DE-NOVO DETERMINATION OF PROTEIN TERTIARY STRUCTURE IN A HHPNX 3D -FACE CENTERED CUBIC LATTICE WITH MEAN FIELD MULTI-AGENT REINFORCEMENT LEARNING

by

ADRIAN COUTSOFTIDES
URN: 6481554

A dissertation submitted in partial fulfilment of the
requirements for the award of

BACHELOR OF SCIENCE IN COMPUTER SCIENCE

May 2020

Department of Computing
University of Surrey
Guildford GU2 7XH

Supervised by: Sotiris Moschoyanis

I declare that this dissertation is my own work and that the work of others is acknowledged and indicated by explicit references.

Adrian Coutsoftides
May 2020

Abstract

The protein folding problem is the search for a function that maps a proteins primary structure, composed of a string of discrete amino acid residues to their respective native conformation in 3D space denoted as the protein's tertiary structure. Recent breakthroughs in the field have utilize techniques such as multiple sequence alignment coupled with residual convolutional neural networks to derive candidate posteriors over the distribution of inter-residue distances from which multiple energetically favourable tertiary structures can be generated; these results are typically annealed to produce a structure of lowest conformational energy. In this work I propose an alternative deep reinforcement learning system that does not rely on the pre-existing data of other sequences and instead approximates the pair-wise local interactions of the residues on a 3D Face-Centered Cubic Lattice model. I show that mean-field approximations effectively model the the free energy landscape of the system with respect to the expectation of the distribution of rewards in the local neighbourhood for each agent. I propose that the architecture of this system effectively incorporates inductive bias into the problem formulation and thus provides a richer training signal. Additional techniques are also employed in combination to reduce the sample complexity of the search space.

Acknowledgements

Write any personal words of thanks here. Typically, this space is used to thank your supervisor for their guidance, as well as anyone else who has supported the completion of this dissertation, for example by discussing results and their interpretation or reviewing write ups. It is also usual to acknowledge any financial support received in relation to this work.

Contents

List of Figures

List of Tables

Glossary

ΔG	Change in Gibbs free energy, measure as the amount of energy in the system than can be turned into work
ΔH	Change in enthalpy as the change in total heat content of the system
ΔS	Change in entropy as the measure of disorder in a system
T	Temperature in Kelvin
ε	Interaction potential of a bond

Abbreviations

MDP	Markov Decision Process
DQN	Deep-Q-Learning
PSP	Protein Structure Prediction
PDB	Protein Data Bank
CASP	Critical Assessment of Structure Prediction
KL	Kullback Liebler
RL	Reinforcement Learning
MSA	Multiple Sequence Alignment
c.d.f	Cumulative Density Function
p.d.f	Probability Density Function

Chapter 1

Introduction

1.1 Problem Background

Proteins are biological molecules that carry out specific functions within the body, they are assembled out of chemical bonds between smaller sub units called amino acid residues to form a poly-peptide chain. Proteins carry out their functions by conforming into specific shapes and fitting into substrates, this is referred to as the "lock and key" model of proteins. A protein's three-dimensional structure has been proven [reference] to been entirely determined from the ordering of a discrete set of 22 residues in a sequence of arbitrary length. The ability to infer a protein's precise[1] three dimensional structure directly from the sequence of residues would unlock the potential for designer drugs that carry out specific actions within the body; it would also open a path for the treatment of illnesses that arise from malformed proteins such as huntington's disease [reference]. This has however been an open problem, as the search space of possible conformations for a given protein is vast and subject to numerous local minima. Traditional methods of protein structure determination such as X-Ray Crystallography [reference] are extremely costly[2] and time consuming, sometimes taking up to *four years* to determine the structure of a protein. This has given rise to multiple computational approaches reference as means to drive down cost and increase the productivity of researchers. Naturally, many approaches using deep learning have been proposed to solve the problem, recently a breakthrough by DeepMind [reference] propelled them to success at the bi-annual Critical Assessment of Structure Prediction (CASP) competition. Though their results were state of

[1] Within 1Å

[2] In the order of millions reference

the art, their methods still relied on building a predictive model of the properties of available proteins in the Protein Data Bank (PDB)[reference]; as opposed to inferring the tertiary structure from the primary sequence alone.

1.2 Project Aims and Objectives

Over the course of this dissertation I aim to explore and contrast modern deep learning approaches for approximating the native conformations of proteins on a discrete lattice structure. I will then go on to propose a novel algorithm based on mean-field approximations that addresses some of the drawbacks and biases inherent in the approaches I have reviewed. The following is a list of the project's aims overall:

1. Provide a succinct introduction to the molecular mechanics that govern the conformations of proteins
2. Introduce and compare different approaches to modelling the problem computationally
3. Provide an extended analysis of lattice models for proteins and their ability to encode correct conformations
4. Introduce reinforcement learning and progress to modern Deep Q-Learning
5. Building on Deep Q-Learning, introduce improvements to the algorithm since it's inception
6. Compare and contrast deep learning approaches to the lattice model with an emphasis on reinforcement learning methods
7. Introduce multi-agent learning within the framework of stochastic games
8. Introduce mean-field approximations in particular reference to mean field games
9. Formulate the conformation of residues on a lattice as a cooperative game of incomplete information
10. Provide a novel approach to de-novo structure determination using mean-field multi-agent learning
11. Benchmark my approach against the results of other groups on the same proteins to evaluate the effectiveness of the novel algorithm

1.3 Success Criteria

In order to evaluate the utility of both my findings and subsequent algorithm, I have defined the following high-level requirements that must be satisfied to mark this project as a success.

1. Provide comprehensive overview of the underlying problem of protein folding
2. Describe markov decision processes (MDPs) and its ties to reinforcement learning
3. Highlight drawbacks to the default DQN algorithms and notable improvements to address those
4. Demonstrate multi-agent learning as a generalisation of single MDPs into markov games
5. Successfully benchmark my multi-agent approach against similar lattice-based approaches to protein folding

1.4 Structure of Report

1. **Introduction**

   In this section I provided an overview the the protein folding problem and introduced the components that I will be synthesising into novel approach.

2. **Literature Review**

   In this section I will provide an overview as-well as an introduction to select topics and that comprise the necessary background knowledge. This section ends with a comparative analysis of related work on the problem.

3. **System Requirements and Specification**

   In this section I will analyse the drawbacks of methods utilized in related work, and from this evaluation derive the requirements of a systems required to address these limitations.

4. **System Design**

   This section seeks to unify the selected systems and concepts into an integrated learning algorithm that coherently reflects the underlying problem's structure.

5. **Testing & Validation**

   In order to verify the efficacy of the learning agents, a set of lattice structure of known proteins will be calculated and evaluated against the results for the same set of proteins in related work.

6. **Discussions**

   Limitations of my implementation are discussed here and possible solutions and research directions are proposed.

7. **Conclusion**

   Here I will present my concluding thoughts and results of my final analyses of the architecture.

Chapter 2

Literature Review

2.1 Proteins

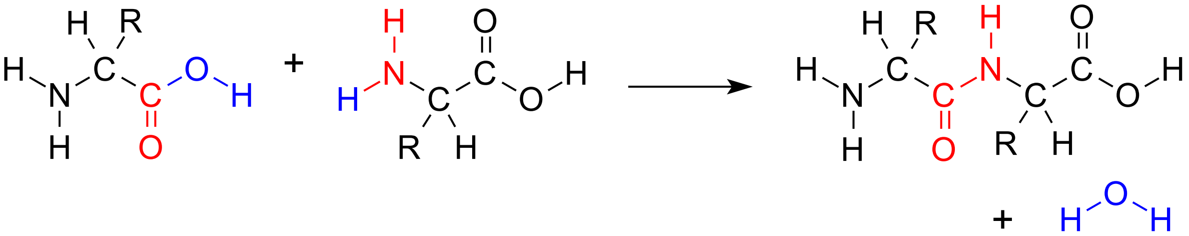
2.1.1 Amino Acids & Poly-Peptides

Proteins are the mechanism by which executive function takes place in the cell. This is includes the implementation of activities such as *"metabolism, growth, architecture and regulation of cell and organism"* -(Lesk 2018). Proteins themselves are composed of discrete molecular units termed amino acids, the precise way in which they amino acids arrange themselves is dependent of the genetic sequence of the parent organism itself. There exist 20 unique amino acids (examples given below) that can be combined that can be sequenced into strings of arbitrary length of any order following chemical laws. These molecules come together to form peptide

Table 2.1

Examples of proteins and their codes	
Glycine	G
Alanine	A
Serine	S
Cysteine	C
Threorine	T
Proline	P
Valine	V
Leucine	K

Figure 2.1: Peptide Bond

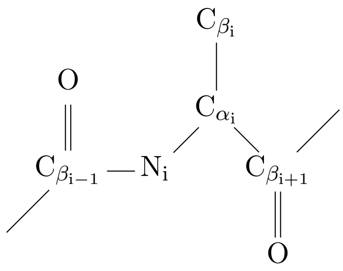


†Image Source: `https://en.wikipedia.org/wiki/Peptide_bond`

bonds, this occurs when the carboxyl group ($^{-}COOH$) bonds to the *amino* group ($^{+}NH_3$) of another amino acid producing water as a by-product of the reaction (Lesk 2018). Despite all amino acids possessing an amino and carboxyl group, every amino acid has a unique side chain, an additional molecular group attached to the main chain. The unique chemical properties of these side chains are responsible for the interactions in that molecule's *neighbourhood*; a term that is expanded on in the next few sections.

2.1.2 Protein Structures

Figure 2.2: Structure of a peptide unit

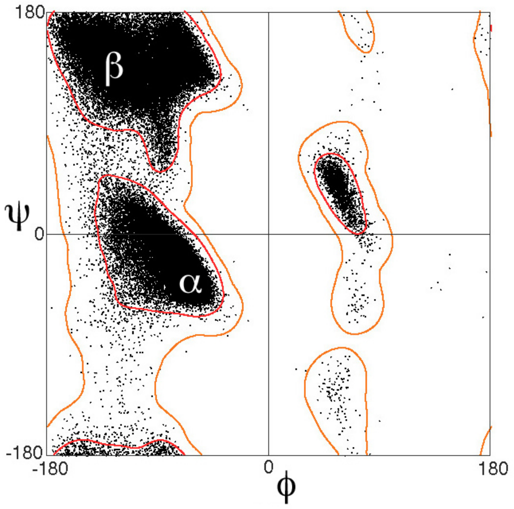


The carbon atom that joins the side chain to the main chain is denoted as the α carbon atom and the carbon atoms in immediate contact with the α carbons are known as β carbons[1]. The bond angles between the $C_{\alpha_i} - C_{\beta_i+1}$ and $C_{\alpha_i} - N_i$ groups are commonly denoted as ψ and ϕ respectively. Stereochemically feasible angles can be described by a Ramachandran plot as in figure 2.3.

Poly-peptide conformations can be described by three structures:

[1] In figure 2.2 subscript i denotes membership to a unique unit

Figure 2.3: Ramachandran plot



†Source: `https://proteopedia.org/wiki/index.php/Ramachandran_Plots`

1. **Primary Structure**

   This is the encoding of a protein as a 1-D sequence of its constituent amino acids.

   I.e:

   $$DTYGYWEPYT$$

2. **Secondary Structure**

   This encoding represents local, repeating structures with respect to the entire conformations. These structures satisfy chemical restraints common to most proteins[2]. In particular, the formation of structures known as α[3] helices and β[4] sheets largely facilitate the energetic and conformational constraints imposed by the interactions amongst the side-chains orthogonal to the backbone. These structures form the dense regions in the Ramachandran plot.

3. **Tertiary Structure**

   A protein's tertiary structure are its 3D atomic coordinates in space. The protein's torsion backbone (main-chain) traces out a curve through space parameterised by all pairs (ϕ, ψ) for each residue pair. Hydrogen bonds, which arise when neighbouring residues are in close proximity although not directly connected by the backbone, hold together the various

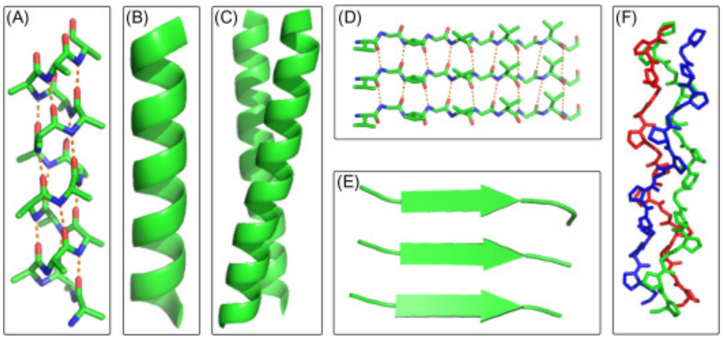
[2] The hydrogen bonding potential of main-chain $N - H$ and $C = O$ groups (Lesk 2018)

[3] B in fig 2.4

[4] E in fig 2.4

secondary structures in specific conformations such that the global energy of the system is minimized (Yang, Ji & Liu 2013).

Figure 2.4: Secondary and tertiary structures



†Source: (Boyle 2018)

The primary goal of protein structure prediction is the *ab initio* determination of a protein's tertiary structure from its primary structure (Yang et al. 2013) which can be described as a mapping:

$$\mathbf{X}^n := \text{Set of all sequences of length } n \tag{2.1}$$

$$\mathbf{F} : \mathbf{X}^n \rightarrow ((\phi, \psi)_1 \ldots (\phi, \psi)_n) \tag{2.2}$$

(Anfinsen 1972) et al showed that the protein's tertiary structure is entirely encoded by its primary structure; the implications of this are explored next.

2.1.3 The Protein's Energy Landscape

Due in part to the continuous spectrum of stereochemically plausible values of (ϕ, ψ), the possible number of conformations for a given sequence are extraordinarily large. Given that each possible conformation has an associated free energy, Levinthal's paradox states that if all conformations were equally energetically favourable, then protein would essentially have to undergo a random walk along the *Gibbs free energy* surface until it has found its native conformation. The free energy surface in this respect refers to the manifold that is formed by taking the *distribution* over conformations $((\phi, \psi)_1 \ldots (\phi, \psi)_n)$ at every time-step and ΔG at every point to form a surface in $2n + 1$ dimensional space. Given that common proteins have in the order of 100-1000+

constituent residues, the combinatorial size of the state space would prohibit any efforts to find a specific conformation by random walk with vanishing probability.

