POLITECNICO DI MILANO

Scuola di Ingegneria Industriale e dell'Informazione
Dipartimento di Elettronica, Informazione e Bioingegneria

Master of Science in
Computer Science And Engineering - Ingegneria Informatica

Reinforcement Learning Force Field

Master Thesis of:
Alireza Ketabdari

Advisor: Prof. Marco Masseroli
Co-advisors: Prof. Modesto Orozco
Tutor: Dr. Milosz Wieczor
Year 2023

Abstract

Molecular Dynamics (MD) simulations play a pivotal role across diverse scientific disciplines, with their accuracy being contingent on the precision of Force Field parameters. Traditional optimization techniques, such as Gradient Descent (GD), often grapple with challenges like local minima entrapment and the intricacies of high-dimensional parameter spaces. This thesis introduces a groundbreaking approach to address these challenges, leveraging Reinforcement Learning (RL) and the Linear Q-function approximation (LQFA). This method offers a computational advantage by approximating Q-values using a linear function, eliminating the need for extensive Q-tables. The study emphasizes the helicity of the Alanine Oligopeptide (20-mer) as a primary metric, a measure of the peptide's propensity to adopt a helical conformation. Preliminary results have shown promising helicity values, with the potential to approach the maximal helicity value of 16 under optimal conditions. The research also delves into prospective developments, suggesting strategies to inch closer to this maximal helicity. In essence, this work paves the way for more accurate and efficient parameter optimization in MD simulations, with implications for broader applications in the scientific community.

Keywords: Reinforcement Learning, Force Field, Q-function, Q-learning, Parameter optimization, Gradient Descent, Helicity, Alanine Oligopeptide, Linear Q-function approximation, Molecular dynamics, State-action pair, Feature vectors, Weight vectors, N-dimensional parameter space.

Abstract in lingua italiana

Le simulazioni di dinamica molecolare svolgono un ruolo fondamentale in diverse discipline scientifiche, con la loro accuratezza che dipende dalla precisione dei parametri del campo di forza. Tecniche di ottimizzazione tradizionali, come la Discesa del Gradiente, spesso affrontano sfide come l'intrappolamento in minimi locali e le complessità degli spazi parametrici ad alta dimensionalità. Questa tesi introduce un approccio rivoluzionario per affrontare queste sfide, sfruttando l'Apprendimento per Rinforzo (RL) e l'approssimazione della Funzione Q-lineare. Questo metodo innovativo offre un vantaggio computazionale approssimando i valori Q con una funzione lineare, eliminando la necessità di estese tabelle Q. Lo studio enfatizza l'elicità dell'Oligopeptide Alanina (20-mer) come metrica primaria, una misura della propensione del peptide a adottare una conformazione elicoidale. I risultati preliminari hanno mostrato valori di elicità promettenti, con il potenziale di avvicinarsi al valore massimo di elicità di 16 in condizioni ottimali. La ricerca approfondisce anche gli sviluppi prospettici, suggerendo strategie per avvicinarsi a questa elicità massima. In sostanza, questo lavoro apre la strada a un'ottimizzazione dei parametri più accurata ed efficiente nelle simulazioni di dinamica molecolare, con implicazioni per applicazioni più ampie nella comunità scientifica.

Parole chiave: Apprendimento per Rinforzo, Campo di Forza, Funzione Q, Q-learning, Ottimizzazione dei Parametri, Discesa del Gradiente, Elicità, Oligopeptide di Alanina, Approssimazione Lineare della Funzione Q, Dinamica Molecolare, Coppia Stato-Azione, Vettori di Caratteristiche, Vettori di Peso, Spazio Parametrico N-dimensionale.

Acknowledgements

I would like to extend my deepest gratitude to Dr. Milosz Wieczor, my direct supervisor for the Reinforcement Force Field (FF) project. His role transcended that of a mere supervisor; he diligently fulfilled all supervisory responsibilities and, beyond that, became a close confidant. His invaluable knowledge and vast experience greatly enriched both my personal growth and the progression of the project. I am profoundly thankful for the significant impact he has had on my academic journey.

Contents

List of Abbreviations

MD	Molecular Dynamics
RL	Reinforcement Learning
FF	Force Field
LQFA	Linear Q-function Approximation
GD	Gradient Descent

Introduction

Molecular Dynamics (MD) simulations are foundational in various scientific fields, and the accuracy of these simulations relies heavily on the precision of Force Field (FF) parameters. However, the optimization of these equations is a complex task, often involving numerous parameters. Traditional methods like Gradient Descent (GD) have shown limitations, such as getting trapped in local minima and challenges in handling high-dimensional parameter spaces.

In response to challenges in optimizing FF equations, this thesis presents a novel Reinforcement Learning (RL) approach using the Linear Q-function Approximation (LQFA). Unlike traditional Q-learning, which relies on extensive Q-tables, this method employs a linear function to approximate Q-values, addressing the computational demands of vast state and action spaces.

The Q-value for a state-action pair is deduced as a linear mix of its associated feature values, which can be either designed or extracted from the environment. This approach uses feature and weight matrices, bypassing the need for individual Q-value storage.

Central to this method is the weight matrix initialization. In this research, the n-dimensional parameter space is grid-discretized, with each point linked to a weight. Proper weight initialization ensures the algorithm's efficiency, preventing convergence to local maxima and capturing state-action relationships effectively.

The methodology reduces the dimensionality of the parameter space by taking two steps, narrowing the focus to sigma and epsilon parameters, and reducing the number of atom types in the system. The parameters space is modeled as an N-dimensional Cartesian coordinate system, discretized into a grid-like structure.

The Results chapter provides a detailed account of the project's achievements, emphasizing the helicity of the Alanine Oligopeptide (20-mer). However, the calculation of the helicity is an easy-to-define trial reward function, and in the target application, it will be substituted with a measure of agreement between the set of simulational estimates and the corresponding experimental reference values. In MD simulations, helicity is indicating the propensity of the Alanine Oligopeptide to adopt a helical conformation. A higher helicity value suggests that the peptide has a stronger tendency to form a helical structure, which is vital for its functional stability and overall behavior in biological systems.

The results underscored a marked enhancement in the system's helicity, a testament to the efficacy of the introduced approach. During one of the evaluations, the agent documented helicity values spanning 6.82, 8.93, 8.17, and 8.64 across distinct states. A helicity peak nearing 9 was observed, a noteworthy accomplishment within the domain of MD simulations. However, it's imperative to contextualize this achievement: the maximal helicity value for the Alanine Oligopeptide (20-mer) stands at 16. This pinnacle represents a state where every parameter is impeccably optimized, embodying the epitome of structural stability for the peptide. The current study, while significant, was conducted under conditions with certain parameter reductions to simplify the computational landscape. As we delve into "Chapter Five: Prospective Developments," we elucidate strategies and refinements that, when implemented, hold the promise of edging closer to that coveted maximal helicity, thereby fully harnessing the peptide's structural potential.

Furthermore, the comprehensive algorithm underwent evaluation using real-world data, with the system's helicity being the primary metric of interest. The study leveraged a complete LQFA to pinpoint the optimal parameters within an N-dimensional parameter space. While the provided examples were limited to a maximum of four dimensions, the algorithm is inherently capable of managing spaces with even more dimensions, the main limitation being computational processing time.

In the realm of related works, two significant contributions stand out. The first, titled "Optimization of Empirical Force Fields by Parameter Space Mapping: A Single-Step Perturbation Approach," [24] introduces the Single-Step Perturbation (SSP) method to refine atomic interactions in simulations. This approach provides a valuable addition to the existing literature, especially for those engaged in molecular simulations. The SSP method offers a fresh perspective on atomic interactions, demonstrating its effectiveness in understanding the properties of specific chemicals and their behavior in various liquids.

The second article, "Empirical optimization of molecular simulation FFs by Bayesian inference," [18] underscores the importance of a systematic approach to FF parameterization. The authors envision a process where FF parameters are validated, updated, and their uncertainties quantified, integrating experimental data and high-level quantum chemical calculations. This systematic and automated approach to FF optimization emphasizes the importance of validation, uncertainty quantification, and the use of Bayesian inference. It stands as a significant contribution to the field, offering insights into molecular conformations and providing a valuable tool for researchers working with diverse molecular systems and simulations.

Both papers align closely with the central theme of this thesis: the optimization of Force Field (FF) parameters. While the methods introduced differ—Single-Step Perturbation (SSP) and Bayesian inference—they each address the overarching challenge of refining atomic and molecular interactions in simulations. This thesis, through the Reinforcement Learning (RL) approach, seeks to tackle similar challenges but from a unique angle using the Linear Q-function Approximation (LQFA). The SSP method from the first paper offers insights into refining specific atomic interactions, which complements the broader goal of RL approach to optimize FF equations. On the other hand, the Bayesian inference method from the second paper emphasizes a systematic approach to FF parameterization, validation, and uncertainty quantification. These components resonate with the objectives of this thesis, especially in the context of ensuring accuracy and precision in MD simulations. Together, these studies underscore the multifaceted efforts in the scientific community to enhance the precision of MD simulations, and they provide valuable context to the innovations introduced in this research.

Preliminary

0.1. Molecular Dynamics Simulations

Molecular dynamics (MD) simulations are computational techniques employed to simulate the movement and behavior of atoms and molecules by solving the equations of motion. In classical MD simulations, the equations of motion are Newtonian, and the classical laws of physics are utilized to model the interactions between individual atoms and their collective dynamics, enabling the study of molecular systems at an atomic scale[1].
MD simulations find widespread use in diverse scientific domains. In the field of biochemistry and drug discovery, MD simulations aid in investigating the structural dynamics of biomolecules such as proteins and nucleic acids, elucidating their folding pathways, conformational changes, and ligand binding interactions. These insights can guide drug discovery efforts and the design of novel therapeutics[1].

The ultimate goal of the project is to provide a framework for systematically improving MD simulations with a combined top-down and bottom-up approach when experimental data are incorrectly predicted from atomistic trajectories. To provide necessary context for later considerations, below follows a step-by-step explanation of how a typical MD simulation works:

1. Initial Geometries: Utilize existing molecular structure databases, such as the Protein Data Bank (PDB) or other specialized databases, to obtain experimentally determined or simulated structures. Alternatively, use molecular modeling software tools to construct the initial geometry of the system. These software packages often offer graphical interfaces for specifying atom types, bond lengths, bond angles, and torsional angles, enabling the creation of custom molecular structures tailored to specific research questions[30].

2. Define Inter-Atomic Forces: Examine the inter-atomic bonded and non-bonded forces through FF equations. This topic will be discussed in detail in section 0.2 of the thesis[30].

3. Simulation Box Setup: The setup of the simulation box in the modeling process,

where a specific volume is defined to enclose the system of atoms or molecules under study. Typically, periodic boundary conditions are employed, allowing atoms that exit one side of the box to re-enter from the opposite side, thereby mimicking an infinite system. The dimensions and shape of the box are carefully selected based on the specific properties and behavior of the system to avoid any artificial boundary effects. It is essential to ensure that the periodic images of the molecule of interest are sufficiently spaced apart to prevent any spurious interactions, either directly or by affecting the solvation shell, such as interfacial water molecules. Appropriate box dimensions are maintaining the desired density of the system, which in turn is essential for accurate representation and study of its macroscopic properties. Therefore, choosing an appropriate box setup and boundary conditions is critical for obtaining physically relevant results in a MD simulation.[30].

4. Energy minimization: Energy minimization is aimed at finding the locally lowest-energy configuration of a molecular system. It involves iteratively adjusting the atomic positions to reduce the total potential energy of the system, thereby reaching a locally stable state[30].

5. Equilibration: Equilibration is an initial stage in MD simulations where the system is allowed to reach a stable and well-defined thermodynamic state before data collection or production runs. It involves a series of simulation steps to gradually adjust the system to the desired temperature, pressure, or other specified conditions[30].

6. Production: The production phase is the main part of MD simulations, following the equilibration stage. During the production phase, the system is simulated over an extended period to generate data for analysis and investigation of the molecular system's behavior, properties, and dynamics[30].

7. The analysis phase of MD simulation involves processing and interpreting the data collected during the simulation to gain insights into the behavior, properties, and dynamics of the molecular system. This phase focuses on extracting meaningful information from the raw simulation data and applying various analytical techniques to understand and quantify the system's characteristics[30].

Among the project components, steps 2 and 7 present distinct challenges in the context of MD. Step 2, which involves defining inter-atomic forces through FF equations, is challenging due to the complexity of selecting appropriate parameters and equations that accurately represent the molecular interactions. The choice of FF can significantly impact the simulation's results and is often subject to rigorous validation. This topic will be discussed in detail in the 0.2 section of the thesis[30].

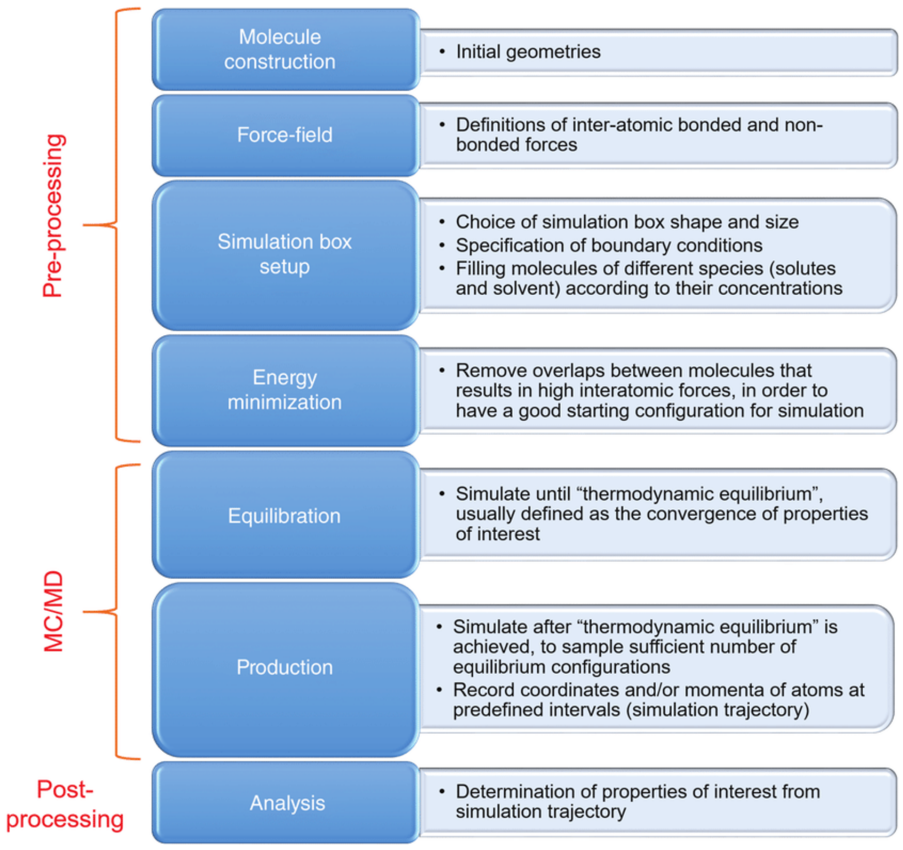

Pre-processing
MC/MD
Post-processing
Molecule construction
• Initial geometries
Force-field
• Definitions of inter-atomic bonded and non-bonded forces
Simulation box setup
• Choice of simulation box shape and size
• Specification of boundary conditions
• Filling molecules of different species (solutes and solvent) according to their concentrations
Energy minimization
• Remove overlaps between molecules that results in high interatomic forces, in order to have a good starting configuration for simulation
Equilibration
• Simulate until "thermodynamic equilibrium", usually defined as the convergence of properties of interest
Production
• Simulate after "thermodynamic equilibrium" is achieved, to sample sufficient number of equilibrium configurations
• Record coordinates and/or momenta of atoms at predefined intervals (simulation trajectory)
Analysis
• Determination of properties of interest from simulation trajectory

Figure 1: MD simulation is a computational technique used to simulate the movement and behavior of atoms and molecules over time.[30]

Step 7, which focuses on the analysis phase of MD simulation, presents challenges of a different nature. This step requires the processing and interpretation of large volumes of raw simulation data to extract meaningful insights. The application of various analytical techniques is necessary to understand and quantify the system's characteristics, making this phase both computationally intensive and intellectually demanding[30].
The remaining components can generally be implemented using an established MD package specifically designed for simulating proteins, lipids, and nucleic acids, such as Gromacs or OpenMM.

0.2. Force Field in Molecular Modeling

FF is mathematical model that describe the interatomic interactions within a system during an MD simulation. It encompass both the functional forms and parameter sets used to calculate the potential energy of the system. These FF parameters are derived from experimental data, quantum mechanical calculations, or a combination of both. The FF determines the accuracy and transferability of the simulation results and influences the fidelity of the modeled system's behavior. Thus, the FF is a critical component in performing MD simulations, as it determines the accuracy and reliability of the simulated MD [13].

0.2.1. Force Field equation: Physical Formulation

In MD simulations, interactions within molecules are typically categorized into two main types: bonded and non-bonded interactions. The following discussion focuses on the bonded interactions. In the physical formulation of a biochemical FF, bonded interactions are described by three key terms. Bond stretching accounts for the stretching of chemical bonds and is represented by a harmonic potential energy function that is parameterized by the force constant (bond stiffness) and the equilibrium bond length. Angle bending considers the deformation of triplets of atoms connected by two chemical bonds and utilizes a harmonic potential energy function with parameters for the force constant (angle stiffness) and the equilibrium angle. Torsional rotation describes the rotation around a chemical bond and incorporates a periodic potential energy function that includes terms for the barrier height, phase, and periodicity of the rotation. These bonded interactions capture the energy associated with the stretching, bending, and rotational movements of atoms in a molecule.[32].

Non-bonded interactions encompass van der Waals forces and electrostatic interactions. Van der Waals forces represent the attractive and repulsive interactions between atoms due to correlated fluctuations in electron density and constant electric dipoles, as well as Pauli exclusion (most of the repulsive term) in their electron density. To account for these, the FF incorporates a Lennard-Jones potential energy function, which features the equilibrium distance at which attractive forces dominate and repulsive forces become significant. Electrostatic interactions arise from charged particles or atoms and are typically modeled using Coulomb's law. The FF assigns partial charges to atoms and calculates the electrostatic potential energy based on the distances and charges of interacting atoms. Together, these non-bonded interactions contribute to the overall potential energy and influence the structural stability and interactions between molecules in a biochemical system[32].

There are many FFs available in the literature, having different degrees of complexity, and designed to treat different kinds of systems. However, a typical expression for a FF may look like this:

Total Energy:

$$E = E_{\text{bonded}} + E_{\text{non-bonded}} \tag{1}$$

Bonded Interactions:

$$E_{\text{bonded}} = \sum_{\text{bonds}} K_b (r - r_0)^2 + \sum_{\text{angles}} K_a (\theta - \theta_0)^2 + \sum_{\text{torsions}} \frac{V_n}{2} [1 + \cos(n\phi - \gamma)] \tag{2}$$

Non-bonded Interactions:

$$E_{\text{non-bonded}} = \sum_{\text{pairs}} 4\epsilon_{ij} \left[\left(\frac{\sigma_{ij}}{r_{ij}} \right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}} \right)^{6} \right] + \sum_{\text{pairs}} q_i q_j \frac{1}{4\pi\epsilon_0} \frac{1}{r_{ij}} \tag{3}$$

Parameters (Constants):

- **Bonded Interactions**:
  - K_b: Bond force constant[35].
  - r_0: Equilibrium bond length[35].
  - K_a: Angle force constant[35].
  - θ_0: Equilibrium bond angle[35].
  - V_n: Barrier height for torsional rotation[35].
  - n: Multiplicity of the torsional potential[35].

  - γ: Phase angle for the torsional potential[35].

- **Non-bonded Interactions**:
  - ϵ_{ij}: Depth of the potential well for van der Waals interactions between atom pairs i and j[35].
  - σ_{ij}: Distance at which the potential energy between atom pairs i and j is zero, representing the position of the potential well[35].
  - q_i, q_j: Partial charges on atoms[35].
  - ϵ_0: Permittivity of free space (a universal constant)[35].

Variables:

- **Bonded Interactions**:
  - r: Current bond length[35].
  - θ: Current bond angle[35].
  - ϕ: Current torsional angle[35].
- **Non-bonded Interactions**:
  - r_{ij}: Current distance between atom pairs i and j[35].

