamics and kinetics”, Nature Communications **10**, 3573 (2019).

[55] W. Chen, H. Sidky, and A. L. Ferguson, “Nonlinear discovery of slow molecular modes using state-free reversible VAMPnets”, The Journal of Chemical Physics **150**, 214114 (2019).

[56] G. Lazzeri, H. Jung, P. G. Bolhuis, and R. Covino, “Molecular free energies, rates, and mechanisms from data-efficient path sampling simulations”, Journal of Chemical Theory and Computation **19**, 9060–9076 (2023).

[57] Y. Yuan and Q. Cui, “Accurate and efficient multilevel free energy simulations with neural network-assisted enhanced sampling”, Journal of Chemical Theory and Computation **19**, 5394–5406 (2023).

[58] V. Ramaswamy, S. Musson, C. Willcocks, and M. Degiacomi, “Deep learning protein conformational space with convolutions and latent interpolations”, Physical Review X **11**, `10.1103/PhysRevX.11.011052` (2021).

[59] X. Liu, J. Xing, H. Fu, X. Shao, and W. Cai, “Analyzing molecular dynamics trajectories thermodynamically through artificial intelligence”, Journal of Chemical Theory and Computation **20**, 665–676 (2024).

[60] L. Franke and C. Peter, “Visualizing the residue interaction landscape of proteins by temporal network embedding”, Journal of Chemical Theory and Computation **19**, 2985–2995 (2023).

[61] P. Eastman, J. Swails, J. D. Chodera, R. T. McGibbon, Y. Zhao, K. A. Beauchamp, L.-P. Wang, A. C. Simmonett, M. P. Harrigan, C. D. Stern, R. P. Wiewiora, B. R. Brooks, and V. S. Pande, “OpenMM 7: rapid development of high performance algorithms for molecular dynamics”, PLOS Computational Biology **13**, edited by R. Gentleman, Publisher: Public Library of Science, e1005659 (2017).

[62] S. C. Musson and M. T. Degiacomi, “Molearn: a python package streamlining the design ofgenerative models of biomolecular dynamics”, Journal of Open Source Software **8**, 5523 (2023).

[63] J. A. Maier, C. Martinez, K. Kasavajhala, L. Wickstrom, K. E. Hauser, and C. Simmerling, “Ff14sb: improving the accuracy of protein side chain and backbone parameters from ff99sb”, Journal of Chemical Theory and Computation **11**, 3696–3713 (2015).

[64] I. Soteras Gutiérrez, F.-Y. Lin, K. Vanommeslaeghe, J. A. Lemkul, K. A. Armacost, C. L. Brooks, and A. D. MacKerell, “Parametrization of halogen bonds in the CHARMM general force field: improved treatment of ligand–protein interactions”, Bioorganic & Medicinal Chemistry **24**, Publisher: Pergamon, 4812–4825 (2016).

[65] W. D. Cornell, P. Cieplak, C. I. Bayly, I. R. Gould, K. M. 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”, Journal of the American Chemical Society **117**, Publisher: American Chemical Society, 5179–5197 (1995).

[66] V. Hornak, R. Abel, A. Okur, B. Strockbine, A. Roitberg, and C. Simmerling, “Comparison of multiple amber force fields and development of improved protein backbone parameters”, Proteins: Structure, Function, and Bioinformatics **65**, 712–725 (2006).

[67] B. Hess, H. Bekker, H. J. C. Berendsen, and J. G. E. M. Fraaije, “LINCS: a linear constraint solver for molecular simulations”, Journal of Computational Chemistry **18**, 1463–1472 (1997).

[68] M. Abraham, A. Alekseenko, V. Basov, C. Bergh, E. Briand, A. Brown, M. Doijade, G. Fiorin, S. Fleischmann, S. Gorelov, G. Gouaillardet, A. Gray, M. E. Irrgang, F. Jalalypour, J. Jordan, C. Kutzner, J. A. Lemkul, M. Lundborg, P. Merz, V. Miletic, D. Morozov, J. Nabet, S. Pall, A. Pasquadibisceglie, M. Pellegrino, H. Santuz, R. Schulz, T. Shugaeva, A. Shvetsov, A. Villa, S. Wingbermuehle, B. Hess, and E. Lindahl, “GROMACS 2024.4 manual”, in *Gromacs 2024.4 manual*, 2024.4 (Oct. 31, 2024), pp. 524–525.

[69] M. Pechlaner and W. F. van Gunsteren, “On the use of intra-molecular distance and angle constraints to lengthen the time step in molecular and stochastic dynamics simulations of proteins”, Proteins **90**, 543–559 (2022).