(Yang et al. 2013) address both the paradox and subsequent critiscisms; part of their work can be summarised by the following lemmas:

Lemma 1. *Not all conformations are equally energetically favourable.*

Proof. Proteins fold spontaneously in the order of *nanoseconds* into their native conformation in a solvent at constant temperature, thus they cannot be traversing the whole state space. □

Lemma 2. *There must therefore be a driving reaction that narrows down the search space.*

Proof. Folding appears to undergo two consecutive stages, tier 1 occurs of a timescale of *nanoseconds* and tier 2 appears to occur over a scale of *picoseconds*, a reduction in 3 orders of magnitude. Thus the peptide must undergo a slower interaction that constrains the state space followed by a final, faster "relaxation" into the native state. □

Lemma 3. *By undergoing hydrophobic collapse, the remaining residues have restricted degrees of freedom, thus a reduced subspace consisting of only energetically feasible conformations is explored, within this subspace there exists one unique solution whose energy is minimized.*

Proof. The relative timescales of the tier 1 and tier 2 interactions indicate a smooth slope in conformational space that greatly constrains the possible conformations, followed by a second tier of faster interactions that enables the system to quickly overcome the local maxima and minima which gives way to the global optimum at the bottom of the subset. □

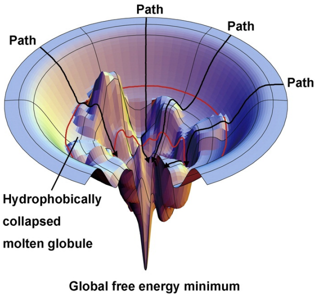
Lemma 4. *If there exists only one unique conformation whose Gibbs free energy is minimal amongst all possible conformations, the manifold must be "funnel shaped".*

Proof. The nature of the hydrophobic collapse "drops" the full state space into a smaller subspace, the unique global minimum lies within this subspace. If there existed more than one unique solution, then a given sequence could form into multiple possible native conformations, this would violate the marginal stability property of the native state so there cannot exist more than one unique solution. □

$$\Delta G = \Delta H - T\Delta S$$

By the second law of thermodynamics, the entropy (amount of disorder) in a closed system can never decrease over time and can remain constant only if the system is in equilibrium, thus a state of maximum entropy (Jaffe 2018). The work of (Yue & Dill 1993, Yang et al. 2013) show that the overall increase in entropy of the peptide-solvent system is primarily motivated by a phenomenon known as the *hydrophobic collapse* whereby all the water produced as by-product of the peptide bond [5] "squeezes" the hydrophobic residues into globular core. Following **Lemma 4**, the process appears to be guided by a heuristic search in this subspace, enabling it to swiftly reach its native state.

Figure 2.5: Gibbs free energy funnel



†Source: (Yang et al. 2013)

Critically, the notion of ΔG is with respect to the distribution over conformations, indicating that at any point the protein exists as an ensemble of possible conformations, and the native state consists of a collection of conformations whose joint distribution minimises the value of ΔG. I reformulate this property as maximum likelihood objective:

Theorem 1.

1. *Let $T \in \{\mathbf{X}\}^n$ denote the native state as a tuple of n torsion angles (follows from eq. 2.1).*
2. *Let $P(\mathbf{F}(\mathbf{X}^n;\theta))$ denote the distribution over all functional mappings from a primary sequence to a tuple of n torsion angles with parameters θ (follows from eq. 2.2).*

The search for a function that maps the protein's primary structure to its tertiary structure can be formulated as the maximization of the posterior probability that function $\mathbf{F}$ with parameters θ

[5] See fig 2.1

maps sequence $\mathbf{X}^n$ *to state* T*.*

$$P(T|\mathbf{F}(\mathbf{X}^n;\theta)) \; = \frac{P(\mathbf{F}(\mathbf{X}^n;\theta)|T) \cdot P(T)}{P(\mathbf{F}(\mathbf{X}^n;\theta))} \tag{2.3}$$

This formulation is further explored in the **System Requirements and Specification** section, however an interesting detail to note is that this is equivalent to sampling the maximum likelihood estimate from the posterior of a *Gaussian process* [6].

2.1.4 Bioinformatics

There exists experimental techniques to determine the protein's native (or *crystalline*) structure[7], however these approaches are out of the scope of this project and are not given a detailed analysis. Otherwise, these techniques can often require years of trial and error and can be very costly (Alberts 2002).

In recent years, advancements in both hardware and processing power have enabled computational methods to play increasingly significant role in the PSP problem. I will focus briefly what's known as homology modelling as this is relevant to understanding related supervised learning methods, and then I will then provide extended review of another class of models used for *ab initio* structure prediction, lattice models.

2.1.4.1 Homology Modelling

All proteins are members of an evolutionary tree and so related proteins tend to demonstrate high structural similarity with their relatives. Although older studies suggest that proteins are in fact random strings only slightly edited by evolution (Weiss, Jiménez-Montaño & Herzel 2000); more recent studies have shown that this is not the case, as certain methods were able to distinguish between random strings and real proteins with an accuracy of up to $88 - 94.36\%$ (Lucrezia, Slanzi, Poli, Polticelli & Minervini 2012, Tsygvintsev 2019). Given the apparent specificity of the sequences, Homology modelling, focuses on trying to predict the tertiary structure of an unknown protein from the structures of closely related proteins (homologues). This involves measuring Hamming distances[8] between closely related proteins and deriving an *inverse scoring matrix* with each pair-wise comparison. When this is done for more than one possible pairing,

[6] See appendix A.1

[7] Such as X-Ray Crystallography (Chayen 2005)

[8] See appendix A.2

this is known as Multiple Sequence Alignment (MSA). Highly correlated proteins with relatively low MSA scores tend to exhibit very similar tertiary structures (Lesk 2018); this property is used as the basis for many machine learning methods which is explored in **Related Work**.

2.1.4.2 Lattice Models

Although approaches that utilise homology modelling in some form have been very successful in recent years[9], they rely on the pre-determined structures of other proteins, this poses a barrier to *de novo* protein design and research into undetermined structures for which we do not have a readily available dataset of relatives.

An alternative model of computation was proposed by (Yue & Dill 1993) that seeks to address the combinatorial conformational space. Instead, they proposed that the space should be discretised to restrict the possible number of conformations a given sequence can take on, a *Bravais* lattice was the perfect tool for this.

2.1.4.3 Bravais Lattices

A Bravais lattice is a simple mathematical description of a lattice structure, whereby each point in space is some linear combination of unit vectors with integer multiples (Kittel 2005).

$$\begin{gathered}\mathbf{r}, \mathbf{a} \in \mathbb{R}^n, u \in \mathbb{Z} \\ \mathbf{r} = u_1\mathbf{a}_1 + u_2\mathbf{a}_2 + u_3\mathbf{a}_3\end{gathered} \tag{2.4}$$

In matrix form:

$$\begin{gathered}\mathbf{r} \in \mathbb{R}^n, \mathbf{u} \in \mathbb{Z}^n \\ \mathbf{a} \in \mathbb{R}^n \times \mathbb{R}^n \\ \mathbf{r} = \mathbf{a} \cdot \mathbf{u}^\top\end{gathered} \tag{2.5}$$

Different values for the unit vectors give rise to different classes of lattices, the number of neighbours for a given vertez is denoted as z. In the interest of brevity 3 lattices are discussed as they are pertinent to the related work and system design.

[9] See **AlphaFold 2.4.2.1**

1. **2D Square Lattice**

   A simple lattice whose unit vectors are two dimensional:

$$\mathbf{a} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{2.6}$$
$$z = 4$$

2. **3D Primitive Cubic Lattice**

   A 3D lattice with unit vectors in each direction of space:

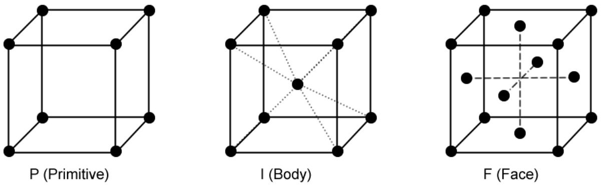
$$\mathbf{a} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.7}$$
$$z = 6$$

3. **3D Face-Centered Cubic Lattice**

   A 3D cubic lattice with additional vertex points at the center of each square face; notably, this lattice has the highest sphere packing density [10]:

$$\mathbf{a} = \begin{bmatrix} 0 & 0.5 & 0.5 \\ 0.5 & 0 & 0.5 \\ 0.5 & 0.5 & 0 \end{bmatrix} \tag{2.8}$$
$$z = 12$$

Figure 2.6: 3D Cubic Bravais Lattices



†Source: `https://biochem.co/2008/08/crystal-structure-studies/`

For clarity, any point within the vector space defined by the lattice structure is an integer multiple of any linear combination of the unit vectors. Nomenclature on the subject dictates that a vertex on the lattice is referred to as a "*site*".

[10] Highest amount of rigid spheres that can be packed per unit space ≈ 0.74 (Hoque, Chetty & Sattar 2009)

2.1.4.4 HP Model

(Yue & Dill 1993) initially proposed the use of a 3D primitive cubic lattice to model the conformations of proteins. Each amino acid was divided into two categories according to their properties: Hydrophobic and Polar, and a conformation was formulated as a self avoiding walk (SAW) on the lattice.
Some definitions:

- A *contact* on the lattice is define as the adjacency of two immediate neighbours on the lattice not connected by the torsion backbone, if two residues are connected through the torsion backbone directly this is referred to as a *bond*.
- A segment is a run of monomers of a particular category (a)[11], a singlet occurs when a monomer is situated between two monomers of another category (b)[12].

$$\underset{(a)}{HPPPPH} \qquad\qquad \underset{(b)}{PHP}$$

- $x \in \{H, P\}$
- n_x is the number of x monomers in the sequence.
  I.e: n_H is the number of H monomers in the entire sequence.
- b_{xx} is number of bonds between xx.
- t_{xx} is the number of *bonds + contacts* between xx.
- h_{term} is the number of segements that terminate in H monomers

These quantities are related by the following equations:

$$2b_{HH} + b_{HP} + h_{term} + (z-2)n_H = 2(t_{HH} + b_{HH}) + b_{HP} + t_{HP} + t_{H-Solvent} \tag{2.9}$$

$$\begin{gathered} G = \frac{b_{HP} + h_{term}}{2} \\ S = b_{HP} + t_{HP} + t_{H-Solvent} \end{gathered} \tag{2.10}$$

- G is the total number of H segments in a sequence

[11] P segment of length 4

[12] H singlet

- S is the total surface area of the hydrophobic core, assuming that the monomers as **unconnected**. This acts as an upper bound of the surface area.

Substituting (2.10) into (2.9) produces the *"folding equation"*:

$$t_{HH} + \frac{S}{2} = G + \frac{(z-2)n_H}{2} \tag{2.11}$$

Where the left side is dependent on the *conformation*, and the right side is a constant that solely depends on the lattice structure and given sequence. Motivated by the role of the hydrophobic collapse, the folding problem within this framework is a function that maximises t_{HH} ; this is equivalent to minimizing S. The logic follows that the most tightly packed core, that with the minimal surface area should also have the densest packing of $H - H$ contacts.
(Dill, Bromberg, Yue, Chan, Ftebig, Yee & Thomas 2008, Lau & Dill 1989) show that an unguided exhaustive search through the coformational landscape is unlikely to find a single candidate native state but instead produce a set of states with high degeneracy; indicating that many conformations shared the same free energy value due to the numerous local minima. However by introducing a heuristic that guided the conformation through as hydrophobic collapse, a much smaller set of candidate conformations with much less degeneracy was produced. Genetic algorithms were developed to overcome the rugged landscape, their fitness function was determined by the favourability of any contact pair derived for experimental data. This resulted in the interaction potential (ε) matrix, which is a measure of that interaction's contribution to the total ΔG of the system (Hoque et al. 2009):

Table 2.2

	H	P
H	-1	0
P	0	0

When (Lau & Dill 1989) first proposed the model, only 2D lattices were explored at the time with exhaustive search techniques and the simulations were evaluated against *mean-field* approximations of the system. (Yue & Dill 1993) then introduced linear integer programming methods that conducted volumetric optimization such as those described by (2.10-2.11) on a 3D primitive cubic lattice. There are some key limitations imposed on the mean-field approximations which are explored in **Related Work** and **System Design and Specification**; subsequent improvements to the original HP model were also introduced and are explored next.

2.1.4.5 hHPNX Model

(Hoque et al. 2009) showed that previously explored variants of the HP model on 2D and 3D lattices were still prone to large degeneracy in the solutions they produced. This was a result of the limitations imposed by having only two possible categories; stochastic methods end up traversing many redundant solutions for each non-native conformation visited in the search-space[13]. The original HP categories were further subdivided according the relative polarities of the side chains:

- $P \mapsto \{P, N, X\}$
- $H \mapsto \{h, H\}$

They then correcte errors in the calculated interaction potentials of the previously proposed HPNX and YhHX models (Bornberg-Bauer 1997) producing the hHPNX interaction matrix :

Table 2.3

	h	H	P	N	X
h	2	-4	0	0	0
H	-4	-3	0	0	0
P	0	0	1	-1	0
N	0	0	-1	1	0
X	0	0	0	0	0

The extended categories were also motivated by experimatal data that emphasises the importance of treating *Alanine* and *Valine*[14] separately from other hydrophobic residues as they consistently observed that these groups underwent different reactions, resulting in different potentials; this was attributed to their geometrical positions in the folded protein (Crippen 1991).

More detailed work has been conducted in the work I has cited thus far, including but not limited to:

[13] This can be proved using Group Theory axioms as lattices form geometric groups - needs citation

[14] $A, V \in H$ in HPNX

- Investigating geometric and volumentric constraints of various lattice structures[15].
- Optimizing linear integer programming constraints with tighter bounds on $H - H$ contacts[16].
- Analysis of conformational lanscapes with respect to free energy landscapes[17].
- Probabilistic thermodynamic analysis of folding pathways[18].

