Deep Reinforcement Learning for Protein Folding in the Hydrophobic-Polar Model with Pull Moves

Haoyang Yu
Duke University
haoyang.yu079@duke.edu

John S. Schreck
The National Center for Atmospheric Research
schreck@ncar.edu

Wann-Jiun Ma
Research.ai, LLC
wannjiun.ma@theresearchai.com

Abstract

The protein folding problem of predicting folded protein structure given an amino acid sequence is a challenging and extensively studied NP-hard problem in computational biology. One common approach is to reduce the complexity using the hydrophobic-polar model. Although this model discretizes and simplifies the conformational space, the problem is still NP-complete. In this study, we formulate the protein folding problem as a deep reinforcement learning problem, using various deep neural networks and combining Monte Carlo tree search with the networks as a replacement for molecular dynamics and Monte Carlo simulation protocols in different protein folding simulated environments.

1 Introduction

Proteins are biomolecules consisting of one or more amino acid residues that are connected together through nearest-neighbor covalent bonds, and they perform a vast array of functions in the body, such as catalysing metabolic reactions and DNA replication. The sequence of amino acids determines the protein's unique three dimensional (3-D) structure as well as its specific biological function. The protein folding problem is the problem of predicting the native structure given an amino acid sequence. The problem's importance stems from the consequences of a failure in protein folding. When proteins do not fold properly, one of two problems usually occur: there may be a loss of function, in which there is a shortage of specialized proteins needed to perform specific job, or the misfolded proteins, which may damage their surrounding cellular environment. This can lead to several known diseases, such as Huntington's and Alzheimer's diseases, Cystic fibrosis, and Tay-Sachs disease. Being able to predict the folded structures and the folding dynamics given the amino acid sequence, therefore, has a wide range of applications, such as disease prevention and computational drug design. The main advantage of creating a computer algorithm to predict protein structures and simulate the folding process is to forgo the high degree of time, computational expenses, and expertise needed to use crystallography and nuclear magnetic resonance to identify the 3-D structures of a folded protein.

The hydrophobic-polar (HP) model, proposed by Lau and Dill (3), is one of the most commonly used coarse-grained models to study protein folding. In this model, the 20 amino acids are classified as hydrophobic (H) or polar (P), depending on their hydrophobicity. The motivation stems from the fact that hydrophobic and hydrophilic interactions play a major role in the folding process. The usual benchmark used to evaluate the protein folding algorithm is the folding (free) energy function, which considers only the interaction between two amino acids that are adjacent on a lattice, but not in the original sequence. In particular, the energy function assigns a value of -1 to every contact between two adjacent and non-sequence-adjacent H-H residues, and a value of zero to all other contacts. The

Third Workshop on Machine Learning and the Physical Sciences (NeurIPS 2020), Vancouver, Canada.

goal of the folding algorithm in its most simplest form is to minimize the energy function, or in other words, to discover the protein structure with the maximum number of H-H bonds on the lattice.

There are two ways to model this simplified problem. The first is to build the protein up one amino acid residue at a time, so that at time-step $t = 0$, the lattice contains only the first residue in the sequence, and at each subsequent time-step, an additional amino acid residue is placed on the lattice, adjacent to the last amino acid placed. We will refer to this as the "lattice environment". A sequence of amino acids is then folded by a self-avoiding walk of amino acids on a 2-D or 3-D lattice. Therefore, protein folding using the HP model in the lattice environment is a transformation from a sequence of amino acids to a folded protein with a lattice structure. It has been proven that while the HP model simplifies the problem, it is still NP-complete (1). Much of the previous work involving machine learning algorithms to solve the protein folding problem use this lattice environment. Most notably, the FoldingZero approach (5) used a coupled approach of a two-headed deep neural network with residual blocks, and an upper confidence bound for trees generated through Monte Carlo Tree Search (MCTS), that was able to give decent folding results in a reasonable amount of run-time. Additionally, Jafari and Javidi (2) used a Long Short-Term Memory (LSTM) network, coupled with a few reinforcement learning algorithms (Deep Q-Learning and it's variants), to outperform previous state-of-the-art approaches that were used to solve the protein folding problem. It should be noted that these aforementioned approaches assumed a 2-D lattice, however, in our work, we also present the results for 3-D lattices.