[70] D. Khurana, A. Koli, K. Khatter, and S. Singh, “Natural language processing: state of the art, current trends and challenges”, Multimedia Tools and Applications **82**, 3713–3744 (2023).

[71] M. Cobos, J. Ahrens, K. Kowalczyk, and A. Politis, “An overview of machine learning and other data-based methods for spatial audio capture, processing, and reproduction”, EURASIP Journal on Audio, Speech, and Music Processing **2022**, 10 (2022).

[72] F.-A. Croitoru, V. Hondru, R. T. Ionescu, and M. Shah, “Diffusion models in vision: a survey”, IEEE Transactions on Pattern Analysis and Machine Intelligence **45**, 10850–10869 (2023).

[73] S. Cohen, “Chapter 1 - the evolution of machine learning: past, present, and future”, in *Artificial intelligence in pathology (second edition)*, edited by C. Chauhan and S. Cohen (Elsevier, Jan. 1, 2025), pp. 3–14.

[74] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning”, Nature **521**, 436–444 (2015).

[75] C. Cortes and V. Vapnik, “Support-vector networks”, Machine Learning **20**, 273–297 (1995).

[76] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning* (MIT Press, 2016).

[77] Y. Bengio, A. Courville, and P. Vincent, *Representation learning: a review and new perspectives*, arXiv:1206.5538, Apr. 23, 2014.

[78] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks”, in Proceedings of the thirteenth international conference on artificial intelligence and statistics, ISSN: 1938-7228 (Mar. 31, 2010), pp. 249–256.

[79] M. M. Hammad, *Deep learning activation functions: fixed-shape, parametric, adaptive, stochastic, miscellaneous, non-standard, ensemble*, arXiv:2407.11090, Version Number: 1, 2024.

[80] P. Kidger and T. Lyons, *Universal approximation with deep narrow networks*, arXiv:1905.08539, Version Number: 2, 2019.

[81] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, *On large-batch training for deep learning: generalization gap and sharp minima*, arXiv:1609.04836, Version Number: 2, 2016.

[82] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors”, Nature **323**, Publisher: Nature Publishing Group, 533–536 (1986).

[83] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, “On the importance of initialization and momentum in deep learning”, in Proceedings of the 30th international conference on machine learning (May 26, 2013).

[84] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization”, Journal of Machine Learning Research **12**, 2121–2159 (2011).

[85] D. P. Kingma and J. Ba, *Adam: a method for stochastic optimization*, arXiv:1412.6980, Jan. 30, 2017.

[86] R. M. Schmidt, F. Schneider, and P. Hennig, *Descending through a crowded valley - benchmarking deep learning optimizers*, arXiv:2007.01547, Version Number: 6, 2020.

[87] J. Martens, “Deep learning via hessian-free optimization”, in Proceedings of the 27th international conference on international conference on machine learning, ICML’10 (June 21, 2010), pp. 735–742.

[88] D. Wolpert and W. Macready, “No free lunch theorems for optimization”, IEEE Transactions on Evolutionary Computation **1**, 67–82 (1997).

[89] A. Y. Ng, “Feature selection, l 1 vs. l 2 regularization, and rotational invariance”, in Proceedings of the twenty-first international conference on machine learning (2004), p. 78.

[90] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, *Spectral normalization for generative adversarial networks*, arXiv:1802.05957, Version Number: 1, 2018.

[91] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting”, The journal of machine learning research **15**, 1929–1958 (2014).

[92] D. E. Rumelhart and J. L. McClelland, “Learning internal representations by error propagation”, in *Parallel distributed processing: explorations in the microstructure of cognition: foundations* (Elsevier, 1987), pp. 318–362.

[93] H. Bourlard and Y. Kamp, “Auto-association by multilayer perceptrons and singular value decomposition”, Biological Cybernetics **59**, Publisher: Springer-Verlag, 291–294 (1988).

[94] U. Michelucci, *An introduction to autoencoders*, arXiv:2201.03898, Version Number: 1, 2022.

[95] J. Feydy, T. Séjourné, F.-X. Vialard, S.-i. Amari, A. Trouvé, and G. Peyré, “Interpolating between optimal transport and MMD using sinkhorn divergences”, arXiv:1810.08278 (2018).

[96] D. P. Kingma and M. Welling, *Auto-encoding variational bayes*, arXiv:1312.6114, Version Number: 11, 2013.

[97] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial networks”, arXiv:1406.2661 (2014).

[98] F. Noé, S. Olsson, J. Köhler, and H. Wu, “Boltzmann generators: sampling equilibrium states of many-body systems with deep learning”, Science **365**, eaaw1147 (2019).