Thus far I have summarised the most important results from these authors within the context of this project. In **Related Work** I will explore how these models have been re-contextualised within the reinforcement learning framework and then propose a novel method that incorporates results covered in this section that have yet to be exploited in recent work.

2.2 Reinforcement Learning

In this section I will introduce the Reinforcement Learning (RL) paradigm. Then, its integration with deep learning is explored, and subsequent improvements in the algorithm are also elaborated upon. The end result, the Rainbow DQN agent, is integrated as part of the **System Design and Specification**.

2.2.1 Markov Decision Processes

A finite markov chain is a process that consists of a set of states $\mathbf{S^n} \coloneqq \{S_1, S_2, \ldots, S_n\}$ and a transition function $F(\mathbf{S})$ that takes the state at the current timestep t and outputs a new state at timestep $t + 1$.

$$F(S_t) \mapsto S_{t+1} \quad \forall S \in \mathbf{S}, \forall t \in \mathbf{T} \tag{2.12}$$

For a given process, the states are linked by a transition probability $P(S_{t+1}, S_t)$. A process is markov only if the markov property holds:

$$P(S_{t+1}, S_t) = P(S_{t+1}|S_t) \tag{2.13}$$

[15] (Yue & Dill 1993, Dill et al. 2008)

[16] (Yue & Dill 1993)

[17] (Dill et al. 2008)

[18] (Yang et al. 2013, Dill et al. 2008)

The next state in a process that obeys the markov property is determined solely by the value of the current state, and for any given sequence, the transition probability between any two states remains the same. Thus, markov chains can begin characterised by a transition matrix $\mathbf{P} := |\mathbf{S}^n| \times |\mathbf{S}^n|$ where each row j is a distribution $P(i, \cdot)$ such that:

$$i \in \mathbf{S}^n, \sum_{j \in \mathbf{S}^n} P(i,j) = 1 \tag{2.14}$$

The product along the column space $j \in \mathbf{S}^n$, $\prod_{i \in \mathbf{S}^n} P(i,j)$ of the transition matrix reveals the **steady state** probability of being in any particular state $P(S_j)$.

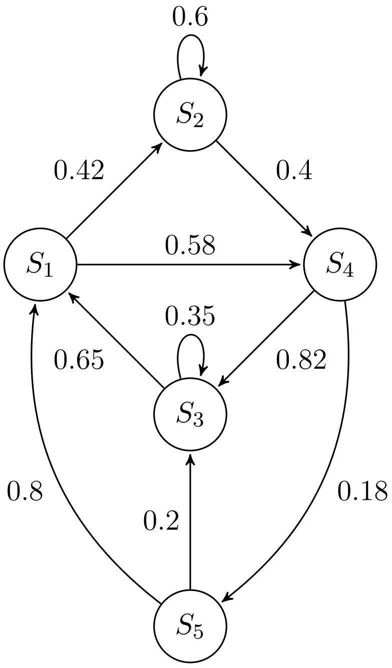


Figure 2.7: Markov chain with state space and transition propabilities

A Markov Decision Process (MDP) is a generalisation of this framework to sequential decision making (Sutton 2018). An MDP consists of an agent-environment interface, where the *Agent* is the learner and decision maker and the *Environment* consists of everything outside of the agent. The agent interacts with the environment bny taking action $a \in \mathbf{A}$ in the current state S_t and receives a reward R_{t+1}[19], the outcome of the agents actions transitions the environment from state S_t to S_{t+1}. The agent's long term reward is maximised by taking actions that move the agents into favourable states that yield higher rewards, the **expected** reward in any given state

[19] Reward received at $t + 1$ to indicate the fact that an action must be taken first to move to a new state in order to obtain a reward

is equal to the probability of entering state S_{t+1} multiplied by the value of the reward R_{t+1} in that state.

$$\mathop{\mathbb{E}}_{S \in \mathbf{S}^n}\Big[R_{t+1} \mid S_t, A_t\Big] = P(R_{t+1} \mid S_t, A_t) \cdot R_{t+1} \tag{2.15}$$

The *value* of a given state $\mathcal{V}(S)$ is the total expected reward for that state for any action taken in that state. This is taken to be the *long run* return of state S if the process was repeated in the infinite limit under a stationary distribution of rewards[20]:

$$\mathcal{V}(S) = \sum_{\forall a \in \mathbf{A}} \mathop{\mathbb{E}}_{S \in \mathbf{S}^n}\Big[R_{t+1} \mid S, a\Big] \tag{2.16}$$

An agent in the environment seeks to take actions that maximise its long term reward at each timestep:

$$\sum_{t \in \mathbf{T}} \underset{a}{max}\Big(\mathop{\mathbb{E}}_{S \in \mathbf{S}^n}\Big[R_{t+1} \mid S_t, a\Big]\Big) \tag{2.17}$$

In episodic settings the set of tasks underlying the MDP contains a *terminal* state that is reached after a finite number of timesteps, whereas continuous processes may not necessarily have a finite number of time steps. These settings can be unified under the notion of an *absorbing* state τ which is one that transitions only to itself with a reward of $R_\tau = 0$. For a given MDP, successive runs from S_0 to S_τ are termed episodes. In applied settings, an agent will repeatedly play through episodes with the *Goal* of maximising the cumulative reward in each episode:

$$G_t \coloneqq R_{t+1} + R_{t+2} + \ldots + R_{\tau-1} = \sum_{t=0}^{t=\tau-1} R_{t+1} \tag{2.18}$$

Equations (2.15-18) assume a stationary distribution of rewards, and (2.18) is incompatible with continuous domains where the cumulative reward that the agent tries to maximise can be infinite if $\tau = \infty$. Many real world tasks such as playing video games (Mnih, Kavukcuoglu, Silver, Rusu, Veness, Bellemare, Graves, Riedmiller, Fidjeland, Ostrovski, Petersen, Beattie, Sadik, Antonoglou, King, Kumaran, Wierstra, Legg & Hassabis 2015) and dialogue generation (Weisz, Budzianowski, Su & Gasic 2018) fall into the category of continuous, non-stationary MDPs.

By instead considering the *discounted cumulative reward*, the analysis of both episodic and continuous tasks with non-stationary distributions of rewards becomes computationally tractable:

$$\begin{gathered} 0 < \gamma \leqslant 1 \\ R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \ldots + \gamma^{\tau-t-1} R_{\tau-1} \end{gathered} \tag{2.19}$$

[20] "Stationary" indicating that the probability of receiving a reward in any state does not change as the process evolves

Where γ is referred to as the *discount factor*. The use of a discount factor contracts rewards further into the future asymptotically towards zero as (2.19) forms a geometric series[21] with ratio $r < 1$:

$$\begin{gathered}\lim_{n\to\infty}\left(\sum_{k=0}^{n}\alpha\cdot r^{k}\right)=\frac{\alpha}{1-r},\quad\forall\, 0<r<1,\quad\alpha\in\mathbb{R}^{+}\\ \lim_{n\to\infty}\left(\sum_{k=0}^{n}1\cdot\gamma^{k}\right)=\frac{1}{1-\gamma}\end{gathered}\tag{2.20}$$

Intuitively, this ensures that the agents values immediate rewards more so than delayed rewards as the rewards in the current timestep can be more accurately estimated than rewards further into the future due to the non-stationarity of the process. And so we can rewrite (2.19) in terms of cumulative reward (2.18):

$$\begin{aligned}G_t &= R_{t+1}+\gamma R_{t+2}+\gamma^{2}R_{t+3}\ldots+\gamma^{\tau-t-1}R_{\tau-1}\\ &= R_{t+1}+\gamma\Big[R_{t+2}+\gamma R_{t+3}+\ldots+\gamma^{\tau-t-2}R_{\tau-1}\Big]\\ &= R_{t+1}+\gamma G_{t+1}\end{aligned}\tag{2.21}$$

From (2.22) we can see that a stationary process (2.18) is a special case of the cumulative discounted reward where $\gamma = 1$. The closer the discount factor is to 1, the more the agent values future rewards.

In order for an agent to choose actions that maximise the discounted future rewards, it must adhere to a *policy* π. A policy is a mapping of states to a distribution of possible actions:

$$\begin{gathered}\pi:\mathbf{S}\to P(\mathbf{A})\\ \sum_{a\in\mathbf{A}}\pi(a\mid s)=1,\quad\forall s\in\mathbf{S}\end{gathered}\tag{2.22}$$

Under this policy, the *value* of a state (2.16) can be rewritten as:[22]

$$\begin{aligned}v_{\pi}(s) &= \sum_{a\in\mathbf{A}}\pi(a\mid s)\cdot\mathbb{E}\Big[R_{t+1}+\gamma G_{t+1}\Big|S_t=s\Big]\\ &= \mathop{\mathbb{E}}_{\pi}\Big[R_{t+1}+\gamma G_{t+1}\Big|S_t=s\Big]\end{aligned}\tag{2.23}$$

Equation (2.24) describes an agent's *state-value* function, which measures the expected return of a given state if the process was repeated infinitely. Similarly, we can define the *action value* function $q_{\pi}(s,a)$ to express the average return of taking a particular action in a given state:

$$q_{\pi}(s,a)=\mathop{\mathbb{E}}_{\pi}\Big[R_{t+1}+\gamma G_{t+1}\;\Big|\;S_t=s,A_t=a\Big]\tag{2.24}$$

[21] See Appendix A.3

[22] Needs further derivation

Formally, the *action value* function describes the expected return of being in state s and taking action a and following π thereafter. It it expected reward described in (2.24) conditioned additionally upon a fixed action being taken in the current state.

2.2.1.1 Bellman Equation

The state value function and the action value fuction for a given MDP can be estimated from experience, the Bellman Equation combines the equations described so far and codifys the relationship between the value of a state and the value of succesor states:

$$\begin{aligned} v_\pi(s) &= \mathop{\mathbb{E}}_{\pi}\left[R_{t+1} + \gamma G_{t+1} \,\middle|\, S_t = s\right] \\ &= \sum_{a \in \mathbf{A}} \pi(a \mid s) \sum_{s' \in \mathbf{S}} \sum_{r \in \mathbf{R}} P(s', r \mid s, a)\left(r + \gamma \mathop{\mathbb{E}}_{\pi}\left[r + G_{t+1} \,\middle|\, S_{t+1} = s'\right]\right) \\ &= \sum_{a \in \mathbf{A}} \pi(a \mid s) \sum_{s' \in \mathbf{S}} \sum_{r \in \mathbf{R}} P(s', r \mid s, a)\left(r + \gamma v_\pi(s')\right) \end{aligned} \tag{2.25}$$

This represents a weighted sum of the expectations over all possible next states given the current state s and action taken a (Sutton 2018).

2.2.1.2 Optimality

Optimal state and value functions (v_*, q_*) are those that maximise the expected return under policy π. A policy π is defined to be better or equal to policy π' if its expected return is greater or equal to π' for all states.

$$\begin{aligned} v_*(s) &= \underset{\pi}{max}\, v_\pi(s) \\ q_*(s) &= \underset{\pi}{max}\, q_\pi(s, a) \\ \Rightarrow \; q_*(s) &= \mathbb{E}\left[R_{t+1} + \gamma v_*(S_{t+1}) \,\middle|\, S_t = s, A_t = a\right] \end{aligned} \tag{2.26}$$

As v_* is the optimal value function, its expected return must equal the expected return for greedily selecting the *best action* in that state everytime, otherwise there would exist $v_{\pi*} > v_*$:

$$\begin{aligned} v_*(s) &= \underset{a \in \mathbf{A}}{max}\, q_{\pi*}(s, a) \\ &= \underset{a}{max} \mathop{\mathbb{E}}_{\pi*}\left[R_{t+1} + \gamma v_*(S_{t+1}) \,\middle|\, S_t = s, A_t = a\right] \qquad \text{by (2.27)} \\ &= \underset{a}{max} \sum_{s' \in \mathbf{S}} \sum_{r \in \mathbf{R}} P(s', r \mid s, a)[r + \gamma v_*(s')] \end{aligned} \tag{2.27}$$

From (2.28) we can see that the optimal value function is defined independently of any policy, indicating that it can be reached starting from any arbitary policy, and improving on that, this is formalised under the framework of *Generalised Policy Iteration* (GPI).

2.2.1.3 Generalised Policy Iteration

GPI consists of an alternating process of making the value function more accurate with respect to the policy and making the policy greedy with respect to the current value function. This process is decomposed into two separate steps:

1. **Policy Evaluation**

   An initial value function υ_0 is parameterised with arbitary values for each $s \in \mathbf{S}$. Given a sequence of value functions $\upsilon_0, \upsilon_1, \upsilon_2, \dots$ each succesive value can be calculated by:

   $$\upsilon_{k+1}(s) = \mathop{\mathbb{E}}_{\pi}\left[R_{t+1} + \gamma \upsilon_k(S_{t+1}) \,\middle|\, S_t = s\right] \tag{2.28}$$

   The value of each state $s \in \mathbf{S}$ is updated "in place" with the revised estimate garnered from the current reward (which may not have been achieved previously) and the expected immediate rewards approximated by the previous value function. A pass over all $\mathbf{S}$ produces a new state value function.