The second way to model the protein folding problem using the HP model is to place the entire unfolded sequence onto the lattice, and through a sequence of "pull" moves, as defined by Lesh et al. (4), fold the sequence into a final folded structure. We will refer to this as the chain environment. Note that the chain dynamics tells us something more about the problem, in particular, about the dynamics of the folding pathway. While we have not yet applied RL to this chain environment, within our knowledge, there is no existing work that has used RL in the chain environment.

In this paper, we first build upon previous work (5; 2) with the lattice environment, testing the validity of different neural network architectures and different deep RL algorithms. We then build and explain the chain environment, in which we plan to implement our DRL algorithms in the future to evaluate the merits of using the relatively more complicated chain environment compared to the more popular lattice environment.

The rest of this extended abstract is formatted as follows. Section 2 includes our results with the different algorithms and neural network architectures in the lattice environment. A more detailed explanation of the chain environment and its pull moves is presented in Section 3. Lastly, in Section 4, we conclude and discuss future developments and approaches.

2 Lattice Environment and Folding Results

We represent the lattice environment as a Markov decision process (MDP) defined by a tuple $(\mathcal{S}, \mathcal{A}, P, R)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $P(s'|s, a)$ is the transition probability, and R is the reward. We define a state at time-step t as s_t which includes information about the positions of the amino acids on the lattice and the folding trajectory at time-step t. The action at time t is denoted by a_t. To avoid the situation that the same amino acid occupies the same vertex, there are only three possible actions (forward, right, and left) are allowed. Adjacent amino acids in the sequence must be neighbors on the lattice and there is at most one amino acid which can occupy the vertex of the lattice. If two H amino acids which are not adjacent in the sequence are closest neighbors on the lattice, i.e., two adjacent and non-sequence-adjacent H-H residues, the HP model assigns negative energy (a value of -1) to such H-H contact of a folded protein. We calculate the total energy by counting the H-H contacts and use this number to be the reward. The objective of our deep RL algorithm is to design a folding policy to maximize the reward (minimize the negative energy) by playing this self-avoiding walk on a 2-D or 3-D lattice.

While there are many different ways to fold the chain and achieve the same maximal reward, the "density" of the resulting structure is also of importance. When proteins fold in the real world, they often fold into the most compact structure with the lowest energy level. We can see this difference in Figure 3 in Appendices when comparing the same sequence folded to the lowest energy level in a 2-D and 3-D lattice. As one might expect, folding in 3-D allows for a more compact structure while achieving the same, if not lower, energy level.

In Appendices, we discuss a few deep RL algorithms used to tackle smaller sequences and limitations of these algorithms. We found that both the value and policy iterations (10) became too expensive, as the time and memory complexity to build P is $O(3^n)$ for the 2-D lattice and $O(5^n)$ for the 3-D lattice, where n is the length of the sequence. It should be noted that for this particular problem, the optimal policy can be directly obtained by enumerating the final states without needing to perform value or policy iteration. This is because when we build P, we compute the reward for each state, and each state keeps tracking the folding trajectory. Therefore, once we have the transition matrix P, we can directly return the optimal policy given a starting state. However, we report results for the value and policy iteration algorithms to demonstrate that while the approach of building a transition probability matrix may work for shorter sequences, it becomes far too costly for longer ones. We were able to implement variants of DQN algorithms (6; 7; 11) for the protein folding problem, but only achieved reasonable results in an acceptable amount of time for sequences with length less than 20 using the methods. Also, we found that the vanilla MCTS (10) with a reasonable number of tree simulations for each folding step was too computationally expensive to converge to a solution reliably for sequences longer than length 20.

The tables below show our results for the different algorithms. We should note that all the algorithms besides AlphaGo Zero with pretraining are trained specifically for one sequence, and therefore the inference time for each sequence includes the training time. However, the AlphaGo Zero algorithm is trained on a set of sequences, and can generalize well, thus there is very little inference time, but also a significantly longer training time.

