BComp Dissertation

Machine Learning Methods in Biomolecule Analysis

By

Tran Hoang Bao Linh

Department of Computer Science

School of Computing

National University of Singapore

2017/2018

BComp Dissertation

Machine Learning Methods in Biomolecule Analysis

By

Tran Hoang Bao Linh - A0112184R

Department of Computer Science

School of Computing

National University of Singapore

2017/2018

Project No: H162030

Advisor: Associate Professor Lee Hwee Kuan [1]

[1]Bioinformatics Institute, ASTAR

Abstract

Understanding the protein folding process and how mutations can affect a chain's native state is important for studying and combating diseases effectively. The main challenge to protein folding research is the long time it takes for simulation models to produce a folded protein chain. It can take up to $10,000,000$ steps for a Monte Carlo simulation to reach the native state of a peptide of 30 atoms. However, some native state characteristics can be inferred from short time relaxation properties. We explore one method to predict the root-mean-squared aligned distance between the native conformations of the mutant and the original chain, based on the correlations between individual atoms after only 100 Monte Carlo steps.

Key words:

Atom chain, folding process, short-term dynamics, correlation matrix, aligned distance, convolutional neural network

Implementation Softwares:

Intel®Distribution for Python (3.6.2), Python 2.7.12, Tensorflow 1.4 CPU, Intel®MPI 5.1.2, National Super Computing Clusters (NSCC)

Acknowledgement

This dissertation would not be possible without the invaluable assistance from various people, both known and unknown to me. I would like to express my gratitude to all of them, and extend my thank personally to those with special contributions.

First of all, I would like to thank my supervisor, Professor Lee Hwee Kuan, for this interesting and practically impactful project, which has stimulated my interest in bioinformatics and machine learning, and for various helpful insights and hints, without which, I would not have been able to complete the project to this extent. I would also like to thank his correspondents, especially Chandra, Daniel and Mustafa, for their assistance in formulating the problem, helpful ideas and advice, which have been direct or indirectly applied in this project.

Thirdly, I would like to thank various people from National University of Singapore (NUS), Immigration Checkpoint for Authority, Tan Tock Seng General Hospital (TTSH) and Clementi Polyclinic (CP) who have helped me overcome tuberculosis and complete my degree. Especially, I extend a special thank to: Ms. Agnes Yuen, student care manager at the Office of Student Affairs, for her crucial assistance and mental support during my struggle to renew my student's pass; my three doctors at TTSH and staff at CP for treating my disease and looking after my health; the Student's Pass officer at ICA for their approval and issuance of my renewed pass.

Next, I would like to express my gratitude to the Principal of NUS, the School of Computing Dean, and other administrative staff who have worked in the background to facilitate the Dissertation program as a platform for senior students, including me, to culminate their academic years into a highly specialized and extensive work.

Furthermore, I also express my deep gratitude to the professors, lecturers and tutors who have shared their knowledge with me and helped train me in my mathematical background, as well as my academic attitude.

Finally, I would like to thank my family, my friends and my past teachers, who have always been by my side and lent support in challenging times, and without whom, I would not be able to achieve a place at NUS to start the journey of my dream. I would like to extend my most special and deepest thank, as well as dedicate this work to my parents, whom have been constantly supporting and educating me, even after I went to study abroad. I also express my warmest gratitude to my sister and close friends, who have been continually providing physical and mental support during my time in Singapore.

Contents

1 Introduction

1.1 Protein Folding

Protein are large biomolecules, formed from chains of amino acid molecules. Thus, proteins are polymers and their monomers are amino acids. The monomers are also called residues. Each pair of consecutive residues are linked by a peptid bond [2] [3]. Each protein type is synthesized in the cells using information in the gene responsible for encoding them. Short proteins are also generated in the laboratory using various organic synthesis techniques [4].
When first generated, a protein chain is unstable and must go through a folding process until it reaches a stable state, which we denote as the **native state** (or native conformation). The folding stage is also called the *conformational process*, during which the changes to the protein's structure are called *conformational changes*. The equilibrium is usually called the *native conformation*. This stable state is the structure in which the chain is able to perform its intended biological functionalities [3]. By the Principle of Minimal Frustration [5], the native conformation state is reached when the total energy of the chain, as a physical particle system, reaches its minimum. This principle has allowed for the discoveries and rapid development of computerized protein folding simulation models.

1.2 Motivation and Goals

1.2.1 Motivations

Proteins play a vital role to our body. A large portion of biophysical functions within organisms are performed by different types of proteins. These functions include including catalysing metabolic reactions, DNA replication, responding to stimuli, and transporting molecules from one location to another [3]. Almost every process within cells involve one to many types of proteins.
As pointed out in [2] [3], it is possible for a protein chain from one type to mutate into a different type. This often renders the chain unable to perform its intended functionalities, and in some cases, exhibits harmful behaviors. Mutant proteins has been known to cause mutations in cells, which produce cancer cells within human body. The topic of protein mutations is therefore of high application potential and worth studying.
Our approach to study protein chain mutation is through the protein **short-term molecular dynamics**. The protein's molecular dynamics describes how the atoms in the chain interact with each other during the folding process. In this approach, the chain is viewed a system of particles interacting with each other. In reality, the chain

can contains thousands of residues, each affects one another through various physical forces. Precisely modelling and performing computations on such a complex system is impractical. Therefore we employ a simplified model, which is described in Simplified Protein Model. Our goal is to study the differences in the short-term dynamics of a normal protein chain and its mutant version. Precisely, below are the questions we aim to answer:

1. Given a protein chain and its mutant version, how can we predict the differences when they reach equilibrium based on their short term dynamics?
2. Given a mutated protein chain, can we tell whether the mutation affect its native state heavily based on its short term dynamics?

1.2.2 Objectives

In order to address the questions posed in Section 1.2.1, we set up some objectives, which serve as intermediate steps to find the answer.

1. Propose a sufficiently simple model for the protein sequence and an appropriate set of parameters to efficiently simulate the folding mechanism while maintaining close resemblance to reality.
2. Propose a method to measure the differences in short-term dynamics of two protein chains of the same length in an accurate and meaningful way.
3. Determine some analyzable properties and propose a conscise method to analyze and compare the long-term dynamics differences based on the short-term dynamics differences.

1.3 Solution Overview

In Section 2, we attempt the first object by giving models for the protein chain and relevant concepts. In Section 3, we formulate the problem by specifying exactly what we aim to predict and what we are given. Finally, in Section 5, we discuss our method to solving the problem with Machine Learning models.

2 Model and Simulations

In this section, we describe our models for the protein chain, the folding process and the short-term molecular dynamics. We will provide the definitions and describe the algorithms where necessary.

2.1 Parameters Overview

Firstly, we define the parameters that will be used in the model. Their values have significance influence on the native state (Section 2.3) and the short-term dynamics (Section 2.4). We will discuss the most impactful parameters in Parameter Selection.

- L - The original length of spring between two consecutive atoms. It is also distance between any pair of consecutive atoms in the original state of the chain, just after being produced. **Note**: We use a straight, evenly spaced chain for the original conformation, so the consecutive distances are the same and all equal to L [1].

- K - The spring constant for the spring link between consecutive atoms. Note that L and K together form the expression for the spring energy of the chain [1].

- σ - The atom radius. It is the radius of each individual residue in the chain. In our simplified model, all residues are identical atoms (but with different charges), so they share the same value for σ [1].

- ϵ - The Lennard-Jones constant, which is also called the Lennard-Jones potential strength, or *potential well*. Note that σ and ϵ are used in the expression for the Lennard-Jones potential energy of the chain.

- T - The temperature at which the folding process occurs, measured in Kelvin. In our simulation model, we assume that this temperature remains unchanged throughout the process.

- K_B - The Boltzmann constant. Note that T and K_B together determine the stability of the chain and the rate at which it converges to its native state.

- R_{mc} - The range of variation for the chosen atom at every Monte Carlo step. Each coordinate of the atom is allow to vary uniformly in a radius R_{mc} when chosen.

- Q - The positive charge used to configure the charge sequence of the chain. We will define it in more details in Section 2.3 and discuss its impact in Parameter Selection.

2.2 Simplified Protein Model

In this section, we describe in details our model of the protein chain folding process. As mentioned in Introduction, a typical chain in the real world can contain up to a few thousands atoms, which affect each other through various physical forces. To accurately account for all the forces involved, one would have to solve thousands of Newtonian motion equations. Solving such a large system either rigourously or numerically proves impractical given the amount of time and computational resources available to us. Therefore, we employ a simplified model [1], which focuses on the energy landscape of the chain rather than forces among the atoms. Note that this approach still incorporates the interactions between atoms, which will be expressed in terms of energy quantities. Furthermore, we eliminate the energies whose influence are numerically superficial, leaving only 3 types of potential energy to consider.

2.2.1 The Protein Chain

A simplified protein chain consists is in fact, a pair of two sequences: a *charge* sequence and an atom sequence.

$$P = [A, C] = \left[\{a_i\}_{i=0}^{n-1}, \{q_i\}_{i=0}^{n-1} \right] \tag{1}$$

where each q_i is a scalar value representing an electrical charge, and each a_i is a tuple of three real coordinates representing the atom's spatial position.

$$a_i = (x_i, y_i, z_i) \quad \forall i = 1, 2, \cdots, n$$

We define the *distance* between the i^{th} and j^{th} atoms, denoted by d_{ij}, as the Euclidean 3D distance between them:

$$d_{ij} = \|a_i - a_j\|_2 = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} \tag{2}$$

In this project, as shown later, we are concerned with one of the simplest and shortest types of proteins, which has 30 atoms. Such a short type chain is usually called a *peptide* chain. It then corresponds to $n = 30$ in our model.

2.2.2 The Chain's Energy

After eliminating forces with negligible effects, we are left with 3 types of energy, the spring, Lennard-Jones and electrica; potential energy.
The total energy of the chain can be computed approximately using the following Hamiltonian expression [1]:

$$H(P) = K \sum_{i=0}^{n-2} (d_{i(i+1)} - L)^2 + 4\varepsilon \sum_{i<j<n} \left[\left(\frac{\sigma}{d_{ij}} \right)^{12} - \left(\frac{\sigma}{d_{ij}} \right)^{6} \right] + \sum_{i<j<n} \frac{q_i q_j}{d_{ij}} \tag{3}$$

Or

$$H = H_{sp} + H_{Len} + H_{elec} \tag{4}$$

where

$$\begin{aligned} H_{sp} &= K \sum_{i=0}^{n-2} (d_{ii+1} - L)^2 \\ H_{Len} &= 4\varepsilon \sum_{i<j<n} \left[\left(\frac{\sigma}{d_{ij}} \right)^{12} - \left(\frac{\sigma}{d_{ij}} \right)^{6} \right] \\ H_{elec} &= \sum_{i<j<n} \frac{q_i q_j}{d_{ij}} \end{aligned} \tag{5}$$

are the spring potential energy [1], Lennard-Jones potential energy [1], and electric potential energy [1], respectively.
Here the parameters K, L, ϵ, σ are all defined in Section 2.1.
Net, we will model the folding process, or more precisely, how we simulate this process.

2.3 Folding Process (Long Simulation)

As mentioned in Section 1.1, the equilibrium is a sufficiently small neighborhood around the native state, at which the energy level is near the minimum, and in which the chain remains as the folding process goes on. Hence, our goal when designing the simulation algorithm is to maintain the following conditions:

1. The total energy of the chain, as defined in (3), is reduced gradually during the simulation.

2. Once a sufficiently small energy level is reached, the chain has negligible probability of escaping a neighborhood around its current state. We can then declare this state the native state.

To solution to maintain both these properties is a Monte Carlo algorithm that adjusts the chain gradually, atom by atom. From an initial state, the system is updated at each Monte Carlo step in a fashion similar to the Metropolis algorithm [1][6], and finally get into an equilibrium after sufficiently many steps. We will describe in details this simulation and make use of the constants K_B and T in Section 2.1.
Below is the step-by-step description of the folding simulation process:

Algorithm 2.1 (Long Simulation)**.** Given a charge sequence $C_0 = \{q_0, q_2, \cdots, q_{n-1}\}$ and an initial chain $A_0 = \{a_0, a_2, \cdots, a_{n-1}\}$, we begin the following process, which will return an atom chain $\mathsf{REF}\,(A_0, C_0)$.
$\mathsf{REF}\,(A_0 : \text{atom chain},\ C_0 : \text{charge sequence})$:

1. Initially we start with a protein chain made from A_0 and C_0:

$$P_0 = [A_0, C_0] = \left[\{a_i\}_{i=0}^{n-1}, \{q_i\}_{i=0}^{n-1}\right]$$

2. Loop for $N_{mc} = 10^7$ steps, where at each step:

   2.1. Uniformly pick an $i \in \{0, 2, \cdots, n-1\}$, choose uniformly random $d_x, d_y, d_z \in [-R_{mc}, R_{mc}]$ where R_{mc} is a model parameter.