2. **Policy Iteration**

   For a given value function, (2.28) shows that the optimal policy is that which greedily selections actions that maximise the expected future returns at each state. It follows that:

   $$\begin{aligned} \pi \in \{\Pi\} &:= \text{ Set of all policies} \\ \upsilon \in \{\Upsilon\} &:= \text{ Set of all value functions} \\ q \in \{\mathcal{Q}\} &:= \text{ Set of all action value functions} \end{aligned} \tag{2.29}$$

   $$\forall\, \pi\ \exists\, q_* \mid q_* \geq (q' \neq q_*) \iff \forall\, \pi\ \exists\, \upsilon_* \mid \upsilon_* \geq (\upsilon' \neq \upsilon_*) \tag{2.30}$$

   $$\begin{aligned} \pi' \geq \pi \ \rightarrow\ \upsilon_{\pi'}(s) \geq \upsilon_\pi(s) \quad \forall s \in \mathbf{S} \\ \therefore\ \ q_*^{\pi'}(s,a) \geq q_*^{\pi}(s,a) \end{aligned} \tag{2.31}$$

   $$\Rightarrow q_\pi(s, \pi'(s)) \geq \upsilon_\pi(s) \tag{2.32}$$

This inductive line of logic reasons that if $\exists\ \pi' \geq \pi$ then the expected returns of a given state calculated under policy π' will be higher, thus if the less optimal policy had instead chosen the action according to the optimal policy, then the expected returns would be greater (2.33). The optimal policy is obtained by making the current policy greedy with respect to the action value function:

$$\begin{aligned}\pi'(s) &= \underset{a}{argmax}\ q_\pi(s,a) \\ &= \underset{a}{argmax} \sum_{s' \in \mathbf{S}} \sum_{r \in \mathbf{R}} P(s', r\ , \mid s, a) \Big[r + v_\pi(s') \Big]\end{aligned} \tag{2.33}$$

By succesively cycling through policy evaluation and improvement, we continuously update our estimate of the value function and then greedily select actions in each state according to our updated estimates. This cycle is referred to as *Generalised Policy Iteration* and can be proven to monotonically converge to an optimal value function and an optimal policy. Upon convergence, the process reaches an equilibrium where the policy does not change because the value of the states has not changed (Sutton 2018).

2.2.1.4 Exploration vs Exploitation

The full reinforcement learning problem can be characterised by the models we have described so far with the addition of the exploration - exploitation trade off. So far we have considered agents with full knowledge of the environment, but for many MDPs, the agent may never access all available state action pairs, or indeed the number of state action pairs may be unenumerable. In order to maximise returns, the agent must strike a balance between exploring new, unseen states and exploiting the greedy strategy it has already learned. A naive exploration strategy[23] is to choose the greedy action with fixed probability ε and otherwise choose a random action with probability $1 - \varepsilon$; this is called an ε **greedy strategy**. However the agent would still require an infinite amountof episodes to form the optimal policy and value functions, they must instead be approximated in such a way that preserves the gurantees of GPI.

2.2.1.5 Monte Carlo Estimation

Monte carlo estimation enables an agent to *learn* optimal policies and value functions from experience alone. In order to take advanatge of Monte Carlo methods, the MDP is modelled in

[23] More recent and general exploration strategies are discussed in **2.2.3.2** and **Related Work**.

the following way:

- Each state has its own stationary distribution of rewards.
- Each state is inter-related, i.e taking an action in one state depends on actions taken in later states.
- As all action selections are undergoing learning, the distribution of rewards becomes *non-stationary* from the point of view of later states.

Monte Carlo methods sample average returns for a given state-action pair for a given episode and use those samples to update estimates of action values. Two distinct approaches enable us to do so:

1. **On Policy Methods**
   In "*On Policy Control*" we aim to evaluate or improve the policy that is used to make decisions, which in turn produces data in the form of the outcomes of these decisions.
2. **Off Policy Methods**
   In "*Off-Policy Control*" attempt to improve a policy different than the one used to generate the data.

Within the scope of this project we will focus on *Off Policy* methods. First we consider two policies $\pi \neq b$ and we refer to the former as the *target* policy and the latter as the *behaviour* policy. The goal is to estimate v_π or q_π. We also assume that $\pi(a\,|\,s) > 0 \to b(a\,|\,s) > 0$ implying that every action taken under π is also taken at least once under b; this is the assumption of *coverage*. From coverage it follows that b must be stochastic in states where it is not identical to π as π_*, being the optimal policy, is greedy and deterministic.

First I will introduce *importance sampling*, a method for estimating expected values under on distribution given samples from another distribution. This is a general technique used in off-policy methods (Sutton 2018) as well as in later discussed *Prioritised Experience Replay*[24]. Given a starting state S_t the probability of a subsequent state action trajectory $A_t, S_{t+1}, A_{t+1}, \dots$ is :

$$\begin{aligned} &\pi(A_t \mid S_t) \cdot P(S_{t+1} \mid S_t, A_t) \cdot \pi(A_{t+1} \mid S_{t+1}) \dots P(S_T \mid S_{T-1}, A_{T-1}) \\ &= \prod_{k=t}^{T-1} \pi(A_k \mid S_k) \cdot P(S_{k+1} \mid S_k, A_k) \end{aligned} \tag{2.34}$$

[24] See **Memory**

Therefore, in order to calculate the relative probability (importance samplig ratio) of a trajectory given the target policy and and the behaviour policy:

$$\begin{aligned}\rho_{t:T-1} &= \frac{\prod_{k=t}^{T-1} \pi(A_k \mid S_k) \cdot P(S_{k+1} \mid S_k, A_k)}{\prod_{k=t}^{T-1} b(A_k \mid S_k) \cdot P(S_{k+1} \mid S_k, A_k)} \\ &= \prod_{k=t}^{T-1} \frac{\pi(A_k \mid S_k)}{b(A_k \mid S_k)}\end{aligned} \tag{2.35}$$

Intuitively, this is a ratio of how much more or less probably the trajectory is under π relative to b, given that $\upsilon_b(s) = \mathbb{E}\left[G_t \middle| S_t = s\right]$ we can transform the returns under b to have to correct expected value by multiplying them by the importance sampling ratio:

$$\mathbb{E}\left[\rho_{t:T-1} G_t \middle| S_t = s\right] = \upsilon_\pi(s) \tag{2.36}$$

In the case of full monte carlo methods, the returns are averaged at the end of each episode, *weighted importance sampling* applies the importance sampling ratio as weighted average of the returns over the whole episode with terminal state $T(t)$to form our *estimate* V:

$$V_\pi(s) = \frac{\sum_{t \in T(t)} \rho_{t:T-1} \cdot G_t}{\sum_{t \in T(t)} \rho_{t:T-1}} \tag{2.37}$$

This can be rewritten as an incremental update that occurs on an episode by episode basis where we start with random weights W_i and for each state maintain the cumulative sum of the weights given to the first n returns (C_n):

$$\begin{aligned}W_i &= \rho_{t_i:T(t_i)-1} \\ C_n &= \sum_{k=1}^{n} W_k \\ Vn + 1(s) &= V_n(s) + \frac{W_n}{C_n}\left[G_n - \upsilon_n(s)\right]\end{aligned} \tag{2.38}$$

Where the text in red is a *target update* as measurement of the error betweenthe actual returns for that episode G_n and the previously estimated returns $\upsilon_n(s)$. This is then multiplied proportionally by a weighted importance sampling ratio. The original estimate of $\upsilon_n(s)$ is the updated proportionally in the direction towards the actual expected returns.

2.2.1.6 Model Free vs Model Based

Sometimes it is possible to provide the agent with access to a model of the environment. If an agent has access to a model of the environment, typically in the form of a simulator, it can simulate states n steps ahead and use that to inform its estimated returns trivially using the approaches described so far. The trouble with this approach however is that when modelling a real world task, no coarse grained model will ever approximately the task perfectly. A poorly designed model can also introduce unintended biases into the agent (Sutton 2018). On the other hand model free methods, such as monte carlo estimation, can approximate the state value or action value function directly, and have shown better generalization properties with regard to acting in unforseen scenarios.

In this project I will focus on the model-free off-policy methods as I will show that the PSP problem on a lattice is a non-stationary MDP for which a model is not computationally tractable. With the foundations lain, I will introduce the central algorithm to contemporary reinforcement learning.

2.2.2 Deep Q Learning

2.2.2.1 Temporal Difference Error

The temporal difference (TF) error is a special case of monte carlo estimation where the action value functions and the state value functions are updates with each timestep instead of at the end of each episode.

$$V(S_t) \leftarrow V(s_t) + \alpha \Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \Big] \tag{2.39}$$

This is similar to the target update in monte carlo estimation, however instead our we update our existing estimate based on our current estimate, the target in (2.39) is expanded using (2.22). In this particular case for each episode the agent takes an action, observes the outcome of this action at S_{t+1} and evaluates the value function at this timestep; this evaluation is then used as part of a target update that is weighted by some constant step size parametere a.

2.2.2.2 Q Learning

We can go a step further and unpack (2.40) some more by observing (2.28) and (2.32 - 34) to form an estimate of our action value function Q directly forming an off policy TD control update rule:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \cdot \underset{a}{max}\, Q(S_{t+1}, a) - Q(S_t, A_t) \Big] \tag{2.40}$$

This update target is formally known as the *Bellman Error* denoted δ_t. I introduce the central algorithm (Sutton 2018): The Q learning algorithm forms the core of methods discussed and

Algorithm 1: Q Learning

```
$\alpha \in (0, 1]$;
$0 < \varepsilon < 1$;
Init Q(s,a) $\forall\ s \in \mathbf{S},\ a \in \mathbf{A}(s)$ randomly;
Q(terminal, $\cdot$) = 0;
forall episodes do
    Init S;
    forall steps in episode do
        $r \backsim U(0, 1)$; // Sample random number r
        if $r > \varepsilon$ then
            $A \leftarrow \underset{a}{argmax}\ Q(s, a)$;
        else
            $A \leftarrow rand(a \in \mathbf{A})$ ; // Choose a random action
        end
        Take action A, observe R, S' ;
        $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \cdot \underset{a}{max} Q(S', a) - Q(S, A)]$ ;
        $S \leftarrow S'$
    end
end
```

proposed throughout this project. After a brief review of neural networks, I will show how they can parameterise the Q function.

2.2.2.3 Neural Networks

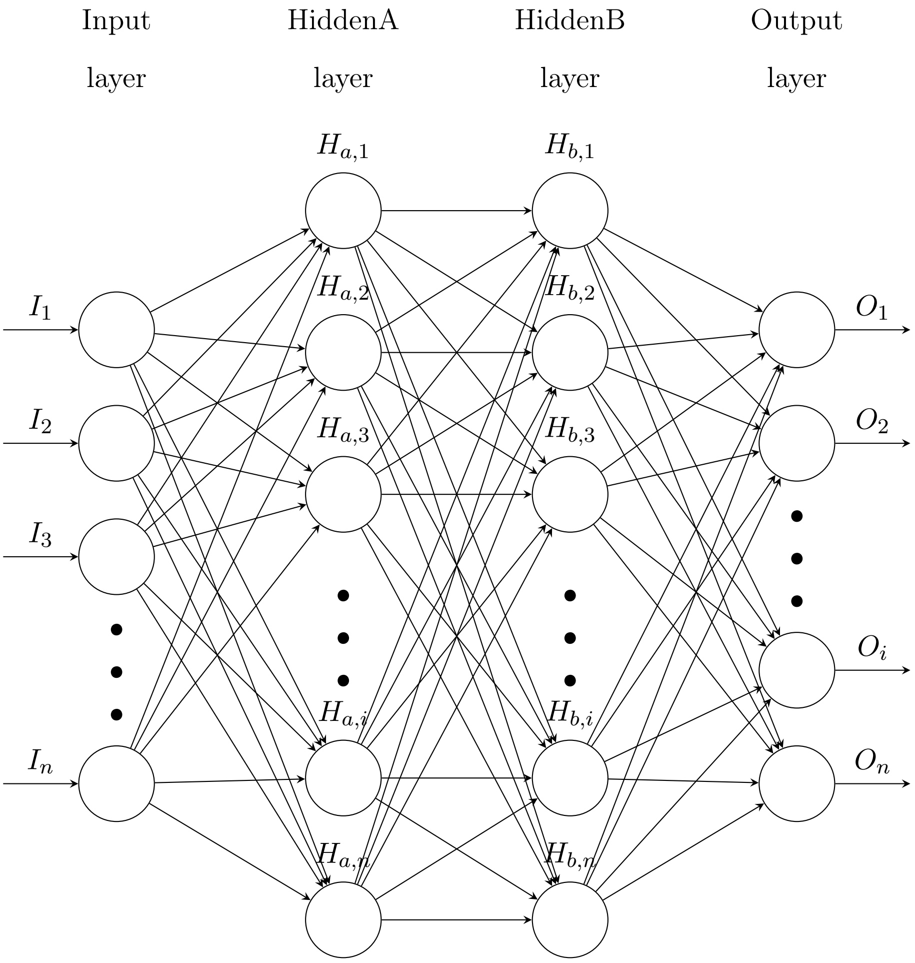
A neural network is a universal non-linear function approximator. It consists of smaller sub units called perceptrons. Each perceptron acts as linear function with respect to its inputs $\mathbf{x}$ and can be parameterised by a weight vector $\mathbf{w}$ and bias b.

$$\hat{\mathbf{y}} = \mathbf{x} \cdot \mathbf{w}^{\top} + b \tag{2.41}$$

Perceptrons canbe organised into layers hence become multi-layer perceptrons MLP. If an MLP contains more than a single intermediate layer between the input and the output layer, it is known as a deep neural net. When organised into layers, the output of each perceptron must be passed through a non-linearity

$$ReLU(\hat{\mathbf{y}}) = max(\hat{\mathbf{y}}, 0)$$

to remove the correlation between layers, unlocking its potential as a non-linear function approximator. Contemporary architecture use non-linearities such as rectified linear units (ReLU) as they demonstrate faster convergence due to sparser representations induced by zeroing weights less than zero.