For the DQN algorithms, the Q function network was able to converge in 100k time steps, with a batch size of 32, updating the target network every 1000 time steps. We tried feeding the 2-D lattice to the networks both as an image and as a flattened vector representation, using a CNN and an MLP network respectively, but found no significant differences between the two. For the 3-D lattice, the input to the MLP is a flattened vector representation of a tensor representing the current state.

For the AlphaGo Zero algorithm, the input is encoded as an $M \times N \times N$ tensor, where M denotes the number of channels and $N \times N$ the grid size. Each state s_t is encoded with 3 channels: the first is the 2-D image, with H's as 1's and P's as 0's, the second is a 2-D image representing all vertical H-H bonds, and the third is a 2-D image representing all the horizontal H-H bonds. We concatenate the tensors for s_t and s_{t-1}, and as well as four channels of 1's, -1's, or 0's, corresponding to H's, P's, and nothing, respectively, denoting the next four molecules to be folded. Thus, we have 10 total channels, and the final state variable for the AlphaGo Zero algorithm is a concatenation of these 10 channels.

Table 1: Max sequence length and inference time to optimal fold comparisons

Method	**Time / Sequence Length (2-D)**	**Time / Sequence Length (3-D)**
Policy Iteration	15 mins/14	37 mins/5
Value Iteration	15 mins/14	37 mins/5
DQN	30 mins/20	10 mins/15
Prioritized DQN	30 mins/20	10 mins/15
Dueling DQN	30 mins/20	10 mins/15
MCTS	5 mins/20	3 mins/12
AlphaGo Zero + pretraining	<1 min/40	–

We compare our folding results with the results of traditional Monte Carlo simulations using Gillespie's acceptance rule (e.g. here comparing the number of H-bonds in the current state with that in the state accessed through a proposed move). These simulations were implemented using the chain environment described in the following section. The sequences used are from a well-known 2D HP model benchmark dataset [1]. For Policy/Value Iteration, DQN, and MCTS, we found Monte Carlo simulations performed better by finding the lowest energy state configuration more reliably in a shorter amount of time. We group these algorithms under DRL in Table 2, and report the best result out of these different algorithms. However, AlphaGo Zero with pretraining outperformed the Monte

[1] `http://www.brown.edu/Research/Istrail_Lab/hp2dbenchmarks.html`

Carlo simulations by finding the same, if not lower, energies, in a much shorter time. The inference times for DRL and AlphaGo Zero with pretraining were similar to those presented in Table 1, and the inference time for Monte Carlo simulations was approximately 5 minutes.

Table 2: 2D Free Energy Comparisons

Length	Benchmark	Monte Carlo Simulations	DRL	AlphaGo Zero + pretraining
18	-4	-2	-2	-3
18	-8	-7	-6	-8
18	-9	-8	-7	-8
20	-9	-8	-6	-8
20	-10	-7	-8	-9
24	-9	-7	-6	-8
25	-8	-6	–	-7
36	-14	-10	–	-13

The AlphaGo Zero with pretraining algorithm was run on a Tesla P100 GPU, and the DQN algorithms were run using an NVIDIA GeForce RTX 2060 GPU. The Monte Carlo Simulations as well as the MCTS were run using an AMD Ryzen 9 CPU.