   2.2. Let
   $$a_i^* = a_i + (d_1, d_2, d_3) = (x_i + d_x, y_i + d_y, z_i + d_z)$$
   and let
   $$P^* = [(a_0, a_2 \cdots, a_{i-1}, a_i^*, a_{i+1}, \cdots, a_{n-1}), C_0]$$
   be a copy of P but a_i is replaced by a_i^*.

   2.3. Let H and H^* respectively be the total energy of P and P^*, computed using (3). Let
   $$p = \exp\left(-\frac{H^* - H}{K_B T}\right)$$
   with K_B, T being Boltzmann constant and temperature.

   - If $p > 1$, meaning $H^* < H$, update $P \leftarrow P^*$.
   - If $p \leq 1$, update $P \leftarrow P^*$ with probability p.

3. Assume the chain has reached equilibrium at this stage. Repeat Step 2 for another 10^6 times, randomly selecting $N_{sam} = 1000$ states to reserve in memory.

4. Apply centroid method, describe in The centroid method to these N_{sam} states to get the most representative equilibrium state $P_{ref} = [A_{ref}, C_{ref}]$, which we term the reference state.

5. Set $\mathsf{REF}\,(A_0, C_0) = C_{ref}$ and return $\mathsf{REF}\,(A_0, C_0)$.

2.4 Short Term Correlation (Short Simulation)

To discuss the short-term dynamics of the folding process, as mentioned, we use the correlation in spatial position between pairs of atoms in the chain. More precisely, our objects of interest will be the **correlation matrices** generated from repeated short simulations, taken at some specific Monte Carlo steps.

Remarks: It is necessary to repeat the short simulation procedure a large amount of times independently to obtain a sufficient set of samples, from which patterns can be

extracted. This set of samples need to be large enough so that the patterns found are statistically significant.
Before describing the short simulation procedure, it is necessary to properly formulate the term **correlation matrix** of a protein chain. We begin with the definition of the **correlation block** of two individual atoms:

Definition 2.2 (Correlation Block)**.** Given two atoms

$$a_1 = (x_1, y_1, z_1) \quad \text{and} \quad a_2 = (x_2, y_2, z_2)$$

moving in a process. This process is being repeated many times to positional data of the two atoms. The **correlation block** of a_1 and a_2 is a 3×3 matrix below:

$$C(a_1, a_2) = \begin{bmatrix} c_{xx} & c_{xy} & c_{xz} \\ c_{yx} & c_{yy} & c_{yz} \\ c_{zx} & c_{zy} & c_{zz} \end{bmatrix}$$

where each cell's value is the correlation coefficient between the corresponding coordinates, e.g. $c_{xx} = \mathsf{corr}\,(x_1, x_2)$, $c_{yz} = \mathsf{corr}\,(y_1, z_2)$. Note: Definition of the correlation coefficient can be found at Appendix A.1

This definition allows us to define the correlation matrix, which is basically a big correlation block of the flatten chain.

Definition 2.3 (Short-term Correlation Matrix)**.** Given a folding protein chain

$$P = [A, C] = \left[\{a_i\}_{i=0}^{n-1}, \{q_i\}_{i=0}^{n-1}\right]$$

whose state at timestep t is

$$P^t = \left[A^t, C^t\right] = \left[\{a_i^t\}_{i=0}^{n-1}, \{q_i^t\}_{i=0}^{n-1}\right]$$

The **correlation matrix** of P at time t is a $3n \times 3n$ matrix

$$C(P, t) = \begin{bmatrix} c_{00} & c_{01} & c_{02} & \cdots & c_{0(n-1)} \\ c_{10} & c_{11} & c_{12} & \cdots & c_{1(n-1)} \\ c_{20} & c_{21} & c_{22} & \cdots & c_{2(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ c_{(n-1)0} & c_{(n-1)1} & c_{(n-1)2} & \cdots & c_{(n-1)(n-1)} \end{bmatrix}$$

where for each $i, j = 0, 1, \cdots, n-1$, we have:

$$\begin{bmatrix} c_{(3i)(3j)} & c_{(3i)(3j+1)} & c_{(3i)(3j+2)} \\ c_{(3i+1)(3j)} & c_{(3i+1)(3j+1)} & c_{(3i+1)(3j+2)} \\ c_{(3i+2)(3j)} & c_{(3i+2)(3j+1)} & c_{(3i+2)(3j+2)} \end{bmatrix} = C(a_i^t, a_j^t)$$

Essentially, this matrix is just a pairwise correlation matrix constructed from the atom chain flatten:

$$\mathsf{Flatten}(A^t) = \{x_0^t, y_0^t, z_0^t, x_1^t, y_1^t, z_1^t, \cdots, x_{n-1}^t, y_{n-1}^t, z_{n-1}^t\}$$

Remarks. We can express $C(P)$ as in the following equivalent form:

$$C(P,t) = \begin{bmatrix} C(a_0^t, a_0^t) & C(a_0^t, a_1^t) & \cdots & C(a_0^t, a_{n-1}^t) \\ C(a_1^t, a_0^t) & C(a_1^t, a_1^t) & \cdots & C(a_1^t, a_{n-1}^t) \\ \vdots & \vdots & \vdots & \vdots \\ C(a_{n-1}^t, a_0^t) & C(a_{n-1}^t, a_1^t) & \cdots & C(a_{n-1}^t, a_{n-1}^t) \end{bmatrix}$$

Having defined clearly the correlation matrices, we can describe in details the short simulation algorithm below:

Algorithm 2.4 (Short Simulation and Correlation)**.** Given a charge sequence $C_0 = \{q_1, q_2, \cdots, q_n\}$, an atom chain $A_0 = \{a_0, a_1, \cdots, a_n\}$, and a timestep $t \in \mathbb{N}$, we begin the following process, which will return a $20 \times 3n \times 3n$ 20-fold correlation matrix $\mathsf{CORR}\,(A_0, C_0)$.

$\mathsf{CORR}\,(A_0$: atom chain, C_0 : charge sequence):

1. Initially we start with a protein chain made from A_0 and C_0:

$$P^0 = [A_0, C_0] = [\{a_0\}_{i=1}^n, \{q_i\}_{i=1}^n]$$

2. Create in memory an array $\mathbb{A}$ of shape $20 \times 100000 \times 3n$

3. For $r = 1, 2, 3, \cdots, N_{rep} = 10^5$:

   3.1. Simulate the folding process using step 2 of Algorithm 2.1 with $N_{mc} = 100$

   3.2. Save $P^5, P^{10}, P^{15} \cdots, P^{100}$, i.e. every 5 MC steps, to memory.

   3.3. Update $S[t, r] \leftarrow \mathsf{Flatten}(A^{5t})$ for $t = 1, 2, \cdots, 20$.

4. After $\mathbb{A}$ is fully filled, create another array $\mathbf{C}$ with shape $20 \times 3n \times 3n$.

5. For $t = 1, 2, \cdots, 20$, update $\mathbf{C}[t] \leftarrow C(P, 5t)$.

6. Let $\mathsf{CORR}(A_0, C_0) = \mathbf{C}$ and return $\mathsf{CORR}(A_0, C_0)$.

Now that we have a complete model to describe the protein folding process and short-term dynamics, the next step is to translate the questions in accordance to this model, and specify the tools to solve it.

3 Problem Formulation

In this section, we will formulate the exact problem we attempt to solve in this project. As mention in Section 1.2.2, we are interested in predicting the effects of a mutation of the protein chain on its native state. We will translate this objective into a prediction task, which can be solved by Machine Learning models, using the model developed in the last section.
Below is an overview of this section:

1. Describe the model we use the for mutation of protein chains, and the question this model poses.

2. Define the "impact" of a mutation, how to measure it, and formulate the details of the prediction task.

3. Analyse the parameters in the model and select them to improve the significance of the problem and demonstrative power of our method.

3.1 Mutation Model

We begin by describing the sample protein chain whose mutations are concerned in this project.

Definition 3.1 (Wild Type)**.** Let $Q > 0$ be a parameter to the model. Let $\mathsf{WildType}$ be the following charge sequence of 30 charges:

$$\begin{aligned}\mathsf{WildType} &= \underbrace{\{0, +Q, 0, -Q, \cdots, 0, +Q\}}_{\text{30 charges}} \\ &= \{q_i\}_{i=0}^{29} \quad \text{where} \quad q_i = \begin{cases} 0 & \text{if } i \equiv 0 \mod 2 \\ (-1)^{(i-1)/2} Q & \text{if } i \equiv 1 \mod 2 \end{cases}\end{aligned}$$

We are concerned with mutations on a native state of $\mathsf{WildType}$, where this native state is the folded from a straight atom chain and $\mathsf{WildType}$. Let us give this native state a name below.

Definition 3.2 (Reference State)**.** Let $\mathsf{St}(L)$ be the following straight chain

$$\mathsf{St}(L) = [(0, 0, i \cdot L)]_{i=0}^{29}$$

where L is the original spring length defined in Section 2.1 The **reference state** of $\mathsf{WildType}$, denoted by $\mathsf{RefState}$, is defined as

$$\mathsf{RefState} = \mathsf{REF}\left(\mathsf{St}(L), \mathsf{WildType}\right)$$

We consider this state an approximation of the native state for $\mathsf{WildType}$ and the straight chain. The straight chain is chosen as a starting point since most protein chains are in a straight shape when first created in the cell.

Now we will define a mutant. Speaking broadly, a mutant can be any atom chain and charge sequence that is different from $[\mathsf{RefState}, \mathsf{WildType}]$. However, in this project we shall only consider the following class of mutants:

Definition 3.3 (Mutant)**.** A charge sequence C of 30 charges is called a **mutant**, if it is of the following form:

$$C = \{q_i\}_{i=0}^{29} \;\text{ where }\; q_i \in \{0, +Q, -Q\}\ \forall i < 30 \;\text{ and }\; C \neq \mathsf{WildType}$$

Next, we would like to discuss the impact of the mutants of this class on $\mathsf{WildType}$, so we need to define a *mutation scheme*, i.e. when does the mutation occur and how is the state of the chain when it occurs. In particular, we are interested in the following mutation scheme:

Definition 3.4 (Mutation Scheme)**.** Given a mutant C. The mutation scheme of C occurring on $\mathsf{WildType}$ in three steps:

1. The protein chain $[\mathsf{St}(L), \mathsf{WildType}]$ folds to its reference state $\mathsf{RefState}$. The folding process is modeled in Section 2.3.

2. From $\mathsf{RefState}$, the charge sequence is replaced with A.

3. The new protein chain $[\mathsf{RefState}, C]$ folds to its reference state $\mathsf{RefState}_C = \mathsf{REF}\left(\mathsf{RefState}, C\right)$.

We call $\mathsf{RefState}_C$ the **mutation state** of C.

To sum up this part, we have defined the set of mutants we are interested in, and defined with each mutant C, a corresponding state $\mathsf{RefState}_C$, which is treated as the native state resulted from folding the mutated $\mathsf{RefState}$. Next, we would like to define the scheme to measure the impact of the mutation.

3.2 Measuring Mutation Impact

The approach we use is one of most direct approaches, which is to measure the deviation of the mutation $\mathsf{RefState}_C$ from the original $\mathsf{RefState}$. For this deviation, we will use the aligned distance defined in Appendix A.2. Later in Section 5.7, we push this idea further by also classifying whether a mutation has "significant" effect by setting up a threshold for the aligned distance.

Definition 3.5 (Mutation Impact)**.** We define the **impact** of a mutant C as the aligned distance between $\mathsf{RefState}_C$ and $\mathsf{RefState}$.

$$\mathsf{Imp}(C) = \mathsf{AD}\left(\mathsf{RefState}_C, \mathsf{RefState}\right)$$

This quantity $\mathsf{Imp}(C)$ will be the labels of our prediction. From Section 1.2.2 and Section 2.4, the input to our prediction models will be the 20-fold correlation matrices, which describe the short-term dynamics of each mutant.

Definition 3.6 (Mutation Short-term Correlation)**.** We define the **short-term correlation**, or simply **correlation** of a mutant C as the 20-fold correlation matrix, computed using Algorithm 2.4, of the folding process starting from $\mathsf{RefState}$ with charge sequence C:

$$\mathsf{CORR}\left(C\right) = \mathsf{CORR}\left(\mathsf{RefState}, C\right)$$

Note that this corresponds to the first 100 Monte Carlo steps of the folding process after the native state of $\mathsf{WildType}$ has been mutated.