A network can be represented as a function f parameterised by weight matrix $\mathbf{W}$ and bias matrix $\mathbf{b}$ with the outputs of each layer being passed into the next layer in a fully connected network.

$$\hat{\mathbf{y}} = f(\mathbf{x}; \mathbf{W}, \mathbf{b}) \tag{2.42}$$

The network is optimised against a target output Y using stochastic gradient descent.

$$\begin{aligned} L(Y, \hat{\mathbf{y}}) &= \|Y - \hat{\mathbf{y}}\|_2 \\ &= \frac{1}{N} \cdot \sum_{i=1}^{N} (Y_i - (\mathbf{x}_i \cdot \mathbf{w}_i^\top + b))^2 \end{aligned} \tag{2.43}$$

$$\begin{aligned} \frac{\partial E}{\partial \mathbf{w}} &= -\frac{2}{N} \cdot \sum_{i=1}^{N} \mathbf{x}_i (Y_i - (\mathbf{x}_i \cdot \mathbf{w}_i^\top + b)) \\ \frac{\partial E}{\partial b} &= -\frac{2}{N} \cdot \sum_{i=1}^{N} (Y_i - (\mathbf{x}_i \cdot \mathbf{w}_i^\top + b)) \end{aligned} \tag{2.44}$$

$$\nabla_{\mathbf{W}} E = \frac{\partial L(Y, \hat{\mathbf{y}})}{\partial \mathbf{W}}, \quad \nabla_{\mathbf{b}} E = \frac{\partial L(Y, \hat{\mathbf{y}})}{\partial \mathbf{b}} \tag{2.45}$$

The error is calculated first for the network's output layers, the weights are then updated in the direction of negative gradient, steepest descent with respect to error E with some constant stepsize η:

$$\mathbf{W}_{k+1} = \mathbf{W}_k - \eta \nabla E_{\mathbf{W}_k} \tag{2.46}$$

The gradients for the biases are updates likewise. This error is then back-propagated using reverse mode differentiation on a computational graph (Margossian 2019) through the remaining nodes, where the weights adjusted after the gradient updates are used as the target for the preceeding layer.

Remark. *In recent years additional properties of neural networks have been discovered relating to their nature as universal function approximators. Neural networks with a single hidden layer can approximate any function, and as the number of nodes tends to infinity it simplifies to a linear model (Lee, Xiao, Schoenholz, Bahri, Novak, Sohl-Dickstein & Pennington 2019). Alternatively, infinitely wide, deep neural networks are equivalent to a gaussian process (Lee, Bahri, Novak, Schoenholz, Pennington & Sohl-Dickstein 2017). The have also demonstrated the ability to sort lists in practically $O(1)$ time (Zhu, Cheng, Zhang, Liu, He, Yao & Zhou 2019).*

2.2.2.4 Deep Q-Networks

Neural networks can be used to approximate the Q function directly using Q learning, however the structure of the gradient update introduces complexity when trying to estimate expected value of rewards. We begin by parameterising our Q fuction with the weights of our network:

$$\begin{gathered} Q(s, a) = f(\mathbf{s}; \mathbf{W}, \mathbf{b}) \quad s \in \mathbf{S} \\ \Rightarrow Q(s, a; \mathbf{W}, \mathbf{b}) \rightarrow Q(s, a; \mathbf{W}) \end{gathered} \tag{2.47}$$

We can develop a loss function in the off policy setting using the Bellman Error δ_t. This holds because the update target $R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$ acts as a "peek ahead" from the perspective of the current state. That is we take an action, observe the outcome, and update the action-value of our old state with this new information.

$$\begin{aligned} L(\delta_t, Q(s,a;\mathbf{W})) &= \|\delta_t - Q(s,a;\mathbf{W})\|_2 \\ \frac{\partial L(\delta_t, Q(s,a;\mathbf{W}))}{\partial \mathbf{W}} &= \frac{\partial(\delta_t - Q(s,a;\mathbf{W}))^2}{\partial \mathbf{W}} \\ &= (\delta_t - Q(s,a;\mathbf{W})) \cdot \frac{\partial(\delta_t - Q(s,a;\mathbf{W}))}{\partial \mathbf{W}} \\ &= (\delta_t - Q(s,a;\mathbf{W}))\nabla_{\mathbf{W}} Q(s,a;\mathbf{W}) \end{aligned} \tag{2.48}$$

The target acts as a constant because the maximal estimated Q value at the next state is taken to be the value of the state under the optimal policy (see 2.32). In their original paper, (Mnih et al. 2015) used a neural network to take in the state as input and then output a vector of action values, one for each action. If we used the intepretation of Q learning developed in (2.41), this structure could lead to unstable gradient updates or could cause the action values to diverge. This can be described in terms of the correlation between various elements of the model, consider two dependent random variables X and Y:

$$Correlation(X,Y) = \frac{Covar(X,Y)}{\sqrt{Var(X)Var(Y)}} \tag{2.49}$$

The corelation between two dependent random variables is proportional to the covariance of the random variables normalised by their joint standard deviatiation $\sqrt{Var(X)} = \sigma$. As correlation is a dimensionless constant on the interval $[-1,1]$, if X and Y are correlated $Corr(X,Y) \approx 1$, then they increasingly co-vary across each dimension, then their standard deviation must increase likewise, and as the co-variance goes to infinity, so does their individual variance. And so under (2.41) we would be using a very similar distribution of weights to calculate $\max_a Q(S_{t+1}, a)$ and $Q(S_t, a)$ which are both dependent according to the definition of our MDP. In addition to that, if we consider successive frames of pixels on a screen as our states, it is trivial note that these observations will also be highly correlated; in the case of the atari game "Pong", the position on the paddle on the next frame is highly informed by position of the paddle in the next frame.

Instead, (Mnih et al. 2015) propose two solutions to this problem. A biologically inspired mechanism termed *Experience Replay* and a separate *Target Network.* To remove the correlations between succesive observations $(S_t, A_t, R_{t+1}, S_{t+1})$, they instead store these tuples in a *replay buffer.* In this case the replay buffer is a FIFO array of constant size that overwrites older memories with new ones. The target network is a seperate neural network whose weights θ^- are initialised with the weights of the behaviour network and then frozen. Periodically, this target network is updated with the current weights of the behaviour network θ. Every timestep, a batch of memories are uniformly sampled, the target network is used to compute the Bellman Error for each memory and the output of the behaviour network is updated according to the loss defined in (2.49), averaged across the whole batch. This algorithm proceeded to lead a breakthrough in contemporary deep reinforcment learning by outperforming human benchmarks at a super human level for the first time (adapted from the original authors):

Algorithm 2: Deep Q Network (DQN)

```
Init replay memory $D$ to capacity $N$ ;
Init $Q$ with random weights $\theta$;
Init $\hat{Q}$ with $\theta^- = \theta$;
forall episodes do
    Init state $s_1$;
    Preprocess sequence $\phi_1 = \phi(s_1)$;
    forall timesteps in episode do
        $r \backsim U(0,1)$;
        if $r > \varepsilon$ then
            $a_t \backsim A$ ; // Randomly sample action from action space
        else
            $a_t = \underset{a}{argmax}\ Q(\phi(s_t), a; \theta)$;
        end
        Execute action $a_t$ observe $R_{t+1}, s_{t+1}, d$ ; // $d = 1$ if state is terminal else 0
        Store $(\phi_1, a_t, r_{t+1}, \phi_{t+1})$ in $D$;
        Sampled random minibacth from $D$;
        $\delta_t = r_t + (1-d)\gamma \max_{a'} \hat{Q}(\phi_{t+1}, a'; \theta^-)$;
        Perform gradient update $(\delta_t - Q(s,a;\theta))\nabla_\theta Q(s,a;\theta)$;
        Every $C$ steps reset $\hat{Q} = Q$;
    end
end
```

2.2.3 Improvements to Vanilla Deep Q-Networks

Although the initial algorithm was very successful, it came with various stability issues arising from different sources. In particular, bootstraped values can be inherently unstable (Sutton 2018) due to the nature of trying to estimate a guess from a guess. Additionaly, sampling memories uniformly at random implicitly samples are larger number of unsuccesful memories (i.e ones that did not attain a reward) than successful memories as the action selections are initially random and uninformed at the start of training. The naïve ε greedy policy is also prohibitive; it monotonic decrease across the training run indicates that the exploration coefficient decreases for all states uniformly. Another way of stating this is that no matter what state the agent is in, the likelihood of choosing a random action to explore more of the state space decreases over time. This may not be a desireable attribute, as we may want to explore more in some states over others, and this preference for exploration may change over time as learn new information and update our estimates (via bootstrapping).

In follow up work (Hessel, Modayil, van Hasselt, Schaul, Ostrovski, Dabney, Horgan, Piot, Azar & Silver 2017) combine subsequent improvements to the original DQN algorithm addressing the issues listed above into a single agent termedthe "Rainbow" agent. I will go over each of the suggested improvements.

2.2.3.1 Prioritised Experience Replay

The motivation behind *Prioritised Experience Replay* (Schaul, Quan, Antonoglou & Silver 2015) is the notion that given a batch of pre-recorded memories, the agent should ideally prioritise learning from certain experiences over others. Their work follows from neuroscientific studies suggesting that the sequences of memories selected for experience replay in animals are chosen proportionally to the rewards associated with that sequence and the magnitude of the δ_t error. The magnitude of the δ_t error can be interpreted as the amount of "surprise" associated with the that sequence, if our estimated Q value for a state action pair was far from our target, the the result of our actions will always deviate from expectations because we did not form a reliable way to ascertain the outcome of events. Updating our estimates for events in which we were surprised is a necessary part of the learning cycle, and so prioritising memories according to this surprise is warrented.

The authors assign a probability of being played to each memory as such:

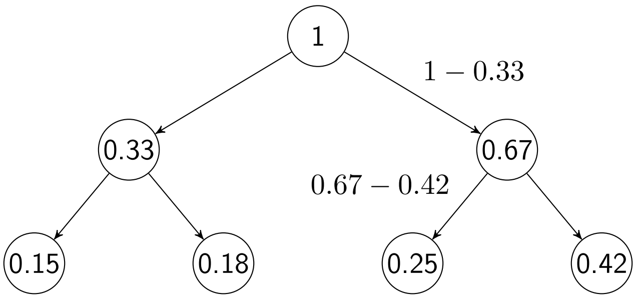
$$\begin{aligned} p_i &= |\delta_i| + \epsilon \\ P(i) &= \left(\frac{p_i}{\sum_k p_k}\right)^{\alpha} \end{aligned} \tag{2.50}$$

Where ϵ is some small positive constant to prevent the priority p_i from being zero and $P(i)$ is the probability of the transition being chosen for replay. The exponent α determines the amound of prioritisation used where $\alpha = 0$ is uniform (regular experience replay). However the use of such a distribution introduces bias into the updates of the Q value. The estimation of an expected value with stochastic updates relies on those updates come from the same distribution as our estimatiom, by construction the distribution as in (2.51), we are now drawing on observations from a different distribution than the one underlying the MDP. This is corrected for with importance sampling weights introduced in **2.2.1.5**.

$$\begin{aligned} w_i &= \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)^{\beta} \\ \nabla_\theta E &= w_j \delta_j \nabla_\theta Q(S_{j-1}, A_{j-1} \ ; \ \theta) \end{aligned} \tag{2.51}$$

The bias is fully compensated for at $\beta = 1$ this ofcourse is coupled with a significant increase in variance. In practice, the authors additionaly normalise the weights with $\frac{1}{\max_i w_i}$ so that the updates only scale downwards to reduce the overall magnitudes of the gradient updates and β is initialised to a low value $\backsim 0.4$ which is linearly annealed up to 1, this encourages additional stability during the training procedure. The observations themselves are stored in a binary heap, specifically a *Sum Segement Tree* rather than a typical array or buffer: This offers the added benefit of $O(1)$ max priority read, the root of the tree, and $O(logN)$ time to update the priorities of each transition which resides on the leaves of the tree. To traverse the tree for a batch size

Figure 2.8: Sum Segment Tree



of k memories, the interval $[0, p_{total}]$ is divided into k equal intervals. Each of these intervals is then uniformly sampled to produce a priority in that interval. The transition whose value is the closest to each sampled priority is then chosen for replay by succesively subtracting results until you arrive at a leaf as least as big as the query. Using this binary heap structure eliminates the need for sorting, enabling the speed up in time complexity.

2.2.3.2 Dueling Networks

(Wang, Schaul, Hessel, van Hasselt, Lanctot & de Freitas 2015) observed that the action value function can be decomposed further into a value and advantage function:

$$Q(s, a) = V(s) + A(s, a) \tag{2.52}$$

Where the advantage is a measure of the advantage of taking an action in that state relative to other possible actions. The expected value of a state is an estimate that takes into account all possible action values of that state. Given than some values will be more valuable than others, the expected value of a state can be taken to represent the weighted mean of all action values. Therefore the maximal value of the optimal action of a state, will be greater than the mean, this difference is expressed as the advantage of that action. Ofcourse, in an optimal setting:

$$\begin{gathered} A(s,a) = 0 \Longleftrightarrow V^*(s) = \max_a Q^*(s,a) \\ \rightarrow A(s,a) = Q^*(s,a) - V^*(s) = 0 \end{gathered} \tag{2.53}$$

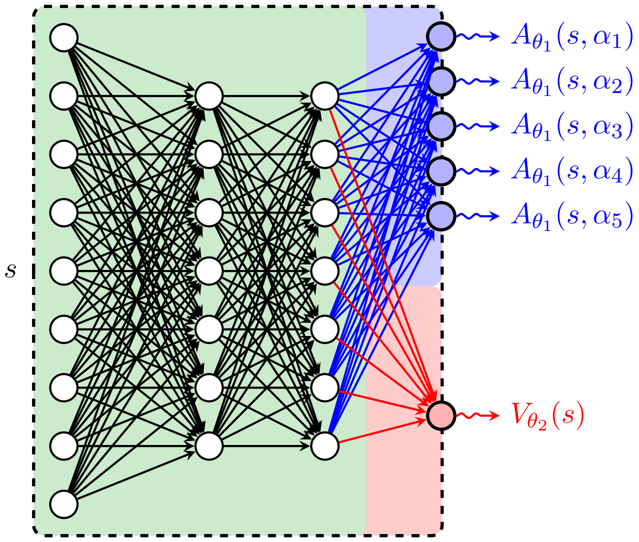
As the value of a state is the value of the most optimal action, the relative advantage of the action to itself is 0. Given only a Q function we cannot recover the value and advantage and value uniquely due to the nature of the inter-connectedness of the neural network. Changes in the weights affect the entire space of approximation. Instead the authors decompose the output layer of the DQN into two seperate heads, one that calculates the advantage and another to calculate the value (as a scalar).

The action value is then calculated:

$$Q(s, a\,;\,\theta) = V(s\,;\,\theta_2) + (A(s, a\,;\,\theta_1) - \mathbb{E}[A(s, a\,;\,\theta_1)]) \tag{2.54}$$

We fix this lack of identifiability by forcing the function approximator to maximise the value of the state by having a 0 advantage at the chosen action, hence we subtract the mean. The Q values for bothe the behaviour and target network are calculated this way.