0.2.2. Force Field Parameter selection

The FF formulation in MD inherently involves a multitude of parameters, typically one or two orders of magnitude more than atom types for each molecular system. This expansive parameter space leads to virtually limitless possibilities, rendering direct optimization infeasible. To address this complexity, it becomes necessary to restrict ourselves to an "active" subset of the optimizable parameters, thereby constraining the number of atom types and parameters, to make the optimization problem tractable. So we aim to focus only on the most significant parameters during trajectory simulations. Identifying these important parameters involves two steps: first, selecting from the challenging parameters in the FF formula, and second, choosing among the atom types that have the largest impact on the quantity we intend to maximize. This approach will help us tackle the problem efficiently and effectively [20].
Determining accurate values for parameters like torsional barrier heights (V_n) and non-bonded interaction well depths (ϵ_{ij}) can be challenging in FF calculations. Precisely capturing the complex potential energy surface of torsional motions and accurately bal-

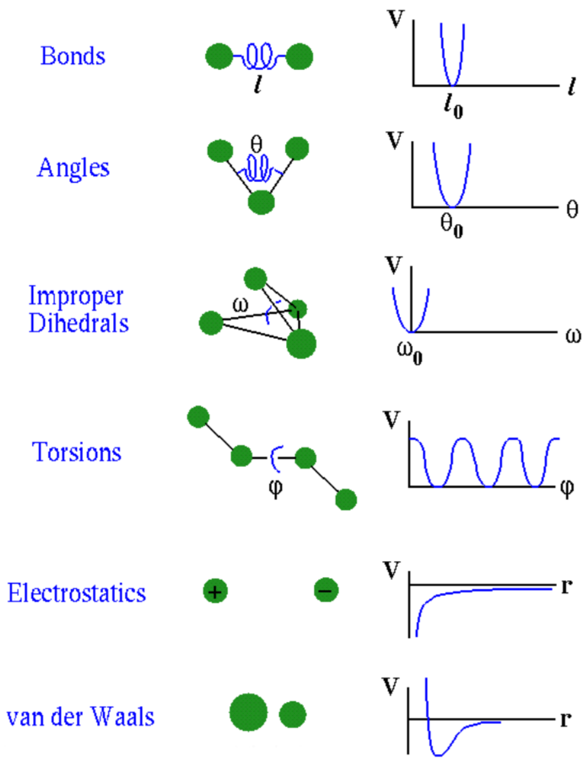


Figure 2: Metaphorically illustrating the diverse FF parameters.

ancing attractive and repulsive forces in non-bonded interactions necessitate advanced computational methods. When using QM methods to derive these parameters, trade-offs arise between employing high-precision calculations in vacuum and using approximate calculations that incorporate varying levels of complexity in the solvent model. Additionally, fitting these parameters to experimental data is crucial for ensuring their accuracy. Refining parameters related to van der Waals interactions, such as the parameter σ_{ij} (which is distinct from the van der Waals radii), and the treatment of regular dihedrals are active research areas, with ongoing efforts to improve the accuracy of these parameters.[16][35]. The remaining parameters in the FF formulation, such as bond lengths (r), equilibrium bond lengths (r_0), bond angles (θ), and equilibrium bond angles (θ_0), are generally considered to be less challenging to determine compared to torsional barrier heights, periodicity (n), phase shift (δ), and non-bonded interaction parameters. The simpler parameters can often be estimated from experimental measurements or theoretical calculations with a reasonable degree of accuracy. When it comes to deriving high-quality dihedral parameters, the optimization procedure involves fitting to a sufficiently accurate QM energy profile. This procedure typically optimizes the barrier heights, multiplicities, and phase shifts together, making the entire optimization process challenging. The intricacy arises not from fitting one or the other number individually but from the collective optimization of these parameters. However, it's important to note that the precise values of these parameters can still depend on factors like molecular environment, molecular conformation, and the specific FF being used. Therefore, while values for these parameters are available, they may not always be universally precise and can still be subject to refinement and adjustment based on experimental or computational evidence[16][35].

Because of the above considerations, this thesis focuses solely on the investigation and experimental characterization of the parameters sigma (σ_{ij}) and epsilon (ϵ_{ij}).

To reduce the number of parameters, we utilize sensitivity analysis by employing the ThermoDiff library in Python. This technique allows us to calculate derivatives of free energy or observables concerning specific FF parameters such as atomic charges, Lennard-Jones parameters σ and ϵ, and bonded terms like bond lengths, angles, and force constants[19]. The methodology requires extensive data, including trajectories featuring various molecular configurations, and involves the selection of particular FF parameters for investigation. The analysis yields a 'sensitivity matrix,' an informative tool that estimates the response of individual quantities to changes in FF parameters. This matrix guides the modification of model attributes that need improvement while preserving well-performing characteristics in alignment with existing data[19].

ThermoDiff functions through numerical differentiation and reweighting schemes based on statistical thermodynamics, mainly employing free energy perturbation (FEP) techniques. By evaluating the "local sensitivity" of free energies to specific FF parameters, represented generically as σ, we can enhance FF development, atomistic model coarse-graining, and lead optimization in drug design[19].

0.3. Reinforcement Learning

Reinforcement Learning (RL) is a type of machine learning paradigm that enables an agent to learn and make decisions in an environment to achieve a specific goal. The key characteristic of RL is its focus on learning through interaction with the environment rather than relying on labeled data[33].

In RL, the agent takes actions in the environment, and the environment provides feedback in the form of rewards or penalties based on the agent's actions. The agent's objective is to maximize the cumulative reward it receives over time, learning from its experiences to make better decisions[33].

The RL process can be summarized as follows:

- Agent: The learner or decision-maker that interacts with the environment. It takes actions and learns from the consequences[33].

- Environment: The external system with which the agent interacts. It can be as simple as a grid world or as complex as a real-world scenario[33].
- State: The current situation or configuration of the environment at a given time. It serves as an input for the agent to make decisions[33].
- Action: The choices made by the agent based on the observed state. The agent selects actions with the aim of maximizing the expected cumulative reward[33].
- Reward: The immediate feedback from the environment to the agent's actions. It indicates how favorable or unfavorable the action was in a given state[33].
- Policy: The strategy or set of rules that the agent follows to decide on actions based on the current state. The policy can be deterministic or stochastic[33].
- Value Function: A function that estimates the expected cumulative reward an agent can achieve from a given state following a particular policy. It helps the agent evaluate the desirability of different states[33].
- Model (Optional): In some RL approaches, a model of the environment is used to simulate possible interactions and learn from them before executing actions in the real environment. This is called a model-based RL approach[33].

The RL process typically involves an iterative learning loop, where the agent collects experiences by interacting with the environment, updates its policy and value function based on these experiences, and continuously improves its decision-making abilities[33].
RL has applications in various fields, such as robotics, game playing, autonomous vehicles, recommendation systems, and more. It has shown great potential for solving complex problems in dynamic and uncertain environments where traditional supervised learning approaches are less effective[33].

0.3.1. Reinforcement Learning: Q-Learning

This thesis concentrates on exploring various RL approaches, with a specific emphasis on Q-learning and the use of LQFA. The research involves a comprehensive comparison of a number of concept within this approach in the context of our real-world problem. Through this comparison, we demonstrate the rationale behind selecting Q-learning with LQFA as the most suitable technique to address the challenges posed by our specific problem domain.
Q-learning is an off-policy RL algorithm. In off-policy learning, the agent learns the value

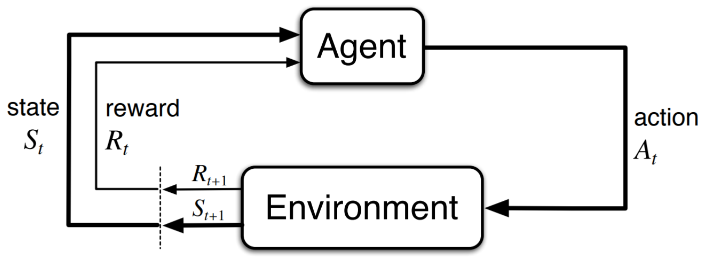


Figure 3: n this diagram, the RL process is depicted succinctly. The RL agent's decision-making is contingent on the current state (S_t) it is in and the immediate reward (R_t) it receives. Based on this information, the agent selects an action (A_t) to take. The action leads to a transition to a new state, and in this new state, the agent is presented with a new reward. This iterative process persists until the agent achieves its desired states, rewards, or both through learning from the consequences of its actions in the environment[33].

function based on a policy different from the one it uses to behave, often referred to as the target policy and the behavior policy, respectively. Specifically, Q-learning learns the optimal value function, called the Q-function, which represents the expected cumulative reward for taking a specific action in a given state and following the optimal policy thereafter [33]. Here is a step-by-step explanation:

1. Q-learning Initialization: Q-learning starts by initializing a Q-table, where each entry $Q(s, a)$ represents the estimated cumulative reward when taking action 'a' in state 's'[33].
2. Exploration and Exploitation: The agent interacts with the environment and decides which actions to take in each state. During the learning process, it employs an exploration-exploitation tradeoff to balance between exploring new actions (exploration) and exploiting its current knowledge to take the best actions (exploitation)[33].
3. Action Selection: The agent selects actions based on a constantly updated policy that has to balance exploration and exploitation, for that reason the epsilon-greedy is chosen as a simple implementation. In this case the agent take a random action with a small probability ϵ and otherwise selects the action with the highest Q-value[33].
4. Updating Q-Values: After taking an action, the agent receives a reward and observes the next state. It then updates the Q-values in the Q-table using the Bellman

equation, which is a key characteristic of Q-learning:

$$Q(s,a) = Q(s,a) + \alpha * [R(s,a) + \gamma * \max(Q(s',a')) - Q(s,a)][33] \quad (4)$$

- $Q(s,a)$ is the Q-value of taking action 'a' in state 's'[33].
- α (alpha) is the learning rate, controlling how much the agent should update the Q-values with each iteration[33].
- R(s, a) is the immediate reward received for taking action 'a' in state 's'[33].
- γ (gamma) is the discount factor, determining the importance of future rewards compared to immediate rewards[33].
- $max(Q(s',a'))$ represents the maximum Q-value among all possible actions in the next state (s')[33]. In the context of Q-learning, this concept is closely tied to its off-policy nature. Off-policy learning means that the agent learns the value of the optimal policy independently of the policy it follows while exploring. Specifically, while the agent might take actions based on a behavior policy during exploration, it learns about the value of the best possible action (as indicated by $max(Q(s',a'))$) that would be taken under an optimal policy. This distinction allows Q-learning to update its Q-values based on the best available action in the next state, even if that action wasn't the one taken during exploration.

5. Convergence: Through repeated interactions with the environment and Q-value updates, Q-learning eventually converges to the optimal Q-values, and the agent learns the optimal policy[33].

0.3.2. Reinforcement Learning: Linear Q-function Approximation

LQFA is a variant of the Q-learning algorithm that approximates the Q-values using a linear function. In traditional Q-learning, a Q-table is used to store and update the Q-values for each state-action pair. However, for problems with large state and action spaces, maintaining such a table can become computationally infeasible. LQFA addresses this limitation by employing function approximation techniques to estimate Q-values more efficiently [11].

In the LQFA, the Q-value for a given state-action pair is represented as a linear combination of feature values associated with that state-action pair. These feature values are manually designed or automatically extracted from the environment to capture relevant information about the state-action space[11].

In LQFA, the representation of Q-values is achieved through the use of feature vectors and weight vectors instead of directly storing Q-values for individual state-action pairs. The goal is to learn the significance or importance of each feature for each specific action, which is captured by the weight vector[11].

In this context, we employ two vectors:

- Feature Vector ($f(s,a)$): The feature vector, denoted as $f(s,a)$, is a composite vector comprising $n * |A|$ distinct functions, where n represents the number of state features, and $|A|$ denotes the number of actions available to the agent. Each function $f_i(s,a)$ within this vector extracts the value of a specific feature from the state-action pair (s,a). Therefore, the entire feature vector consists of multiple functions, each responsible for extracting a particular feature for every possible combination of state and action[11].
- Weight Vector (w): The weight vector, represented as w, has a size of $n * |A|$, containing one weight value (w_i^a) for each feature-action pair. This weight vector plays a crucial role in linear Q-learning as it determines the significance of each feature for each action. The value of (w_i^a) indicates the weight assigned to the i_{th} feature concerning the corresponding action, effectively quantifying the influence of the feature on the Q-value estimation[11].

Given a feature vector f and a weight vector w, the Q-value of a state is a a scalar product of the respective vectors of features and weights:

$$Q(s,a) = f_1(s,a) \cdot w_1^a + f_2(s,a) \cdot w_2^a + ... + f_n(s,a) \cdot w_n^a = \sum_{i=0}^{n} f_i(s,a) \cdot w_i^a \tag{5}$$

To use approximate Q-functions in RL, there are two steps we need to change from the standard algorithsm: (1) initialisation; and (2) update.

For initialization, initialize all weights to 0. Alternatively, you can try Q-function initialization and assign weights that you think will be “good” weights.

For update, we now need to update the weights instead of the Q-table values. The update rule is now:

For each state-action feature i [11]:

$$w_i^a \longleftarrow w_i^a + \alpha \cdot \delta \cdot f_i(s,a) \tag{6}$$

[11] where δ depends on which algorithm we are using; e.g. Q-learning, SARSA. In this thesis Q-learning is used, then we have a complete expansion of the weights update formula as follows [11]:

$$w_i^a \longleftarrow w_i^a + \alpha \cdot [r + \gamma \max_{a'} Q(s',a') - Q(s,a)] \cdot f_i(s,a) \tag{7}$$

1 | Chapter one: Model

1.1. Definition and modeling of the problem

The main objective of this thesis is to optimize the FF parameters by determining the optimal parameters using a RL approach. The focus is on utilizing the QF method due to its suitability for handling multiple parameters and its capacity to discover the best solution in the solution space. Before applying QF, to manage the high dimensionality of the problem, I employ two parameter reduction steps. Firstly, the focus is narrowed to the sigma and epsilon parameters, as they are more likely to have adjustable values compared to the other parameters (0.2.2). Secondly, the number of atom types in the system is reduced using sensitivity analysis with the ThermoDiff library in Python.
To model the parameter optimization problem, an analogy to a video game is used. The problem can be represented as follows:

1.1.1. Model: Parameter Space

The environment serves as the setting where the algorithm explores different values of the parameters, evaluating each combination to determine if it yields satisfactory results (similar to navigating a game to find clues or rewards to progress in the mission). The parameter space is modeled as an N-dimensional Cartesian coordinate system, where each dimension represents a specific parameter. To navigate within this N-dimensional Cartesian coordinate system, representing the parameter space, we must discretize it into a grid-like structure, hereafter referred to as an ND grid. The rationale behind this discretization is to to make it more consistent with the function approximation. In a continuous coordinate system, an infinite number of possible values could be assigned to the parameters, leading to challenges in the modification process. By transforming this space into an ND grid, we impose a structure that limits the possibilities, thereby allowing a more controlled and systematic exploration of parameter values [24] [6].

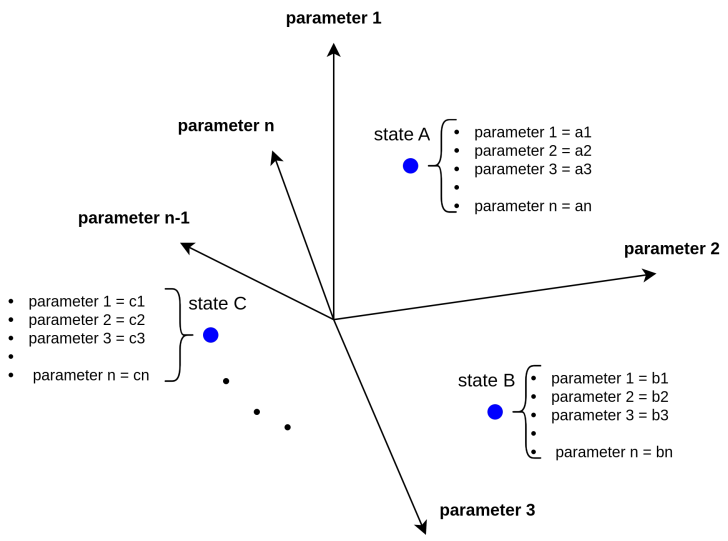


Figure 1.1: 3D model of the parameter space. Each state is considered as a point in this Cartesian coordinate system with specific parameters' value setting. The states and the dimensions in this diagram are schematic illustrations of the parameter space.

1.1.2. Model: States

Each configuration of parameters constitutes a state in this setup. For every state that is being visited, a simulation is run based on the parameter values of that state. The outcome of the simulation, which can vary based on the observables considered, serves as the reward. In this thesis, the calculation of the helicity is an easy-to-define trial reward function, and in the target application it will be substituted with a measure of agreement between the set of simulational estimates and the corresponding experimental reference values [6].

1.1.3. Model: Agent

In this RL implementation, the agent is not explicitly visible in the environment like a character in a video game. Rather, it can be thought of as an abstraction of the FF. The agent interacts with the environment by receiving the reward, influencing the Q-function parameters, and deciding the next action based on the policy defined. The ultimate goal is to discover the optimal parameter values that lead to the best possible parameters in the FF equation [6].

1.1.4. Model: Actions

In the context of policy-besed decision making, the agent's possible actions are determined by specific strategies. An action is just a change of the values of the parameters, so our model has a deterministic mapping from action space to the state space; furthermore, the moves have to be on the grid, and cannot escape beyond it. These actions can include moving to the location of one of its neighboring states, where the choice of neighbor is influenced by the local-radius variable, typically set between 2 to 4 (so the agent can jump into one of the neighbors in its local-radius distance, take a look at figure 1.5) [6].

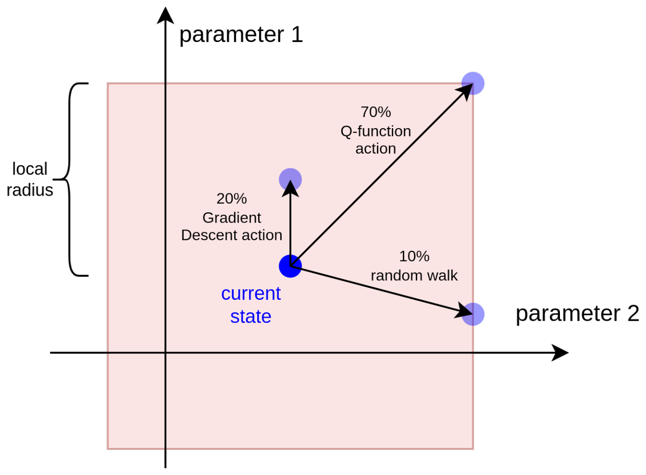


Figure 1.2: This is a very simple illustration of decision making (policy) in the agent's world. From an exact state the agent can jump to one of its neighbors included in the area that determined by local-radius constant variable. A random variable in every iteration is chosen and guide the agent to jump based on RL, GD or random walk.

1.1.5. Model: Policy

In this implementation, the policy adopted is a combination of greedy based action with a 70 percent probability, GD with a 20 percent probability, and random walk with a 10 percent probability (see figure 1.2)[28]. The approach that we have employed was inspired by the epsilon-greedy algorithm, which is widely recognized in the field of RL, particularly in the context of Q-learning. This algorithm's primary function is to maintain equilibrium between two fundamental processes: exploration, which pertains to the identification of novel actions that have the potential to enhance future rewards, and exploitation, the act of executing the currently optimal action according to known information[28].
However, our implementation introduces an additional complexity. Alongside the actions determined by the epsilon-greedy approach, we have incorporated another category of

actions dictated by the GD methodology. This inclusion augments the conventional functioning of the epsilon-greedy algorithm, extending its capabilities in the context of our specific implementation.

This policy is designed to introduce diverse types of movements to the agent's decision-making process, aiming to avoid over-fitting to the environment and enabling the agent to explore new, potentially more efficient paths in the simulation. By integrating GD into the policy, the agent can employ a mathematically simple approach to find local minima in the state space. We add GD to make sure that at least in the local neighborhood, the agent drifts on average in the right direction, but one cannot fully depend on GD because the gradients we calculate are anyway noisy and the reward function can be highly non-linear with multiple minima [5]. By incorporating these different types of actions into the policy, the agent gains the flexibility to adapt and explore various strategies, improving its overall performance and adaptability in the RL environment. More specific details about GD will be discussed in the Implementation section[28].