To sum up, for each mutant C, our prediction model will have:

1. Output: the aligned distance of the native state of C with the original native state $\mathsf{AD}\left(\mathsf{RefState}_C, \mathsf{RefState}\right)$ and use it to indicate the impact of the mutation.

2. Input: the short-term correlation matrix of the first 100 steps of the mutation scheme and use it to indicate the short-term dynamics of atoms in the mutated chain.

The following diagram visualize the workflow for each mutant C:

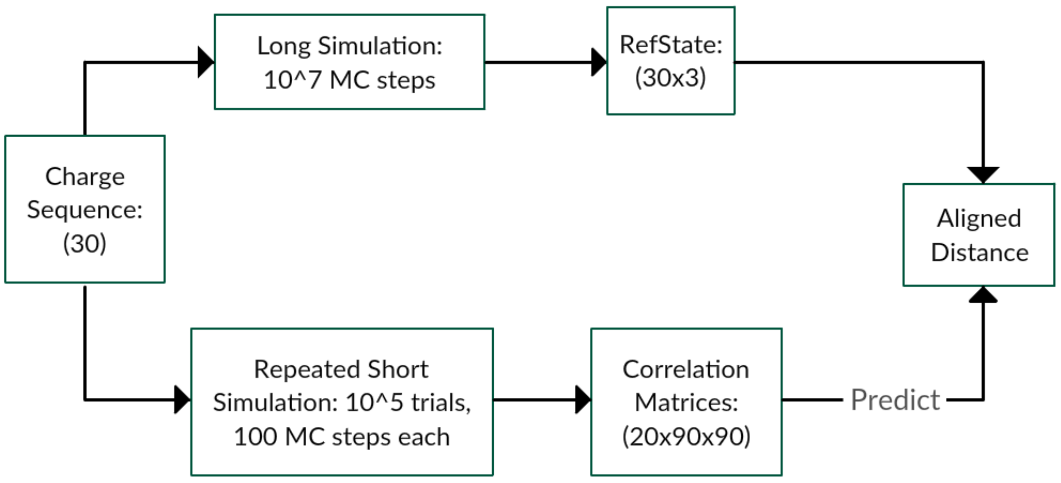

Long Simulation: 10^7 MC steps
RefState: (30x3)
Charge Sequence: (30)
Aligned Distance
Repeated Short Simulation: 10^5 trials, 100 MC steps each
Correlation Matrices: (20x90x90)
Predict

Figure 1: The worflow of our model

Next, we will discuss how the parameters to the atom chain and folding models (in Section 2.1) affect both the input and output of the predictions, and how to tune the parameters to obtain meaningful results.

3.3 Parameter Selection

The most obvious choice for the parameters are 1.0 for everything. However, there are two specific parameters that affect the outcome of the folding process and the short-term correlation, which in turns affect the prediction model.
Note that we are not attempting to increase the predictive power of our model. Instead, its predictive power remains robust regardless of the parameters. What we are aiming to improve is the *significance* of the model, or precisely, to increase the influence of the class of mutants of interest on the native state. We will elaborate further in Section 3.3.2. Before that, we will discuss the affect of the Temperature T on the folding speed of the chain.

3.3.1 Temperature Selection for Optimal Folding Rate

Recall the update rule for Monte Carlo simulation in Algorithm 2.1:
Let H and H^* respectively be the total energy of P and P^*, computed using (3). Let

$$p = \exp\left(-\frac{H^* - H}{K_B T}\right)$$

with K_B, T being Boltzmann constant and temperature.

- If $p > 1$, meaning $H^* < H$, update $P \leftarrow P^*$.

- If $p \leq 1$, update $P \leftarrow P^*$ with probability p.

Observation: When T is large, p is closer to 1, meaning the chain is more likely to be updated when $H^* > H$, i.e. updated to a state with higher energy level. While this allows the chain to escape non-global local minimum energy levels, it is also more likely to jump out of the equilibrium due to being easy to change.
Conversely, when T is small, p is closer to 0, meaning the chain is less likely to be updated when $H^* > H$, i.e. it tends to stick to a low energy level. Likewise, the chain maintain its energy level in the equilibrium, but bears the risk of being stuck in a local minimum, which could be far from the global one.
In short, the larger T means more chaotic and unstable native state but smaller T risks being in stuck in a local minimum and possibly far from the native state, which leads to inaccuracy in our model.
In this project, we consider two values for T: 1.0 and 0.5. To determine which is better, we would need to run test simulations with them and compare. Below we describe the experiments and results.

Experiment 3.7. In this experiment, we perform two long simulations following Algorithm 2.1, starting from a straight chain $\mathsf{St}(L)$ and charge sequence $\mathsf{WildType}$. We then plot the graphs of energy levels for both simulations and compare them.
Part I. We use the following parameters:

Parameter	Value	Parameter	Value
K	1.0	ε	1.0
L	1.0	σ	1.0
R_{mc}	1.0	K_B	1.0
Q	2.0	T	1.0

In particular, the simulation is run on the following atom chain:

$$\mathsf{St}(1.0) = [(0., 0., 0.), (0., 0., 1.), \cdots, (0., 0., 29.)]$$

and the following charge sequence:

$$\mathsf{WildType}(Q = 2.0) = [0, +2, 0, -2, 0, \cdots, 0, +2]$$

As mentioned in Algorithm 2.1, the simulation use $N_{mc} = 10^7$ Monte Carlo steps.

Part II. We use the following parameters:

Parameter	Value	Parameter	Value
K	1.0	ε	1.0
L	1.0	σ	1.0
R_{mc}	1.0	K_B	1.0
Q	2.0	T	0.5

The settings are the same as from Part I, but the value of T is switched to 0.5.

Results. The resulting energy graphs are shown in Figure 2 and Figure 3. From these graphs, we see that the simulation for $T = 1.0$ does not perform well as expected, compared to $T = 0.5$.

- When $T = 1.0$, the simulation is unstable as expected, having a wide fluctuation range for the energy. However, it fails to descend to a good minimum value, hence losing its only possible theoretical advantage. This is possibly due to it spending too much time wandering the $(-50, -100)$ and always happening to ascend up before reaching low enough.

- When $T = 0.5$, the simulation is stable with low energy fluctuation range as expected. Furthermore, it is able to descend to a much lower energy level (-140) than when $T = 1.0$ (-100) and stays stable in the region $(-110, -140)$.

Based on the above observations, the value $T = 0.5$ is practically better, given the other parameters are all 1.0. Choosing $T = 0.5$ will not only give us a smaller equilibrium range, hence more accurate estimate of the native state, but also gain more efficiency in time and computing resources.

Below are the summary of the parameter values at the end of this part:

Parameter	Value	Parameter	Value
K	1.0	ε	1.0
L	1.0	σ	1.0
R_{mc}	1.0	K_B	1.0
Q	2.0	T	0.5

3.3.2 Charge Selection for Significant Influence on the Native State

Now we will discuss the influence of the value the parameter Q on the native state and the significance of our model.

Firstly, the significance of the model means the representative power of our class of mutants. Since the mutants only alter the charges, if the Coulomb component H_{cou}

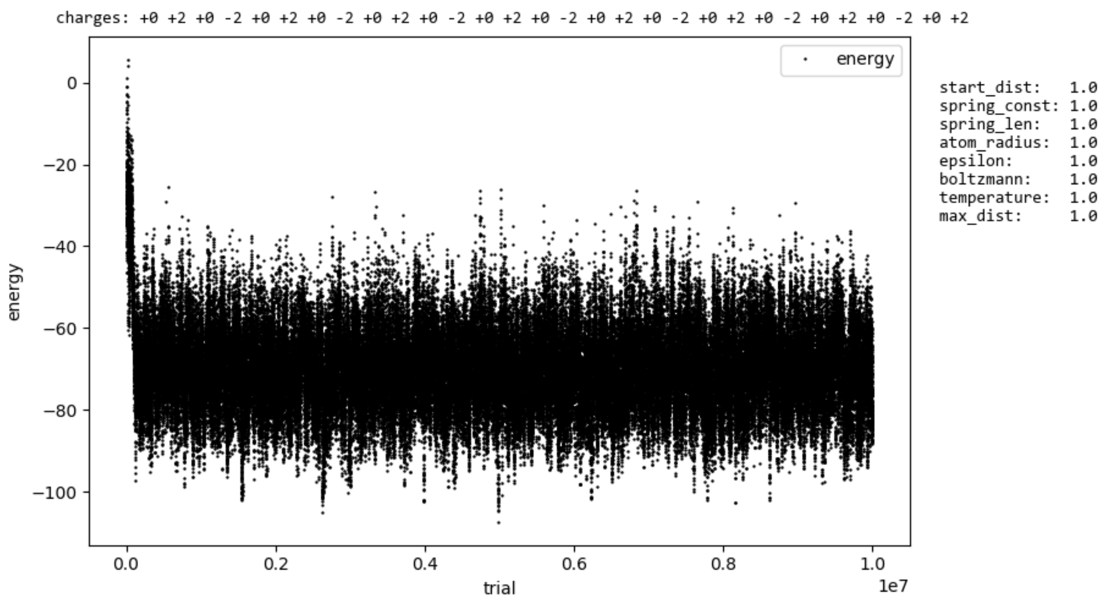


Figure 2: Energy graph for T = 1.0.

in the Hamiltonian in (3) ends up having little influence over the sum, which leads to low impact on the native state, the charges, and hence this class of mutants, will not produce many mutations that deviate much from the original RefState. In other words, if we measure a mutation by its native state aligned distance as shown in Section 3.2, and the Coulomb component has litter effect on the native state, we will not have enough mutants with noticeable impact, and the purpose of our project becomes obsolete.

Thus, it is important to adjust the charge value Q so that the mutants we are interested in have non-negligible impact and increase the demonstrative power of our model. Note that in the original report [1], the author used $Q = 2.0$. We attempt to double this value, and observe the change in the native state and short-term dynamics following the change.

Experiment 3.8. This experiment also consists of two parts. We run two independent sets of short-term simulations, following Algorithm 2.4, each having 10^5 trials with 100 Monte Carlo steps per trial, one with $Q = 2.0$ and the other $Q = 4.0$. The 20-fold correlations are then generated, plotted and compared with the correlations of the corresponding RefState.

The following is the procedure of each part of this experiment:

1. Define a mutant

$$\mathsf{Alt0pQ} = [0, +Q, 0, +Q, \cdots, 0, +Q]$$

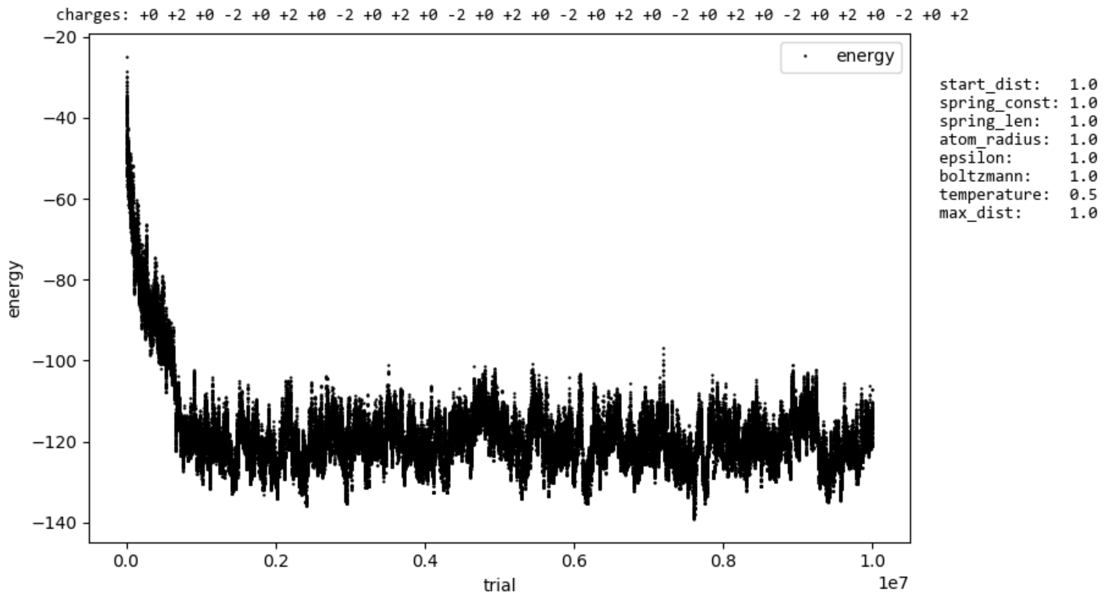


Figure 3: Energy graph for T = 0.5.

i.e. all charges $-Q$ in WildType are switched to $+Q$.