[99] D. Berthelot, C. Raffel, A. Roy, and I. Goodfellow, “Understanding and improving interpolation in autoencoders via an adversarial regularizer”, arXiv:1807.07543 (2018).

[100] B. Alipanahi, A. Delong, M. T. Weirauch, and B. J. Frey, “Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning”, Nature Biotechnology **33**, 831–838 (2015).

[101] J. Zhou and O. G. Troyanskaya, “Predicting effects of noncoding variants with deep learning–based sequence model”, Nature Methods **12**, 931–934 (2015).

[102] O. Denas and J. Taylor, “Deep modeling of gene expression regulation in an erythropoiesis model”, in Representation learning, icml workshop (2013).

[103] H. R. Roth, L. Lu, J. Liu, J. Yao, A. Seff, K. Cherry, L. Kim, and R. M. Summers, “Improving computer-aided detection using convolutional neural networks and random view aggregation”, IEEE Transactions on Medical Imaging **35**, 1170–1181 (2016).

[104] P.-P. Ypsilantis, M. Siddique, H.-M. Sohn, A. Davies, G. Cook, V. Goh, and G. Montana, “Predicting response to neoadjuvant chemotherapy with PET imaging using convolutional neural networks”, PLOS ONE **10**, edited by R. J. Anto, e0137036 (2015).

[105] T. Zeng, R. Li, R. Mukkamala, J. Ye, and S. Ji, “Deep convolutional neural networks for annotating gene expression patterns in the mouse brain”, BMC Bioinformatics **16**, 147 (2015).

[106] B. Zhang, J. Li, and Q. Lü, “Prediction of 8-state protein secondary structures by a novel deep learning architecture”, BMC Bioinformatics **19**, 293 (2018).

[107] S. Long and P. Tian, “Protein secondary structure prediction with context convolutional neural network”, RSC Advances **9**, 38391–38396 (2019).

[108] J. Hou, B. Adhikari, and J. Cheng, “DeepSF: deep convolutional neural network for mapping protein sequences to folds”, Bioinformatics **34**, edited by A. Valencia, 1295–1303 (2018).

[109] R. Zamora-Resendiz and S. Crivelli, *Structural learning of proteins using graph convolutional neural networks*, arXiv:10.1101/610444, Apr. 16, 2019.

[110] M. Gao, H. Zhou, and J. Skolnick, “DESTINI: a deep-learning approach to contact-driven protein structure prediction”, Scientific Reports **9**, 3514 (2019).

[111] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, A. Bridgland, C. Meyer, S. A. A. Kohl, A. J. Ballard, A. Cowie, B. Romera-Paredes, S. Nikolov, R. Jain, J. Adler, T. Back, S. Petersen, D. Reiman, E. Clancy, M. Zielinski, M. Steinegger, M. Pacholska, T. Berghammer, S. Bodenstein, D. Silver, O. Vinyals, A. W. Senior, K. Kavukcuoglu, P. Kohli, and D. Hassabis, “Highly accurate protein structure prediction with AlphaFold”, Nature **596**, 583–589 (2021).

[112] A. R. Leach, *Molecular modelling: principles and applications*, 2nd ed (Prentice Hall, Harlow, England ; New York, 2001), 744 pp.

[113] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, “Fast and accurate deep network learning by exponential linear units (ELUs)”, arXiv:1511.07289 (2015).

[114] J. A. Bertrand, E. Fanchon, L. Martin, L. Chantalat, G. Auger, D. Blanot, J. Van Heijenoort, and O. Dideberg, ““open” structures of MurD: domain movements and structural similarities with folylpolyglutamate synthetase”, Journal of Molecular Biology **301**, 1257–1266 (2000).

[115] J. A. Bertrand, G. Auger, L. Martin, E. Fanchon, D. Blanot, D. Le Beller, J. Van Heijenoort, and O. Dideberg, “Determination of the MurD mechanism through crystallographic analysis of enzyme complexes”, Journal of Molecular Biology **289**, 579–590 (1999).

[116] R. Šink, M. Kotnik, A. Zega, H. Barreteau, S. Gobec, D. Blanot, A. Dessen, and C. Contreras-Martel, “Crystallographic study of peptidoglycan biosynthesis enzyme MurD: domain movement revisited”, PLOS ONE **11**, edited by I. G. Boneca, e0152075 (2016).

[117] M. T. Degiacomi, I. Iacovache, L. Pernot, M. Chami, M. Kudryashev, H. Stahlberg, F. G. van der Goot, and M. Dal Peraro, “Molecular assembly of the aerolysin pore reveals a swirling membrane-insertion mechanism”, Nature Chemical Biology **9**, 623–629 (2013).