Figure 2.9: Dueling Network Heads

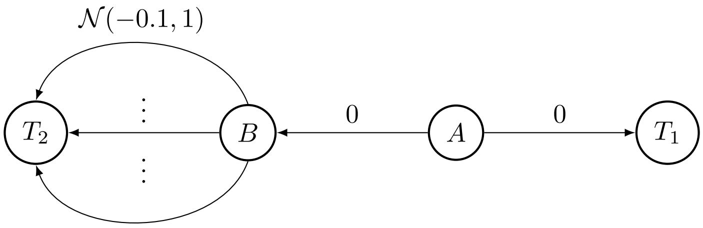

s
$A_{\theta_1}(s, \alpha_1)$
$A_{\theta_1}(s, \alpha_2)$
$A_{\theta_1}(s, \alpha_3)$
$A_{\theta_1}(s, \alpha_4)$
$A_{\theta_1}(s, \alpha_5)$
$V_{\theta_2}(s)$

†Source: `https://github.com/PetarV-/TikZ/blob/master/A3C%20neural%20network/a3c_neural_network.tex`

2.2.3.3 Double Q Learning

Consider the MDP given by fig (2.10), the agent begins in state A and can either move left or right for a reward in 0. Moving right will terminate the process with reward 0. Moving left (with reward 0) puts the agent in a state where it can choose amongst a multitude of actions whose rewards are normaly distributed with a mean of 0.1 and variance of 1. During policy iteration we select actions according to $\underset{a}{argmax}\, Q(s, a)$, this incurs a significant *positive* bias due to the nature of the maximization, in fig (2.10) the agent will always choose the left action as it seeks to maximise the expected return and certain returns from going to B will be greater than 0;

Figure 2.10: Maximization Bias Example



†Source: (Sutton 2018)

instead the agent should choose to go right, because the expected value of that action is greater than the expected value of going to $B \backsim -0.1$. The error occurs because the same samples are used to determine both the value of each action and to select the maximising action. (van Hasselt, Guez & Silver 2015) address this within the context of deep Q learning by proposing the following variant of the temporal difference δ_t error:

$$y_i = R_{t+1} + \gamma Q(S_{t+1}, \underset{a}{argmax}\, Q(S_{t+1}, A_{t+1}\ ;\ \theta)\ ;\ \theta^{-}) \tag{2.55}$$

The Q value of the next state is calculated first by finding the most optimal action for that next state according to our behaviour policy parameters θ and then selecting the Q value of the action according to its value under the target network's parameters θ^{-}. By doing so we decouple the estimation of an actions value from its selection. The authors show that this is effective in combination with the previously discussed improvements.

2.2.3.4 Distributional Reinforcement Learning

Recalling equation (2.41), the expected future discounted reward is a weighted mean, whose final output is a scalar value. This mean ignors the variance and possible multi-modality of the underlying distribution. Assuming that the process of generalized policy iteration reaches a fixed point $\pi^* = \pi_\theta$ where the policy under the neural network's parameters aproximates the optimal policy arbitarily well, this approximation is only in their means, and equality of means does gurantee that the "shape" of the underlying distributions are the same. Motivated by these shortcomings, (Bellemare, Dabney & Munos 2017) proposed an alternative distributional perspective on reinforcement learning. The authors reinterpret equation (2.41) under a distributional variant; first they contextualize the Q learning update under the *Bellman operator* $\mathcal{T}^\pi$ and in the control setting as the application of the *"optimality operator"* $\mathcal{T}$:

$$\begin{gathered} x \in \mathbf{S},\ \ x' := x_{t+1} \\ \mathcal{T}^\pi Q(x,a) = \mathbb{E}[R(x,a)] + \gamma \underset{P,\pi}{\mathbb{E}}[Q(x',a')] \\ \rightarrow \mathcal{T} Q(x,a) = \mathbb{E}[R(x,a)] + \gamma \underset{P,\pi}{\mathbb{E}}[\max_{a' \in A} Q(x',a')] \end{gathered} \tag{2.56}$$

This reflects the same objective as (2.41), where successive applications of the Bellman operator produce new policies paramterised by the updated estimates to the Q value function. Additionally we recognise that the reward $R(x,a)$ the agent receives depends on the action taken in that particular state. Under the distributional setting the state is denoted $x' \backsim P(\cdot \mid x,a)$

as the state is assumed to be sampled from the underlying steady state distribution of the MDP[25]. Considering the markov property[26], the discounted future returns γG_{t+1} of a particular state can be considered a random variable Z distrbuted according to the probability of attaining certain returns dependent on the action taken in each state, formally $R \in \mathbf{Z}$, where $\mathbf{Z}$ represents a mapping from state-action pairs to *distributions* of returns. Furthermore the authors describe "transition operator" as an operator that transforms the current distribution of discounted returns into the distribution of discounted returns for the next state:

$$\begin{gathered}\mathcal{P}^{\pi} : Z \mapsto Z' \\ \mathcal{P}^{\pi} Z(x, a) \overset{D}{:=} Z(x', a') \\ x' \backsim P(\cdot \mid x, a), \;\; a' \backsim \pi(\cdot \mid x')\end{gathered} \tag{2.57}$$

In this case, the transition operator will map the distribution of returns under the current state and action, to the distribution returns under the next state action pair which is distributed according to the same law (denoted by $\overset{D}{:=}$) as the previous distribution. They then reformulate the Bellman operator:

$$\mathcal{T}^{\pi} Z(x, a) \overset{D}{:=} R(x, a) + \gamma \mathcal{P}^{\pi} Z(x, a) \tag{2.58}$$

Three sources of randomness arise under this interpretation:

- Randomness in reward $R(x, a)$.
- Randomness in the transition $\mathcal{P}^{\pi}$.
- Randomness in the next state-value distribution $Z(x', a')$.

These sources are important to observe when considering if the distributional variant of the Bellman equation converges to a fixed point Z^{π}. In order to measure the relative distance between an arbitary distribution, the authors utilise the *Wasserstein* distance defined on cumulative density functions (c.d.f) of arbitary probability density functions (p.d.f). If a random variable is distributed according to $F(x)$ then it's c.d.f, also known as a quantile function, is $F^{-1}(x)$. The *Wasserstein* distance is an L_p metric for some $p \in \mathbb{N}$ that measures the distance between two random variables U, V on the space of possible samples (values the variable can take on) $\omega \in \Omega$ is the smallest possible difference between the probability density of each sample whose support

[25] See equation (2.41)

[26] See **2.2.1**

is the metric space $[0, 1]$:

$$\begin{aligned} W_p(F, G) &= \left(\int_0^1 |F_U^{-1}(\omega) - F_V^{-1}(\omega)|^p \, \mathrm{d}x \right)^{\frac{1}{p}} \\ &= \inf_{U,V} \|U - V\|_p \end{aligned} \tag{2.59}$$

To measure the distance between value distributions Z_1, Z_2, they define the maximal form of the Wasserstein metric as the highest element-wise absolute difference between the two:

$$\bar{d}_p(Z_1, Z_2) \coloneqq \sup_{x,a} W_p(Z_1(x, a), Z_2(x, a)) \tag{2.60}$$

The authors show that similarly to equation (2.20) the $\bar{d}_p$ metric is a contraction in γ by *Banach's fixed point theorem*[27]:

$$\begin{aligned} \bar{d}_p(\mathcal{T}^\pi Z_1, \mathcal{T}^\pi Z_2) &\leqslant \bar{d}_p(R + \gamma \mathcal{P}^\pi Z_1, R + \gamma \mathcal{P}^\pi Z_2) \\ &\leqslant \bar{d}_p(\gamma \mathcal{P}^\pi Z_1, \gamma \mathcal{P}^\pi Z_2) \\ &\leqslant |\gamma| \bar{d}_p(\mathcal{P}^\pi Z_1, \mathcal{P}^\pi Z_2) \\ &\leqslant \gamma \bar{d}_p(Z_1, Z_2) \end{aligned} \tag{2.61}$$

In theory, these results can be applied directly by substituting (2.56) for the Q learning update, however the issue arises when incorporating function approximation in an effort to parameterise the distribution of returns for each state-action pair. Starting from random, the network's parameters are continually updated to more closely reflect the actual distribution of returns for each state action pair. As a result, the support[28] of the predicted distribution, the output of the neural network, will drift over time. Generalised policy iteration continuously produces new policies, in this case, the result is that each policy is defined on a different support. This poses an issue, as the distance between two random variables on disjoint supports is infinite (Bellemare et al. 2017). In practice the authors overcome this by utilising a $\mathbf{\Phi}_2$ projection of the output distribution of the network onto a discrete support with N atoms and then minimzing the *Kullback Liebler* divergence between the target and the prediction.

However as (Dabney, Rowland, Bellemare & Munos 2017) note in their follow up work, the conjugation of the $\mathbf{\Phi}_2$ operator with the Bellman operator $\mathcal{T}^\pi$ does not necessarily converge to the *same fixed point* as the repeated application of the $\mathcal{T}^\pi$. Additionally, in practice the $\mathbf{\Phi}_2$

[27] See Appendix

[28] Range of values that the expected discounted return of a state action pair can take on with a non-zero probability

projection requires conducting a nearest neighbour search between the atoms of the predicted distribution and the target distribution, and assigning credit proportionally to the 2 nearest neighbours, this can be unwieldy to implement. (Dabney et al. 2017) instead propose *quantile regression* as a way around these limitations.

Where (Bellemare et al. 2017) choose to fix the support and vary the probabilities,(Dabney et al. 2017) invert this formulation to instead fix the probabilities and vary the support. The complete metric space of probability $[0, 1]$ is discretized into τ uniform intervals and the appropriate density of returns is allocated to each interval. First they show that the projection Π_{W_1} of any arbitary value distribution Z onto a distribution of τ number of quantiles Z_Q, preserves the convergence to a fixed point under the conjugation of $\bar{d}_p \circ \Pi_{W_1}$:

$$\bar{d}_p(\Pi_{W_1}\mathcal{T}^\pi Z_1, \Pi_{W_1}\mathcal{T}^\pi Z_2) \leqslant \gamma \bar{d}_p(Z_1, Z_2) \tag{2.62}$$

The Q network aims to predict the quantiles θ_i for each action resulting in a vector $Z_\theta \in Z_Q$ for each action such that $Z_{\theta_i} = P(y_i < \theta_i)$[29]. In order to project an arbitary distribution onto a distribution of N quantiles it follows that the most suitable distribution is that which minizes the *Wasserstein* metric $\Pi_{W_1} Z = \arg\min W_p(Z, Z_Q)$. The authors show that the value on the interval of each quantile that minimizes this distance is the median of the interval:

$$\begin{gathered} \arg\min \int_{\tau}^{\tau'} |F^{-1}(\omega) - \theta| \, d\omega \\ \text{Is given by} \\ \theta : F(\theta) = \frac{\tau + \tau'}{2} \end{gathered} \tag{2.63}$$

They utilise *quantile regression loss* developed within economics, to fit the predicted quantile distribution of action-values to a target quantile distribution. Formally, a dirac delta function positioned on the quantiled is scaled proportionally when the distance $\xi = \hat{Z} - \theta$ is overestimated $\xi \geq 0$ or underestimated $\xi < 0$. Quantile regression is used to fit the predicted quantile to the target by minimizing the following objective:

$$\begin{gathered} \rho_\tau(u) = \begin{cases} \tau \cdot u & u \geq 0 \\ (1 - \tau) \cdot u & u < 0 \end{cases} \\ \mathcal{L}_{QR}^{\tau} := \sum_{i=1}^{N} \mathop{\mathbb{E}}_{\hat{Z} \backsim Z} [\rho_\tau(\hat{Z} - \theta_i)] \end{gathered} \tag{2.64}$$

[29] For instance if $\tau = 6$ then $Z_{\theta_5} = y_i$ s.t $P(y_i < Q(\frac{5}{6})) \mid F^{-1}(Q(\frac{5}{6})) = \frac{5}{6}$

This is in turn applied to the Bellman update:

$$\begin{aligned} \mathcal{T}Q(x,a) &= \mathbb{E}[R(x,a)] + \gamma \mathbb{E}_{z' \backsim P}[\max_{a'} Q(x',a')] \\ \rightarrow \mathcal{T}Z(x,a) &= R(x,a) + \gamma Z(x',a') \\ x' \backsim P(\cdot \mid x,a), a' &= \arg\max_{a'} \mathbb{E}_{z \backsim Z(x',a')}[z] \end{aligned} \tag{2.65}$$

From which the authors derive the TD control update:

$$\theta_i(x) \leftarrow \theta_i(x) + \alpha(\hat{\tau}_i - \delta_{r+\gamma<\theta_i(x)}) \tag{2.66}$$

(Dabney et al. 2017) results show that learners that incorporate the distribution of returns (**risk-sensitive learners**) perform much better and converge to optimal policies much faster, this increase in performance is no doubt due to the greater representational power of distributions.

2.3 Multi-Agent Reinforcement Learning

I will now turn to examine the generalization of single-agent MDPs to settings that involve multiple agents and the dynamics between them. Although the field of multi-agent reinforcement learning spans many possible approaches, the unifying framework underlying all of them is that of a Stochastic Game which bares its roots in Game Theory. Within the scope of this project, a particularly recent advancement at the intersection of reinforcement learning and games with an infinite number of players, newly developed mean field multi agent learning algorithms are also discussed.

2.3.1 Stochastic Games

Many of the concepts introduced within the framework of a markov decision process have their origins in decision theory, particularly expected utility theory (Sutton 2018). As such many of the concepts introduced so far have synonymous terminology in game theory, which I will introduce and make use of for the convenience of describing later topics.