2. Compute CORR (WildType) and CORR (Alt0pQ).

3. Plot the matrices as colored map and compare these maps visually.

Note that step 2 cannot be shared for the two parts, since the value of Q differs, hence the WildType and RefState are two both different in each part.
Part I. Re-using the parameters at the end of the previous section:

Parameter	Value	Parameter	Value
K	1.0	ε	1.0
L	1.0	σ	1.0
R_{mc}	1.0	K_B	1.0
Q	2.0	T	0.5

In particular, we find the correlation matrices of the following charge sequence

$$\mathsf{WildType}(Q = 2.0) = [0, +2, 0, -2, \cdots, 0, +2]$$

given its reference state

$$\mathsf{RefState}(Q = 2.0) = \mathsf{REF}\left(\mathsf{St}(L = 1.0), \mathsf{WildType}(Q = 2.0)\right)$$

and the correlation matrices of the following mutant

$$\mathsf{Alt0pQ}(Q = 2.0) = [0, +2, 0, +2, \cdots, 0, +2]$$

given the same reference state $\mathsf{RefState}(Q = 2.0)$.

Part II. Now we keep all parameters from Part I, except Q, except that we double the value of Q: $Q = 4.0$.

In particular, we find the correlation matrices of the following charge sequence

$$\mathsf{WildType}(Q = 4.0) = [0, +4, 0, -4, \cdots, 0, +4]$$

given its reference state

$$\mathsf{RefState}(Q = 4.0) = \mathsf{REF}\left(\mathsf{St}(L = 1.0), \mathsf{WildType}(Q = 4.0)\right)$$

and the correlation matrices of the following mutant

$$\mathsf{Alt0pQ}(Q = 4.0) = [0, +4, 0, +4, \cdots, 0, +4]$$

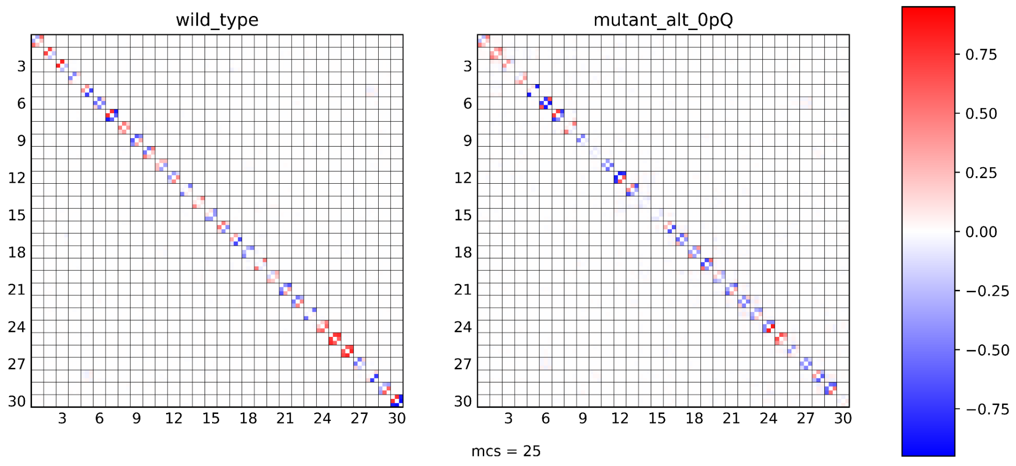
given the same reference state $\mathsf{RefState}(Q = 4.0)$.

Results. We only show the correlation matrices at $t = 25, 50, 75, 100$ as examples. In the figures below, the color scaling are uniform, and the right column are correlations of Alt0pQ, while the left are that of WildType.

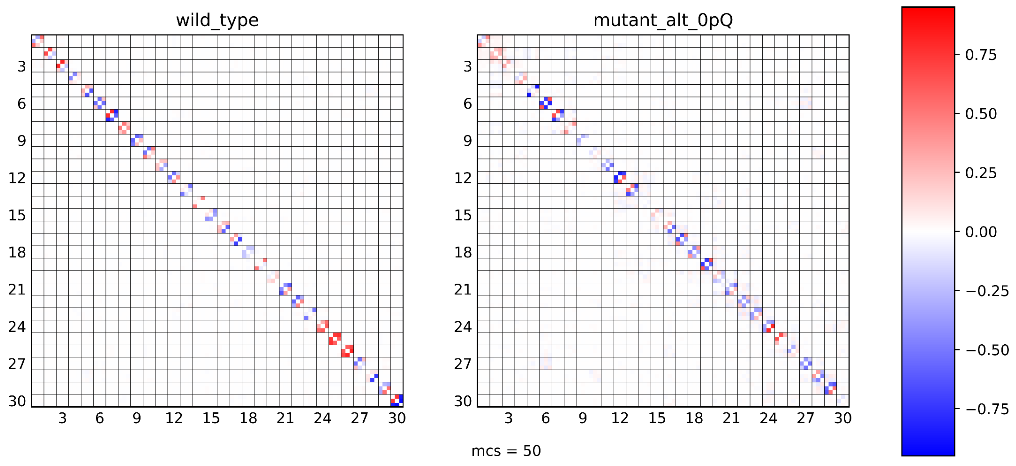
Figure 4 and Figure 5 show the correlation comparison for $Q = 2.0$, while Figure 6 and Figure 7 show that for $Q = 4.0$.

Observations. In Figure 4 and Figure 5, it is hard to detect any significant difference in the correlation matrices for Alt0pQ and WildType. Both are bold at the diagonal, which translates to strong self-correlation of the atoms, and extremely dim elsewhere. This implies atoms in the chain mainly vibrate in their own, independent directions.

On the other hand, Figure 6 and Figure 7 show a clear difference between the correlations of Alt0pQ and WildType, which gets stronger as t increases. The diagonal terms for Alt0pQ are noticeably dimmer than those of WildType, while the one-offet diagonal are more visible. This implies atoms move more chaotically with less clear directions, but are still strongly influence by the movement of their next and previous atoms in the chain.

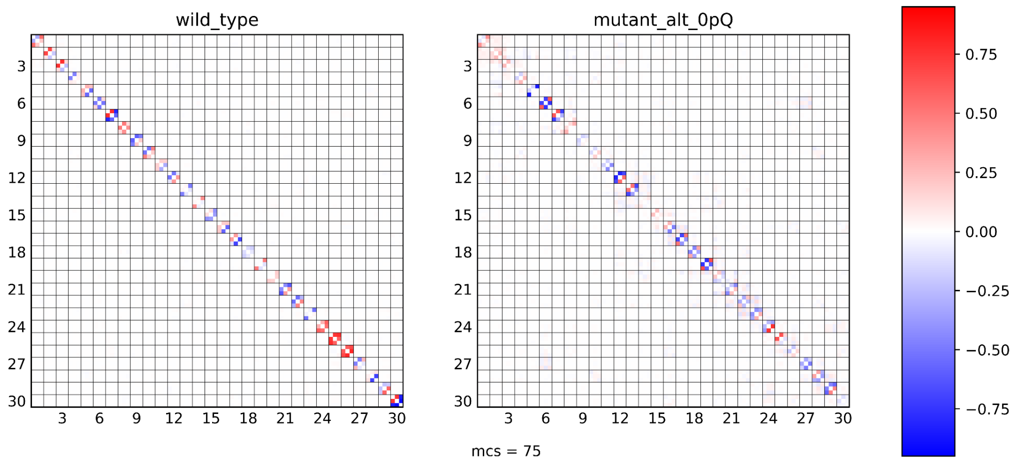


(a) mcs=25.

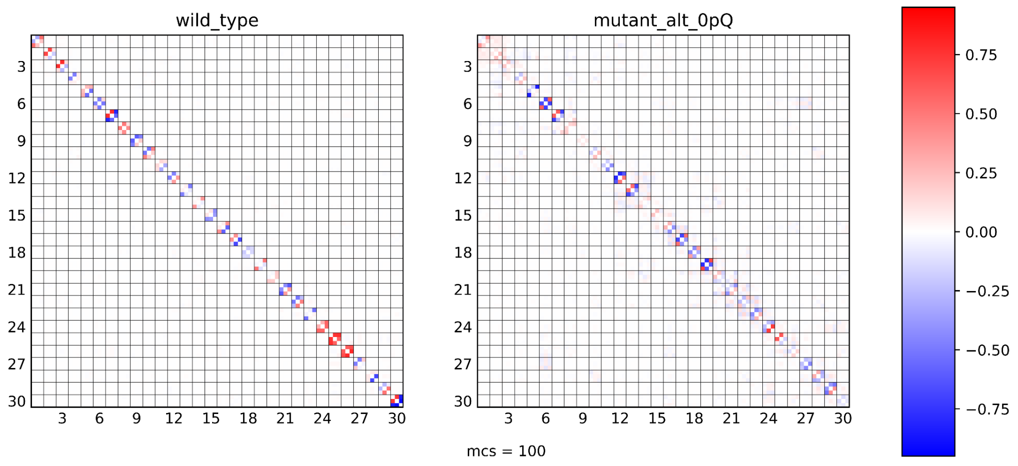


(b) mcs=50.

Figure 4: Correlation Matrices of WildType and Alt0pQ for $Q = 2.0$ (1).



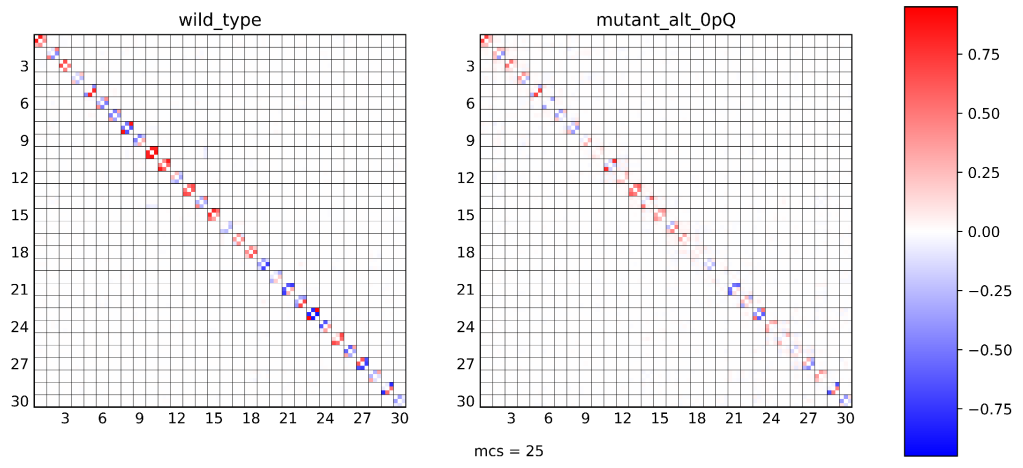
(a) mcs=75.



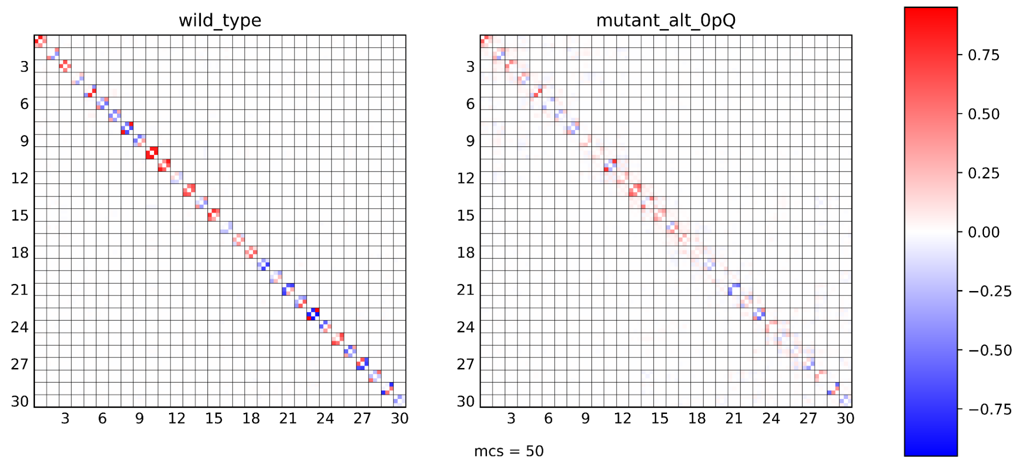
(b) mcs=100.

Figure 5: Correlation Matrices of WildType and Alt0pQ for $Q = 2.0$ (2).