[118] A. N. Calabrese, B. Schiffrin, M. Watson, T. K. Karamanos, M. Walko, J. R. Humes, J. E. Horne, P. White, A. J. Wilson, A. C. Kalli, R. Tuma, A. E. Ashcroft, D. J. Brockwell, and S. E. Radford, “Inter-domain dynamics in the chaperone SurA and multi-site binding to its outer membrane protein clients”, Nature Communications **11**, 2155 (2020).

[119] W. Humphrey, A. Dalke, and K. Schulten, “VMD: visual molecular dynamics”, Journal of Molecular Graphics **14**, 33–38 (1996).

[120] Schrödinger, LLC, “The PyMOL molecular graphics system, version 1.8”, Nov. 2015.

[121] J. D. Hunter, “Matplotlib: a 2d graphics environment”, Computing in Science & Engineering **9**, 90–95 (2007).

[122] J. A. Bertrand, “Crystal structure of UDP-n-acetylmuramoyl-l-alanine:d-glutamate ligase from escherichia coli”, The EMBO Journal **16**, 3416–3425 (1997).

[123] R. Šink, H. Barreteau, D. Patin, D. Mengin-Lecreulx, S. Gobec, and D. Blanot, “MurD enzymes: some recent developments”, BioMolecular Concepts **4**, 539–556 (2013).

[124] T. M. Belete, “Novel targets to develop new antibacterial agents and novel alternatives to antibacterial agents”, Human Microbiome Journal **11**, 100052 (2019).

[125] M. Shen and A. Sali, “Statistical potential for assessment and prediction of protein structures”, Protein Science **15**, 2507–2524 (2006).

[126] W. B. Arendall, W. Tempel, J. S. Richardson, W. Zhou, S. Wang, I. W. Davis, Z.-J. Liu, J. P. Rose, W. M. Carson, M. Luo, D. C. Richardson, and B.-C. Wang, “A test of enhancing model accuracy in high-throughput crystallography”, Journal of Structural and Functional Genomics **6**, 1–11 (2005).

[127] S. Monaco-Malbet, C. Berthet-Colominas, A. Novelli, N. Battaı, N. Piga, V. Cheynet, F. Mallet, and S. Cusack, “Mutual conformational adaptations in antigen and antibody upon complex formation between an fab and HIV-1 capsid protein p24”, Structure **8**, 1069–1077 (2000).

[128] F. A. Rey, F. X. Heinz, C. Mandl, C. Kunz, and S. C. Harrison, “The envelope glycoprotein from tick-borne encephalitis virus at 2 å resolution”, Nature **375**, 291–298 (1995).

[129] C. Bagnéris, O. Bateman, C. Naylor, N. Cronin, W. Boelens, N. Keep, and C. Slingsby, “Crystal structures of -crystallin domain dimers of b-crystallin and hsp20”, Journal of Molecular Biology **392**, 1242–1252 (2009).

[130] E. Bitto and D. B. McKay, “Crystallographic structure of SurA, a molecular chaperone that facilitates folding of outer membrane porins”, Structure **10**, 1489–1498 (2002).

[131] Z. Zhang, S. Witham, and E. Alexov, “On the role of electrostatics in protein–protein interactions”, Physical Biology **8**, 035001 (2011).

[132] J. Adams, “Bonding energy models”, in *Encyclopedia of materials: science and technology* (Elsevier, 2001), pp. 763–767.

[133] M. Baek, F. DiMaio, I. Anishchenko, J. Dauparas, S. Ovchinnikov, G. R. Lee, J. Wang, Q. Cong, L. N. Kinch, R. D. Schaeffer, C. Millán, H. Park, C. Adams, C. R. Glassman, A. DeGiovanni, J. H. Pereira, A. V. Rodrigues, A. A. Van Dijk, A. C. Ebrecht, D. J. Opperman, T. Sagmeister, C. Buhlheller, T. Pavkov-Keller, M. K. Rathinaswamy, U. Dalwadi, C. K. Yip, J. E. Burke, K. C. Garcia, N. V. Grishin, P. D. Adams, R. J. Read, and D. Baker, “Accurate prediction of protein structures and interactions using a three-track neural network”, Science **373**, 871–876 (2021).

[134] M. Frassek, A. Arjun, and P. G. Bolhuis, “An extended autoencoder model for reaction coordinate discovery in rare event molecular dynamics datasets”, The Journal of Chemical Physics **155**, 064103 (2021).