3 Chain Environment with Pull Moves

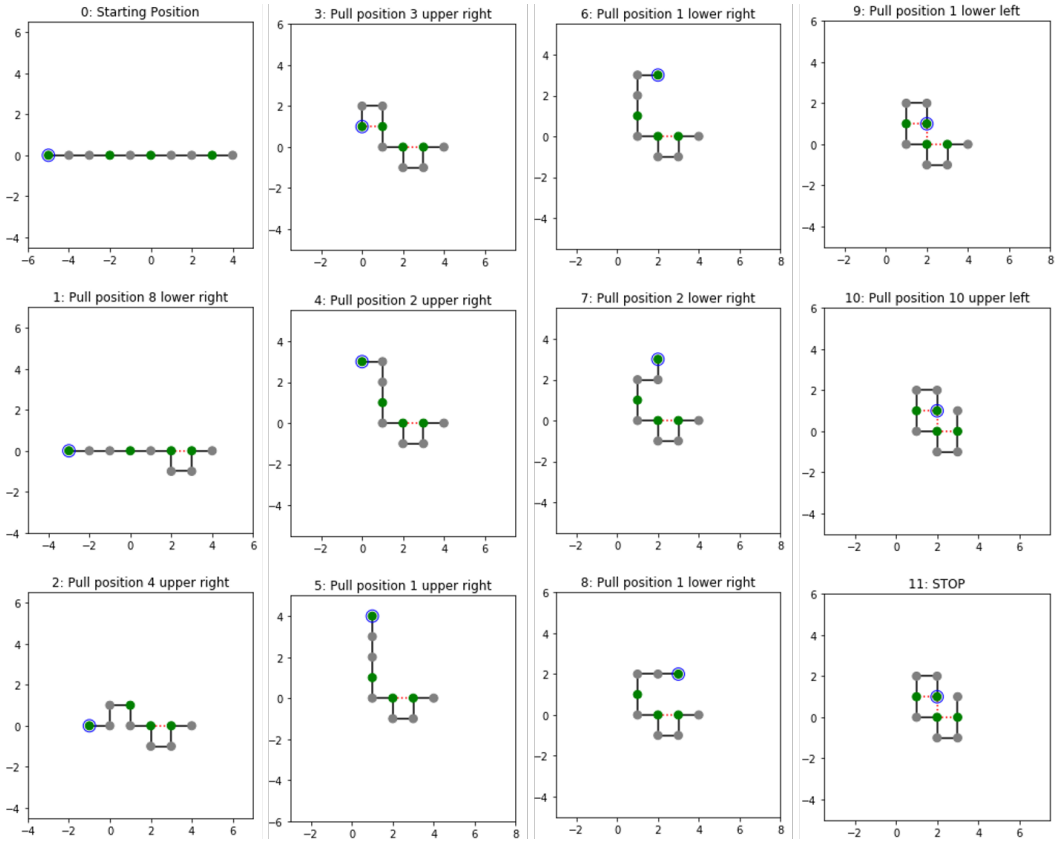


Figure 1: Example series of pull moves in the chain environment. The circled molecule represents position 1 of the sequence. Notice the Stop action; with no predetermined game length as in the lattice environment, under the chain environment, an agent must choose to stop moving.

The chain environment, as shown in Figure 1 on the previous page, introduces a different approach to folding proteins: rather than building up the protein sequentially as in the lattice environment, we start with a configuration for the entire sequence, and apply pull moves to produce different configurations. It has been shown that pull moves is a set of moves that is complete, reversible, and local (4), so it is in theory possible to find the optimal solution using this environment.

We will describe pull moves in terms of how it is implemented. Consider a vertex i at time t at a location $(x_i(t), y_i(t))$, and let L be a free lattice point adjacent to $(x_{i+1}(t), y_{i+1}(t))$ and diagonally adjacent to $(x_i(t), y_i(t))$. Notice that the three points $(x_i(t), y_i(t))$, $(x_{i+1}(t), y_{i+1}(t))$, and L are the vertices of a unit square. Let C be the fourth vertex of this square. If C is occupied, then a pull move consists of moving i to L. If C is also free, then a pull move consists of moving i to L, $i-1$ to C, and until we reach a valid configuration (every vertex k is adjacent to vertices $k-1$ and $k+1$), vertices are pulled two spaces forward until we reach a valid configuration. Mathematically, we start with vertex $j = i-2$ and move down to vertex 1, and set $(x_j(t+1), y_j(t+1)) = (x_{j+2}(t), y_{j+2}(t))$.