1.1.6. Model: Rewards

In the development stage of the project, we initially adopted the alpha-helical content of an Alanine Oligopeptide (20-mer) as a trial reward for each state in our RL model[36] [34]. This choice served as an easy-to-define and computationally tractable reward function. However, in the broader context of our research, the actual reward is a measure of agreement between the set of simulational estimates and the corresponding experimental reference values (see equation 1.1). The computation of the reward requires the creation of a system trajectory governed by new parameters, where each state encapsulates the structural information of the system, as delineated within the .pdb and .top files of the given simulation (more information in 2.1.1). As the system transitions from one state to another, only the values of the FF parameters are altered, defined by a unique vector of parameters within each individual state. In this modeling, each configuration of parameters constitutes a state, and for every state, a simulation is run based on the parameter values of that state. The outcome of the simulation, which varies based on the observables considered, serves as the reward. In the target application, the trial reward function based on helicity will be substituted with the aforementioned measure of agreement, providing a more comprehensive and meaningful evaluation metric.

$$r = \left[\sum_{i}^{number\_of\_targets} (t_i(\theta) - \mu_i)^2 \right]^{-1} \tag{1.1}$$

The equation 1.1 defines a reward function, r, which quantifies the agreement between simulation outcomes and experimental data. In this formula:

- $t_i(\theta)$ represents the target values derived from the simulation for a given parameter set θ. Each target value corresponds to a specific aspect or property of the system being simulated.
- μ_i denotes the corresponding experimental data for each target. These are the reference values obtained from actual experiments, serving as a benchmark for the simulation's accuracy.

The term $(t_i(\theta) - \mu_i)^2$ computes the squared difference between each simulated target and its experimental counterpart. By squaring the differences, the formula emphasizes larger discrepancies, ensuring that the reward function is sensitive to significant deviations from the experimental data.

In essence, this reward function offers a quantitative measure of how closely the simulation's outcomes align with real-world experimental data, making it a valuable tool for assessing and refining the simulation's accuracy and reliability.

The trajectory duration is a critical constant in this research, owing to its role in balancing the need for sufficient system change observation time and maintaining reasonable simulation lengths.

In this research, the methodology employed involved determining the time constant through an analysis consistent with first-order kinetics of relaxation. This assumption is predicated on the notion that a small perturbation in parameters will lead, on average, to a steady flow of probability from the initial 'unadapted' state to the final 'adapted' probability density. The utilization of an exponential function in this context serves as a fitting model to describe this process. The time constant, a characteristic indicative of a system's rate of change, is the reciprocal of the decay constant, b, in an exponential decay function of the form $a*exp(-b*x)+c$. For systems exhibiting exponential relaxation, this time constant signifies the duration required for the system to decay to approximately 63.2% of its initial state to steady-state or equilibrium. The function $a * exp(-b * x) + c$ is frequently adopted as a model for systems demonstrating exponential decay or growth with a baseline or steady-state value c. In the presence of a decreasing trend that does not descend to zero, this model can potentially provide an accurate description[8].
Implementing this methodology by fitting an exponential decay function to the data and determining the time constant serves as a strategy to maximally exploit the available information, allowing the quantification of the system's change rate over time (see Fig-

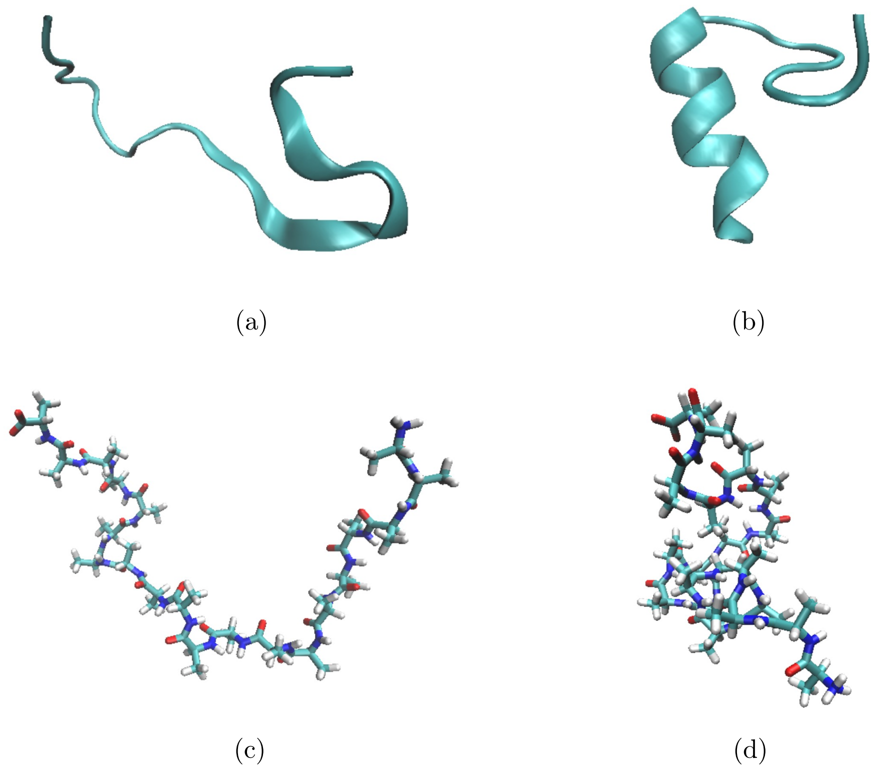


Figure 1.3: The Alanine Oligopeptide (20-mer) is depicted in two distinct structural conformations across the provided images. In figures a and c, the molecule is illustrated in its un-helical form, signifying an unstable state. Conversely, figures b and d showcase the peptide in a fully helical configuration. The helical structure of Alanine is intrinsically linked to the stability of the molecular system. A helix formation often results in a lower energy state, enhancing the structural robustness and resilience of the peptide against perturbations. This transformation underscores the critical role of conformation in determining molecular stability [22].

ure 1.4). This approach strikes a balance between too short simulations, which may not provide sufficient insights, and overly long ones, that risk yielding redundant information. This information could be pivotal for understanding the system or forecasting its future behavior. Moreover, it enables a comparison of the behavior of disparate systems or a single system under varying conditions [8].

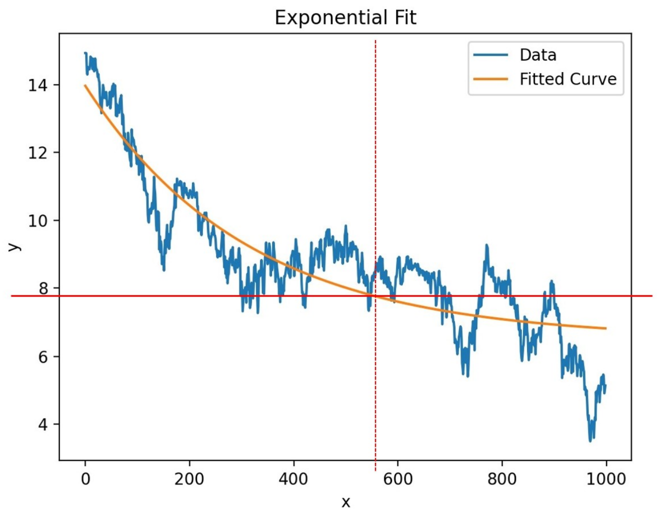


Figure 1.4: The figure depicts the fitting of an exponential function to the mean helicity measures derived from six distinct metrics of the system, tracked over a 10 ns trajectory. This methodology involves the application of plumed forces at the trajectory's onset, followed by a free evolution of the system for the subsequent 10 ns [12]. The primary objective is to ascertain the time required for the system to undergo first-order kinetic relaxation. For systems that exhibit exponential relaxation, a crucial parameter is the time constant, often denoted as τ. This constant represents the time span necessary for the system to decay to approximately e^{-1} (or roughly 63.2%) of its initial deviation from the equilibrium state. As illustrated, the system approaches this specified decay threshold approximately 5.5 ns post-initiation. Consequently, the deduced time constant, τ, for this particular system is 5 ns.

In the context of a molecular simulation, the time constant could be associated with the time scale of specific molecular events or processes. This association is crucial for understanding the system's physics as it necessitates the independent estimation of the time constant. Such an estimation is integral for making informed decisions about the optimal length of our trajectories, striking a balance between cost and accuracy, and fine-tuning the simulation parameters. For example, the simulation may need to run for a minimum duration equivalent to several time constants to accurately depict the system's behavior[8].

1.1.7. Model: Q-function

In the preceding chapter, the subsection titled "Reinforcement Learning: Linear Q-function Approximation" 0.3.2, we delved into the concept of LQFA, underscoring its indispensability for managing extensive and intricate environments with featuring numerous states. The subsequent discussion pertains to the relevance of this method in the context of our project and its consequent execution.
Primarily, although we are testing our approach with a manageable set of 4 parameters, in principle the method is not designed with a specific number of degrees of freedom. Consequently, we are dealing with an N-dimensional grid in which there are a minimum of 20 steps in each dimension. The size of each step can be calculated by considering the values of that parameter in the topology file. In a simplistic system with four parameters and a 20-step range for each dimension, we encounter $20^4 = 160000$ states. Hence, it is crucial to implement a method that permits a degree of generalization of our findings, at least to immediately neighboring states. Contrasting with the conventional Q-learning method that focuses on a single state and assesses its Q-value, LQFA provides an approach to evaluate a bunch of states together [26].
During project implementation, we deal with two distinct radii: a global-radius and a local-radius (1.5). The global-radius defines the ranges of the steps in each dimension. For instance, in an execution with a global-radius of 10, we have a grid with 20 steps across all dimensions. In layman's terms, if the grid step size is 0.01, the parameter value can fluctuate from $-10*0.01$ to $10*0.01$ during the simulation, the same applies for other parameters. The second radius, the local-radius, determines the neighbors impacted by any alterations in the current state due to the LQFA (or generalization). This radius, associated with the current state, determines the states that are proximate to it. Only states with a distance equal to or less than this radius can be immediately accessed from the current state. For instance, if the local-radius is 2, and we are operating in a specific state, all modifications in the current state influence its neighbors within a radius of 2. Thus, if we are going to alter the value of a state located at (5,5,5), it also affects its neighbors at locations $(5 \pm 2, 5 \pm 2, 5 \pm 2)$, although to lesser degree the farther the neighbor is from the current state.

A key aspect of LQFA is the selection of a suitable feature function f(s,a), which aids us in understanding how a Q-value of a state impacts neighboring q-values. In our execution, we utilized the Gaussian function (σ should be chosen based on the local-radius variable, sigma defines how far-reaching the effect of an update is in the parameter space).
In line with equation 6, we also need to introduce weights. The weights matrix in the LQFA method represent the importance or contribution of each feature to the predicted

Q-value. Adjusting these weights during training enables the model to learn the relationships between different states and actions and their corresponding Q-values. In essence, the weights determine how each feature influences the estimated value of the Q-function, and they are tuned during the learning process to minimize the error between the approximated Q-values and the actual returns received. In simple terms, the updating process of the weights is an effort to align the estimated Q-value with the sum of the immediate reward and the discounted expected return of the next state. According to the Q-learning's Bellman Equation (4), the next state is selected from all the neighbors (covered by local-radius) possessing the highest Q-value. Primarily, when the sum of the immediate reward and the expected return of the next state's Q-value is higher than the current estimated Q-value, the weights are adjusted to increase the current Q-value estimate. Conversely, if the sum of the immediate reward and the expected return of the next state is lower than the current estimated Q-value, the weights are adjusted to decrease the current Q-value estimate. These adjustments are guided by the relationships between states, actions, and features encoded in the LQFA, aiming to align the estimated Q-values with the observed returns. As per equation (7), it directly impacts the weight of that state and its neighbors. Therefore, if we witness a negative update of weights, it is less likely to move again towards that state or its neighbors, and the agent is more inclined to explore that area in the current and subsequent iterations.
Weights have to be initialized in some arbitrary way, and if you optimistically initialize them to high values, this encourages exploration. Since we want to encourage exploration close to the original parameter set (which effectively has a role similar to regularization), we set optimistic initial weights as a (Gaussian) function of distance from the initial set of values. However, there can be instances where we possess preliminary knowledge of the environment and initialize the weights unevenly (when we know certain areas in the environment are more likely to be selected by the agent). At each state, the agent computes the system's reward or helicity in that simulation and takes into account the expected value of the next state and updates its weight and its neighbors. This procedure continues until certain areas have high weights where optimized parameters are likely to be found and the rest of the environment with low weights, it's less likely to find the optimum parameters.

1.1.8. Model: Gradient Descent

GD methods are in general an optimization method for finding the closest local extrema of multidimensional functions for which gradients are available. In the context of molecular simulations and parameter optimization for the FF equation, it is employed as an

iterative optimization algorithm to minimize the discrepancy between simulated results and experimental data. The objective function represents the error or difference between the two sets of data, and the primary goal is to find an optimal set of FF parameters that minimize this objective function[5].
To apply GD for parameter optimization, sensitivity analysis is initially conducted to identify the most influential parameters in the FF equation. This analysis specifically calculates the local estimate of the gradient, which is the vector of derivatives of the objective function with respect to the selected subset of the underlying FF parameters. In doing so, it quantifies the impact of individual parameters on the FF equation.
The explanation of the steps are written as follow:

- Gradient Calculation: The sensitivity analysis yields information about the gradients of the objective function concerning each parameter. The gradients indicate the direction and magnitude of change required for each parameter to improve the fit between simulated and experimental data[5].
- Learning Rate (Alpha): A learning rate (often denoted as "alpha") is introduced to control the step size of the parameter updates during the optimization process. The learning rate determines how large or small the adjustments to the parameters should be in each iteration, it can also be dynamic, i.e. it can change between optimization steps[5].
- Parameter Update Rule: By multiplying the gradients with the negative value of the learning rate (-alpha) and subsequently updating the parameters accordingly, the optimization algorithm iteratively moves towards parameter values that minimize the objective function[5].

While GD is effective to some extent, it exhibits certain limitations that may hinder its ability to reach the global minimum of the objective function. GD can become trapped in local minima, where the objective function reaches a low point, but not necessarily the global minimum. This restricts the algorithm's ability to explore better solutions in the parameter space. Furthermore, by utilizing the Q-function algorithm, we can construct a well-trained environment. This environment will not only be invaluable for more in-depth explorations but also serve as a foundation for examining other systems. [2].

To overcome these limitations, RL approaches, particularly those utilizing QFs, can complement GD in the parameter optimization process. RL methods offer benefits such as global search capabilities, adaptability to non-differentiable objective functions, and the ability to manage probabilistic or stochastic outcomes. By employing RL techniques, researchers can enhance the optimization process and potentially achieve improved pa-

rameter values that better match experimental data and enhance the performance of the FF equation in molecular simulations.
To leverage the advantages of GD along with RL-based optimization and random exploration, a hybrid approach is adopted, where the optimization process is guided by a predefined policy. This policy determines the probability of choosing different actions during each iteration of the optimization process.
Specifically, the policy assigns a 70 percent probability to greedy Q-function based actions, a 20 percent probability to GD-based actions, and a 10 percent probability to completely random walk actions. The inclusion of random walk actions is essential for promoting the exploration of the parameter space, enabling the discovery of new and potentially more promising parameter configurations [28].
Combining GD with RL and random actions demands meticulous planning. For GD to be effective when selected as an action, it's crucial to consistently update the gradient values in every iteration where it's used, especially within the 20% likelihood of opting for the GD action. It's pertinent to mention that our chosen 70-20-10 policy distribution is an initial estimate at this stage. While fine-tuning these percentages might offer improved results, delving into such optimization isn't the focus of this thesis. We began our study with certain foundational values, and we recognize that our current distribution stems from these initial assumptions. The reason for frequent updates is that when using other approaches like RL or random walks, the optimization process moves away from the direction of GD, altering the parameters. Consequently, the previously computed GD values might no longer be accurate for the current parameter space (for better understanding take a look at the diagram 1.6).

However, recalculating GD from scratch in every iteration can be computationally expensive and inefficient (a gradient calculation can take an order of magnitude longer than a regular simulation). To overcome this issue, a time-saving approach is employed. For instance, consider the scenario where the GD values for two locations in the parameter space, (x1, x2, x3) and (y1, y2, y3), have already been calculated. Now, the optimization process aims to compute the GD for the location (z1, z2, z3). If (z1, z2, z3) lies close enough to (x1, x2, x3) and (y1, y2, y3) or in the direction of the gradients of one or both of those locations, it is possible to estimate the direction and value of the gradients using a linear combination of known gradients.
By employing such estimation techniques, the need for full recalculations is reduced, enabling more efficient and less time-consuming optimization. This integration of RL, GD, and random exploration, along with the clever handling of GD updates, provides a well-rounded optimization strategy that combines the strengths of different techniques for more

effective parameter tuning of the FF equation in molecular simulations.

1.1.9. Model: Initialization

In this section, we detail the essential initializations required for our simulation process [15]:

- Dimensions: We need to determine the number of FF parameters to be considered in the simulation.
- Grid Size (global-radius): The grid size represents the number of steps we can take in each dimension. Typically, a value of 10 or more is reasonable.
- The step size, crucial for parameter optimization in the FF, is derived from the analysis of the PDB (Protein Data Bank) file of the molecular system. The PDB file provides detailed information about the molecular structure, including atomic coordinates and other essential parameters. By analyzing these parameters, one can gauge the range and scale of values within the molecular system. This analysis aids in determining an appropriate step size for the n-dimensional parameter space of the FF parameters. Essentially, the step size should be chosen such that it is sensitive enough to capture the nuances of the molecular system while ensuring computational efficiency.
- Weights Matrix: The weights matrix is of the same size as the environment (coordinate system). Each element of the matrix is initialized using a Gaussian distribution. Two methods are employed for initialization: pessimistic and optimistic. Pessimistic initialization sets all weights to zero. This is considered pessimistic because zero represents the lowest value that the parameter can attain, not merely because it indicates no initial preference among the states. Optimistic initialization sets all weights to a higher value (e.g., 16, the maximum helicity of a 20-mer alanine oligopeptide) [17], considering all states as possibly good options. Depending on the method, the agent adjusts the weights of the current state and its neighbors based on rewards received.
- Local-radius: The local-radius determines the number of neighboring states from each dimension that should be considered in Q-value calculations, where the current state can influence them.
- Parameters of Q-function and GD: The Q-function parameters consist of Alpha, Gamma, and Gaussian Sigma. For GD, we only have Alpha. These parameters are to be initialized by the technician overseeing the simulation process.

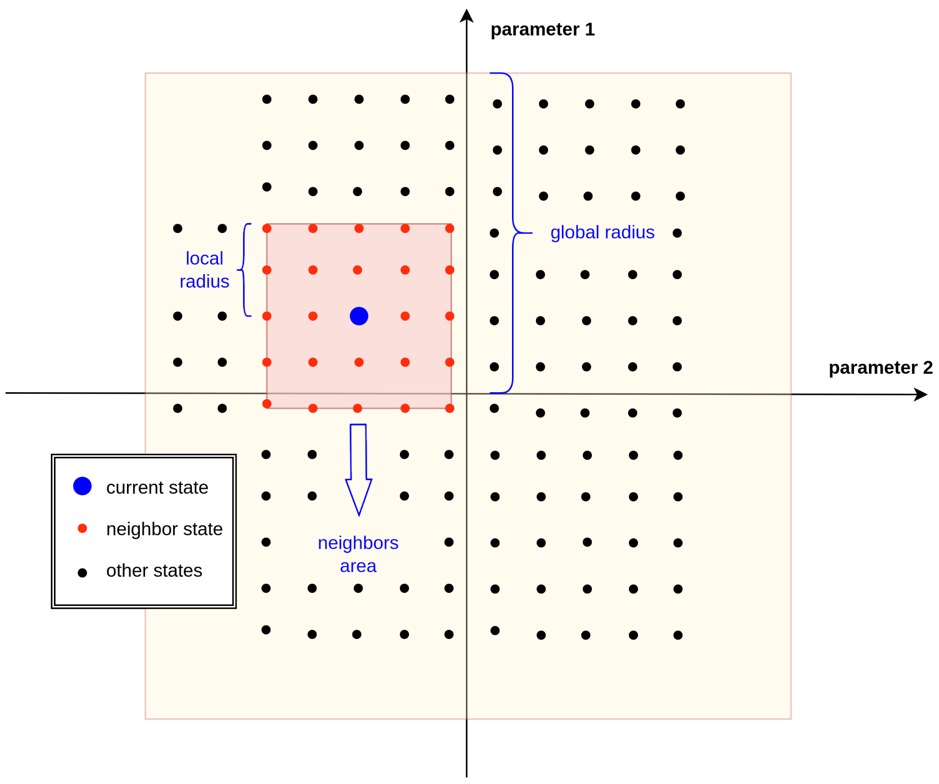


Figure 1.5: The provided diagram depicts a rudimentary representation of a two-dimensional environment, characterized by a local-radius of 2 and a global-radius of 7. These parameters, intrinsic to the problem structure, are subject to modification by the user, conferring a degree of flexibility in the model. Should the user's requirements extend to the utilization of more than two parameters, the environment's dimensionality would correspondingly expand, equating to the number of parameters engaged. As delineated in the illustrative diagram, the red points symbolize the potential successive states that the agent could transition into, thereby representing the neighboring locations. Conversely, the black points designate the set of states that are unattainable from the current state, denoted in blue in the given visual representation. This distinction underlines the fundamental states and transitions inherent to the illustrated agent-environment interaction.