Therefore, these matrices show that there is a difference in the short-term dynamics of the mutant from that of WildType, and this difference increase positively with the value of Q. It goes from hardly noticeable to clearly visible when Q goes from 2.0 to 4.0. Furthermore, the numerical difference also leads to the difference in interpretation, where the actions and interactions of atoms in the mutant are completely different from WildType when $Q = 4.0$, while almost the same when $Q = 2.0$.



(a) mcs=25.



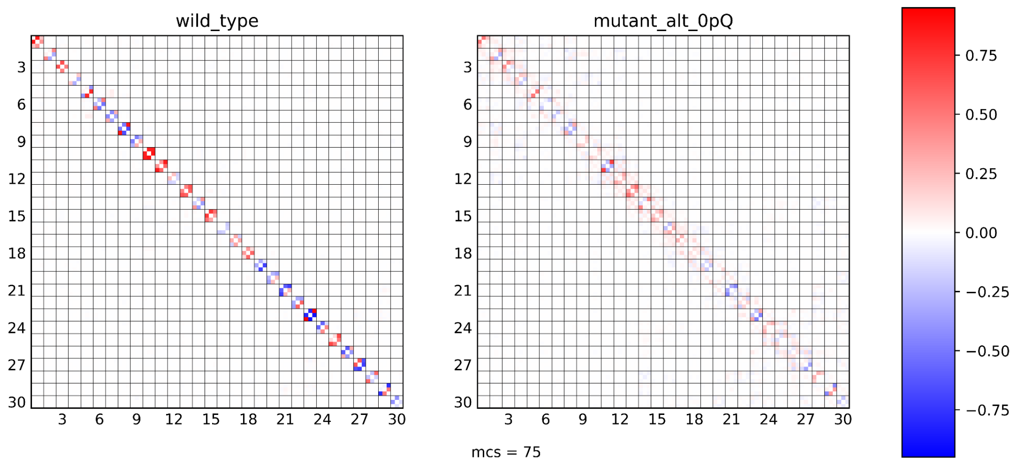
(b) mcs=50.

Figure 6: Correlation Matrices of WildType and Alt0pQ for $Q = 4.0$ (1).

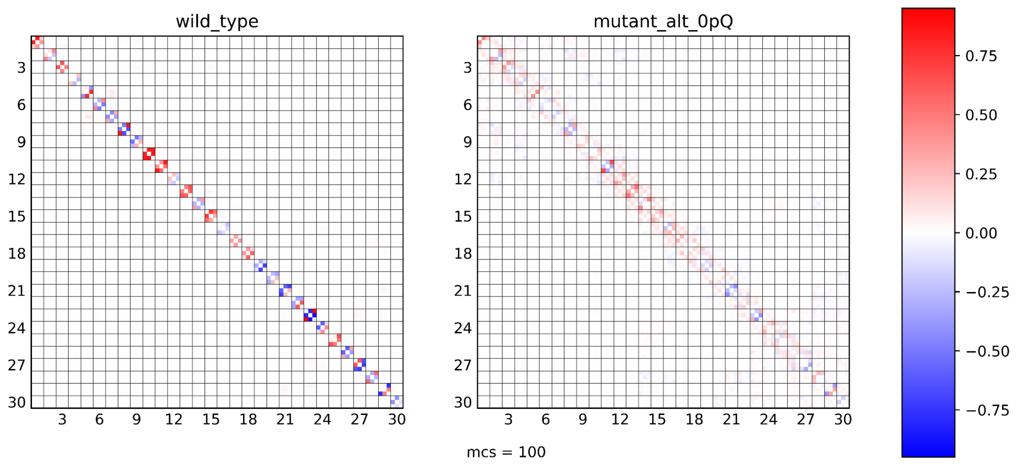
Conclusion. We decide to use the value $Q = 4.0$ for our simulations. In particular, we use this value to generate mutants in Generating Mutants. The following table summarizes all parameters for our simulations:

Parameter	Value	Parameter	Value
K	1.0	ε	1.0
L	1.0	σ	1.0
R_{mc}	1.0	K_B	1.0
Q	4.0	T	0.5

Table 1: Parameters used for all simulations.



(a) mcs=75.



(b) mcs=100.

Figure 7: Correlation Matrices of WildType and Alt0pQ for $Q = 4.0$ (2).

4 Data Generation

Before talking about prediction models, we need a sufficient amount of training and testing data for any model we would build. We will discuss the data generation process in this section.

4.1 Objectives

Recall the workflow to generate a training data point:

1. Generate a mutant C from the class described in Section 3.1.

2. Using the procedures in Section 3.2, generate the correlation matrix $\mathsf{CORR}(C)$ and the mutation state $\mathsf{RefState}_C$.

3. From $\mathsf{RefState}_C$, compute the aligned distance $\mathsf{AD}(\mathsf{RefState}_C, \mathsf{RefState})$.

4. Set $\mathsf{CORR}(C)$ as input, $\mathsf{AD}(\mathsf{RefState}_C, \mathsf{RefState})$ as labels and add this point to the dataset.

In order to obtain a sufficient and quality dataset, we have three tasks:

1. Generate a set of mutants with enough representative power while not being too large to reduce the time and computational cost.

2. Generate $\mathsf{CORR}(C)$ and $\mathsf{RefState}_C$ for all generated mutants C efficiently, i.e. save time and computational cost as much as possible.

We will describe our data generation method and explain our design choices in the following sections.

4.2 Generating Mutants

4.2.1 Challenges

Recall the definition of mutants in Section 3.1: The set of mutants is the following sets:

$$\begin{aligned}\mathsf{MUTANTS} &= \left\{C = [q_i]_{i=0}^{29} : q_i \in \{0, \pm Q\}\ \forall i \wedge C \neq \mathsf{WildType}\right\} \\ &= \{0, +Q, -Q\}^{30} - \{\mathsf{WildType}\}\end{aligned}$$

The cardinality of $\mathsf{MUTANTS}$ is then:

$$3^{30} - 1 = 205891132094648 \approx 2 \times 10^{14}$$

As this number is astronomical, we can only afford to generate an infinitesimal portion of possible mutants. In fact, we aim to generate a set with less than $100,000$ mutants but still retain high representative power.

4.2.2 Solution

The first step of our approach is restrict ourselves only to the following subsets of mutants:

$$\mathsf{MUTANTS}_0 = \{C \in \mathsf{MUTANTS} : |C[2i+1]| = |\mathsf{WildType}[2i+1]|\ \ \forall 0 \leq i \leq 14\}$$

In more details, a mutant $C = \{q_i\}_0^{29} \in \mathsf{MUTANTS}_0$ if and only if $q_i \in \{\pm Q\}$ for all i such that $\mathsf{WildType}[i] \in \{\pm Q\}$.

Now, the cardinality of $\mathsf{MUTANTS}_0$ is:

$$3^{15} \times 2^{15} - 1 = 470184984575 \approx 4.7 \times 10^{11}$$

Again, we cannot afford to generate such amount of mutants. Therefore, we employ a random sampling method to randomly generate mutants of this set. With random sampling, we can control the number of samples to obtain with little difficulty.
The idea is to first pick the group of atoms to mutate, i.e. group of charges to switch. We classify each group of charges, as subsets of $\{0, 1, \cdots, 29\}$, by their *influence* over the total energy in (3). Our assumption is that charge groups with similar influence will give the mutated chain similar short-term dynamic schemes and native states. Under this assumption, we will only need to generate a very few mutants based on each group, and greatly reduce the number of mutants to generate.
Thus, the first step is to find a metrics to compare and classify charge groups into classes of similar influence.

Definition 4.1. The **influence** of a subset of charges $S \subseteq \{0, 1, \cdots, 29\}$ is the following sub-sum of the Hamiltonian in (3):

$$\begin{aligned}\mathsf{Infl}(S) = {} & K \sum_{i=0}^{28} \left(d_{i(i+1)} - L\right)^2 \mathbb{1}_{\{i \in S \vee i+1 \in S\}} \\ & + 4\epsilon \sum_{0 \le i < j \le 28} \left[\left(\frac{\sigma}{d_{ij}}\right)^{12} - \left(\frac{\sigma}{d_{ij}}\right)^{6} \right] \mathbb{1}_{\{i \in S \vee j \in S\}} \\ & + \sum_{0 \le i < j \le 28} \frac{q_i q_j}{d_{ij}} \mathbb{1}_{\{i \in S \vee j \in S\}}\end{aligned} \tag{6}$$

where the atom chain is $\mathsf{RefState}$ and the charges are $\mathsf{WildType}$.
Simply put, $\mathsf{Infl}(S)$ is the sum of all terms in the Hamiltonian expression that involves terms d_{ij} with one of the indices belong to S.

Next, we define the concept of "similar influence"

Definition 4.2. Two subsets $S_1, S_2 \subset \{0, 1, \cdots, 29\}$ are said to have have **similar influence** if $|S_1| = |S_2|$ and

$$\mathsf{ROUND}(\mathsf{Infl}(S_1), 20) = \mathsf{ROUND}(\mathsf{Infl}(S_2), 20)$$

where the $\mathsf{ROUND}(x, 20)$ rounds x to the nearest multiple of 20, i.e. $\mathsf{ROUND}(105.2, 20) = 100$, $\mathsf{ROUND}(216.3, 20) = 220$, $\mathsf{ROUND}(50.0, 20) = 40$.
The number 20 is called the **base**.

We will discuss the choice of the base later. We now give an algorithm to generate the necessary charge sequence

Algorithm 4.3. Given RefState and WildType.

1. Let $Z = \{0, 2, 4, \cdots 28\} = \{i : \mathsf{WildType}[i] = 0\}$.
2. Let $P = \{1, 5, 9, \cdots, 29\} = \{i : \mathsf{WildType}[i] = +4\}$.
3. Let $N = \{3, 7, 11, \cdots, 27\} = \{i : \mathsf{WildType}[i] = -4\}$.
4. Iterate throught subsets of Z, and create a list of similar influence classes $\mathsf{SimInfl}\,(Z)$.
5. Create $\mathsf{SimInfl}\,(P)$ and $\mathsf{SimInfl}\,(N)$ in the same manner.
6. Actual output at this state shows:

Subgroup	No. of classes sorted by member size															
Size	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Z	1	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1
P	1	1	2	3	4	4	4	3	1							
N	1	2	2	3	3	3	2	1								

Table 2: Number of classes of similar influence sorted by member size

7. We will pick one small subset in each class in each subgroup, then combine them to get a larger subset of $\{0, 1, \cdots, 29\}$. The corresponding mutants will be generated by mutating the charges positioned in this subset in the following scheme:
8. For each class $\mathcal{C}_Z$ of Z:
   8.1. Repeat for $2^{m/3}$ times, where m is the size of members in $\mathcal{C}_Z$:
      i. Sample $Z' \in \mathcal{C}_Z$, $N' \in \mathcal{C}_N$, $P' \in \mathcal{C}_P$ randomly.
      ii. Store Z' in memory.
   8.2. This results in $2^{m/3}$ parts being stored. Store the container of all of them in memory.
9. For each class $\mathcal{C}_N$ of N:
   9.1. Sample $N' \in \mathcal{C}_N$ randomly.
   9.2. Store N' in memory.
10. For each class $\mathcal{C}_P$ of P:
   10.1. Sample $P' \in \mathcal{C}_P$ randomly.

10.2. Store P' in memory.

11. The table below show the number of parts generated for each member size:

Subgroup	No. of parts generated sorted by member size															
Size	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Z	1	2	2	4	4	4	8	8	8	16	16	16	16	16	16	32
P	1	1	2	3	4	4	4	3	1							
N	1	2	2	3	3	3	2	1								

Now we can combine at triplet of parts Z', N', P' to form a set of charges to be switched, thus make a mutant from them.

12. For each triplet of parts Z', N', P', let $S = Z' \cup N' \cup P'$. We generate a random mutant from S as follows:

12.1. Let $C = \mathsf{WildType}$.

12.2. For each $i = 0, 1, \cdots 29$, update:

$$C[i] = \begin{cases} -4 & \text{if } i \in P' \\ +4 & \text{if } i \in N' \\ \begin{cases} -4 & \text{w. prob. } 1/2 \\ +4 & \text{w. prob. } 1/2 \end{cases} & \text{if } i \in Z' \\ C[i] & \text{otherwise} \end{cases}$$

12.3. Store C in memory.

13. Return the container of all the mutants generated.

The algorithm above generate a total of 66079 mutants. We will refer to this set of mutants as $\mathsf{MUTANTS}^*$ from now on.
Under the assumption that charge groups of "similar" influence, as computed in Definition 4.1, indeed affects the native state in similar ways when mutated, this data set is guaranteed to have enough expressive power, while having a manageable size for our prediction model.
We shall describe this model in the next section.