Similarly, we can consider pull moves in the other direction (i.e. let L be a free lattice point adjacent to $(x_{i-1}(t), y_{i-1}(t))$). It is also worth noting that in order to make the set of pull moves reversible, we add a few special pull moves at the end vertices: we take C to be a free location adjacent to the end vertex, and L to be a free location adjacent to C, then proceed according to the action plan previously described.

4 Conclusion and Future Work

In this study, we implemented different deep RL algorithms to solve a challenging NP-Complete protein folding problem using the lattice environment, and tested the validity of different neural network architectures and algorithms. We also implemented a more complicated chain environment with pull moves for simulating the dynamics of the protein folding process.

In future work, we plan to implement the deep RL algorithms in the chain environment to evaluate the merits of using a physically motivated approach compared to the simpler, more popular lattice environment.

Broader Impact

Owing to its simplicity, the model is a good test case to determine whether a rewards-based algorithm can learn from biopolymer sequences the relevant dynamics to find a ground-state structure, as quickly as possible. This effort could significantly enhance our ability to design folded structures, investigate mutations to the strand sequences, and predict the range of structures that small disordered protein clusters could reach after rearrangement. This approach could also pave the way for a similar effort aimed at predicting DNA or RNA ground state structures, where the oxDNA/oxRNA model may be used in hypothetical folding (self-assembly) games.(8; 9) We therefore believe that our work will benefit those working in fields including molecular simulation of biopolymers, protein folding and protein aggregation, nucleic acid folding, and drug design, to name just a few. By open-sourcing our code, we will make it possible for anyone to build upon our work. Although we have discretized and simplified our model, we believe that the results will give researchers a stronger degree of accuracy in predicting folded protein structure in simplified lattice-based models.

Acknowledgments

J. S. S. thanks Prof. Petr Šulc for useful discussions and for supplying a GPU node that was used for training throughout this work.

References

[1] B Berger and T Leighton, *Protein folding in the hydrophobic-hydrophilic (hp) model is np-complete*, J. Comput. Biol. **5** (1998), no. 1, 27–40.

[2] R. Jafari and Mohammad Masoud Javidi, *Solving the protein folding problem in hydrophobic-polar model using deep reinforcement learning*, SN Appl. Sci. **2** (2020), 1–13.

[3] Kit Fun Lau and Ken A. Dill, *A lattice statistical mechanics model of the conformational and sequence spaces of proteins*, Macromolecules **22** (1989), no. 10, 3986–3997.

[4] Neal Lesh, Michael Mitzenmacher, and Sue Whitesides, *A complete and effective move set for simplified protein folding*, Proceedings of the Seventh Annual International Conference on Research in Computational Molecular Biology (New York, NY, USA), RECOMB '03, Association for Computing Machinery, 2003, p. 188–195.

[5] Yanjun Li, Hengtong Kang, Ketian Ye, Shuyu Yin, and Xiaolin Li, *Foldingzero: Protein folding from scratch in hydrophobic-polar model*, CoRR **abs/1812.00967** (2018).

[6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller, *Playing atari with deep reinforcement learning*, CoRR **abs/1312.5602** (2013).

[7] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver, *Prioritized experience replay*, 2015, cite arxiv:1511.05952Comment: Published at ICLR 2016.

[8] Benedict EK Snodin, Ferdinando Randisi, Majid Mosayebi, Petr Šulc, John S Schreck, Flavio Romano, Thomas E Ouldridge, Roman Tsukanov, Eyal Nir, Ard A Louis, et al., *Introducing improved structural properties and salt dependence into a coarse-grained model of DNA*, J. Chem. Phys. **142** (2015), no. 23, 06B613_1.

[9] Petr Šulc, Flavio Romano, Thomas E Ouldridge, Jonathan PK Doye, and Ard A Louis, *A nucleotide-level coarse-grained model of RNA*, J. Chem. Phys. **140** (2014), no. 23, 06B614_1.

[10] Richard S. Sutton and Andrew G. Barto, *Reinforcement learning: An introduction*, second ed., The MIT Press, 2018.

[11] Ziyu Wang, Nando de Freitas, and Marc Lanctot, *Dueling network architectures for deep reinforcement learning*, CoRR **abs/1511.06581** (2015).