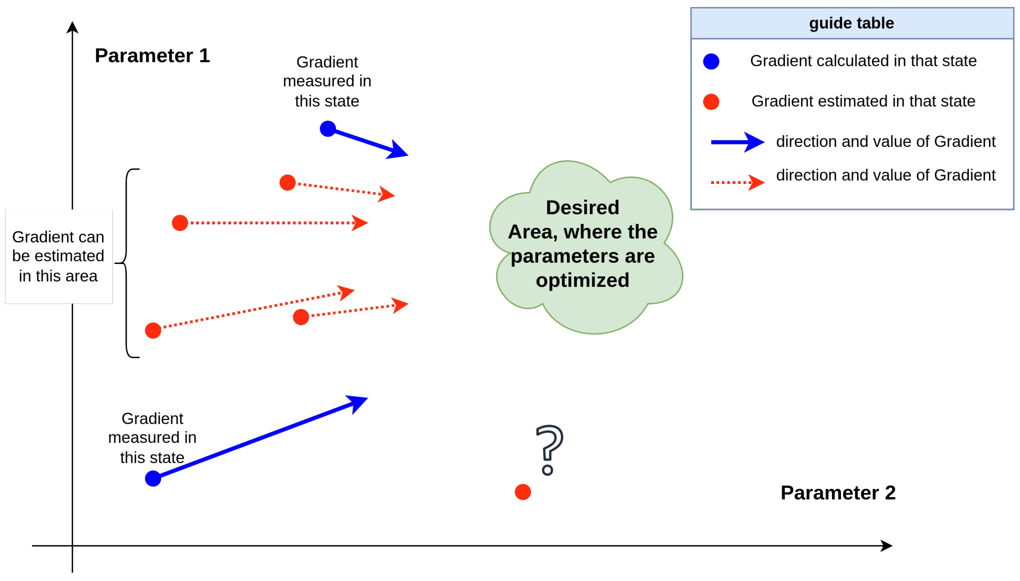

Parameter 1
Gradient measured in this state
Gradient can be estimated in this area
Desired Area, where the parameters are optimized
Gradient measured in this state
?
Parameter 2
guide table
● Gradient calculated in that state
● Gradient estimated in that state
direction and value of Gradient
direction and value of Gradient

Figure 1.6: Assume that the states at the blue points are already visited by the agent and we calculated the GD at those states. Now we are visiting one of the red states and we need to take a GD based action. In this situation, the information about the direction and magnitude of the gradient in the blue states can easily be used to estimate the value and direction of the red points through simple interpolation. But for the red state with question mark, as we never discovered that area previously, it is not possible to estimate, then we need to calculate the GD for that point again.

2 | Chapter two: Implementation and Coding

This section outlines the step-by-step implementation of the project, explaining the algorithms by using pseudo-code. We first go over simulation input files and their roles in simulating a biological system. Then, we detail the initial steps needed after obtaining these files, with a special focus on creating a molecular trajectory, which is a key part of this stage. Once all the pieces are in place on the biological side, we can move on to the application of RL.
The project unfolded in the following steps:

1. My first task was to establish a connection with supercomputers in Barcelona to set up code repositories and simulation protocols. While this was time-consuming, it's not closely tied to the core ideas of the thesis, so it won't be discussed in depth.
2. I then gathered and customized useful codes from previous work done by others:
   (a) The trajectory producer: Given a PDB and Topology file, this code generates a molecular trajectory, which we will discuss further in the next section (2.1.1).
   (b) Sensitivity analysis: This function takes a trajectory file and returns a list of atom types along with their sensitivities or derivatives with respect to chosen FF parameters, allowing us to identify the most important atom types for parameter optimization.
3. I linked different parts of these previous codes and organized them in a coherent function library.
4. I developed code to update parameters using only the GD approach.
5. I successfully performed GD as a foundational step for future RL implementations.
6. I then started to develop a model based on RL, beginning with a 2-dimensional space (2 parameters). We used artificial rewards instead of real rewards.

7. I started developing for more than 2 dimensions using real rewards (helicity of the system in the given trajectory).

8. Lastly, I fine-tuned the algorithm and made several improvements, including adjusting the time constant, updating the derivatives, implementing multi-level logging, and improving visualization and monitoring.

2.1. pre-programming

In this segment, we discuss pre-existing entities, files, and codebases that function as dependencies for our code. This includes universally accessible files, retrievable from online sources, as well as pre-developed code components contributed by fellow team members and students.

2.1.1. Biological Files and Entities

In this section, we will delve into the critical biological files and entities that are fundamental for the successful execution and understanding of our research project.

1. PDB file (.pdb) [14]: The Protein Data Bank (PDB) format is a widely used text format for describing three-dimensional structures of molecules, particularly proteins and nucleic acids. The information contained in a PDB file includes atom names, residue names, chain identifiers, and coordinates in 3D space. This data is critical for computational molecular modeling, including MD simulations, docking studies, and structure-based drug design. The PDB file format was originally established by the Protein Data Bank, a repository of protein and nucleic acid structures. One can freely access and download PDB files from the Protein Data Bank's website (www.rcsb.org). Once the PDB file is downloaded, it can be viewed using various molecular visualization software such as PyMOL, UCSF Chimera, or VMD (Visual MD).

2. Topology File (.top) [25]: In molecular simulations, a topology file serves as a critical component that contains the empirical parameters of the system under study. This file typically contains information about atom types, bonds, angles, dihedrals, and other parameters relevant to the molecular system. Essentially, it details the connections and interactions among atoms within the molecule or system. The generation of a topology file involves the selection of a FF, which is a set of parameters and equations employed to calculate the system's potential energy. The choice of FF depends on the type of system being simulated and the required level

of detail. Popular FFs include AMBER, CHARMM, and GROMOS. Typically, the creation of a topology file starts with a coordinate file (such as a PDB file), and software-specific tools are used for its generation. For example, in GROMACS, the 'pdb2gmx' tool might be utilized to generate a topology file from a PDB file. These files are generally in plain text format and can be viewed or edited using any text editor. However, caution must be exercised when editing these files, as incorrect modifications can result in simulation errors.

3. Plumed files (.dat): PLUMED files are additional instructions to the MD engine, specifying how to modify the simulation by adding external forces, or calculating custom quantities. They both deal with a protein, as described in a PDB file, and analyze its alpha-helical structure. An examination of the functions of each file is presented below [12].

   - Plumed file for sensitivity analysis: The first PLUMED file is designed to conduct a sensitivity analysis on a protein's structure. It forces the protein to adopt different configurations and observes how it behaves under these conditions. The aim is to sample states for the calculation of free energy derivatives between them. This detailed exploration helps to create a reliable foundation for methods such as GD, as it prevents the analysis from getting trapped in local minima.

   - Plumed file for analysing system's behaviour during simulation: It monitors the alpha-helical structure of a protein without imposing any restrictions or manipulations. The aim here is to observe the protein's natural behavior under simulated conditions. The observations from these simulations could be used as an input for an RL algorithm to optimize a function related to the protein's structure. Furthermore, it can be used either during the simulation or as a post-processing analysis tool (as opposed to the previous one which has to be used during the execution of the simulation)

   The first file actively enforces various conditions by altering the Hamiltonian to investigate the system's behavior, while the second file passively observes the system's natural behavior for optimization purposes.

4. Trajectory file (.xtc) [29]: A trajectory file is a standard output file type for MD simulations. This file records the positions of all atoms or particles in the system as a function of time, effectively tracing their "trajectory" through space over the course of the simulation. MD simulations are often performed to study the physical movements of atoms and molecules. The data provided by the simulations can

be used for a wide range of purposes, such as studying the folding of proteins, the formation of chemical complexes, or the behavior of materials under various conditions. The generation of a trajectory file primarily requires two inputs: the PDB file and the Topology file. The PDB file provides the initial conditions, while the Topology file defines the Hamiltonian of the system. With both these inputs, when you run an MD simulation, the simulation package will integrate the equation of motion, applying physical laws (like Newton's laws of motion) to calculate the forces between atoms and their subsequent movement. At specified time steps, the simulation will write the current positions (and sometimes velocities and forces) of all the atoms to the trajectory file. The result is a detailed record of how every atom in the system moved over the course of the simulation. The trajectory file format can vary depending on the MD simulation software being used. Some common formats include .dcd, .xtc, and .trr. This trajectory file can then be analyzed post-simulation using various analysis tools to understand different aspects like RMSD, RMSF, hydrogen bonding, radial distribution function, and more.

2.1.2. Primary and prepared functions

Next, the focus shifts to functions developed by others that are incorporated in this thesis.

1. trajectory_producer function: The function trajectory_producer is a crucial part of the simulation pipeline, written by other developers, and then modified by me, primarily in the conditional (if-else) section to handle various types of processing. The function creates MD trajectories, where the input parameters are the Plumed file (plumed_file), an identifier (id), the duration of the simulation in nanoseconds (duration_ns), and a directory path (path), where the trajectory file should be saved. The role of the function is determined by the id value, with each id corresponding to a different operation. If id=0, the function initiates a sensitivity calculation. If id=1, it begins a time constant calculation. If id=2, the function initiates the first iteration of the process, where the Plumed file is simple and does not contain forces. For id>2, it handles subsequent iterations of the process, in these cases, the simulation is supposed to read the last checkpoint file and continue the process with the exact output settings of the last iteration.

2. The sensitivity_calc function in Python is used for performing a sensitivity analysis on a molecular system. The method identifies the influence of individual FF parameters on a chosen molecular observable. It takes in an xtc file (a trajectory file detailing atomic motion), the molecular property based on which reward will be

calculated such as the alpha-helical content in a polypeptide, and a list of atoms to exclude from the analysis. The function first adds all possible modifications of the Lennard-Jones sigma parameter, which are changes in the non-bonded forces between atoms, to the list of calculated sensitivities, excluding the specified atoms. Next, it specifies the trajectory and adds relevant datasets, such as the helicity dataset loaded from the input file. The core sensitivity calculation is then run, which may require significant computation time. After the calculation, it computes discrete derivatives of the system's free energy with respect to the helicity dataset, both in the lower and upper halves of the helicity range. This provides insight into how the free energy as a function of helicity changes with variation in the values of sigma. Finally, it calculates the difference between the free energy derivatives in the upper and lower halves of the helicity range for each atom. These values, stored in the list hel_by_atom, represent the sensitivity of each atom. The output of this function is a list containing each atom's type and its corresponding sensitivity value. These sensitivity values serve as crucial input for subsequent MD simulations or optimizations. The sensitivity analysis thus helps identify the atoms with the greatest impact on the system, guiding parameter selection and system modifications.

2.2. Project Implementation: Entities, Functions and Algorithms

The current iteration of the project encompasses two principal Classes: System and RL_qfunction. The System class encapsulates all entities and functionalities pertaining to the molecular system. Upon receiving the system's pdb and topology, it facilitates trajectory production, helicity calculation, time-constant computation, and the evaluation of the final reward in the current state. Notably, the state entity, which alters upon transitioning to a new state, is represented by the system's topology. The System class also facilitates sensitivity analysis and includes a system modifier function, which permits the agent to adjust current parameters in accordance with the primary algorithm, thereby returning a new state with an updated pdb file and topology. Additional functions are implemented to preclude the occurrence of bugs and erroneous results.
The second class, RL_qfunction, incorporates all entities and functionalities germane to the RL aspect of the code. It comprises functionalities such as Q-value computation and weights updating, which are detailed in the RL section (See: 0.3).

2.2.1. System Object

The code is designed to work with molecular simulations, specifically handling trajectory production, reward calculations based on helix formation, sensitivity analysis, and system modification. It integrates various libraries to facilitate these tasks and provides an object-oriented approach to manage complex biological systems, such as an Alanine Oligopeptide. Below the description of the specific libraries added the project is provided:

- openmm: These are part of OpenMM, a molecular simulation toolkit, and they handle MD simulations, units, and other related tasks.
- mdtraj.reporters.XTCReporter: From the MDTraj library, used for handling trajectory reporting in XTC format, a compressed coordinate file format.
- parmed: Used for loading and manipulating molecular topologies.
- gromologist: A library, handling tasks related to the GROMACS simulation package.
- openmmplumed.PlumedForce: Provides an interface between OpenMM and PLUMED, allowing enhanced sampling simulations using collective variables.

Main functions

- trajectory_producer(self, ...): Produces a trajectory for a given system based on the provided parameters.

Algorithm 2.1 trajectory_producer Algorithm

```
Require: $plumed\_file, id, duration\_ns, path$
Ensure: A trajectory produced according to parameters.
 1: Define filenames for trajectory, checkpoint, and log
 2: Load topology and positions from files
 3: Define integrator, system, barostat, and simulation
 4: if trajectory file exists then
 5:     Remove existing trajectory, checkpoint, and log files
 6: end if
 7: if $id = 0$ then
 8:     Sensitivity calculation: Add PlumedForce
 9: else if $id = 1$ then
10:     Time constant calculation: Load sensitivity checkpoint
11: else if $id = 2$ then
12:     First iteration: Set positions, minimize energy
13: else if $id > 2$ then
14:     Subsequent iterations: Load previous checkpoint
15: end if
16: Define reporter for trajectory (XTC format)
17: Append reporters for simulation state and energy metrics
18: for $j = 1$ to $loop$ do
19:     Perform simulation steps
20:     Save checkpoint
21: end for
22: Load last saved checkpoint if an exception occurs =0
```

- reward_calculation(self, ...): Calculates the reward for the system based on a given directory, time constant, and run time.

Algorithm 2.2 Reward Calculation

```
 1: Multiply time_constant and run_time by 100
 2: Get the list of files in the directory named "helix.dat"
 3: Initialize empty list for extracted data: data_lists
 4: for each file in file_names do
 5:     Initialize empty list for second column data: data_list
 6:     Open the file for reading
 7:     for each line in the file do
 8:         if line starts with # then
 9:             Continue to next iteration
10:         end if
11:         Split the line into columns
12:         if there are more than 1 columns then
13:             Append float value of second column to data_list
14:         end if
15:     end for
16:     Add data_list to data_lists
17: end for
18: Compute h1_index from data_list based on run_time
19: Compute h1 as mean of h1_index
20: Compute numerator and denominator
21: Compute reward = data_list[0] + numerator / denominator
22: Return reward =0
```

- time_constant_cal(self): Calculates the time constant for the system based on provided data files.
- sensitivity_calc(self, ...): Calculates the sensitivity (or GD) of atom types in the system.

Algorithm 2.3 Sensitivity Calculation

```
1: Create a ThermoDiff object: td
2: Add all possible NBFIXes to the list of calculated sensitivities, excluding given atoms
3: Load helicity data: hdata
4: Specify a trajectory and datasets for sensitivity calculation
5: Run the sensitivity calculation (Note: this part may take some time)
6: Calculate the difference between the binned derivatives for the lower and upper half
   of the dataset
7: Initialize empty list for helicity by atom: hel_by_atom
8: for each key in td.discrete_free_energy_derivatives.keys() do
9:     Append type and difference between derivatives to hel_by_atom
10: end for
11: Return hel_by_atom =0
```

- systemmodifier(self, ...): Modifies the system by changing one or more parameters and then creates a new system.

Algorithm 2.4 System Modifier

```
1: Create a Top object: topo
2: Check the PDB file in the topology
3: Prepare a list of atom changes: atoms_changes
4: for each atom change in atoms_changes do
5:     if parameters equal to "sigma" then
6:         Edit atom type in the topology with the given modification
7:     end if
8: end for
9: Save the modified topology and PDB files
10: Create a new SystemObj: newsys
11: Call the trajectory_producer method on the new system with the given parameters
12: Return the new system =0
```

- helix_reward_calc(self, ...): Calculates the reward associated with helix formation in the system.

2.2.2. RL_qfunction Object

The class implements a LQFA for RL. It's designed to manage the Q-values and weights within a particular space, and it supports updating these Q-values according to the provided RL algorithm.

Attributes

- Global Parameters: Parameters like global_dimensions, global_radius, local_radius, and initial_qval define the space and characteristics of the problem.
- Current Location: Represents the current position within the global space.
- Weights and Q-values: Both global and local weights, and Q-values are maintained for calculations.
- Various Series: Lists to keep track of weights, locations, and other data throughout the learning process.

Constructor: __init__ method

This constructor initializes the class with the given parameters and sets up the initial state of the LQFA, including global and local weights.

Main Functions

This constructor initializes the class with the given parameters and sets up the initial state of the LQFA, including global and local weights.

- q_value_calculation: Calculates the Q-value at a specific location using Gaussian smoothing. It manages the bounds and normalization to provide accurate Q-value computation.

Algorithm 2.5 q_value_calculation Algorithm

```
Require: $loc, normalize$
Ensure: The calculated Q-value for the given location.
 1: Call $\_check\_borders$ with $loc$ to get $mins, maxs$
 2: $gaus$ ← Call $\_nd\_gaussian$ with parameters 3, self.local_radius, self.global_dimensions, normaliz
 3: Define $ranges$ as $[(x, y)$ for $x, y$ in zip$(mins, maxs)]$
 4: Define $slices\_list$ as [slice$(min, max + 1)$ for $min, max$ in $ranges$]
 5: $local\_weights$ ← self.global_weights[$tuple(slices\_list)$]
 6: $local\_weights\_copy$ ← np.copy($local\_weights$)
 7: $loc\_tuple$ ← tuple(self.local_radius for $_{\text{in range}}$(self.global_dimensions))
 8: $gaus\_weights$ ← Call $\_operation\_matrices\_with\_location$ with
    $local\_weights\_copy, gaus, loc\_tuple$, 'mul'
 9: $qval$ ← np.sum($gaus\_weights$)
10: return $qval$ =0
```

- update_weights: Updates the weights based on the current state, action, reward, and other parameters. This is guided by the LQFA formula, and it ensures proper update to learn from the environment.

Algorithm 2.6 update_weights Algorithm

```
Require: $id, Alpha, Gamma, gaussian\_sigma, reward, normalize$
Ensure: Next action, data, and the list of locations.
 1: Copy $self.global\_qvalues$ to $global\_qvalues\_copy$
 2: $Gaus \leftarrow$ Call $\_nd\_gaussian$ with $gaussian\_sigma$, self.local_radius, self.global_dimensions, $normali$
 3: if $id == 2$ then
 4:     Call $\_check\_borders$ with $self.current\_location$ to get $mins, maxs$
 5:     Calculate $denominator$ using $q\_value\_calculation$ with $self.current\_location$ and
        $normalize$
 6:     Get $numerator$ from $self.global\_qvalues$ indexed by $self.current\_location$
 7:     Calculate $factor$ as $numerator/denominator$
 8:     Update $self.global\_weights$ by multiplying with $factor$
 9: else
10:     Call $\_check\_borders$ with $self.current\_location$ to get $mins, maxs$
11: end if
12: Define $ranges$ as $[(x, y + 1)$ for $x, y$ in zip$(mins, maxs)]$
13: Calculate all combinations for the ranges
14: for $idx, combination$ in $combinations$ do
15:     Update $global\_qvalues\_copy[combination]$ with $q\_value\_calculation$ using
        $combination$ and $normalize$
16: end for
17: Define $ranges$ using $mins$ and $maxs$
18: Construct $slices\_list$ from $ranges$
19: Extract $qvalues\_local$ and $weights\_local$ from the global matrices
20: Copy $qvalues\_local$ to $qvalues\_local\_copy$
21: Create $loc\_tuple$
22: Set the central element of $qvalues\_local\_copy$ to 0.0
23: Determine $maxQ\_index$ and $gamma\_maxQ$ values
24: Calculate $diff$ and $Delta$
25: Compute $weights\_update$ using $Alpha, Delta$, and $Gaus$
26: Get next action as the difference between $maxQ\_index$ and $loc\_tuple$
27: Compute $current\_qval$ and $next\_qval$
28: Update the global matrices using $\_operation\_matrices\_with\_location$
29: Initialize an empty list $data$
30: Append rounded values of $reward, current\_qval, next\_qval, diff, Delta$, _recursive_average($weights$
    to $data$
31: Append $self.current\_location$ to $self.locations\_list$
32: return $next\_action, data$, self.locations_list $= 0$
```

Helper Functions

- _nd_gaussian: Creates an n-dimensional Gaussian kernel.
- _normalize_matrix: Normalizes a given matrix.
- _gaussian_multiplier: Multiplies the Gaussian kernel with a given matrix.
- _check_borders: Checks the bounds of the given location.
- _operation_matrices_with_location: Applies an operation between two matrices at a specific location.
- _recursive_average: Calculates the average of a list.
- gradients2action_convertor: Converts gradients to action based on grid steps.
- action2changes_convertor: Converts action to changes in the grid.