[135] H. Sidky, W. Chen, and A. L. Ferguson, “Molecular latent space simulators”, Chemical Science **11**, 9459–9467 (2020).

[136] S. Mehdi, D. Wang, S. Pant, and P. Tiwary, “Accelerating all-atom simulations and gaining mechanistic understanding of biophysical systems through state predictive information bottleneck”, Journal of Chemical Theory and Computation **18**, 3231–3238 (2022).

[137] L. S. P. Rudden, S. C. Musson, J. L. P. Benesch, and M. T. Degiacomi, “Biobox: a toolbox for biomolecular modelling”, Bioinformatics **38**, edited by A. Valencia, 1149–1151 (2022).

[138] D. Beazley, *SWIG (simplified wrapper and interface generator)*, version 4.0.2, Sept. 1, 2020.

[139] W. Falcon, *PyTorch lighting*, version 1.4, Mar. 30, 2019.

[140] R. W. Grosse-Kunstleve, N. K. Sauter, N. W. Moriarty, and P. D. Adams, “The *Computational Crystallography Toolbox* : crystallographic algorithms in a reusable software framework”, Journal of Applied Crystallography **35**, 126–136 (2002).

[141] C. R. Harris, K. J. Millman, S. J. Van Der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. Van Kerkwijk, M. Brett, A. Haldane, J. F. Del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, “Array programming with NumPy”, Nature **585**, 357–362 (2020).

[142] N. Michaud-Agrawal, E. J. Denning, T. B. Woolf, and O. Beckstein, “MDAnalysis: a toolkit for the analysis of molecular dynamics simulations”, Journal of Computational Chemistry **32**, 2319–2327 (2011).

[143] H. Nguyen, D. A. Case, and A. S. Rose, “NGLview–interactive molecular graphics for jupyter notebooks”, Bioinformatics **34**, edited by A. Valencia, 1241–1242 (2018).

[144] B. E. Granger and F. Perez, “Jupyter: thinking and storytelling with code and data”, Computing in Science & Engineering **23**, 7–14 (2021).

[145] P. Eastman, *OpenMM PyTorch plugin*, version 1.4, Sept. 10, 2023.

[146] Y. Yang, C. Feng, Y. Shen, and D. Tian, *FoldingNet: point cloud auto-encoder via deep grid deformation*, arXiv:1712.07262, Version Number: 2, 2017.

[147] P. Eastman, *Openmm/pdbfixer*, version 1.10, Aug. 26, 2024.

[148] B. Webb and A. Sali, “Comparative protein structure modeling using MODELLER.”, Current protocols in bioinformatics **54**, Publisher: NIH Public Access, 5.6.1–5.6.37 (2016).

[149] A.-C. Gavin, M. Bösche, R. Krause, P. Grandi, M. Marzioch, A. Bauer, J. Schultz, J. M. Rick, A.-M. Michon, C.-M. Cruciat, M. Remor, C. Höfert, M. Schelder, M. Brajenovic, H. Ruffner, A. Merino, K. Klein, M. Hudak, D. Dickson, T. Rudi, V. Gnau, A. Bauch, S. Bastuck, B. Huhse, C. Leutwein, M.-A. Heurtier, R. R. Copley, A. Edelmann, E. Querfurth, V. Rybin, G. Drewes, M. Raida, T. Bouwmeester, P. Bork, B. Seraphin, B. Kuster, G. Neubauer, and G. Superti-Furga, “Functional organization of the yeast proteome by systematic analysis of protein complexes”, Nature **415**, Publisher: Nature Publishing Group, 141–147 (2002).

[150] Y. Bengio, A. Courville, and P. Vincent, “Representation learning: a review and new perspectives”, IEEE Transactions on Pattern Analysis and Machine Intelligence **35**, 1798–1828 (2013).

[151] D. Bahdanau, K. Cho, and Y. Bengio, *Neural machine translation by jointly learning to align and translate*, arXiv:1409.0473, Version Number: 7, 2014.

[152] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, *An image is worth 16x16 words: transformers for image recognition at scale*, arXiv:2010.11929, Version Number: 2, 2020.

[153] Y. Gong, Y.-A. Chung, and J. Glass, *AST: audio spectrogram transformer*, arXiv:2104.01778, Version Number: 3, 2021.

[154] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Kaiser, and I. Polosukhin, “Transformer: attention is all you need”, Advances in Neural Information Processing Systems 30, 5998–6008 (2017).

[155] J. Ho, A. Jain, and P. Abbeel, *Denoising diffusion probabilistic models*, ArXiv:2006.11239, Version Number: 2, 2020.

[156] Pylint contributors, *Pylint*, version 2.17.7, Mar. 1, 2025.