A Deep Reinforcement Learning Algorithms

In general, RL is a machine learning technique that trains an agent in an interactive environment or simulator by taking samples and using trial and error to maximize a reward function. After training, the agent can apply a learned policy to make decisions in the testing environment. Deep RL refers to using deep neural networks as function approximators to approximate the policies and value functions in RL algorithms. Below, we will introduce a few of these algorithms applied to our protein folding.

A.1 Policy and Value Iterations

In policy iteration, we start with a random policy, and within each iteration, we update the value function evaluated by the current policy using the Bellman expectation equation. At convergence of each iteration, we act greedily with respect to the value function and use this greedy policy in the next iteration to further improve the policy at each state until the policy converges.

In value iteration, we consider the Bellman optimality equation instead of the Bellman expectation equation to update the value function. Similarly, we act greedily to improve the policy until convergence. Both policy and value iterations are dynamic programming approaches which consider shallow but full backups. Dynamic programming approach usually cannot be applied to a sequential decision making problem with a very large state space such as the protein folding problem considered here.

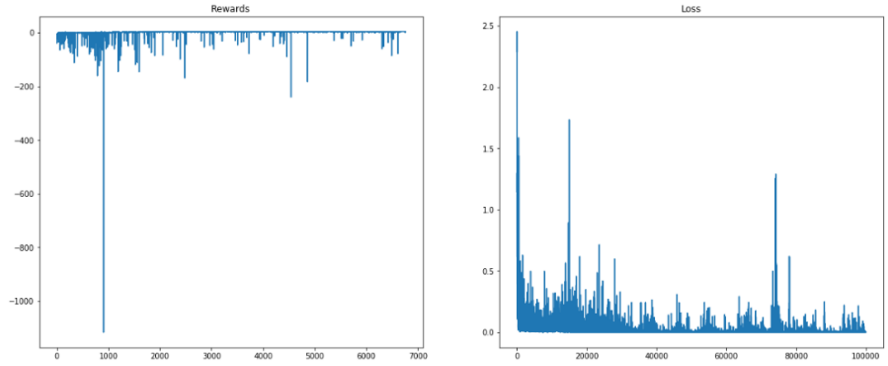
A.2 DQN and Variants

Q-learning applies the Bellman optimality equation and learns the action-value function $Q(s, a)$, which represents how good an action a is at a particular state s in a model free and off-policy manner. The Q function is updated by the Bellman optimality equation similar to the value iteration, but the policy is determined by acting ϵ-greedily with respect to the Q function. For Q-learning, we build a table $Q(s, a)$ to store the Q-values for all possible combinations of s and a in the memory. However,

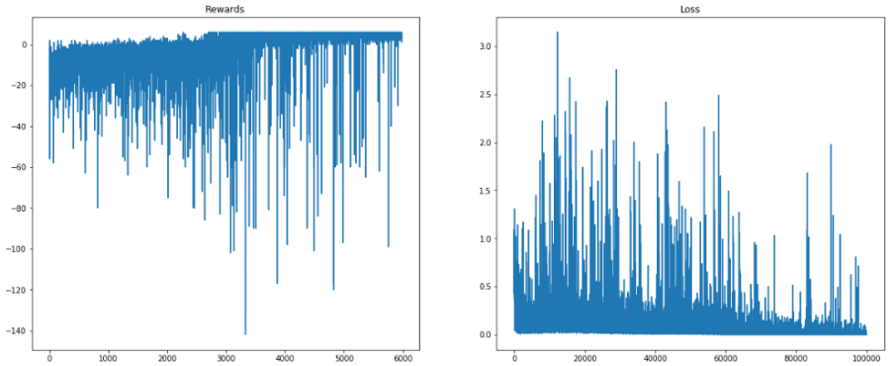
when there are too many combinations of states and actions to store, it becomes infeasible to create such a table and store it in the memory. To address this issue, we train a deep Q neural network to approximate the action-value function $Q(s, a)$. This method is called Deep Q-Networks (DQN).