3 | Chapter three: Results

In this chapter, the outcomes of implementing various algorithms using the Python programming language and specialized libraries are discussed. The development process began with a basic model, which was systematically refined to incorporate all necessary algorithms, thereby enhancing both the model and the resulting data.

Existing research in the field suggests that GD is a viable algorithm for optimizing FF parameters. To gain a comprehensive understanding of the problem and the steps involved in its implementation, a preliminary test was conducted using the GD approach with real rewards. In the context of this thesis, the helicity of the system structure serves as the primary metric for evaluating the efficacy of the implemented algorithms, functioning as a simplified yet informative test result.

Following this, a simplified LQFA model was developed as a reference model. This environment has two primary characteristics: it operates in a two-dimensional parameter space and generates artificial rewards. The focus on two dimensions allows for easier coding and visualization, serving as a foundation for more complex, N-dimensional models. Artificial rewards were used to expedite the process, eliminating the need for time-consuming trajectory production.

After successful testing in this simplified environment, an N-dimensional model was implemented, again by utilizing artificial rewards. This model was then integrated with the initial GD approach with real rewards. Both algorithms operate in tandem, guided by an epsilon-greedy policy: 70% of actions are based on LQFA, 20% on GD, and 10% are random actions. Due to the use of multiple approaches, including LQFA and random actions, periodic updates to derivative values became necessary. This is particularly crucial when the agent deviates significantly from the last point of derivative calculation, as the different approaches may not align with the direction indicated by GD. In such cases, a re-calculation of derivatives, or sensitivity analysis, is required to maintain the integrity of the simulation.
The chapter concludes by presenting and discussing the results obtained at each stage of implementation.

3.1. Results: Gradient Descent Approach With Real Rewards

The first goal of this thesis was to set up the basic code needed for running a simulation. This involved creating a long trajectory, applying various forces to the system (by plumed library forces), and measuring how each type of atom responds to these forces.

It is crucial to note that not all atom types within the system are relevant for sensitivity analysis in FF optimization. Generally, ions such as sodium (Na+), chloride (Cl-), and potassium (K+) are excluded, as they primarily serve to neutralize the system rather than contribute to molecular interactions under study. Similarly, solvent atoms like water ('OW' for oxygen, 'HW' for hydrogen) are typically omitted to focus on the solute [7]. The exclusion criteria commonly encompass:

- Solvent atoms
- Ions
- Atoms not integral to the molecule's covalent structure
- Atoms with minimal impact on the interactions or properties under investigation

In the following table, the derivatives for each atom type in the Alanine Oligopeptide (20-mer) are shown, excluding 'OW', 'HW', 'Cl', and 'K'. These were left out because the first two are solvents and the last two are ions, which aren't the focus of this analysis. These exclusions are common in sensitivity studies to keep the focus on the molecule's structure and interactions. The table provides details on the remaining atom types and their derivatives. In the table above, the derivatives for various atom types are presented. It

Atom Type	Derivative Value
C	600.74
CT	-265.11
H	-62.78
H1	-437.97
HC	151.86
HP	-4.40
N	1007.14
N3	-3.18
O	-424.03
O2	-2.00

Table 3.1: Derivatives of all the atom types in Alanine Oligopeptide (20-mer)

is advisable to focus on atom types with the highest absolute values of their derivatives. The number of atom types considered in the simulation is contingent upon computational efficiency and time constraints. Including additional atom types introduces more dimensions to the system, as each atom type necessitates the modification of its Lennard-Jones parameters ϵ and σ, as well as potentially other FF parameters. While this can enhance the accuracy of the simulation, it also increases computational complexity and time requirements[16].

In the scope of this thesis, attention is restricted to the four atom types exhibiting the highest sensitivity.

Atom Type	Derivative Value
N	0.7620
C	0.4545
H1	-0.3314
O	-0.3208

Table 3.2: Derivative values for the four most sensitive atom types in the Alanine Oligopeptide (20-mer). These values are normalized using the L2 norm to focus on the direction of parameter adjustments, improve numerical stability, and facilitate comparison.

In the optimization algorithm, each gradient component is scaled by $-\alpha_{\text{gr}}$, which acts as the learning rate. This scaling serves multiple purposes: it sets the step size for the optimization, ensures that parameter adjustments occur in the direction that minimizes the objective function, and improves the numerical stability of the algorithm. Consequently, the choice of α_{gr} is crucial for both the speed and accuracy of the algorithm's convergence.

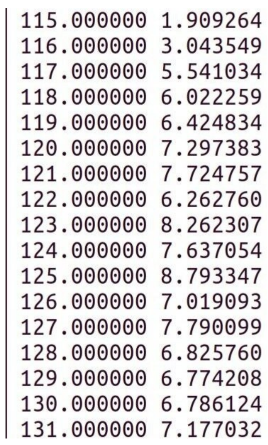
In order to establish an appropriate environment for our RL (RL) model, the continuous parameter space was discretized into a grid [23]. Each point on this grid represents a state in the RL framework. The grid step size, determined based on the values of the FF parameters, was set at 0.005. On the other hand, the magnitude of the actions within the RL model is influenced by the derivative values. A learning rate of $-\alpha_{\text{gr}} = 0.03$ was selected for scaling these derivatives. After this scaling, the derivatives were converted into actions, as detailed in the subsequent table.

In the RL model, the actions are derived from the gradients by first scaling rounding them to the nearest integer. This process converts the continuous gradient values into discrete actions that the RL agent can take. For instance, as shown in Table 3.3, the raw gradients for atom types O, H1, C, and N were converted into the corresponding actions through this process.

Atom Type	Gradient	Action
N	-0.022860	-4
C	-0.013635	-2
H1	0.009941	1
O	0.0096248	1

Table 3.3: Conversion of gradients to actions for different atom types in the RL model.

In the final stage of the algorithm, a loop is employed to iteratively adjust the FF parameters based on the actions derived from the gradients. For the purposes of this study, a simplified example that relies solely on GD was executed over a sufficient number of iterations (In the latest simulation, following eight iterations and actions along the derivatives) to evaluate its effectiveness. The objective was to assess whether this optimization technique could yield a high helicity value within a meaningful range. Given that the maximum attainable helicity is 16, achieving a value close to this upper limit is unlikely. The figure below illustrates the helicity output file, where the average helicity achieved was 5, with peak values reaching up to 8 at certain points. While this outcome may not represent an optimal solution, it serves as a promising indicator that GD can guide the agent to a local maximum.



```
115.000000 1.909264
116.000000 3.043549
117.000000 5.541034
118.000000 6.022259
119.000000 6.424834
120.000000 7.297383
121.000000 7.724757
122.000000 6.262760
123.000000 8.262307
124.000000 7.637054
125.000000 8.793347
126.000000 7.019093
127.000000 7.790099
128.000000 6.825760
129.000000 6.774208
130.000000 6.786124
131.000000 7.177032
```

Figure 3.1: The figure displays a segment of the helicity output file, generated from a trajectory of an Alanine Oligopeptide (20-mer). This file is produced in each iteration following the modification of the parameters for four atom types identified as highly sensitive. The helicity output file provides a time-resolved measure of the system's helicity at each simulation step.

3.2. Results: 2-Dimensional Linear Q-function Approximation with artificial rewards

In this section, as well as the subsequent one, artificial rewards are employed to evaluate the performance of the Q-learning algorithm. Initially, the algorithm is tested in a two-dimensional parameter space, and later extended to an N-dimensional space. The transition from artificial to real rewards is straightforward, requiring only a substitution of the reward-generating function.

The rationale for utilizing artificial rewards in a two-dimensional parameter space is based on the likelihood of certain regions possessing higher helicity values than others. Specifically, when the parameters are near their optimized values, the system is expected to exhibit elevated helicity. To simulate this, two locations within the two-dimensional space are designated to have exceptionally high helicity values (see the Figure 3.2), approaching the theoretical maximum of 16. Surrounding these locations, neighboring points are also assigned elevated helicity values, calculated using a Gaussian function. As one moves away from these designated high-helicity locations, the helicity values gradually decrease. The distribution of these artificial rewards is illustrated in the figure below.

In the context of this research, a critical aspect warranting detailed discussion is the initialization of the weights matrix in LQFA. This matrix plays a pivotal role in the optimization of FF parameters, which is the focus of this study. In this approach, we employ an n-dimensional parameter space discretized into grids. Each grid point has a weight associated with it, and higher weights in a particular region indicate a higher probability of receiving substantial rewards there by locally increasing the value of the Q-function.

The initialization of the weights matrix is of paramount importance for the efficacy of the algorithm [38]. Initializing the weights matrix with low or zero values can lead the LQFA to converge to a local maximum. This behavior can be understood by examining Equation 7, which is an extension of the Q-learning algorithm. When the agent receives a high reward at a particular state, the Q-value for that state tends to increase, unless the state has been previously visited and lacks expected return from that state. Conversely, if the agent receives a low reward, the Q-value for that state tends to decrease, unless the state has sufficiently large expected return. This can lead to a feedback loop where the agent continually revisits high-reward states, potentially getting stuck unless it employs an ϵ-greedy policy to explore other states.

Alternatively, initializing the weights matrix with high values (greater than 10) offers a

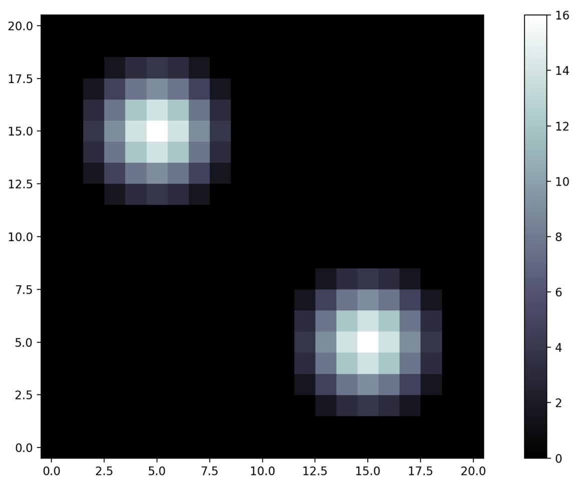


Figure 3.2: The figure above illustrates the distribution of artificial rewards within our two-dimensional parameter space, which consists of 20 steps along each axis. The agent commences its exploration from the exact center of this space, representing the initial parameter configuration. Each action taken by the agent moves it to a new location within this grid. The objective is for the agent to traverse the parameter space in search of regions with higher reward values, as indicated by lighter shades in the figure. It is important to note that the reward distribution merely sets the average reward that the agent can expect to receive at a given location. For instance, if the agent moves to a high-potential area (depicted in lighter colors), it will receive a random reward whose average is higher than that in other regions of the space.

different dynamic. In this scenario, the agent is predisposed to explore various regions of the parameter space. Upon encountering a region, even with high rewards, the weights in that area will typically decrease (when the neighboring states are initialized with higher weight values, the rewards attained by the agent may still be lower than the weights and Q-values of these neighboring states). This reduction in weights causes the most promising point in the neighborhood to shift away from the agent and further promoting exploration. This behavior is visually represented in the figures below (3.3 and 3.5), which compare the outcomes of low and high initial weights matrix values.

Based on Equation 7 and the examples discussed earlier, it is advisable to initialize the weights matrix with values that are either close to the maximum or higher than the average. This initialization strategy has the disadvantage of being more computationally intensive in the search for high-potential areas. The reason is that the agent is required to explore the environment extensively. In this process, it weakens low-potential areas by assigning them lower weights while maintaining or increasing the weights of high-potential

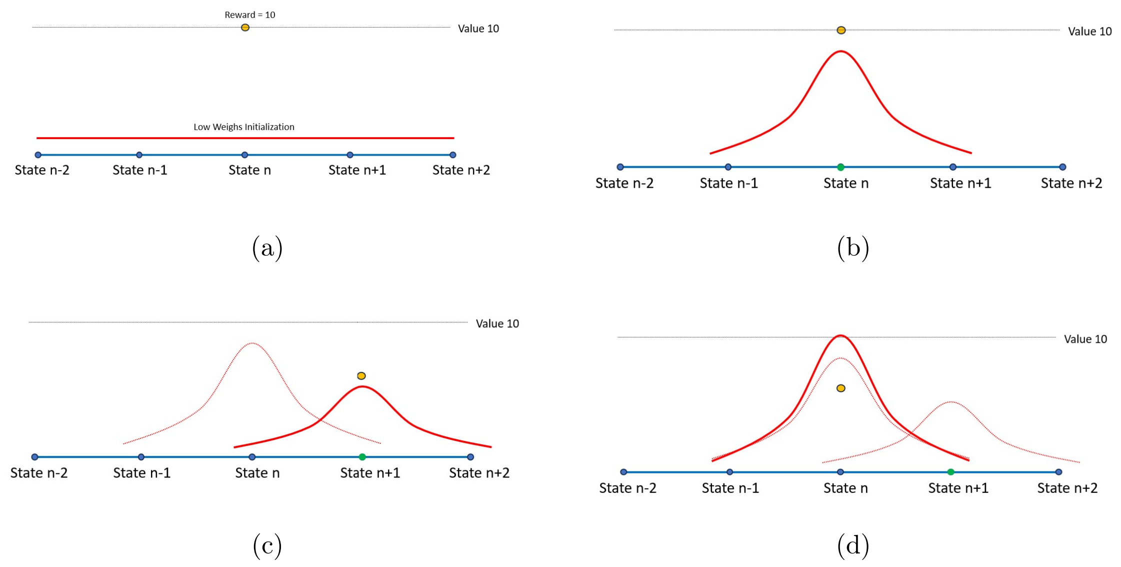


Figure 3.3: In these figures, for the sake of better understanding, we have plotted the Gaussian curves separately and have focused exclusively on those states whose weights have undergone changes. However, it is important to note that in a real-world scenario, the actual weights are the sum of the products of each neighboring Gaussian function value and its corresponding weight. In mathematical terms, the weights are magnitude numbers that are multiplied by the Gaussian matrix function $f(s,a)$. In this example, for the sake of simplicity, the status of the environment is depicted in a one-dimensional space. The initial matrix of weights is configured to have low magnitudes, either set to zero or close to zero. Initially, state n is selected, and the agent receives a reward of high numerical value. Subsequently, state $n+1$ is selected, and the agent receives a reward of lower value. Finally, the agent tends to revert to the previous state n due to the increased weights associated with it, while the neighboring states maintain lower weights and, consequently, lower Q-values. The agent may choose other neighboring states for reasons such as taking alternative actions, employing GD, making random moves, or encountering another state with a previously calculated Q-value that is higher than that of state n [38].

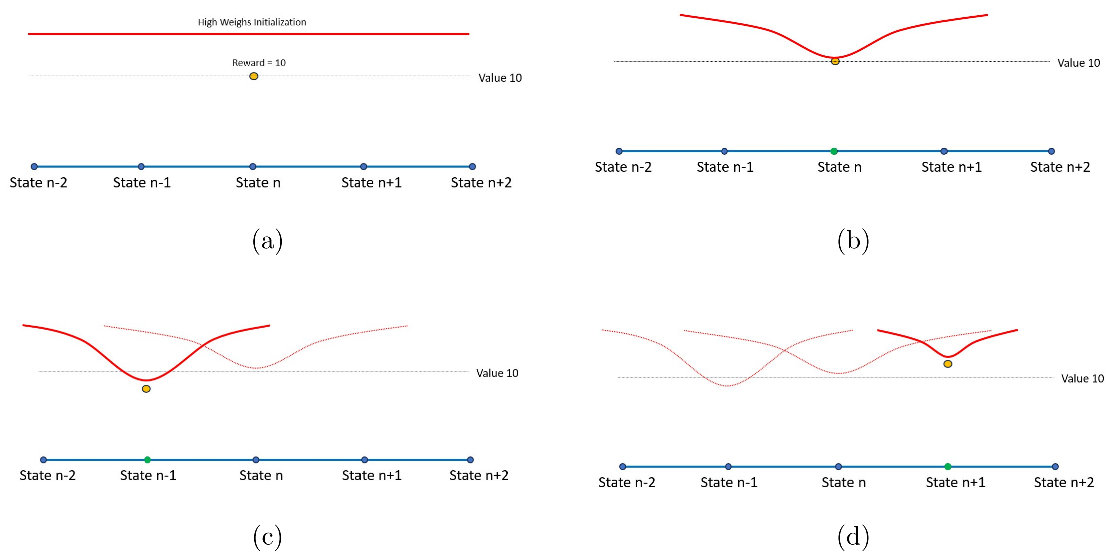


Figure 3.4: In the second example, initially, all the weights are set to their maximum value, which in the context of our project is 16, or close to this maximum value. It is improbable for the agent to receive a reward that is higher or equal to this maximum during the initial iterations. Consequently, the weight associated with the initially selected state n is reduced. This change not only affects the weight of state n but also has an impact on the weights of its neighboring states. However, the weights of the neighboring states still remain higher than that of state n, thereby rendering them probable candidates for selection by the agent in subsequent iterations. In this scenario, state $n-1$ is chosen, and it yields a reward that is lower than that of state n. Hence, between states n and $n-1$, state n is more likely to be selected in future iterations. However, the remaining states, which still have high weights, have not yet been evaluated. As such, we provide an opportunity for these neighboring states to be chosen in forthcoming iterations. For instance, in image (d), state $n+1$ is selected. It should be noted that other states could also potentially be selected in lieu of state n, which was previously selected and now has a diminished weight and, consequently, a lower Q-value [38].

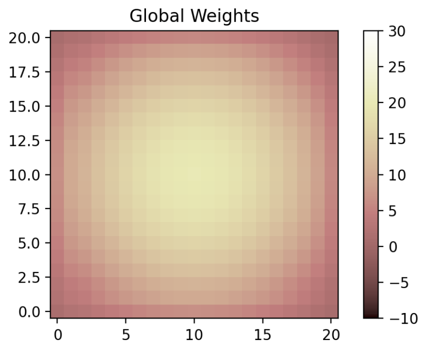
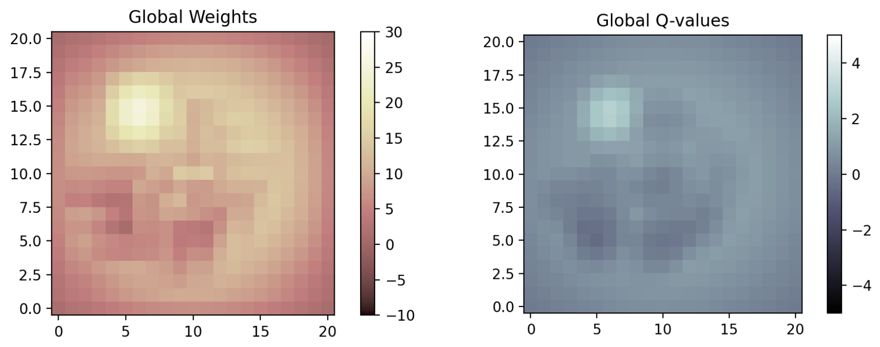


Figure 3.5: This figure serves as a graphical representation of the initialization of the weights matrix in a two-dimensional space, with the grid comprising 20 discrete steps along each axis. The center, which is represented by lighter hues, corresponds to higher values in the weights matrix. As one moves away from the center, the values diminish and are denoted by progressively darker shades [38].

areas. As a result, high-potential areas become more likely to be selected in subsequent iterations, thereby encouraging the agent to persist in those regions.
In the context of our study, we employ a two-dimensional weights matrix, as depicted below. The reward distribution, referenced in Figure 3.2, was initialized with relatively high weight values. However, it is crucial to highlight that as one moves away from the agent's initial position, serving as the center point of our environment, the initial weights diminish in value. This method of initialization acts as a form of regularization, effectively discouraging exploration far from the original parameter set.

The agent may now commence the exploration of the environment. Specifically, the agent should probe its immediate neighbors to identify regions characterized by high reward values.