5 The Neural Network

Our approach to the prediction task from Section 3.2 is to use a neural network (NN). Neural networks are known for their wide variety, flexibility and extensive library support. Theoretically, for any prediction problem that can be solved by some statistical or regression model, one can build a neural network with equivalent or stronger predictive performance, thanks to the freedom one has in structuring the network. Therefore, we are hopeful that we can build a neural network suitable for our problem, if it is indeed solvable by predictive models.

In fact, the input format in our problem are 3D matrices of shape $20 \times 90 \times 90$, which can be thought of a 3D images, or 2D images with channels. This leads naturally to the use of **Convolutional Neural Networks** (**CNN**s).

We begin by describing our training and testing scheme, universal for all models, followed by a description of our method for rescaling data to match the output range of popular activation functions.

5.1 Training and Testing

As discussed in Generating Mutants, we have already generated a set $\mathsf{MUTANTS}^*$ of 66079 mutants and their correlation matrices, as well as their distance to $\mathsf{RefState}$.

We employ a standard train test split algorithm, which divides the data set into a training set (80%) and testing set (20%). The algorithm also shuffles the data set before splitting it, so we are guaranteed randomness in both the training and testing set. The API we use for this algorithm is inside `sklearn` [13], one of the most popular and efficient Python package for scientific computing and machine learning.

Furthermore, on the training data set, we repeat the same splitting algorithm with train test ratio $80 - 20$ to further divide it into a smaller training set and a validation set. In summary, the training, validation and test sets' ratio are:

$$64 : 16 : 20 \quad (\%)$$

It is clear what the training set and test set are for just from the names. The validation set's role is similar to the test set, except that we are able to track the network's performance on it after each backpropagation loop. To clarify completely, below is the summary of the roles of the datasets:

- **Training dataset:** 64% of the whole dataset, i.e. 44290 data points, each contains a correlation matrix and a value of the aligned distance.
- **Validation dataset:** 16% of the whole dataset, i.e. 10573 data points. This dataset works like a "test" for the model after every "lesson" (i.e. each completed

training batch and backpropagation). The performance of the model on this dataset is updated regularly during training, but it is not used for training. One of the most important role of the validation set is to prevent overfit: if the training loss decreases but validation loss increases, it is likely an overfit has happened.

- **Test dataset:** 20% of the whole dataset, i.e. 13216 data points. This dataset is not present in training and will be only be used to evaluate the robustness of the model after training is finished.

To finalize the training procedure, we need to specify the optimizer, the loss function and evaluation metrics. We use the ADAM optimizer, which has been shown to achieve accurate and fast convergence in many cases [7].
For the loss function, we choose a standard mean-squared-error function [8]:

$$\mathsf{MSE}(\text{Label}, \text{Pred}) = \frac{1}{n}\sum_{i=1}^{n}(\text{Label}[i] - \text{Pred}[i])^2$$

where $n = \mathsf{length}(\text{Label})$, and two metrics [9]

- Mean absolute error:

$$\mathsf{MAE}(\text{Label}, \text{Pred}) = \frac{1}{n}\sum_{i=1}^{n}|\text{Label}[i] - \text{Pred}[i]|$$

- Mean absolute percentage error:

$$\mathsf{MAPE}(\text{Label}, \text{Pred}) = \frac{100}{n}\sum_{i=1}^{n}\left|\frac{\text{Label}[i] - \text{Pred}[i]}{\text{Label}[i]}\right|$$

5.2 Output Rescaling and Transformation

The currently popular activation functions for the last hidden layers of neural networks are Sigmoid and Hyperbolic tangent (tanh). However, these functions have range $(0, 1)$ and $(-1, 1)$ respectively, making output transformation mandatory for regression tasks with larger label ranges. In this part, we explore two methods of output rescaling, **linear rescaling** and **unbounded rescaling**, to fit the properties of our data.
Firstly, we plot all the labels in the training set to check their value range. The result is shown in Figure 8

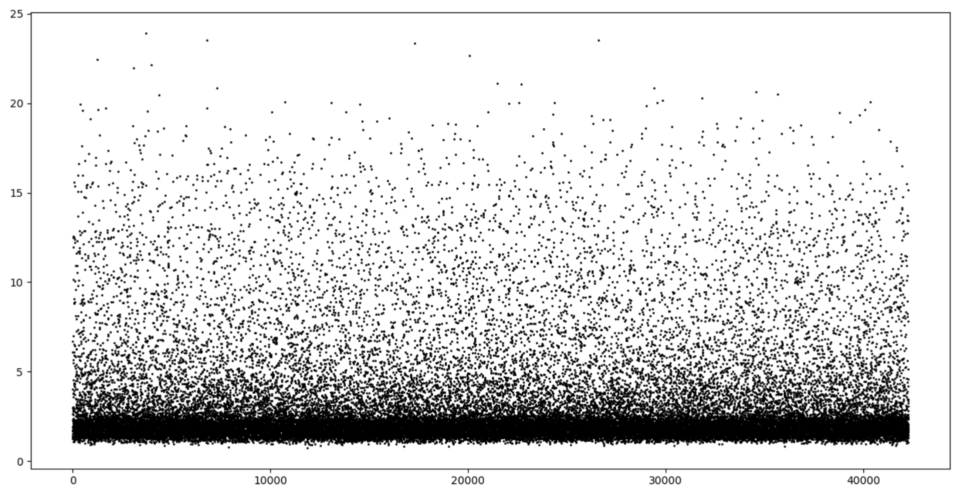


Figure 8: Distribution of the aligned distances of the mutants.

Figure 8 indicate that all of the aligned distance values are in the range $[0, 25]$. Hence we can linearly rescale the output in the following way:

- $x \to 25x$ if x is the output of sigmoid function.
- $x \to 12.5(x + 1)$ if x is the output of `tanh`.

However, to account for possible outliers in the testing set and practice, we will increase the multiplier to 30 instead. Thus the finalized operation to apply on the output is

- $x \to 30x$ if the last hidden layer uses Sigmoid activation.
- $x \to 25(x + 1)$ if the last hidden layer uses `tanh` activation.

Now, the motivation for unbounded rescaling is: On theory, the aligned distance between two atom chains can go unbounded, the labels in testing and practice may still fall above 1. To account for this possibility, we use the following transformation:

- $x \to \dfrac{x}{1-x}$ if the last hidden layer uses Sigmoid activation.
- $x \to \dfrac{1+x}{1-x}$ if the last hidden layer uses `tanh` activation.

Since the output of Sigmoid is strictly in the range $(0, 1)$, $\lambda(x) \in (0, +\infty)$ for all output x of the sigmoid function. Moreover, for any (no matter how large) $M > 0$, we have:

$$1 > x > \frac{M}{M+1} \implies \lambda(x) = \frac{x}{1-x} > M$$

The proof of the positiveness and unboundedness for `tanh` is analogous.

Hence, theoretically, we this transformation has unbounded range. We attempt to use it to capture all possible values for the labels, both in testing and practice.
In subsequent parts, we will describe several neural networks, which makes use of the four output transformations above.

5.3 Toy Model and Problems with Data

In Section 5.3.1 only, we will replace the dataset generated from MUTANTS* by a small subset, denoted by M, consisting of only 6600 data points (out of ≈ 66000). Our goal in this section is to describe our first vanilla network, its problems, and how we fixed them in subsequent networks.

5.3.1 Vanilla Convolution Network

Our first attempt is a sequential model consisting of two parts. The first part (convolution part) consists of 3 `Conv-Conv-Pool` blocks, namely, each block has two convolution layers followed by a max pooling layer. Its output is then flattened and fed into the second part (fully connected part), which consists of four fully connected layers with decreasing sizes, followed by an unbounded transformation layer (as described in Section 5.2) before the output. It is summarized in Figure 9.

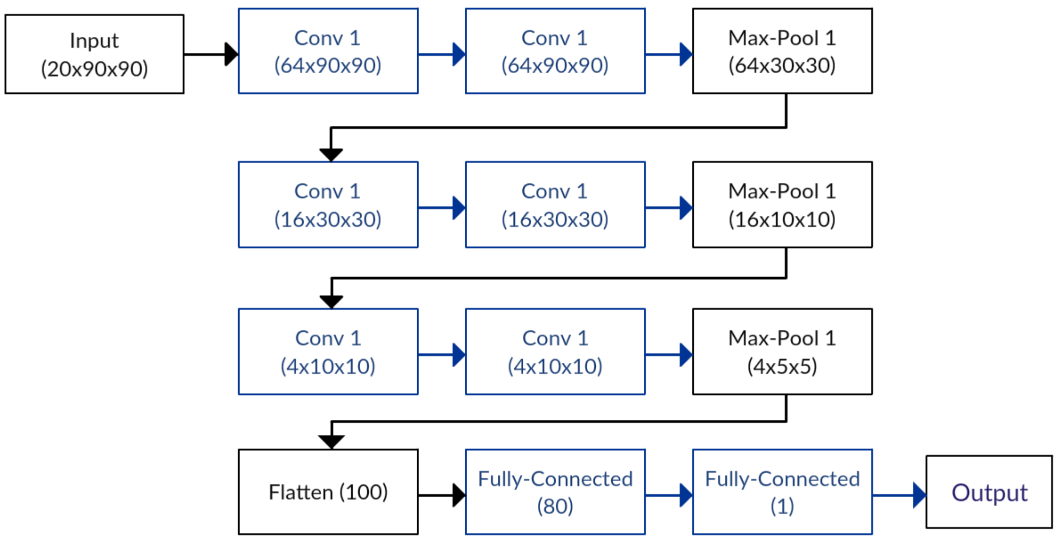


Figure 9: Structure of the Vanilla Model

We trained the network for 100 epochs, with a batch size of 2000, meaning 3 batch updates per epoch (since we are using the toy data set). With this configuration, the network takes 3 hours to train on a 48-core NSCC node with 640 GB RAM. Training with the full batch is impossible even with 1 TB RAM since the memory consumption exceeds the limit. Figure 10 shows the loss, error and accuracy curves.

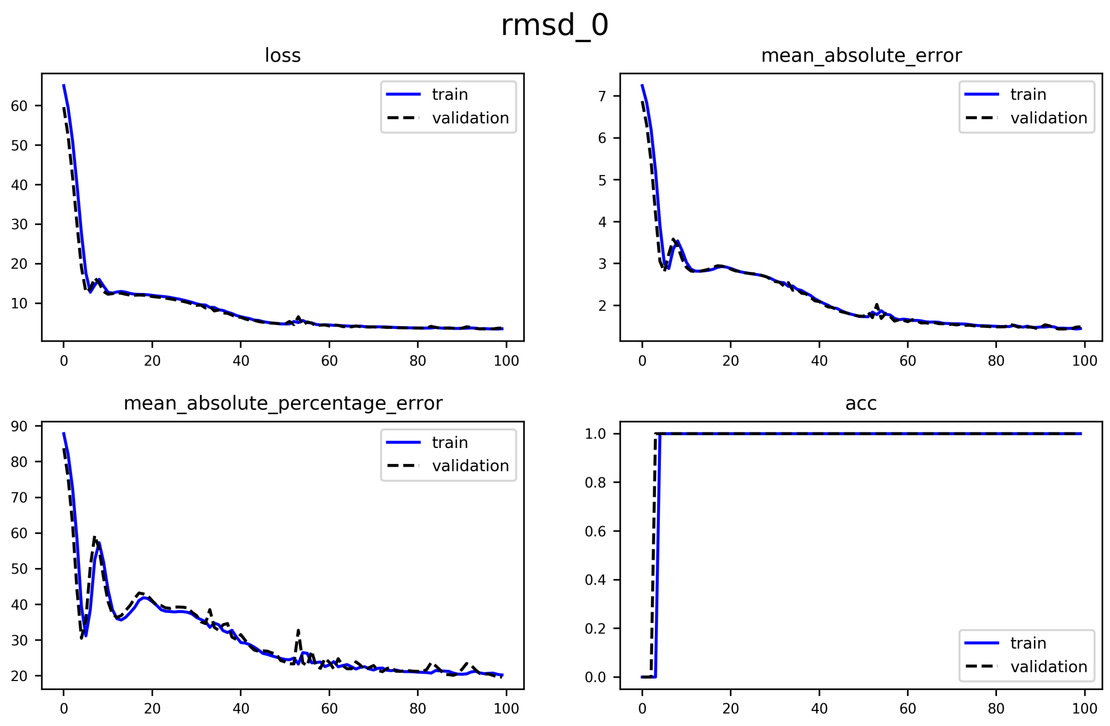


Figure 10: Loss, Error and Accuracy curves for Vanilla Model

This graph shows that the curves for training and validation are almost identical at all times. This is due to the first few values are very large for the errors and loss, which massively increase the scale. In order to “zoom in”, we need to go past the first few epochs. Let us plot the curves for epochs from 50 only in Figure 11.