For a vanilla DQN, we use a replay buffer to store the training samples (transitions) and randomly draw the samples from the buffer to train the Q function network. Also, the parameters of the target Q network are fixed to avoid a moving target issue. There are many variants to the vanilla DQN described above, such as using a prioritized experience replay (Prioritized DQN) or using a specialized architecture (Dueling DQN).

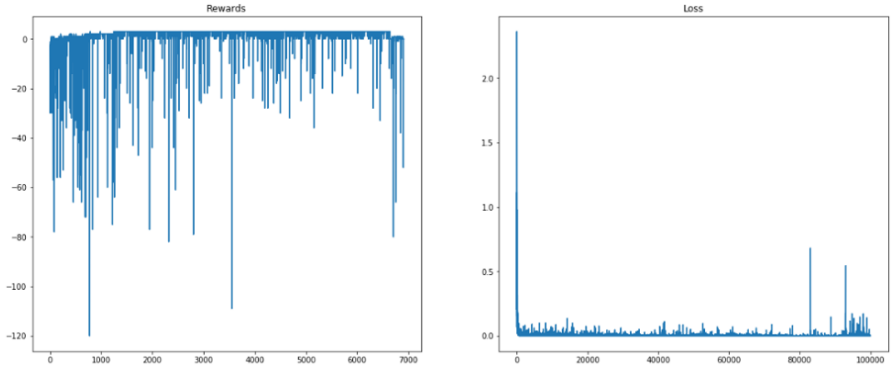
The learning curves for the DQN algorithms applied to the protein folding problem are shown below:



(a) DQN Results



(b) Dueling DQN Results



(c) Prioritized DQN Results

Figure 2: Learning curves and rewards for the various DQN algorithms in the 2-D environment.

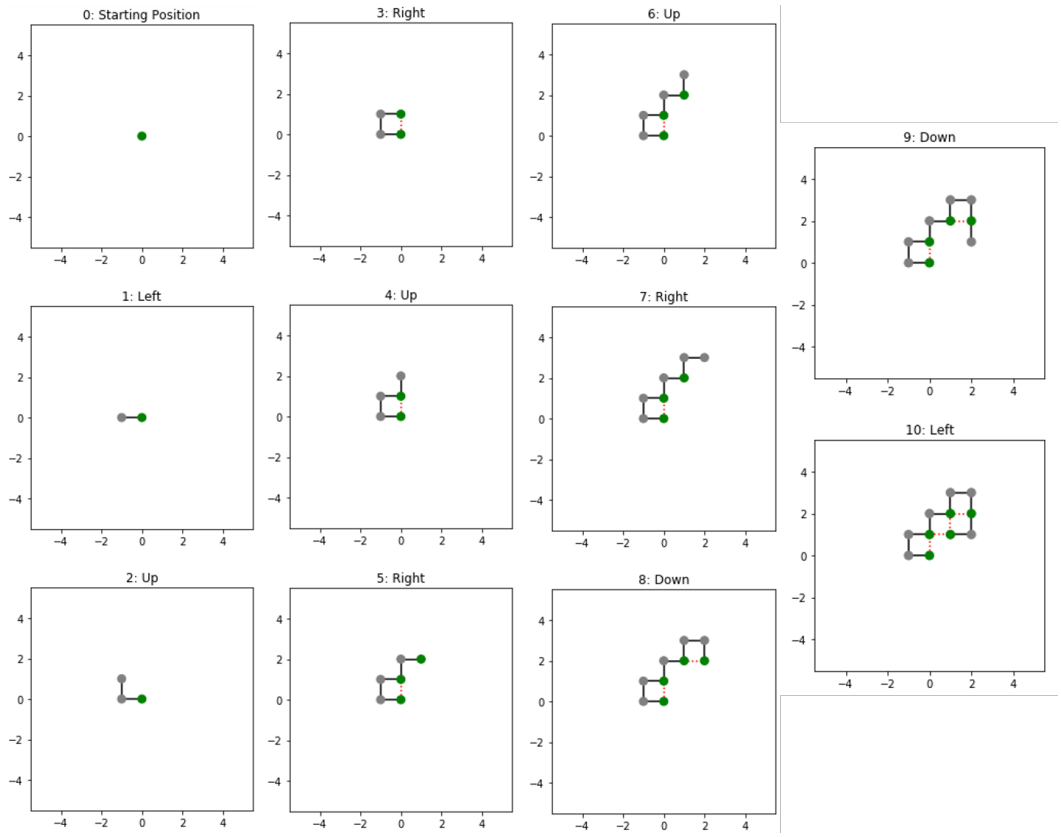
A.3 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is a search algorithm usually used in games to predict the moves that should be taken to reach the optimal outcome. MCTS at its core simply analyzes the most promising moves, expanding the search tree by randomly sampling the action space. We used a vanilla MCTS to verify our results for DQN as well as policy and value iterations, but also adapted MCTS to work with neural networks, which is described more in detail in Section A.4.