Observing Figure 3.6, it becomes evident that the agent has deviated from the central position in pursuit of more advantageous regions. The adoption of LQFA allows the agent to explore on a region-by-region basis rather than a state-by-state basis, thus substantially reducing the time complexity of the search operation. In Figure 3.7, there are points with relatively high rewards; the optimistic initial values were still sufficient to attract the agent towards further exploration. Ultimately, the agent identified a high-potential area, indicated by a lighter color, signifying greater weight values, while other areas darkened in comparison.
Further insights can be gleaned from this run. Figure 3.6 highlights a correspondence between the Q-values and weights, especially in high-potential areas where higher Q-values are observed. This similarity is also corroborated by Figure 3.7, where the Q-value



(a) (b)

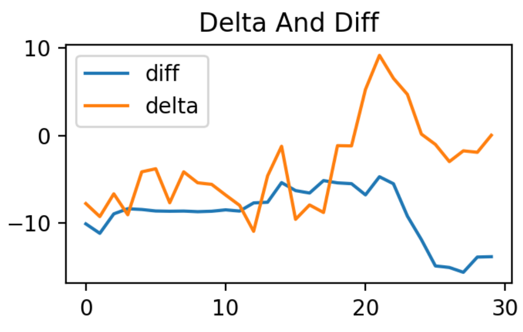
Figure 3.6: In the aforementioned figure, the outcomes of the agent's exploration-exploitation activities on the Weights matrix, as well as the Q-values associated with each state, are illustrated. Areas depicted in darker shades signify regions with a low probability of yielding high rewards. Conversely, lighter shades indicate regions where there is a higher likelihood of encountering substantial rewards.

plot is presented. The term *diff* represents the difference between the current Q-value and γ multiplied by the next Q-value. Here, γ serves as a constant factor that quantifies the consideration given to the future Q-value in decision-making, as detailed in Section 0.3.1. The term *Delta* is $reward + \gamma \times$ (next Q-value $-$ current Q-value), which updates the Q-value of that particular state. Consequently, the gap between *diff* and *Delta* in Figure 3.7 is effectively filled by the reward value. As the agent converges to a high-potential area, both the actual value of the result and the average reward exhibit an increase.

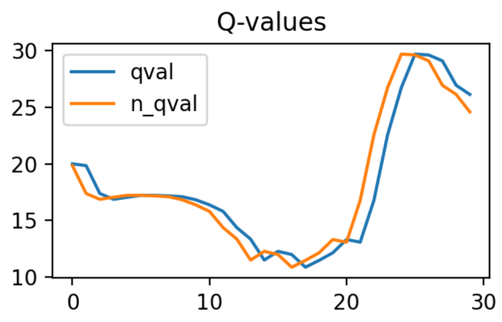
In summary, it appears that our algorithm is capable of identifying areas with high reward potential within approximately 20 iterations. Furthermore, the algorithm effectively distinguishes these high-potential areas from those with lower potential. This prepares the groundwork for extending the implementation to accommodate an N-dimensional parameter space.

3.3. Results: N-Dimensional Linear Q-function Approximation with artificial rewards

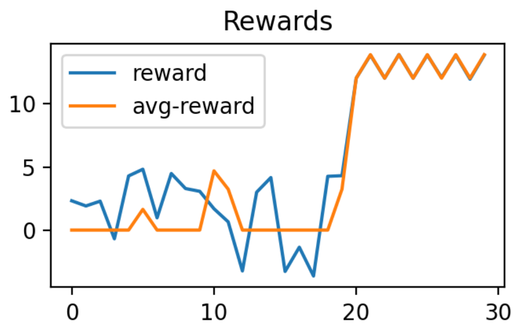
In this segment of the Results chapter, we concentrate specifically on two instances within an N-dimensional space. Our objective is to ascertain whether these instances yield appropriate outcomes. This serves as a preliminary assessment, preparing our algorithm for its full-fledged version: N-dimensional RL that incorporates genuine rewards. It is crucial



(a)



(b)



(c)

Figure 3.7: In the rewards Figure, the blue line represents the actual value of the reward, whereas the orange line delineates the average reward value. Notably, fluctuations are evident in the early stages, particularly before iteration 20. During this phase, the actual reward value approached 5 at certain instances, which is considerably high. However, the agent continued its exploratory behavior, maintaining high weight values for these areas while also venturing into new areas in search of higher rewards. This behavior can be attributed to the high initial weights, as lower initial weights would have resulted in the agent's prolonged stay in these areas.Post iteration 20, the agent identifies an area characterized by exceptionally high rewards. This area exerts sufficient influence to override the agent's inherent tendency for exploration, resulting in its stationary behavior within that locale. Simultaneously, significant fluctuations in the Delta and diff values are observed prior to iteration 20. These fluctuations stabilize after the agent locates the high-reward area. Importantly, the gap between Delta and diff represents the reward value. Before reaching this high-potential area, the rewards were not substantial enough to create a noticeable difference between Delta and diff. Upon entering the high-reward zone, a significant divergence between Delta and diff manifests, attributable to the elevated reward values.

to ensure the scientific accuracy and effectiveness of our approach before advancing to more complex scenarios.
All aspects pertaining to the system, reward distribution, weight initialization, and the algorithms remain consistent with the 2-dimensional example. The primary modification involves the addition of more dimensions, thereby expanding our parameter space. Consequently, to avoid redundancy, we will not reiterate the underlying concepts and the rationale behind them.
In the initial test, we employed a 4-dimensional parameter space. The parameters `local_radius` and `global_radius` were initialized with values of 2 and 5, respectively. This configuration results in a $10 \times 10 \times 10 \times 10$ space. The rewards were strategically positioned at coordinates $(2, 2, 2, 2)$ and $(7, 7, 7, 7)$. As anticipated, when the agent approaches these specified areas, it should identify them and emphasize them with augmented weights.

It's pertinent to note that the initial weight was set at 10. This means that the central weight possesses a value of 10, which diminishes progressively according to a Gaussian function as one moves away from the center.

Referring to figure 3.8, it's evident that after approximately 25 iterations, the agent successfully localized the desired area. Despite encountering intermittent reward fluctuations earlier in the process, the agent persisted in its exploration. By the 25th iteration, the agent transitioned from exploration to exploitation of the identified area. Concurrently, the Q-values mirrored the behavior of the high-value rewards, increasing in tandem with the rewards and weights.

The variable `Delta` serves as an indicator, determining whether the subsequent update will result in an increase or decrease in weights. Initially, its value was negative, signifying the agent's inclination towards exploration over exploitation. However, post the 25th iteration, this value transitioned to positive. The subsequent plots underscore this shift, illustrating that the update value became positive and the local weights of the targeted area registered elevated values.

In our subsequent test, we initialized the system with a `global_radius` of 8, a `local_radius` of 2, and expanded it to 4 dimensions. The rewards were strategically positioned at coordinates $(3, 3, 3, 3)$ and $(14, 14, 14, 14)$. The agent commenced its operations from a central location, specifically $(8, 8, 8, 8)$. The visual data indicates that due to the enlarged parameter space, as a result of the increased `global_radius`, the agent required close to 50 iterations to locate the desired region. This observation underscores the efficiency gains achievable by minimizing the number of parameters and reducing the grid's dimensions. A sensitivity analysis, coupled with a concentrated focus on pivotal FF parameters, proved

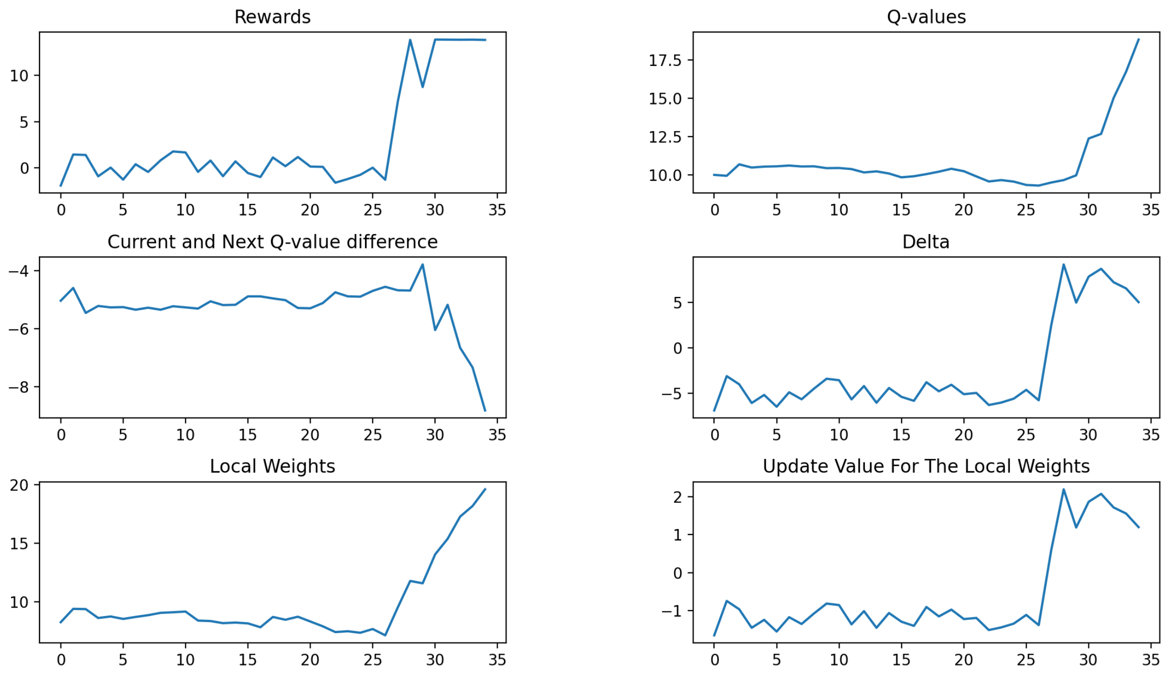


Figure 3.8: In this Figure, we present the visual representations of the outcome variables from our first 3D test. These plots offer insights into the performance and behavior of the system under study.

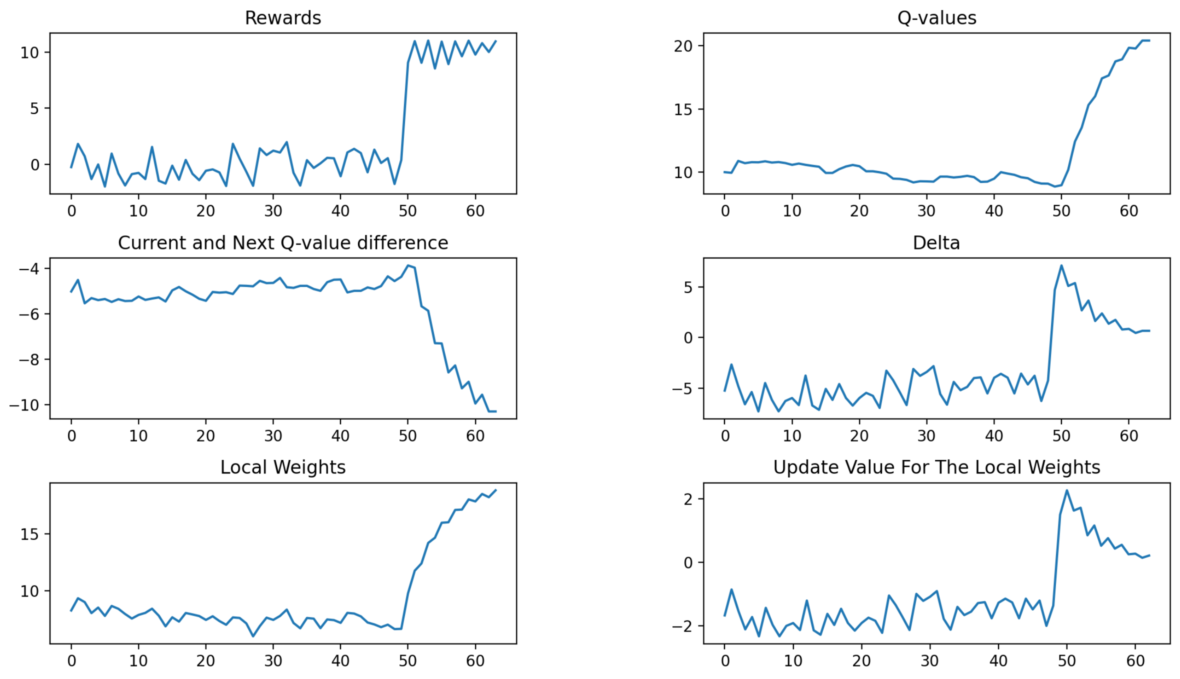


Figure 3.9: In this Figure, we present the visual representations of the outcome variables from our second 3D test. These plots offer insights into the performance and behavior of the system under study.

indispensable in this scenario.

Upon examining the plots from the second example 3.9, one can observe a decline in the values of both `Delta` and `Update Weights` post their peak around the 50th iteration. This trend is attributed to our design choice of setting the gamma parameter, ensuring that the weights do not grow indefinitely. Specifically, when gamma is set close to 0, it leads to convergence to the pure reward, whereas a value close to 1 can potentially lead to divergence to infinity. For values in between, convergence occurs to a multiplicity of the reward. In our design, the weights are constrained to plateau approximately at the point indicating the maximum expected reward. To elucidate further, both the Q-values and Weights stabilize around a value of 20. After this stabilization, due to the gamma setting, the environment introduces updates to these areas with values nearing zero to maintain them around this threshold. Consequently, `Delta`, which signifies the positivity or negativity of the update, also gravitates towards a value close to zero, mirroring the behavior of `Update Weights`.

This particular characteristic offers significant advantages, especially when integrating other actions such as GD or Random. Imagine a scenario where the agent identifies an area with a maximum reward of 6. If the agent remains in this region without further exploration, it will continue to update its Weights and Q-values. However, these updates will only elevate them to levels corresponding to a reward of 6, and no further. This implies that if the agent deviates from this area due to other actions (like GD or Random), it stands a higher chance of discovering regions with even greater potential rewards.

3.4. Results: Complete N-Dimensional RL Model With Real Rewards

In this segment of the Results chapter, we evaluate the comprehensive algorithm using real-world data. It's essential to note that the primary metric under consideration here is the system's helicity. While helicity serves as a straightforward measure to assess the system, there exist other potential metrics that could provide further insights. However, a detailed exploration of these additional metrics falls outside the purview of this thesis, presenting an object of future research or subsequent investigations.

For this study, we employ a complete LQFA to identify the optimal parameters within an N-dimensional parameter space. Although the examples detailed in this section are confined to a maximum of four dimensions, the algorithm and its underlying code have been designed to seamlessly handle spaces with more than four dimensions. The primary constraint in expanding the dimensions is the associated increase in computational processing time.

In the first test, as depicted in Figure 3.10, the agent experienced rapid growth over four states, with values of 6.82, 8.93, 8.17, and 8.64. The peak value reached was approximately 9, which is a significant result. It's important to note the difference between the results based on GD (refer to Section 3.1) and this section. In the former, the helicity values were not averaged, and the best result achieved was around 5 over an entire trajectory. While better results might be achievable with GD, relying solely on derivatives for parameter optimization can lead to local maxima, without exploring potentially superior areas. Conversely, combining Q-learning, GD, and Random moves allows for the exploration of more promising regions. For instance, in this test, the agent moved away from the potential areas near (7,3,5,7) and (9,1,5,9) to explore the more distant state (10,2,3,5), influenced by a random move. A notable aspect of this approach is that after discovering a peak, the algorithm gains some knowledge about the space, making it easier to identify other promising areas. This is evident from the subsequent rewards, which generally increase until another peak is reached at (10,6,1,2) with a value of 7.2. The agent then moves away from this location, again influenced by a random action, to explore further. Towards the end, we observe several significant values around 5.2 in states (5,4,0,9), (4,4,0,10), and

(3,4,0,10), but there's a sharp decline in reward value at (2,4,0,10).

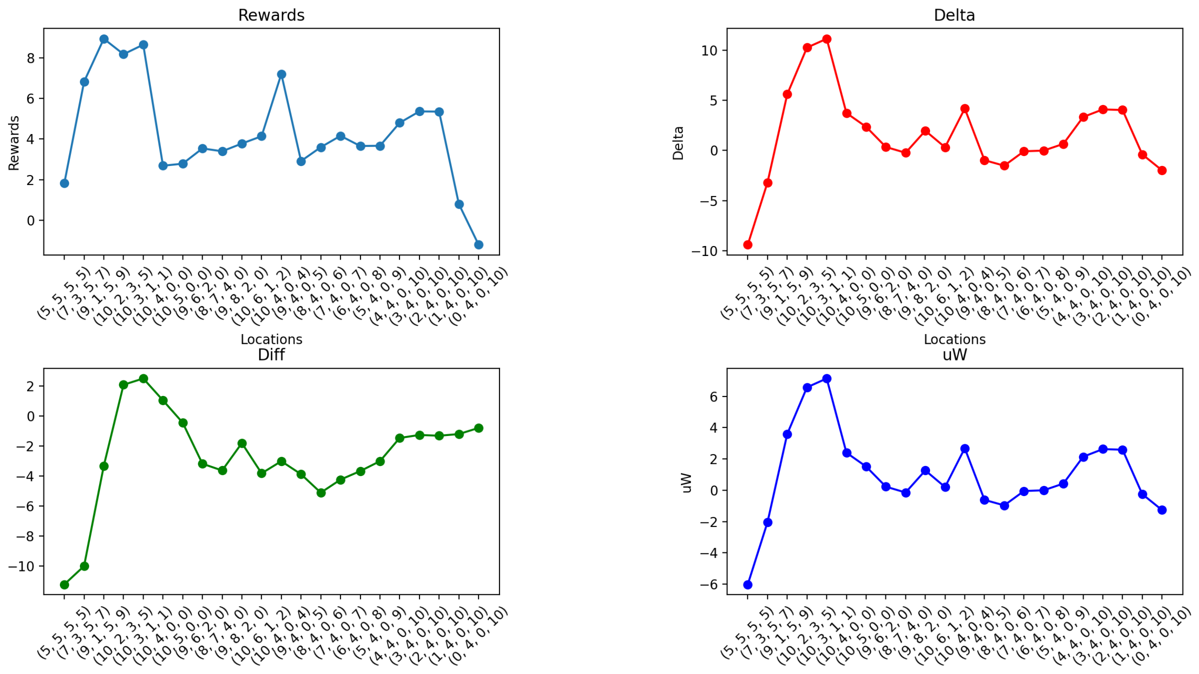
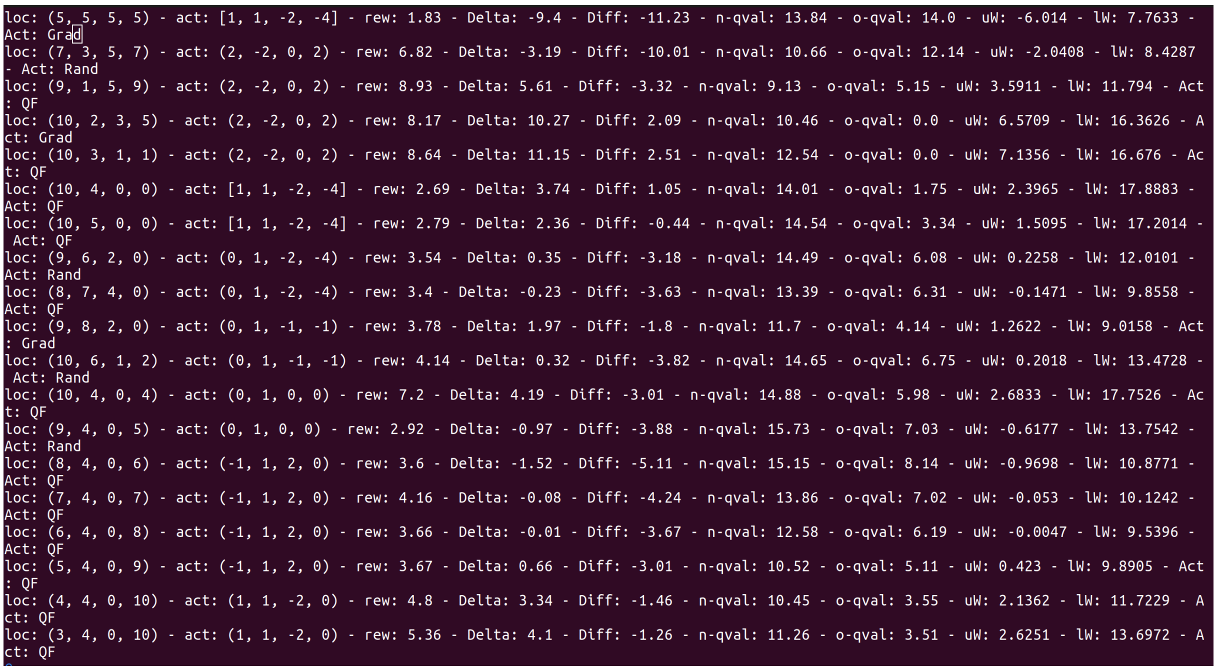


Figure 3.10: This figure presents the graphical results from the first implementation.