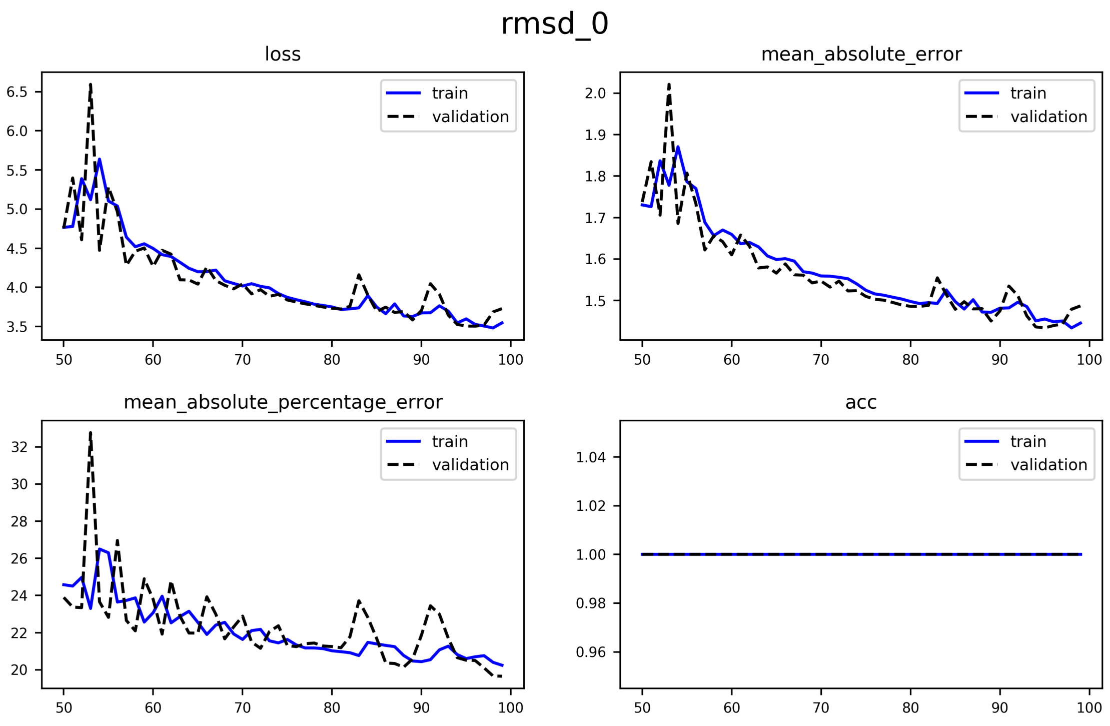


Figure 11: Curves for Vanilla Model starting from epoch 50

Although the result in Figure 11 is acceptable, it is impractical for our project to continue with this model due to the tremendous time and computational cost. Therefore in the next part, we identify and exploit several key properties of the dataset to reduce the data size and retain the most useful data to build new models with fast training and low memory consumption.

5.3.2 Problems

The toy model fails for several reasons:

1. **Data size:** The input data exceeds the RAMs of even super computers in public domain (without special configuration for large memory handling). The whole input set, which are of shape $66079 \times 20 \times 90 \times 90$, take up about 81GB of memory. This input is then separated and duplicated, and distributed multiple times over the network during training, making the real amount of data consumption proportionally higher. Even the memory it takes to train with 10% of the data exceeds most personal computer's RAM. Hence this model is unscalable for a school project.

2. **Redundant data:** When we sample a mutant in M and plot its correlations, we found that the correlation matrices change slowly with time steps, i.e. the matrix at $t = 5$ is mostly similar to the matrix at $t = 10, 15$. Figure 12 shows the correlation matrices at $t = 50, 55, 60, 65, 70, 75$. There is not much difference between them

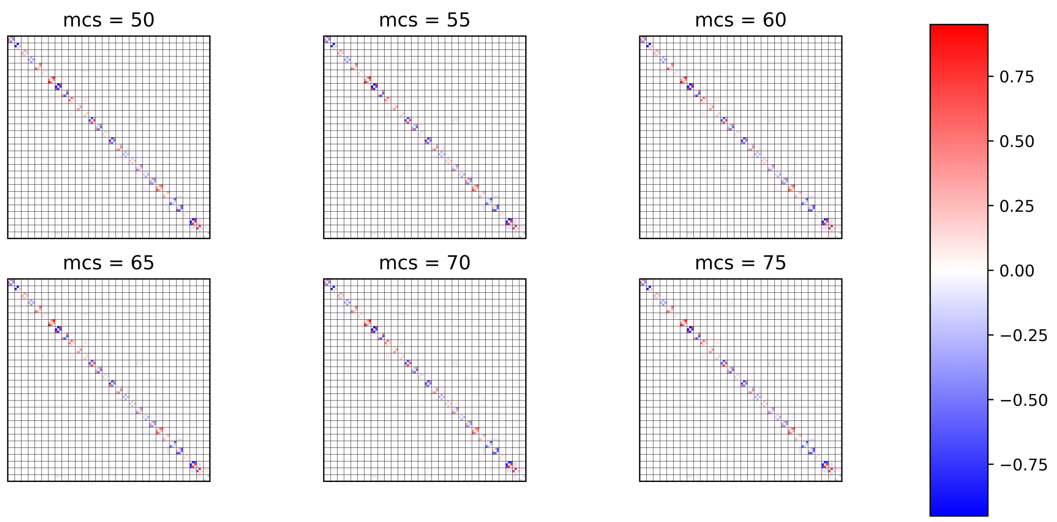


Figure 12: Correlation matrices of a mutant at $t = 50, 55, 60, 65, 70, 75$

Hence, since subsequent channel of the input are highly correlated, only selecting a few of them at time steps away from each other will most likely suffice. Eliminating redundant data will also reduce the amount of memory and time consumption in training.

From the figure above, we also observe that most of the significant (i.e. those with sufficient absolute value) correlation coefficients reside very close to the main diagonal of the matrices. This implies the atoms are most strongly correlated with themselves, which mean they are moving in a consistent general direction during the short simulation. Outside the main diagonal, only the first offset diagonal appears to have any visible correlation, which indicates atoms are affected by their neighbors in the chain (this is not surprising, since they are pulled by the spring force according to (3)).

The challenge here is that, two completely different short-term dynamics will result in correlation matrices 90% similar to each other, only differ near the diagonals. This massive similarity will confuse the network, which leads to extremely slow convergence (to the point that it appears to have converged with bad performance, like our model).

3. **Loss of information:** As shown in the figure below, the correlation block (defined in Definition 2.2), when two atoms are highly correlated, have some strongly positive and some strongly negative cells.

$$\mathsf{corr}\,(a_1, a_2) = \begin{bmatrix} +0.9 & -0.8 & -0.1 \\ +0.1 & +0.5 & -0.6 \\ +0.0 & -0.5 & -0.5 \end{bmatrix}$$

If we apply a convolution kernel onto this block, the result is a linear combination of the values in the cells. For instance, we use the following filer:

$$F = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

Then the convoluted block is:

$$\begin{bmatrix} +0.9 & -0.8 & -0.1 \\ +0.1 & +0.5 & -0.6 \\ +0.0 & -0.5 & -0.5 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} = 0.0$$

where $\otimes$ denote the sum of the cell-wise products. Thus, the feature value that this filter extract from this block would be 0.0. However, consider the same filter and a weakly correlated block:

$$\begin{bmatrix} +0.1 & -0.3 & -0.1 \\ +0.0 & -0.5 & -0.2 \\ -0.0 & +0.0 & -0.1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} = 0.0$$

Hence, this implies that the filter fails to extract different feature values for supposedly far distinct type, i.e. strong vs weak correlation, of inter-atom dynamics. The reason is the large positives and negatives in the block cancels each other in a linear combination with appropriate coefficients, which results in the same 0.0 result in the end. This phenomenon will confuse the network since some weakly correlated interactions can be indistinguishable from strongly correlated ones and make the next layer after the first convolution layer noisy. Here, the information on the *absolute values* of the cells, which indicate the strength of the correlation, has been lost. We call this problem the loss of information. Its solution will be discuss in the next part.

5.4 Input Transformation and Reduction

In this section, we fix problems with the vanilla model by giving algorithms to transform the data properly. In particular, we will apply transformations to the correlation matrices to reduce their size, eliminate redundant and confusing data, and retain only the most import part. These transformations are as follows:

1. **Channel Filtering**: Normally, each chronological correlation matrix $\mathbf{C}$ has shape $(20 \times 90 \times 90)$. We apply the operation:

$$\mathsf{FilterChn}(\mathbf{C}) = \begin{bmatrix} \mathbf{C}[4,:,:] & \mathbf{C}[9,:,:] & \mathbf{C}[14,:,:] & \mathbf{C}[19,:,:] \end{bmatrix}^T$$

   In other words, we retain the correlation matrices at times $t = 25, 50, 75, 100$ and discard the rest. This guarantees sufficient variance among them, hence make each of them meaningful input to the model.

2. **Sub-channel Splitting**: To tackle the problem of information loss, we need to separate the positive cells from negative cells. Hence, we apply the following transformation, which takes one matrix of any shape and returns a pair of two matrices of the same shape:

$$\mathsf{SubChnSplit}(\mathbf{C}) = [\max(\mathbf{C}, 0), \max(-\mathbf{C}, 0)]$$

For example, the following 3×3 block is splitted into two sub-channels:

$$\begin{bmatrix} +0.9 & -0.8 & -0.1 \\ +0.1 & +0.5 & -0.6 \\ +0.0 & -0.5 & -0.5 \end{bmatrix} \rightarrow \left[\begin{bmatrix} +0.9 & +0.0 & +0.0 \\ +0.1 & +0.5 & +0.0 \\ +0.0 & +0.0 & +0.0 \end{bmatrix}, \begin{bmatrix} +0.0 & +0.8 & +0.1 \\ +0.0 & +0.0 & +0.6 \\ +0.0 & +0.5 & +0.5 \end{bmatrix}\right]$$

With these two sub-channels, if we re-use the same filter from the last section, we will get:

$$\mathsf{SubChnSplit}\left(\begin{bmatrix} +0.9 & -0.8 & -0.1 \\ +0.1 & +0.5 & -0.6 \\ +0.0 & -0.5 & -0.5 \end{bmatrix}\right) \otimes \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} = [1.0, 1.0]$$

And

$$\mathsf{SubChnSplit}\left(\begin{bmatrix} +0.1 & -0.3 & -0.1 \\ +0.0 & -0.5 & -0.2 \\ -0.0 & +0.0 & -0.1 \end{bmatrix}\right) \otimes \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} = [0.1, 0.1]$$

Now there is a clear and significant difference in the resulting convoluted block. The opposite information does not cancel each other anymore, and both sub-channels serve as meaningful input to the model.

3. **Diagonal Extraction**: To solve the data redundancy problem, we will apply the following transformation, which reduce a matrix of size $20 \times 90 \times 90$ to a 2-tuple of sizes $(m \times 90 \times 3, m \times 87 \times 3)$:

$$\mathsf{DiagExt}(\mathbf{C}) = (\mathbf{C}_0, \mathbf{C}_1)$$

where for each channel $i = 1, 2, \cdots, m$, if

$$\mathbf{C}[i] = \begin{bmatrix} c_{00} & c_{01} & c_{02} & \cdots & c_{0(89)} \\ c_{10} & c_{11} & c_{12} & \cdots & c_{1(89)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ c_{89(0)} & c_{89(1)} & c_{89(2)} & \cdots & c_{89(89)} \end{bmatrix}$$

Then

$$\mathbf{C}_0[i] = \left[\begin{array}{ccc|ccc|cc} c_{00} & c_{01} & c_{02} & c_{33} & c_{34} & c_{35} & c_{66} \cdots & c_{87(89)} \\ c_{10} & c_{11} & c_{12} & c_{43} & c_{44} & c_{45} & c_{76} \cdots & c_{88(89)} \\ c_{20} & c_{21} & c_{22} & c_{53} & c_{54} & c_{55} & c_{86} \cdots & c_{89(89)} \end{array}\right]^T$$

and

$$\mathbf{C}_1[i] = \left[\begin{array}{ccc|ccc|cc} c_{03} & c_{04} & c_{05} & c_{36} & c_{37} & c_{38} & c_{69}\cdots & c_{84(89)} \\ c_{13} & c_{14} & c_{15} & c_{46} & c_{47} & c_{48} & c_{79}\cdots & c_{85(89)} \\ c_{23} & c_{24} & c_{25} & c_{56} & c_{57} & c_{58} & c_{89}\cdots & c_{86(89)} \end{array}\right]^T$$

Each 3×3 block in $\mathbf{C}_0$ is a self-correlation block of an atom, and each block in $\mathbf{C}_1$ is a correlation block between an atom and its next neighbor. Note that we do not extract the -1-offset diagonal since it is identical to the 1-offset diagonal due to the symmetry of correlation matrices.