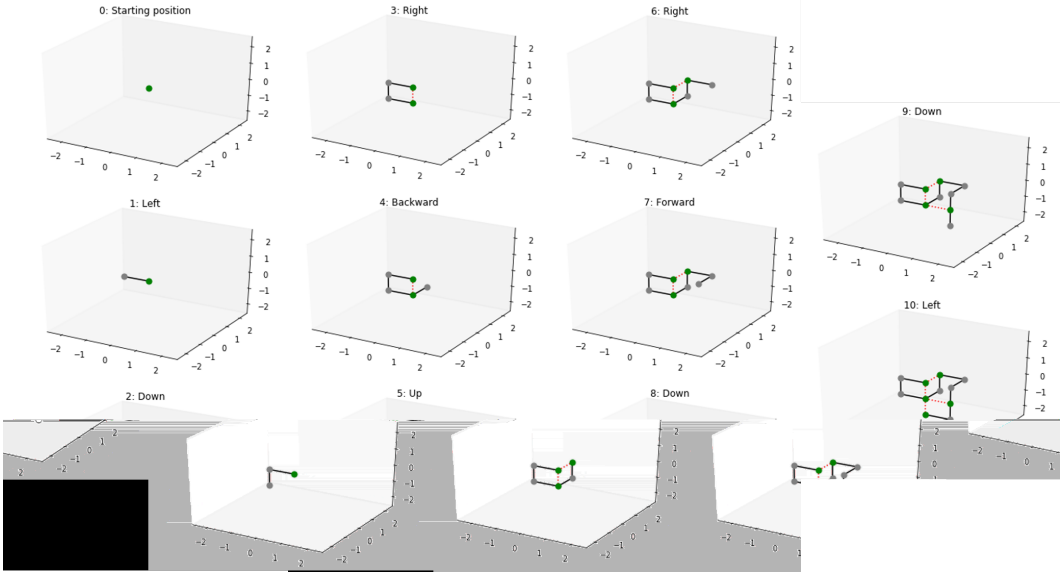
A.4 AlphaGo Zero with MCTS Pretraining

A decision making problem such as the protein folding considered here needs careful planning especially for a long chain of amino acids to reach the native state with minimum energy. MCTS combined with deep neural networks as policy and value function approximators are suitable for this task. MCTS has great success for simulation-based planning tasks by utilizing forward search and samples to reduce the size of search space. The approach is similar to what has been used in recent achievements in using deep RL to play Go, e.g., AlphaGo Zero. We combine MCTS with deep neural networks to approximate the policy and value function to further improve the folding solution. The primary difference between this algorithm and AlphaGo Zero is that we speed up the initial training process by pre-training the agent with MCTS without the neural network before transitioning to self-play using the neural network to evaluate states found with MCTS.

B Sequence Folded in 2-D and 3-D Lattice Environments



(a) 2D Lattice Environment



(b) 3D Lattice Environment

Figure 3: Folding the sequence $HPPHPHPPHPH$ in the 2D and 3D lattice environments. A green dot represents an H, gray represents P, black lines connect adjacent molecules in the sequence, and red dot lines represent non-sequence-adjacent $H - H$ bonds.