A normal form game is defined by a set of players (agents) who take actions within an environment that maximise their expected discounted payoffs $\mathbb{E}[\gamma G_{t+1}]$ according to some strategy π. Normal form games are defined to be **non-cooperative**, **static** and of **complete information**, indicating that although agent's priorities may align, they each are self interested with respect to their own reward function. Static defines the property that each player chooses their action without knowledge of the actions being played by other players at the current time-step. Complete knowledge is defined not only in terms of complete knowledge of the environment, but also the knowledge that every agent knows that every agent knows that every other agent has knowledge of the environment,*ad infimum.*

A stochastic game is a generalization over both MDPs and repeated games (games played over a span of time) which can be described as a collection of normal form games which are played by the agents where the game played at any point in time probabilistically depends on the previously game played and the actions taken therein (Shoham 2009). Formally, a stochastic game is defined as the tuple $\Gamma \coloneqq (\mathcal{S}, \mathcal{A}^1, \ldots, \mathcal{A}^n, r^1, \ldots r^N, p, \gamma)$ where $\mathcal{S}$ denotes the familiar state space in which the agents act in, $\mathcal{A}^j$ represents the action space of agent $j \in 1, \ldots, N$, the actions in turn are drawn from some stationary distribution $p = \mathcal{S} \times \mathcal{A}^1 \times \ldots \times \mathcal{A}^n \rightarrow \Omega(\mathcal{S})$ where $\Omega(\mathcal{S})$ represent the collection of distributions over the state space (Yang, Luo, Li, Zhou, Zhang & Wang 2018). The primary departure of interest from the previously introduced setting is the reward function r^j of each agent as they now take into consideration the actions of all the other agents:

$$r^j : \mathcal{S} \times \mathcal{A}^1 \times \ldots \times \mathcal{A}^n \rightarrow \mathbb{R} \tag{2.67}$$

Likewise, we also define the joint strategy of all agents:

$$\boldsymbol{\pi} \triangleq [\pi^1, \pi^2, \ldots, \pi^N] \tag{2.68}$$

From this we can derive the Q and value function of such agents:

$$\begin{gathered} v^j_{\boldsymbol{\pi}}(s) = v^j(s; \boldsymbol{\pi}) = \sum_{t=0}^{\infty} \gamma^t \mathop{\mathbb{E}}_{\boldsymbol{\pi}, p}[r^j_t \mid s_0 = s, \boldsymbol{\pi}] \\ Q^j_{\pi}(s, \mathbf{a}) = r^j(s, \mathbf{a}) + \gamma \mathop{\mathbb{E}}_{s' \backsim p}[v^j_{\boldsymbol{\pi}}(s')] \end{gathered} \tag{2.69}$$

In multi-agent reinforcement learning, each agent aims to learn an optimal policy to maximize its respective value function, indicating that any optimal value function must take into account not only the agent's policy, but the joint policy of all other agents; following the logic set out in equations (2.29-32):

$$v^j(s; \boldsymbol{\pi}_*) = v^j(s; \pi^j_*, \boldsymbol{\pi}^{-j}_*) \geq v^j(s; \pi^j, \boldsymbol{\pi}^{-j}_*) \tag{2.70}$$

Where $\mathbf{a} \triangleq [a^1, \ldots, a^N]$ is defined as the joint action over all agents. Here the optimal joint policy $\boldsymbol{\pi}_*$ as been decomposed into the policy of the individual agent π^j and the joint policy of all other agents except j $\boldsymbol{\pi}_*^{-j}$. In game theoretic terms, the optimal joint policy represents the *Nash equilibrium* of the sytem, defined as the deterministic joint optimal strategy from which the agents do not deviate from because it maximises the expected discounted payoff for all agents. In a nash equilibrium, each agent acts with the best response π_*^j to other provided that that all other agents adhere to $\boldsymbol{\pi}_*^{-j}$. (Hu & Wellman 2003) defined an iterative procedure as follows:

- Solve the Nash equilibrium of the current state game
- Bootstrap the current estimation of the Q value for each learning with the new Nash equilibrium value

The conjugation of these operations is collectively define the *Nash operator* $\mathscr{H}^{Nash}$:

$$\mathscr{H}^{Nash}Q(s, \mathbf{a}) = \underset{s' \backsim p}{\mathbb{E}}[\mathbf{r}(s, \mathbf{a}) + \gamma \mathbf{v}^{Nash}(s')] \tag{2.71}$$

Where $\mathbf{v}$ defines the joint value function governing the value of states under the Nash equilibrium and $\mathbf{r}$ represents joint rewards of all agents. (Hu & Wellman 2003) show that this operator forms a contraction mapping in similar fashion to (2.61), indicating that the Q function will eventually converge to the values that constitute the Nash equilibrium of the game, the *Nash Q value.*

One can see that the dimension of the dimension of cartesian product of the joint action $\mathbf{a}$ grows proportionally with respect to the number agents, this limitation was initially thought to be computationally intractable for a large to infinte number of players, (Yang et al. 2018) circumvent this limitation by introducing the mean field game framework.

2.3.2 Mean Field Games

Owing it's origin to mean field theories in physics, mean field games first pioneered by (Lasry & Lions 2007) examine classes of games that have extremely large numbers of players,particularly in the infinite limit. Many of the intractabilities associated with games of many players simplify in the infinite limit; instead of considering the interaction of the agent with all other agents, we consider the pairwise interaction between the agent and some virtual *mean* agent. Under the standard assumptions of mean-field approximation, each agent is assumed to be *indistinguishable* and *interchangable*, indicating that given an indexed set of agents, the ordering of their indices does not have an effect on their interactions, this assumption will be relaxed later when multiple types of agents are introduced. Within the context of reinforcement learning, (Yang et al. 2018) factorize the Q function into only the pairwise local interactions in the *neighbourhood* of the agent:

$$
\begin{gathered}
Q^j = \frac{1}{N^j} \sum_{k \in \mathcal{N}(j)} Q^j(s, a^j, a^k) \\
N^j = |\mathcal{N}(j)|
\end{gathered}
\tag{2.72}
$$

Where $\mathcal{N}(j)$ is the index set of neighbouring agents determined by different applications and a^j is the discrete acion space of agent j and a^k represents the *empirical distribution* of the one-hot encoded neighbouring agent's actions:

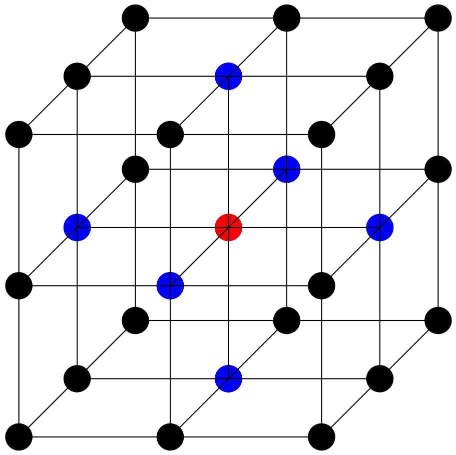
$$
\begin{gathered}
a^1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}_1 \quad a^2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}_2 \quad \dots \; a^N = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}_N \\
\mathbf{a}^k \approx \mathbf{a}^{-j} \text{ where } a^{-j} = \frac{1}{N^j} \sum_k a^k
\end{gathered}
\tag{2.73}
$$

Although this reduces the complexity significantly, it still preserves global interactions implicitly (Blume 1993). Assuming that the joint Q function is *Lipschitz continuous* [30], (Yang et al. 2018) show that if we examine the interaction between a central agent j and an arbitary neighbour k as $Q^j(s, a^j, a^k)$ and evaluate the second order Taylor approximation of this interaction, we find that from the perspective of agent j, it's immediate neighbour will choose it's actions based on its own respective neighbours. This is tantamount to the action taken by immediate neighbour k being determined according to some external action *distribution* of it's own neighbours, and so by the findings of (Blume 1993) we can approximate the pairwise interactions of each neighbour as that between j and the virtual mean agent k that captures the mean effect of all of the neighbours within j's neighbourhood while conserverving global interactions implicitly.

This is depicted in figure (2.11) where the central agent j is shown as the node in red, the central node only considers the actions taken by its immediate neighbours on the lattice shown in in blue, rather than taking into considerations the actions of all nodes on the lattice. (Yang et al. 2018) model the distribution over the actions of the virtual mean agent with a *Boltzmann*

[30] See Appendix

Figure 2.11: Mean field interactions



policy at time t parameterised by the empirical distribution (2.73) of the neighbouring actions:

$$\pi_t^j(a^j \mid s, a^{-j}) = \frac{\exp(-\beta Q_t^j(s, a^j, a^{-j}))}{\sum_{a^j \in \mathbf{A}^j} \exp(-\beta Q_t^j(s, a^j, a^{-j}))} \tag{2.74}$$

At each time step, the tuple (s, a^k, r^j, s') is stored in memory, each agent then computes the empirical distribution of the neighbouring agents actions that have been optimalling selected according to a policy π_t^k parameterised by the previous distributions of actions a_-^{-k} read from memory:

$$a^{-j} = \frac{1}{N^j} \sum_k \arg\max_{a^k} \pi_t^k(\cdot \mid s, a_-^{-k}) \tag{2.75}$$

Then the policy is updated accordingly using eq. (2.75), this completes the generalised policy iteration procedure. (Yang et al. 2018) produce the **mean field** value function $\mathbf{v}^{MF} \triangleq [v^1(s), \ldots, v^N(s)]$ at time t:

$$v_t^j(s') = \sum_{a^j} \pi_t^j(s^j \mid s', a^{-j}) \mathbb{E}_{a^{-j}(\mathbf{a}^{-j}) \backsim \boldsymbol{\pi}_t^{-j}} [Q_t^j(s', a^j, a^{-j})] \tag{2.76}$$

This is consolidated into a Q value update for some learning rate α where the Q values are assumed to be greey w.r.t the policy:

$$Q_{t+1}^j(s, a^j, a^{-j}) = (1-\alpha) Q_t^j(s, a^j, a^{-j}) + \alpha[r^j + \gamma v_t^j(s')] \tag{2.77}$$

The authors then prove that this collective procedure, defined as the mean-field operator is also a contraction mapping, converging to the optimal Q values for each agent:

$$\mathscr{H}^{MF}\mathbf{Q}(s, \mathbf{a}) = \mathop{\mathbb{E}}_{s' \backsim p}[\mathbf{r}(s, \mathbf{a}) + \gamma \mathbf{v}^{MF}(s')] \tag{2.78}$$

In practice they construct a DQN as described previously, and the agent is trained to minimize the following loss function in the same fashion as equation (2.48):

$$\begin{gathered} L(y^j, Q(s, a^j, a^{-j}))) = (y^j - Q(s, a^j, a^{-j}))^2 \\ \nabla_{\mathbf{W}} L(y^j, Q(s, a^j, a^{-j}) = (y^j - Q(s, a^j, a^{-j}))\nabla_{\mathbf{W}} Q(s, a^j, a^{-j}) \end{gathered} \tag{2.79}$$

Where $y^j = r^j + \gamma v_{\theta^-}^{MF}(s')$ is the TD target constructed from the weights θ^- of the target network.

2.3.3 Spatial Congestion Games

(Mguni, Jennings & Munoz de Cote 2018) develop results similar to (Yang et al. 2018) for different settings, particularly that of spatial congestion games. A spatial congestion game is a variant of a stochastic game where the agent's rewards are based on some shared resource usage, in this a case a region of space in $\mathbb{R}^2$.

> In the spatial congestion (SC) game there are N agents, given some initial position $x_0 \in \mathbf{S}$, each agent chooses an action in order to move to a desired location $x_T \in \mathbf{S}$ which is a terminal state. Certain areas of $\mathbf{S}$ are more desirable to occupy than others, however the agents are averse to occupying crowded areas ((Mguni et al. 2018))

They define the joint state of the agents as their combined density as anypoint in space as the average dirac deltas centered on the position of neighbouring agents at coordinate ${x_k}$[31]:

$$m_{x_k} \triangleq \frac{1}{N} \sum_{i=1}^{N} \delta_{x_k^i} \tag{2.80}$$

The go on to model the desireability of a coordinate x_t with the following loss function:

$$L(x_t, m_{x_t}) = \frac{1}{2\pi\sqrt{|\Sigma|}} \frac{\exp(-(x_t - \mu)^\top \Sigma^{-1} (x_t - \mu))}{(1 + m_{x_t})^\alpha} \tag{2.81}$$

With $\mu \in \mathbb{R}^2$ representing the centroid and $\Sigma \propto 1_{2\times 2}$ derived from the covariance of the coordinates. The hyperparameter α measures the agent's averseness to dense areas, $\alpha > 0$ representing greater averseness and $\alpha < 0$ indicating an affinity for dense areas. The reward for each agent is determined based on the combined expected desireability of the position of each agent:

$$\mathbb{E}\left[\sum_{k=t}^{T} L(x_k, m_{x_k}, a_k) \,\middle|\, a_k \backsim \pi\right] \tag{2.82}$$

Although the authors develop the results specifically for the Actor-Critic architecture, as (Yang et al. 2018) shows, the same convergence gurantees can be extended to off-policy Q-Learning.

[31] This can be interpreted the average of the one-hot encoded action vectors

2.3.4 Mean Field Multi Type Reinforcement Learning

(Subramanian, Poupart, Taylor & Hegde 2020), build directly ontop of the results of (Yang et al. 2018) by extending the framework to incorporate multiple heterogenous types of agents. They introduce the loose term of *type*, within the paper a unique type refers to agents that share the same action space but have different reward structures, this will be the definition used throughout here. They extend the Q function developed by (Yang et al. 2018) by partioning the *empirical distribution* of the actions of neighbours, into an empirical distribution of the actions for each type of neighbour:

$$Q^j(s, \mathbf{a}) = \frac{1}{M} \sum_{i=1}^{M} [Q^j(s, a^j, a_1^{-j}, a_2^{-j}, \ldots, a_M^{-j})] \tag{2.83}$$

For each central agent j, there exist M total types of surrounding agents, and a_m^{-j} denotes the empirical distribution of the actions of neighbours of type m. The rest of (Yang et al. 2018) results follow accordingly. The primary contribution of this paper to this work was to show that although (Yang et al. 2018)'s work converged in settings with heterogenous agents, by explicitly considering types, agents often perform better and exhibit greater accumulated rewards in the long run as compared to settings that do not incorporate types explicitly.