Upon examining the plot corresponding to uW, which essentially represents the average updating value of the weights of neighboring states, alongside $Delta$ and $Diff$, several insights can be drawn. Notably, when the agent achieves a high result value, the surrounding environment of that specific location updates the weights to values predominantly exceeding 6. Consequently, this region is designated as a high-potential area, making it a prime candidate for consideration in subsequent iterations. This trend persists across other peaks and desired regions, distinguishing them from the surrounding areas. It's imperative to highlight that, should we decide to investigate a system distinct from the Alanine Oligopeptide (20-mer) yet bearing close structural and atomic similarities, the weighted space derived from the former could serve as an initial space for the latter, obviating the need to commence from a blank slate. Such an approach paves the way for the potential incorporation of Neural Networks in future studies, particularly for supervised tasks.
Below 3.11, one can find the log file generated and updated throughout the implementation process. While its structure may not be immediately intuitive, perusing the logs can offer insights into the values assessed during the course of the implementation.

The results of the second test are illustrated in Figures 3.12 and 3.13. In this test, the agent identified a potential region after approximately 28 iterations, reaching a peak at



```
loc: (5, 5, 5, 5) - act: [1, 1, -2, -4] - rew: 1.83 - Delta: -9.4 - Diff: -11.23 - n-qval: 13.84 - o-qval: 14.0 - uW: -6.014 - lW: 7.7633 -
Act: Grad
loc: (7, 3, 5, 7) - act: (2, -2, 0, 2) - rew: 6.82 - Delta: -3.19 - Diff: -10.01 - n-qval: 10.66 - o-qval: 12.14 - uW: -2.0408 - lW: 8.4287
- Act: Rand
loc: (9, 1, 5, 9) - act: (2, -2, 0, 2) - rew: 8.93 - Delta: 5.61 - Diff: -3.32 - n-qval: 9.13 - o-qval: 5.15 - uW: 3.5911 - lW: 11.794 - Act
: QF
loc: (10, 2, 3, 5) - act: (2, -2, 0, 2) - rew: 8.17 - Delta: 10.27 - Diff: 2.09 - n-qval: 10.46 - o-qval: 0.0 - uW: 6.5709 - lW: 16.3626 - A
ct: Grad
loc: (10, 3, 1, 1) - act: (2, -2, 0, 2) - rew: 8.64 - Delta: 11.15 - Diff: 2.51 - n-qval: 12.54 - o-qval: 0.0 - uW: 7.1356 - lW: 16.676 - Ac
t: QF
loc: (10, 4, 0, 0) - act: [1, 1, -2, -4] - rew: 2.69 - Delta: 3.74 - Diff: 1.05 - n-qval: 14.01 - o-qval: 1.75 - uW: 2.3965 - lW: 17.8883 -
Act: QF
loc: (10, 5, 0, 0) - act: [1, 1, -2, -4] - rew: 2.79 - Delta: 2.36 - Diff: -0.44 - n-qval: 14.54 - o-qval: 3.34 - uW: 1.5095 - lW: 17.2014 -
 Act: QF
loc: (9, 6, 2, 0) - act: (0, 1, -2, -4) - rew: 3.54 - Delta: 0.35 - Diff: -3.18 - n-qval: 14.49 - o-qval: 6.08 - uW: 0.2258 - lW: 12.0101 -
Act: Rand
loc: (8, 7, 4, 0) - act: (0, 1, -2, -4) - rew: 3.4 - Delta: -0.23 - Diff: -3.63 - n-qval: 13.39 - o-qval: 6.31 - uW: -0.1471 - lW: 9.8558 -
Act: QF
loc: (9, 8, 2, 0) - act: (0, 1, -1, -1) - rew: 3.78 - Delta: 1.97 - Diff: -1.8 - n-qval: 11.7 - o-qval: 4.14 - uW: 1.2622 - lW: 9.0158 - Act
: Grad
loc: (10, 6, 1, 2) - act: (0, 1, -1, -1) - rew: 4.14 - Delta: 0.32 - Diff: -3.82 - n-qval: 14.65 - o-qval: 6.75 - uW: 0.2018 - lW: 13.4728 -
 Act: Rand
loc: (10, 4, 0, 4) - act: (0, 1, 0, 0) - rew: 7.2 - Delta: 4.19 - Diff: -3.01 - n-qval: 14.88 - o-qval: 5.98 - uW: 2.6833 - lW: 17.7526 - Ac
t: QF
loc: (9, 4, 0, 5) - act: (0, 1, 0, 0) - rew: 2.92 - Delta: -0.97 - Diff: -3.88 - n-qval: 15.73 - o-qval: 7.03 - uW: -0.6177 - lW: 13.7542 -
Act: Rand
loc: (8, 4, 0, 6) - act: (-1, 1, 2, 0) - rew: 3.6 - Delta: -1.52 - Diff: -5.11 - n-qval: 15.15 - o-qval: 8.14 - uW: -0.9698 - lW: 10.8771 -
Act: QF
loc: (7, 4, 0, 7) - act: (-1, 1, 2, 0) - rew: 4.16 - Delta: -0.08 - Diff: -4.24 - n-qval: 13.86 - o-qval: 7.02 - uW: -0.053 - lW: 10.1242 -
Act: QF
loc: (6, 4, 0, 8) - act: (-1, 1, 2, 0) - rew: 3.66 - Delta: -0.01 - Diff: -3.67 - n-qval: 12.58 - o-qval: 6.19 - uW: -0.0047 - lW: 9.5396 -
Act: QF
loc: (5, 4, 0, 9) - act: (-1, 1, 2, 0) - rew: 3.67 - Delta: 0.66 - Diff: -3.01 - n-qval: 10.52 - o-qval: 5.11 - uW: 0.423 - lW: 9.8905 - Act
: QF
loc: (4, 4, 0, 10) - act: (1, 1, -2, 0) - rew: 4.8 - Delta: 3.34 - Diff: -1.46 - n-qval: 10.45 - o-qval: 3.55 - uW: 2.1362 - lW: 11.7229 - A
ct: QF
loc: (3, 4, 0, 10) - act: (1, 1, -2, 0) - rew: 5.36 - Delta: 4.1 - Diff: -1.26 - n-qval: 11.26 - o-qval: 3.51 - uW: 2.6251 - lW: 13.6972 - A
ct: QF
```

Figure 3.11: Log file of the fist implementation.

the location (4,8,2,4) with a reward value of 7.38. However, the average weight updates value ("update weight" in the log file) remained relatively low at 0.3715, and the Delta was only 0.58. Analyzing Figure 3.14, it's evident that this state offers a significant reward. Yet, the difference between the current Q-value and the next state Q-value is minimal, indicating that this region might not provide substantial future returns. The local weights in this area average around 8.5729, suggesting that the peak of 7.38 isn't a significant advancement, and it doesn't drastically impact the weight. Conversely, when the agent encounters the location (9, 5, 2, 4), it discovers a new region with Q-values around 7-8. To enhance the weight of this new region, both the Delta and update weights increased by roughly 3 units. A similar trend is observed at the location (10, 6, 0, 0), which has an even higher result, causing a sharp rise in the region's weight by nearly 3.2 and a Delta of 5.1. The final location represents an optimal blend of a high reward and anticipated return, resulting in a Delta of 4.32 and a weight update of 2.7628.

Upon further examination of the plots, it's evident that after 28 iterations, the agent successfully identified regions with substantial rewards and consistently monitored these areas. The values of rewards, Delta, and update weights not only remained high but also showed an increasing trend. This behavior aligns perfectly with the primary objective of the project: for the agent to pinpoint and persistently track regions with high reward potentials and to explore further promising areas. During its exploration, there are instances, such as at location (9,5,2,4), where the agent experiences a dip in rewards after some advancements. However, it's crucial for the agent to navigate away from these less rewarding areas and refocus on the more potential regions.

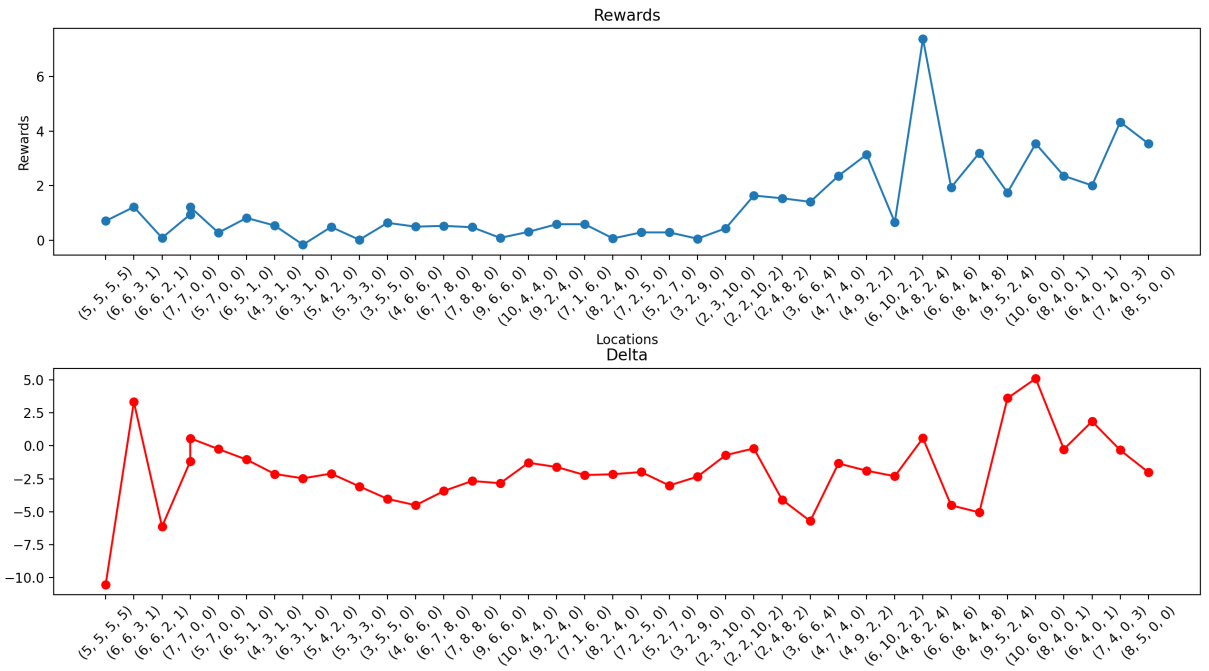


Figure 3.12: Rewards and Delta plots of the second implementation.

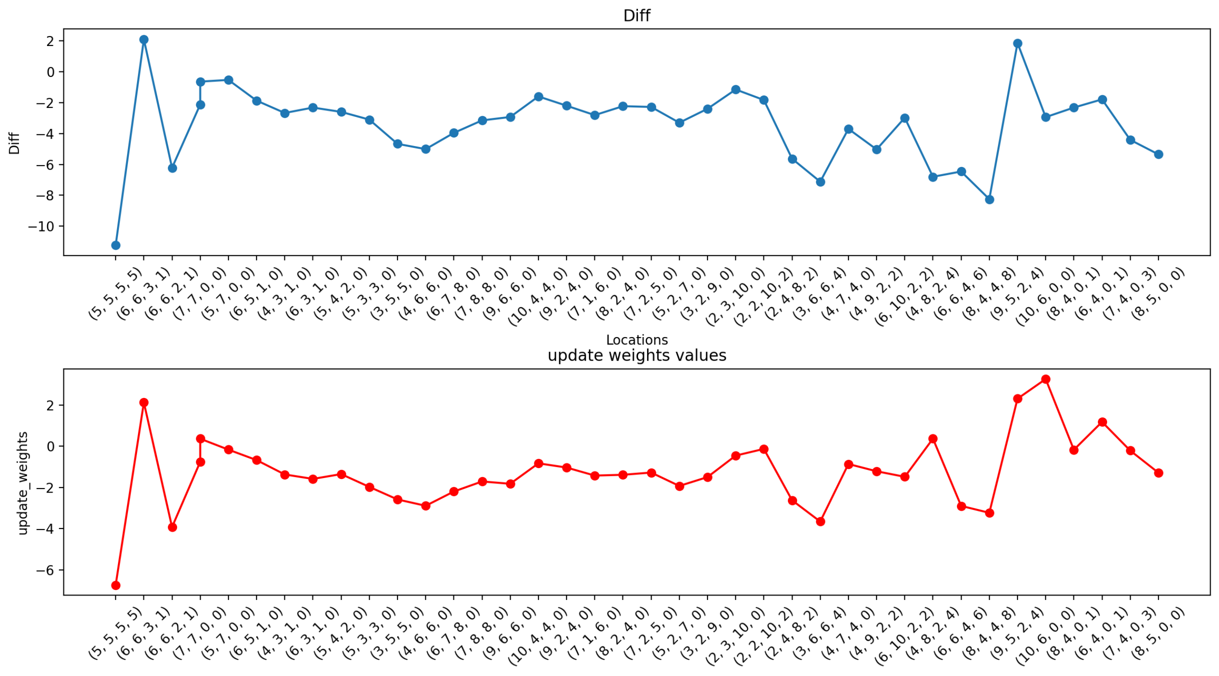
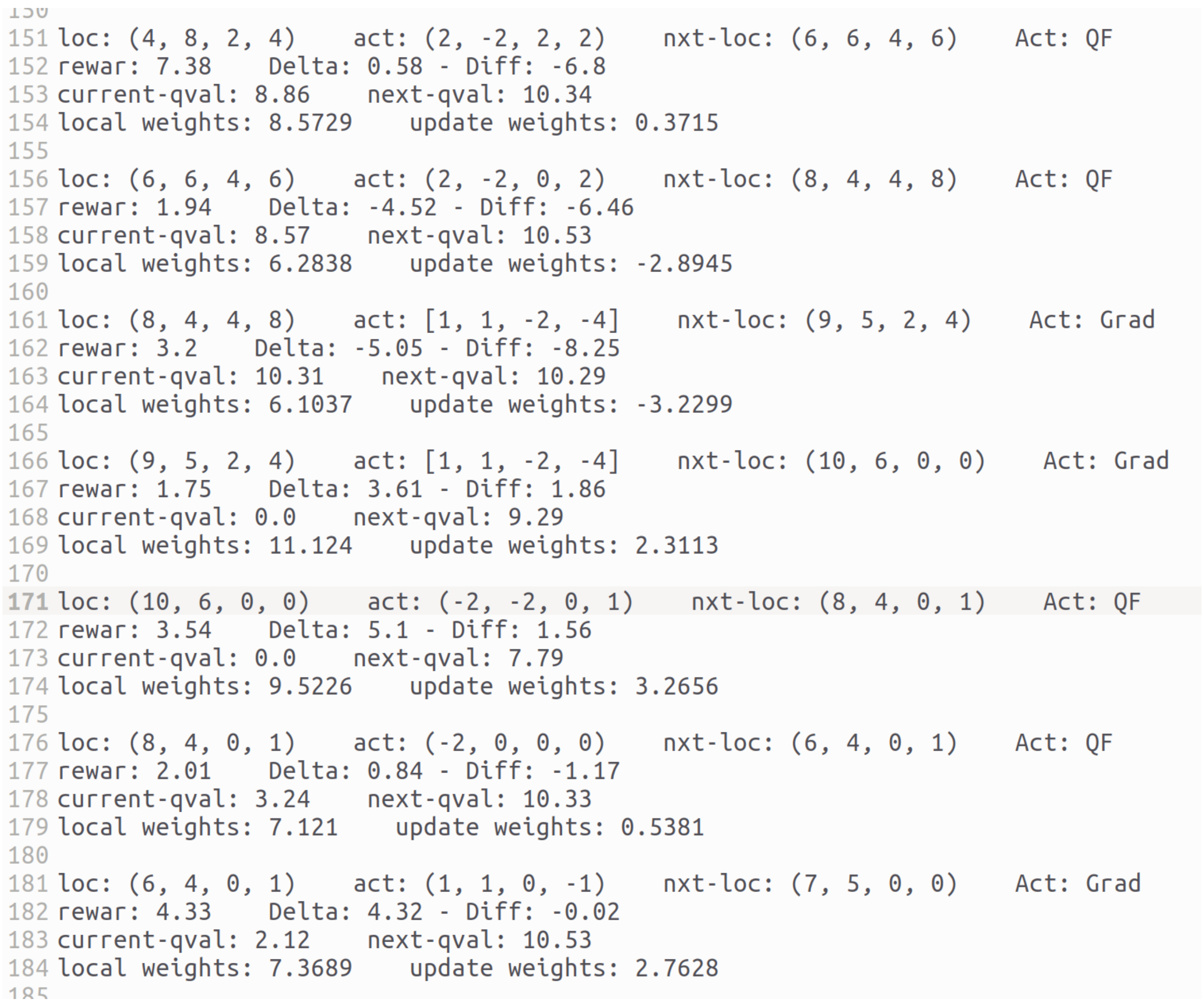


Figure 3.13: Diff and Updated weights values of the second implementation.



```
loc: (4, 8, 2, 4)    act: (2, -2, 2, 2)    nxt-loc: (6, 6, 4, 6)    Act: QF
rewar: 7.38    Delta: 0.58 - Diff: -6.8
current-qval: 8.86    next-qval: 10.34
local weights: 8.5729    update weights: 0.3715

loc: (6, 6, 4, 6)    act: (2, -2, 0, 2)    nxt-loc: (8, 4, 4, 8)    Act: QF
rewar: 1.94    Delta: -4.52 - Diff: -6.46
current-qval: 8.57    next-qval: 10.53
local weights: 6.2838    update weights: -2.8945

loc: (8, 4, 4, 8)    act: [1, 1, -2, -4]    nxt-loc: (9, 5, 2, 4)    Act: Grad
rewar: 3.2    Delta: -5.05 - Diff: -8.25
current-qval: 10.31    next-qval: 10.29
local weights: 6.1037    update weights: -3.2299

loc: (9, 5, 2, 4)    act: [1, 1, -2, -4]    nxt-loc: (10, 6, 0, 0)    Act: Grad
rewar: 1.75    Delta: 3.61 - Diff: 1.86
current-qval: 0.0    next-qval: 9.29
local weights: 11.124    update weights: 2.3113

loc: (10, 6, 0, 0)    act: (-2, -2, 0, 1)    nxt-loc: (8, 4, 0, 1)    Act: QF
rewar: 3.54    Delta: 5.1 - Diff: 1.56
current-qval: 0.0    next-qval: 7.79
local weights: 9.5226    update weights: 3.2656

loc: (8, 4, 0, 1)    act: (-2, 0, 0, 0)    nxt-loc: (6, 4, 0, 1)    Act: QF
rewar: 2.01    Delta: 0.84 - Diff: -1.17
current-qval: 3.24    next-qval: 10.33
local weights: 7.121    update weights: 0.5381

loc: (6, 4, 0, 1)    act: (1, 1, 0, -1)    nxt-loc: (7, 5, 0, 0)    Act: Grad
rewar: 4.33    Delta: 4.32 - Diff: -0.02
current-qval: 2.12    next-qval: 10.53
local weights: 7.3689    update weights: 2.7628
```

Figure 3.14: Log file of the second implementation.

4 | Chapter four: Prospective Developments

4.1. Utilizing Neural Networks in RL for Parameter Optimization [10] [21]

Utilizing Neural Network (NN) in RL for parameter optimization presents a promising avenue for future development of this foundational implementation. This integration leverages the pattern recognition capabilities of NN to identify complex relationships within the parameter space of the FF Equation in MD. By approximating functions essential in RL, such as the value function or policy, NN can guide the exploration towards more promising regions, reducing search time and computational resources.
One specific way to enhance this approach is by extrapolating the long-term equilibrium values using time series data from previous simulations. By incorporating these time series, the neural network can better understand the temporal dynamics and trends within the parameter space. This is especially valuable when considering the energy differentials between individual Hamiltonians. Since each state in the RL framework parametrizes the Hamiltonian slightly differently, understanding these differentials can provide deeper insights into the system's behavior and potential optimizations.
The approach involves designing a tailored neural network architecture that captures the underlying relationships between parameters and outcomes. This architecture should be capable of processing both the current state data and historical time series to make more informed decisions. Coupled with an appropriate RL algorithm like Q-learning or Deep Q-Networks, the system takes the current state, including FF parameters, observables (like the alpha-helix structure of the system), and past simulation data as input. Actions represent possible modifications to the parameters, and the reward function evaluates the quality of these actions based on metrics like the distance between experimental and calculated data, as well as the consistency with historical trends.
In summary, the integration of Neural Networks with RL, especially when enhanced with

time series data and energy differentials, offers a robust and flexible solution for parameter optimization in this thesis. It combines the strengths of both fields to create an efficient and effective method that can handle high-dimensional parameter spaces. By considering the historical context and the nuanced differences between Hamiltonian states, this approach becomes a valuable addition to the prospective developments of the research.