2.4 Related Work

2.4.1 MCMC methods for lattice models

2.4.2 Deep Learning methods for lattice models

2.4.2.1 Alpha-Fold

2.4.3 Reinforcement Learning methods for lattice models

Chapter 3

System Requirements and Specification

3.1 PSP as a cooperative optimisation problem

3.2 Protein structures as distributions

3.3 Overcoming energy barriers after the hydrophobic collapse

Chapter 4

System Design and Analysis

4.1 Environment and actions

In contrast to other approaches on a 2D lattice, I will instead model the environment using a 3D FCC lattice, which has the added benefit of being computationally trivial to model. I make use of the matrix representation:

$$\mathbf{e} = \begin{bmatrix} 0 & 0.5 & 0.5 \\ 0.5 & 0 & 0.5 \\ 0.5 & 0.5 & 0 \end{bmatrix} \tag{4.1}$$

Any point in space is represented as a linear combination of the column space of the bravaise lattice, within a discrete action setting, this can be viewed as a choice among 3^3 possible actions $a = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$ s.t $(x, y, z) \in \{-1, 0, 1\}$. The chains of amino acids are initialised as a straight line in space, and each time-step each residue moves to another position x_t by selecting a movement vector $a_\mathbf{j}$ to be applied to it's current position. The transition function is characterised as:

$$T(x_t, a_\mathbf{j}) = \mathbf{e} \cdot a_\mathbf{j} \tag{4.2}$$

At any point in time, I also consider the joint state m_{x_t} as described by (Mguni et al. 2018) to model the densities of residues in 3D space.

4.2 Revising reward structures

Instead of using the standard rewards associated with the HP model, I will instead divide each of the residues into their respective categories under the $hHPNX$ model with the signs of values inverted, each residue acts as their own agent with their own reward function determined according to the column space in table (2.3). In addition to these rewards, the agents are further penalized with with a reward of -10 for occupying the same space as another residue to discourage overlapping residues:

Table 4.1: Reward structure

Table 4.2

	h	H	P	N	X
h	-2	4	0	0	0
H	4	3	0	0	0
P	0	0	-1	1	0
N	0	0	1	-1	0
X	0	0	0	0	0
$x' = x$	-10	-10	-10	-10	-10

In addition to these rewards, the agents are also weighted proportionally to the spatial congestion as described by (Mguni et al. 2018) as follows:

$$R(m_{x_t}, a') = \psi R_{hHPNX} + (1 - \psi)\frac{\exp(-(x_t - \mu)^{\top}\Sigma^{-1}(x_t - \mu))}{2\pi\sqrt{|\Sigma|}(1 + m_{x_t})^{\alpha}} \tag{4.3}$$

Where $\alpha < 0$ encourages the occupation of dense areas and $0 < \psi < 1$ is a hyperparameter that weights the agent's intrinsic reward against the global reward, with higher values of ψ favouring intrinsic reward more than global reward. This serves two purposes, it offsets the sparse rewards of the neutral group X and it serves to provide a richer training signal in the proteins initial denatured state, guiding it to the hydrophobic collapse, both rewards are complementary and provide different benefits at different stages of training, ψ is set initially low $\backsim$ 0.2 and annealed up to $\backsim$ 0.9. This heterogenous reward structure is afforded by the gurantees provided by (Subramanian et al. 2020), which ensures convergence with lower bounds within the mean-field scheme when considering multiple types of agents.

4.3 Mean field multi-type spatial congestion games

As suggested previously, each residue is modelled as an individual agent belonging to one of the categories as listed in the hHPNX model. The reward structure of each type is heterogenous and so can be considered to be members of disjoint types. I model the inter-domain cooperativity by positively rewarding the agents for the occupation of lattice sites with:

- Favourable contacts with neighbouring residues
- Dense occupation of other residues

Rewards for the occupation of dense areas of space encourages cooperativity amongst all agents and acts as a guide to the most compact states. The reward for favourable contacts also filters the landscape of possible conformations leaving only those that maximise the number of hydrophic contacts while being optimally compact, this mirrors the work of (Yang et al. 2013) regarding the two tiers of interactions. The reward for dense packing is initially high, driving the fast interactions due rapidly accumulating rewards as the density increases. As the value of ψ is annealed, the specific contacts among residues becomes more important as they rearrange themselves while in a compact form, this resembles the tier 2 of slower interactions that guide the proteins to the bottom of the *Gibbs energy funnel.* The use of the *hHPNX* scheme on a 3D FCC lattice also reduces the number of degenerative results as (Hoque et al. 2009) shows.

Each agent's Q function is updated in accordance with the multi-type paradigm which was shown to be provably convergent:

$$Q^j_{t+1}(s, a^j, a^{-j}, a^{-j}_1, \ldots, a^{-j}_M) = (1-\alpha)Q^j_t(s, a^j, a^{-j}, a^{-j}_1, \ldots, a^{-j}_M) + \alpha[R^j(m_{x_t}, a^j) + \gamma v^j_t(s')] \tag{4.4}$$

Where a^j represents the action taken by the central agent, a^{-j} and a^{-j}_m represents the empirical distribution of all neighbours and neighbours of type m respectively.

4.4 Risk sensitive agents

As (Lyu & Amato 2018) show, distributional reinforcement learning proves to be effective in multi-agent settings; an agent is better able to *distribute blame* proportionally among the actions of neighbouring agents. For instance, for a given transition $s \to s'$, if we consider the joint action taken by all agents $\mathbf{a}$, then a central agent j may determine that the action taken by a specific agent $\mathbf{a}_k$ was the most influential factor in the transition to specific state s'; this is also known as the *credit assignment problem.* In their work, they use quantile networks to determine a state-specific learning rate α that changes based on the interaction with other agents, becoming more or less exploratory as the situation deems. I instead take an alternative approach in favour of *reward shaping.*

4.4.1 Reward Shaping with Quantile Estimates

(Devlin, Yliniemi, Kudenko & Tumer 2014) show that

Let $\hat{\mathbf{q}}_{i,k}^{-j}$ denote the quantile distribution of a selected action i by neighbour k. First I average the quantiles returns for each action across all neighbours:

$$\mathbf{a}_i^{-j} = \frac{1}{K} \cdot \sum_K \hat{\mathbf{q}}_{i,k}^{-j} \tag{4.5}$$

The weighted sum of the quantile distributions is then taken *indiscriminately* and *type-wise* by multiplying the average returns for each action by the empirical probability of that action, this produces $M + 1$ quantile estimates:

$$\mathbf{a}^{-j} = \sum_{i=1}^{|\mathbf{A}|} p(a_i) \cdot \mathbf{a}_i^{-j} \tag{4.6}$$

- A quantile estimate for the returns of all the neighbours combined

$$\mathbf{a}_m^{-j} = \sum_{i=1}^{|\mathbf{A}|} p(a_{i,m}) \cdot \mathbf{a}_{i,m}^{-j} \tag{4.7}$$

- A quantile estimate of the contribution of each type to the total returns

Bibliography

Alberts, B. (2002), *Molecular biology of the cell*, Garland Science, New York.

Anfinsen, C. B. (1972), 'The formation and stabilization of protein structure', *Biochemical Journal* **128**(4), 737–749.
URL: `https://doi.org/10.1042/bj1280737`

Bellemare, M. G., Dabney, W. & Munos, R. (2017), 'A distributional perspective on reinforcement learning'.

Blume, L. E. (1993), 'The statistical mechanics of strategic interaction', *Games and Economic Behavior* **5**(3), 387–424.
URL: `https://doi.org/10.1006/game.1993.1023`

Bornberg-Bauer, E. (1997), Chain growth algorithms for HP-type lattice proteins, *in* 'Proceedings of the first annual international conference on Computational molecular biology', ACM Press.
URL: `https://doi.org/10.1145/267521.267528`

Boyle, A. L. (2018), Applications of de novo designed peptides, *in* 'Peptide Applications in Biomedicine, Biotechnology and Bioengineering', Elsevier, pp. 51–86.
URL: `https://doi.org/10.1016/b978-0-08-100736-5.00003-x`

Chayen, N. (2005), 'Methods for separating nucleation and growth in protein crystallisation', *Progress in Biophysics and Molecular Biology* **88**(3), 329–337.
URL: `https://doi.org/10.1016/j.pbiomolbio.2004.07.007`

Crippen, G. M. (1991), 'Prediction of protein folding from amino acid sequence over discrete conformation spaces', *Biochemistry* **30**(17), 4232–4237.
URL: `https://doi.org/10.1021/bi00231a018`

Dabney, W., Rowland, M., Bellemare, M. G. & Munos, R. (2017), 'Distributional reinforcement learning with quantile regression'.

Devlin, S., Yliniemi, L., Kudenko, D. & Tumer, K. (2014), Potential-based difference rewards for multiagent reinforcement learning, *in* 'Proceedings of the 2014 International Conference on Autonomous Agents and Multi-Agent Systems', AAMAS '14, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, p. 165–172.

Dill, K. A., Bromberg, S., Yue, K., Chan, H. S., Ftebig, K. M., Yee, D. P. & Thomas, P. D. (2008), 'Principles of protein folding - a perspective from simple exact models', *Protein Science* **4**(4), 561–602.
URL: `https://doi.org/10.1002/pro.5560040401`

Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M. & Silver, D. (2017), 'Rainbow: Combining improvements in deep reinforcement learning'.

Hoque, T., Chetty, M. & Sattar, A. (2009), 'Extended HP model for protein structure prediction', *Journal of Computational Biology* **16**(1), 85–103.
URL: `https://doi.org/10.1089/cmb.2008.0082`

Hu, J. & Wellman, M. (2003), 'Nash q-learning for general-sum stochastic games.', *Journal of Machine Learning Research* **4**, 1039–1069.

Jaffe, R. (2018), *The physics of energy*, Cambridge University Press, Cambridge, United Kingdom New York, NY.

Kittel, C. (2005), *Introduction to solid state physics*, Wiley, Hoboken, NJ.

Lasry, J.-M. & Lions, P.-L. (2007), 'Mean field games', *Japanese Journal of Mathematics* **2**, 229–260.

Lau, K. F. & Dill, K. A. (1989), 'A lattice statistical mechanics model of the conformational and sequence spaces of proteins', *Macromolecules* **22**(10), 3986–3997.
URL: `https://doi.org/10.1021/ma00200a030`

Lee, J., Bahri, Y., Novak, R., Schoenholz, S. S., Pennington, J. & Sohl-Dickstein, J. (2017), 'Deep neural networks as gaussian processes'.

Lee, J., Xiao, L., Schoenholz, S. S., Bahri, Y., Novak, R., Sohl-Dickstein, J. & Pennington, J. (2019), ‘Wide neural networks of any depth evolve as linear models under gradient descent’.

Lesk, A. (2018), *Introduction to bioinformatics*, Oxford University Press, Oxford New York.

Lucrezia, D. D., Slanzi, D., Poli, I., Polticelli, F. & Minervini, G. (2012), ‘Do natural proteins differ from random sequences polypeptides? natural vs. random proteins classification using an evolutionary neural network’, *PLoS ONE* **7**(5), e36634.
URL: `https://doi.org/10.1371/journal.pone.0036634`

Lyu, X. & Amato, C. (2018), ‘Likelihood quantile networks for coordinating multi-agent reinforcement learning’.

Margossian, C. C. (2019), ‘A review of automatic differentiation and its efficient implementation’, *WIREs Data Mining and Knowledge Discovery* **9**(4).
URL: `https://doi.org/10.1002/widm.1305`

Mguni, D., Jennings, J. & Munoz de Cote, E. (2018), ‘Decentralised learning in systems with many, many strategic agents’.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. & Hassabis, D. (2015), ‘Human-level control through deep reinforcement learning’, *Nature* **518**(7540), 529–533.
URL: `https://doi.org/10.1038/nature14236`

Schaul, T., Quan, J., Antonoglou, I. & Silver, D. (2015), ‘Prioritized experience replay’.

Shoham, Y. (2009), *Multiagent systems : algorithmic, game-theoretic, and logical foundations*, Cambridge University Press, Cambridge New York.

Subramanian, S. G., Poupart, P., Taylor, M. E. & Hegde, N. (2020), Multi type mean field reinforcement learning, *in* ‘Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020)’, Auckland, New Zealand.

Sutton, R. (2018), *Reinforcement learning : an introduction*, The MIT Press, Cambridge, Massachusetts London, England.

Tsygvintsev, A. (2019), 'Natural versus random proteins: Nouvel neural network approach based on time series analysis'.
URL: https://doi.org/10.1101/687558

van Hasselt, H., Guez, A. & Silver, D. (2015), 'Deep reinforcement learning with double q-learning'.

Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M. & de Freitas, N. (2015), 'Dueling network architectures for deep reinforcement learning'.

Weiss, O., Jiménez-Montaño, M. A. & Herzel, H. (2000), 'Information content of protein sequences', *Journal of Theoretical Biology* **206**(3), 379–386.
URL: https://doi.org/10.1006/jtbi.2000.2138

Weisz, G., Budzianowski, P., Su, P.-H. & Gasic, M. (2018), 'Sample efficient deep reinforcement learning for dialogue systems with large action spaces', *IEEE/ACM Transactions on Audio, Speech, and Language Processing* **26**(11), 2083–2097.
URL: https://doi.org/10.1109/taslp.2018.2851664

Yang, L.-Q., Ji, X.-L. & Liu, S.-Q. (2013), 'The free energy landscape of protein folding and dynamics: a global view', *Journal of Biomolecular Structure and Dynamics* **31**(9), 982–992.
URL: https://doi.org/10.1080/07391102.2012.748536

Yang, Y., Luo, R., Li, M., Zhou, M., Zhang, W. & Wang, J. (2018), Mean field multi-agent reinforcement learning, *in* J. Dy & A. Krause, eds, 'Proceedings of the 35th International Conference on Machine Learning', Vol. 80 of *Proceedings of Machine Learning Research*, PMLR, Stockholmsmässan, Stockholm Sweden, pp. 5567–5576.

Yue, K. & Dill, K. A. (1993), 'Sequence-structure relationships in proteins and copolymers', *Physical Review E* **48**(3), 2267–2278.
URL: https://doi.org/10.1103/physreve.48.2267

Zhu, X., Cheng, T., Zhang, Q., Liu, L., He, J., Yao, S. & Zhou, W. (2019), 'Nn-sort: Neural network based data distribution-aware sorting'.