4.2. Exploring Alternative RL Algorithms and Initialization Techniques for Enhanced Optimization: [3] [38]

Exploring alternative RL algorithms and initialization techniques represents a vital avenue for enhancing the optimization process within this project. Different RL algorithms offer unique strengths and weaknesses, and selecting the right approach can significantly impact the efficiency and effectiveness of the optimization.

Among the various RL algorithms, Deep Q-Networks (DQN) leverage deep neural networks to approximate the Q-function, allowing for more complex representations of the state-action space. However, DQN can be computationally expensive and sensitive to hyperparameter tuning, potentially leading to instability. Methods like Policy Gradients provide more flexibility in defining the policy but can suffer from high variance, making convergence slower and potentially less stable. Monte Carlo Tree Search (MCTS) builds a search tree and selects actions based on simulations, offering strategic exploration beneficial in navigating the complex parameter space. However, MCTS can be computationally intensive and may struggle with large action spaces, limiting its applicability.

An alternative and promising approach is model-based RL, which uses a model of the environment to simulate outcomes of various actions without actually taking them. Specifically, within the context of this research, leveraging Boltzmann-reweighted past experiences within the free energy perturbation framework can serve as an effective model. This method provides a way to gain more insights about the Q-function of prospective states by reusing past experiences, thereby reducing the computational cost. The advantage of this approach is that it can efficiently utilize the information from past simulations to guide the exploration of new states. Furthermore, with the increasing availability and power of GPUs, new simulations can run in parallel, further enhancing the efficiency and speed of the optimization process.

Proper initialization of the parameters is crucial for efficient convergence. Random initialization, though simple, can sometimes lead to slow convergence or getting stuck in local minima. Utilizing domain knowledge or heuristic methods to initialize parameters can provide a better starting point but may introduce biases that limit exploration of the entire parameter space. Techniques that leverage prior simulations or data to initialize the parameters, known as smart initialization, can further enhance the optimization process but may require substantial prior knowledge or data, potentially limiting their applicability in novel or less-studied systems.

Additionally, Neural Networks (NN) can be employed to infuse more knowledge into the initialization phase of the project. By training an NN on existing data or simulations, it can discern the underlying relationships and trends within the parameter space. This learned knowledge can then be used to initialize the parameters in a manner more aligned with the expected optimal solutions, potentially accelerating convergence and reducing the risk of suboptimal local minima. This integration of NN into the initialization process represents a sophisticated approach that combines data-driven insights with computational intelligence.

Another promising avenue for initialization involves the strategic deployment of multiple agents, preferably in parallel, but situated in different locations within the parameter space. This approach can be conceptualized as having two distinct agents: one exploration-oriented and the other exploitation-oriented. The exploration-oriented agent, with an "optimistic" weight matrix filled with high values, is designed to extensively probe and traverse the parameter space. In contrast, the exploitation-oriented agent, initialized with a "pessimistic" weight matrix populated with lower values, is inclined to meticulously exploit the region it identifies, seeking optimal solutions within that localized area. By juxtaposing the results from these distinct initializations, one can discern which approach yields superior outcomes. Furthermore, with adequate training, algorithms can be developed to predict which initialization strategy, or sequence of strategies, would be most efficacious for a specific system.

This multi-agent initialization strategy, emphasizing both exploration and exploitation, offers a comprehensive approach to parameter optimization. It not only harnesses the strengths of both exploration and exploitation but also provides a robust framework to compare, contrast, and eventually converge to the most optimal solutions in diverse systems.

In conclusion, exploring alternative RL algorithms and initialization techniques offers a multifaceted approach to enhance the optimization of the FF Equation in MD. While each method presents unique advantages, they also come with specific challenges and drawbacks that must be carefully considered.

4.3. Incorporating Additional Observables into the Reward Function for a Comprehensive Evaluation: [4]

In the context of optimizing the FF Equation in MD, the reward function plays a pivotal role in guiding the learning process. Traditionally, focusing on specific observables like the alpha-helix structure has been the norm. However, this approach may lead to a narrow optimization that overlooks other essential aspects of the system.

Incorporating additional observables into the reward function can provide a more comprehensive evaluation of the system's behavior. Here are some key observables:

- Beta-Sheet Structure: Including beta-sheet structure ensures that the optimization does not overly favor the alpha-helix structure, leading to a more balanced understanding of protein folding. The challenge here is to accurately represent this structure within the reward function without overshadowing other features.
- Energy Landscape: This provides insights into the stability and dynamics of the molecular system, aligning the optimization process with the underlying physics. The complexity of accurately modeling the energy landscape can be a challenge, requiring careful consideration of various energy terms.
- Solvent Accessibility: Reflecting how the molecule interacts with its environment, solvent accessibility can be crucial in understanding molecular behavior. The challenge lies in accurately modeling solvent effects, which can be computationally demanding.
- Hydrogen Bonding Patterns: These provide detailed insights into molecular conformation and stability. Including hydrogen bonds requires careful handling of distance and angle parameters, adding complexity to the model.
- Torsional Angles: Torsional angles offer a detailed view of molecular geometry. The challenge here is the high dimensionality and the potential for increased computational complexity.
- RMSD (Root Mean Square Deviation): RMSD measures the structural difference between the predicted and actual structures, providing a direct measure of accuracy. The challenge is to incorporate RMSD in a way that aligns with other observables without dominating the reward function.
- Electrostatic Interactions: These interactions play a vital role in molecular behavior.

Accurately modeling them requires consideration of charges and dielectric effects, adding complexity to the optimization process.

Incorporating these diverse observables can lead to a more realistic and nuanced understanding of the molecular system. However, balancing multiple observables to ensure that one does not dominate the others can lead to unintended biases. The increased complexity of the reward function may require more sophisticated algorithms and more computational resources.

4.4. Adaptive Policy Modification for Enhanced Convergence and Performance [37]

In the optimization of the FF Equation in MD, the policy that guides the exploration and exploitation of the parameter space plays a critical role in determining the efficiency and effectiveness of the learning process. Traditional policies often rely on fixed rules or heuristics, which may not adapt to the evolving understanding of the parameter space. This can lead to suboptimal performance, slow convergence, or even convergence to suboptimal solutions.

Adaptive Policy Modification represents a dynamic approach that adjusts the policy in real-time based on the ongoing learning process. By continuously monitoring the performance, convergence rate, and other relevant metrics, the policy can be modified to better align with the current state of the optimization.

However, implementing Adaptive Policy Modification is not without challenges. Designing a policy that can effectively adapt requires a deep understanding of the problem and the learning process. It may require sophisticated algorithms that can recognize patterns, assess the reliability of solutions, and make informed decisions about how to modify the policy. The risk of overfitting to specific situations or oscillating between different policy settings is also a concern that must be carefully managed.

4.5. Incorporating Multi-Objective Optimization for a Holistic Approach: [31]

In the context of optimizing the FF Equation in MD, focusing on a single objective, such as minimizing the error between experimental and calculated data, has been a common approach. While this can lead to effective solutions for that specific objective, it may overlook other important aspects of the system, potentially resulting in a narrow or unbalanced optimization.

Incorporating Multi-Objective Optimization represents a shift towards a more holistic approach that considers multiple objectives simultaneously. This can include objectives related to accuracy, stability, computational efficiency, robustness, and other relevant factors. By optimizing across these multiple objectives, the process can achieve a more balanced and comprehensive understanding of the system.

For example, in addition to minimizing the error between experimental and calculated data, a multi-objective approach might also aim to minimize the computational resources required or maximize the stability of the solutions. This can lead to solutions that are not only accurate but also efficient and robust, providing a more well-rounded optimization.

The challenge in Multi-Objective Optimization lies in the inherent trade-offs between different objectives. Improving performance in one area may lead to compromises in others. Designing an optimization process that can navigate these trade-offs requires careful consideration of the relative importance of different objectives, the relationships between them, and the desired outcomes of the optimization.

Techniques such as Pareto optimization play a pivotal role in multi-objective optimization problems. Pareto optimization is rooted in the concept of Pareto dominance, where a solution is considered dominant if it performs better or equally well in all objectives and strictly better in at least one objective compared to another solution. In essence, Pareto optimization seeks to identify solutions that are non-dominated with respect to the entire solution set.

This involves identifying a set of solutions, known as the Pareto optimal set, where improving one objective would require compromising another. Each solution in this set represents a unique combination of trade-offs among the objectives, ensuring that no single solution is universally superior to the others. Visualization tools, such as Pareto front plots, can be instrumental in understanding and selecting among these trade-offs. The Pareto front is a graphical representation of the Pareto optimal set in the objective space, and it provides a clear picture of the relationship between different objectives and the trade-offs involved.

However, implementing Multi-Objective Optimization, especially with Pareto-based methods, can be complex and computationally demanding. The increased dimensionality of the optimization space, the need to evaluate multiple objectives simultaneously, and the intricacy of navigating the trade-offs all add to the challenge. Moreover, as the number of objectives increases, the size of the Pareto front can grow exponentially, further complicating the optimization process.

4.6. Utilizing Ensemble Learning for Robust Parameter Optimization: [9]

In the field of optimizing the FF Equation in MD, achieving robust and reliable parameter optimization is a complex task. Traditional methods may rely on a single model or algorithm, which, while effective in certain scenarios, might be sensitive to the specific characteristics of the data or the initial conditions of the optimization.

Utilizing Ensemble Learning offers a pathway to more robust parameter optimization by combining the predictions or decisions of multiple models or algorithms. Instead of relying on a single approach, ensemble learning leverages the strengths of various methods, potentially reducing the weaknesses and biases of individual models.

Ensemble Learning can be implemented in several ways, and in our specific problem of multiple agents with different initializations, we can leverage this technique effectively. For instance, we can apply different machine learning algorithms or models to each agent, each starting with a distinct set of initial values. These agents can then work on existing data and generate individual predictions or solutions. To further enhance our results, we can create agent committees that quantify uncertainty using the same data but with different initial values for each agent. This ensemble approach allows us to leverage the diversity of our agents and their unique perspectives to improve the overall performance of our system.

The benefits of Ensemble Learning include increased robustness to noise and outliers, reduced risk of overfitting, and often improved overall performance. By combining multiple perspectives on the problem, ensemble learning can achieve a more balanced and nuanced optimization, potentially leading to more accurate and reliable solutions.

However, implementing Ensemble Learning is not without challenges. The complexity of integrating multiple models or algorithms requires careful design and coordination. Decisions about which models to include, how to combine their results, and how to manage the increased computational demands must be made with consideration of the specific problem and goals of the optimization.

4.7. Investigating Alternative and Challenging Parameters in FF Equations

While the challenge of determining specific parameters can depend on the particular system and the force field (FF) model being used, the following three parameters are often considered among the most challenging to calculate and could potentially benefit significantly from machine learning-based optimization techniques:

- **Polarization Parameters**: Polarization parameters describe how the electron cloud of an atom is distorted by its surroundings. Accurate modeling might require complex calculations, and AI could provide a way to optimize these parameters. In mathematical terms, this might involve parameters related to the polarizability tensor, often denoted by α.
- **Charge Parameters**: Determining partial atomic charges (q_i) that represent electrostatic interactions within a molecule is a complex task. AI could be used to optimize these charges by learning from quantum mechanical calculations or empirical data.
- **Torsional Parameters**: Torsional angles describe the energy associated with the rotation around single bonds. Accurately modeling these parameters (ϕ, ψ, ω, etc.) can be complex, and AI could be used to learn the underlying potential energy surface, providing a way to optimize these parameters more accurately.

These parameters are considered challenging due to the complexity of the underlying physics and chemistry, the potential need for high-level quantum mechanical calculations, and the sensitivity of the results to the specific modeling choices. AI offers the potential to navigate these challenges by leveraging its ability to learn from complex data, recognize patterns, and optimize high-dimensional spaces. By applying AI to these challenging parameters, it might be possible to achieve more accurate and robust FF models, opening new possibilities for understanding and manipulating molecular systems.

4.8. Enhancing Scalability through Parallelization and Distributed Computing: [27]

In the field of MD, particularly in the optimization of the FF Equation, computational demands can be substantial. Simulating complex molecular systems, exploring high-dimensional parameter spaces, and employing sophisticated optimization algorithms can require significant computational resources. Traditional serial computing approaches, where calculations are performed sequentially, may be inadequate for large-scale or highly detailed simulations.

Enhancing scalability through parallelization and distributed computing represents a pathway to overcome these computational challenges. By leveraging the power of modern parallel and distributed computing architectures, this approach seeks to dramatically increase the computational capacity and efficiency of the optimization process.

- Parallelization: Parallel computing involves dividing a problem into subproblems that can be solved simultaneously by multiple processors. In the context of MD, this might involve parallelizing the calculations for different parts of a molecule or different steps in an optimization algorithm. By performing these calculations simultaneously, parallelization can significantly reduce the computational time required.
- Distributed Computing: Distributed computing takes parallelization a step further by distributing the computations across multiple computers or clusters. This can enable even larger-scale simulations or optimizations by harnessing the combined computational power of many machines. Distributed computing can be particularly valuable for exploring vast parameter spaces or simulating large and complex molecular systems.

The benefits of Enhancing Scalability through Parallelization and Distributed Computing include the ability to tackle larger and more complex problems, reduced computational time, and the potential for more detailed and accurate simulations. This approach can enable research and optimization that might be infeasible with traditional computing methods.

However, implementing parallelization and distributed computing is not without challenges. Designing algorithms that can effectively divide a problem into parallelizable subproblems requires careful consideration of dependencies, communication overhead, and load balancing. Ensuring that the parallel or distributed computations are coordinated and that the results are correctly integrated can be complex. Additionally, specialized hardware, software, and expertise may be required.

4.9. Conclusion

This chapter has explored several prospective developments with the potential to enhance the research presented in this thesis within the practical scope of our Master's project. These avenues include the integration of Neural Networks and RL into our project as well as the consideration of techniques such as parallelization and distributed computing.

These selected approaches offer innovative solutions to address complex challenges in optimizing the Force Field Equation within the context of Molecular Dynamics.

Each of these developments provides unique advantages and opportunities, but we must carefully weigh the benefits against the constraints of our current project. The application of these strategies should be considered in a way that aligns with the specific needs and goals of our research while remaining feasible within the scope of a Master's project. By doing so, we can aim for more robust, efficient, and insightful solutions tailored to our project's objectives.

It's important to acknowledge that while these prospective developments offer promising pathways, they may also entail significant complexity and resource requirements. Therefore, we will need to assess their suitability for our project and make informed decisions to ensure that our research remains both achievable and impactful within the confines of our Master's program.

As the field of Molecular Dynamics continues to evolve, these selected prospective developments provide opportunities for us to advance our understanding and capabilities while maintaining a realistic and practical approach that aligns with the goals of our Master's project.

Bibliography

[1] D. B. K. H. Anita Rácz, Levente M. Mihalovits and R. A. Miranda-Quintana. Molecular dynamics simulations and diversity selection by extended continuous similarity indices. 2022.

[2] P. Baldi. Gradient descent learning algorithm overview: a general dynamical systems perspective. 1995.

[3] F. C. F. W. Caarls. Parameters tuning and optimization for reinforcement learning algorithms using evolutionary computing. 2018.

[4] A. A. . T. K. Chikako T Nakazawa. Structural studies of amphiphilic oligopeptides composed of alternating alanine and ionizable amino-acid residues using cd and 13c cp/mas nmr spectroscopy. 2012.

[5] W. C. S. Z. D. C. a. Y. W. Chuanlei Zhang, Minda Yao. Gradient descent optimization in deep learning model training based on multistage and method combination strategy. 2021.

[6] J. Clifton and E. Laber. Q-learning: Theory and applications, 2020.

[7] . H. B. M. . D. L. M. . J. I. M. . S. P. . Efrem Braun, 1 Justin Gilmer and D. M. Zuckerman. Best practices for foundations in molecular simulations. 2018.

[8] S. N. . R. R. Encyclopedia. Time constant. URL `https://academic-accelerator.com/encyclopedia/time-constant`.

[9] D. T. R. J. Florian Wenzel, Jasper Snoek. Hyperparameter ensembles for robustness and uncertainty quantification. 2022.

[10] F. M. T. S. A. Gamel2. Rl based hyper-parameters optimization algorithm (roa) for convolutional neural network. 2021.

[11] N. H. B. C. Gentile. Learning theory, 2007.

[12] G. A. T. Giovanni Bussi. *Analyzing and Biasing Simulations with PLUMED*. 2019.

[13] M. González. Force fields and molecular dynamics simulations. 2011.

[14] Z. F. G. G. T. N. B. H. W. I. N. S. Helen M. Berman, a John Westbrook and P. E. Bourne1. The protein data bank. 2000.

[15] Y. S. S. L. L. hong LiGui-fang Zhang. An efficient initialization approach of q-learning for mobile robots. 2012.

[16] D. T. J. Joe G. Greener. Determining force field parameters using a physically based equation of state. *PLOS, Public Library of Science*, 2021.

[17] R. R. G. Jung C. Lee. Helix capping in rna structure. *Commun. ACM*, 4 2014.

[18] G. H. J¨urgen K¨ofingera, 1. Empirical optimization of molecular simulation force fields by bayesian inference. 2021.

[19] M. M. S. K. S. Shefov. Sensitivity analysis in a problem of reaxff molecular-dynamic force field optimization. 2018.

[20] J. K¨ofinger and G. Hummer1. Empirical optimization of molecular simulation force fields by bayesian inference. 12 2021.

[21] M. O. Mariam Kiran. Hyperparameter tuning for deep reinforcement learning applications. 2022.

[22] V. B. J. C. A. B. T. L. B. Martin Bartas, Kristyna Slychko and P. Pečinka. Searching for new z-dna/z-rna binding proteins based on structural similarity to experimentally validated z domain. 2022.

[23] R. Z. A. R. A. H. R. Martin Roesch *, Christian Linder. Smart grid for industry using multi-agent reinforcement learning. 2020.

[24] A. K. M. Martin Stroet, Katarzyna B. Koziara and A. E. Mark*. Optimization of empirical force fields by parameter space mapping: A single-step perturbation approach. 2017.

[25] V. A. C. H. D. M. D. E. G. C. J. L. M. d. A. L. M. R. K. R. L. Mayk C. Ramos, Patrick K. Quoika and B. A. C. Horta*. pypolybuilder: Automated preparation of molecular topologies andinitial configurations for molecular dynamics simulations ofarbitrary supramolecules. 2021.

[26] F. S. Melo and M. I. Ribeiro. *Learning Theory*, chapter Q-Learning with Linear Function Approximation, pages 319–333. Springer-Verlag Berlin Heidelberg, San Diego, CA, USA, 2007.

[27] M. H. A. S. J. H. Milan Vaško1, *. Parallelization of computational algorithms for optimization problems with high time consumption. 2018.

[28] H. N1 and P. A. G1. A brief study of deep reinforcement learning with epsilon-greedy exploration. 2021.

[29] M. T. C. Omar. Data analysis of molecular dynamics simulation trajectories of -sitosterol, sonidegib and cholesterol in smoothened protein with the charmm36 force field. 2020.

[30] P. J. Ratna Sandeep Katiyar. Molecular simulations in drug delivery: Opportunities and challenges. 2018.

[31] A. A. R.-P. a. N.-R. Sandra C. Cerda-Flores. Applications of multi-objective optimization to industrial processes: A literature review. 2022.

[32] S. L. S. M.-Y.-M. K. K. H. C. S.-Y. K. O. Y. K. C. N. Y. Y. K. K. J. H. Y. K.-Y. N. S.-G. K. Y. I. H. H. C. W. E. A. J. J. A. G. K. D. G. M. S. J. H. A. S. Sung Bo Hwang, Chang Joon Lee and K. T. No. Pmff: Development of a physics-based molecular force field for protein simulation and ligand docking. 2020.

[33] R. S. Sutton and A. G. Barto. Reinforcement learning, an introduction, 2018.

[34] L. P. Terence Tang. Recognition of poly(a) rna through its intrinsic helical structure. 2020.

[35] Thijs van Westen, † Thijs J. H. Vlugt and J. Gross*. Determining force field parameters using a physically based equation of state. *The Journal of Physical Chemistry B*, 2011.

[36] Q. Vicens and J. S. Kieft. Thoughts on how to think (and talk) about rna structure. 2022.

[37] J. C. Warren E. Walker, S.Adnan Rahman. Adaptive policies, policy analysis, and policy-making. 2001.

[38] N. A. a.-Y. Q. S. Xi Min Zhang a, Yan Qiu Chen b. Mini-max initialization for function approximation. 2004.

List of Figures

List of Tables