This operation reduce the amount of values we need to deal with per data point from $20 \cdot 90 \cdot 90 = 162,000$ to $20 \cdot 3 \cdot 90 + 20 \cdot 3 \cdot 87 = 10,700$, which is more than 15 times reduction.

The last step is to combine these three transformation. Let

$$\mathsf{Transform} = \mathsf{SubChnSplit} \circ \mathsf{DiagExt} \circ \mathsf{FilterChn}$$

Then $\mathsf{Transform}$ transform the input shape as below:

$$(20 \times 90 \times 90) \to (4 \times 90 \times 3 \times 2,\ 4 \times 87 \times 3 \times 2)$$

Now with this refined dataset, we can proceed to describe our improved networks.

5.5 Aligned Distance Prediction Models

Making use of the data transformation, we design a template CNN, with has four variations differing by the activation function of the last hidden layer and the subsequent output scaling.

1. **Input:** pairs of shape $(4 \times 90 \times 3 \times 2,\ 4 \times 87 \times 3 \times 2)$.
2. 4 separate sub-models. For each sub-model:
   - 2.1. **Sub-input:** pairs of shape $(90 \times 3 \times 2,\ 87 \times 3 \times 2)$.
   - 2.2. For each sub-input channel:
     - **Convolution 1:** 32 filters of size 3×3, stride 3×3 (i.e. apply the filter to each 3×3 atom-wise correlation block).
     - **Convolution 2:** One filter of size 2×1, stride 1×1. This layer is to gather the features extracted in the previous layer and combine them.
     - **Max pooling:** Filter of size 3×1. This layer extracts the most prominent features from the last two layers.

- **Flatten:** Flatten the features into a 1D array. Shape: (10) for the first channel (main diagonal) and (9) for the second channel (offset 1 diagonal).

2.3. **Concatenate:** Feature array of shape (19).

3. **Concatenate** output of 4 sub-models: Feature array of shape (76).
4. **Fully connected 1:** 64 perceptrons with Sigmoid activation. Large number of perceptrons ensures no feature is missed out.
5. **Fully connected 2:** 64 perceptrons with Sigmoid activation. Another big layer to capture any deep features.
6. **Fully connected 3:** 1 perceptron with Sigmoid/Hyberbolic tangent activation.
7. **Transformation:** Unbounded/Linear scaling, as described in Section 5.2 in accordance to the above layer.
8. Output layer.

We name this structure `Model_Template`. Figure 13 summarizes `Model_Template`.

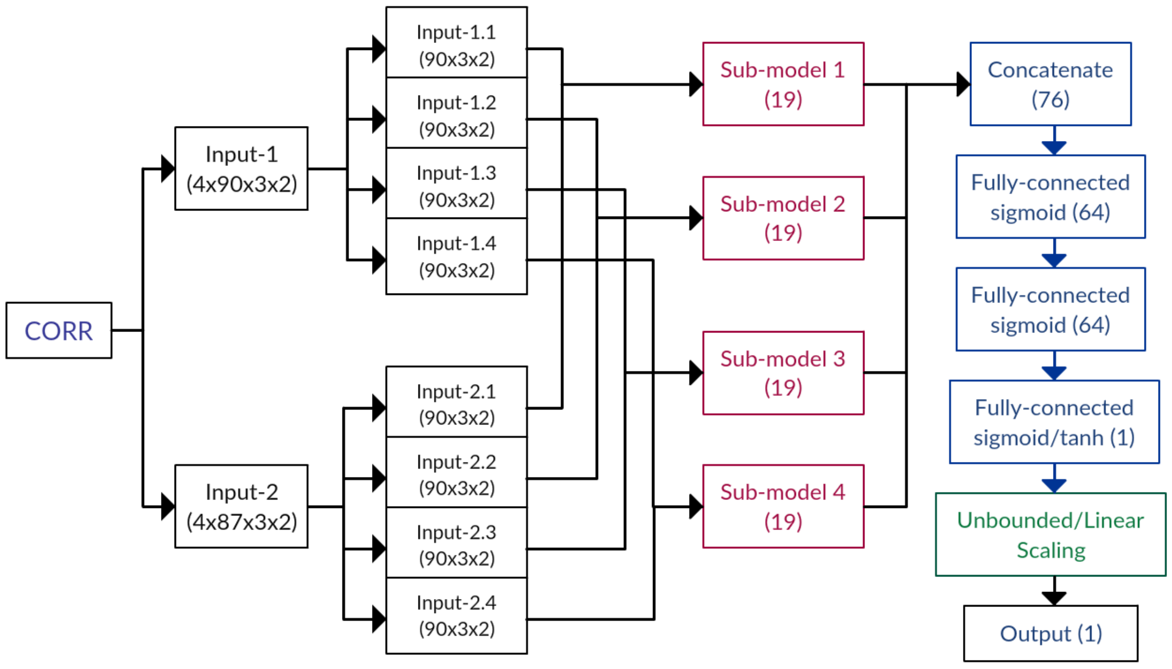


Figure 13: Structure of the main model.

The structure of each sub-model is shown in Figure 14

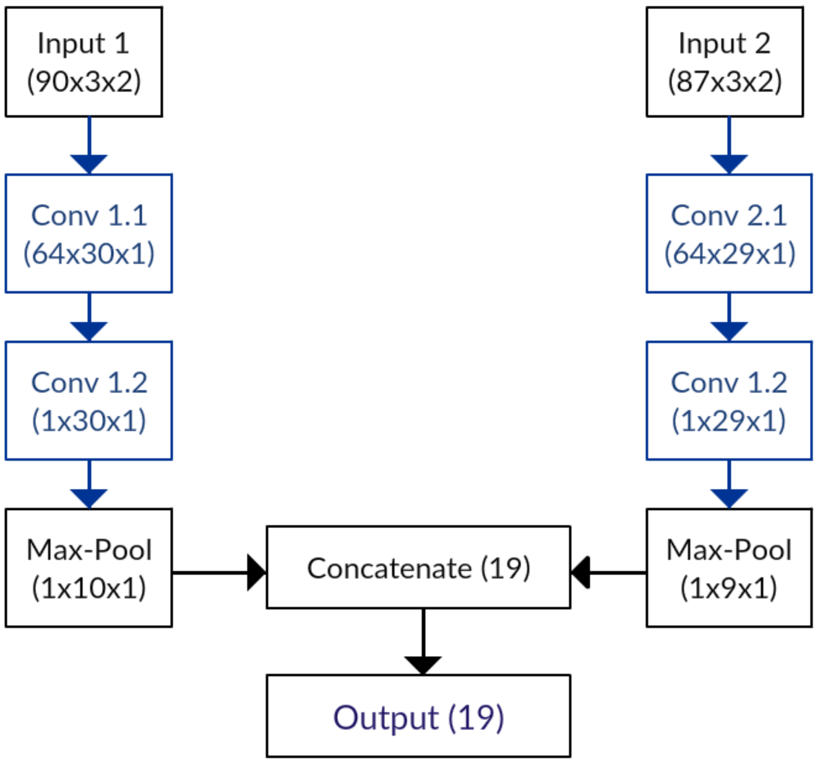


Figure 14: Structure of the sub-models.

We also specify names for the four variations of this template:

1. `Model_linear_sigmoid`: The last deep layer uses Sigmoid activation and Linear scaling.
2. `Model_linear_tanh`: The last deep layer uses `tanh` activation and Linear scaling.
3. `Model_unbounded_sigmoid`: The last deep layer uses Sigmoid activation and Unbounded scaling.
4. `Model_unbounded_tanh`: The last deep layer uses `tanh` activation and Unbounded scaling.

Now that we have four prediction models, we can push the idea further by combining their predictions to get a combined model. We discuss it in the next part.

5.6 Combined Model

The idea is to combine the output of the four models in Section 5.5 via a function to get a single combined output. Since all four are estimates of the actual label, the linear function is a natural choice as the combiner. We expect the optimal output to have the form

$$\text{out} = \alpha_{\texttt{ls}}\text{out}_{\texttt{ls}} + \alpha_{\texttt{lt}}\text{out}_{\texttt{lt}} + \alpha_{\texttt{us}}\text{out}_{\texttt{us}} + \alpha_{\texttt{ut}}\text{out}_{\texttt{ut}} : \alpha_{\texttt{ls}} + \alpha_{\texttt{lt}} + \alpha_{\texttt{us}} + \alpha_{\texttt{ut}} \approx 1$$

where `ls, lt, us, ut` are abbreviations for the model names in Section 5.5. In other words, we expect the output of the models to be in a small range around the label, hence some weighted average of them will lie in an even smaller range, thus we expect a linear combiner to improve the performance by some extent if well-fitted.
We denote the combined model by `Model_combined`. The structure of the combined model is shown in Figure 15

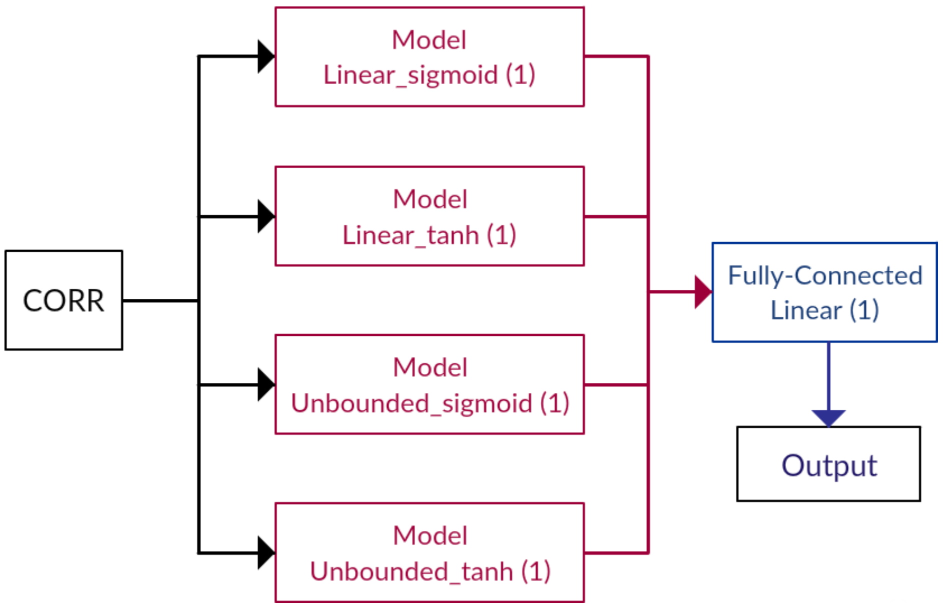


Figure 15: Structure of `Model_combined`

Before discussing the performance of the models in Section 6, we would like to explore extending further the predictive power of our models by deducing the *mutation significance* from them.

5.7 Extending Models to Predict Mutation Significance

The motivation behind this idea is to foretell whether a protein chain will cause serious damage to the cell after mutating. Our goal is to classify mutants in MUTANTS* into two groups, those that cause significant change in structure for RefState, and those that do not cause any noticeable difference. If the structure change is significant, it can be inferred that the protein chain will malfunction and cause damage to the cell that employs it.
Firstly, we must define our metrics for measuring the significance of the structure change. Naturally, we think again of the aligned distance AD $(\text{RefState}_C, \text{RefState})$. Our

idea is to set up a threshold value for $\mathsf{AD}\left(()\,\mathsf{RefState}_C, \mathsf{RefState}\right)$ and if this distance goes above the threshold, we declare the mutation has significant structural change. Therefore, our first task is to determine this threshold experimentally.

5.7.1 Choosing Distance Threshold

We perform the following one-time experiment:

Experiment 5.1. Given RefState, let us find the range and density of the equilibrium region, by looking at the deviation after simulating the folding process for 10^6 MC steps from RefState.

1. Start with RefState and WildType.
2. Repeat for $10,000$ independent loops. At each loop:
   (a) Perform Algorithm 2.1 for $N_{rep} = 10^6$ and $N_{sam} = 100$ to find a representative state $\mathsf{RefState}^*$.
   (b) Record $\mathsf{AD}\left(\mathsf{RefState}^*, \mathsf{RefState}\right)$.
3. Save the records for plotting later.

Results. The result of Experiment 5.1 is shown below:

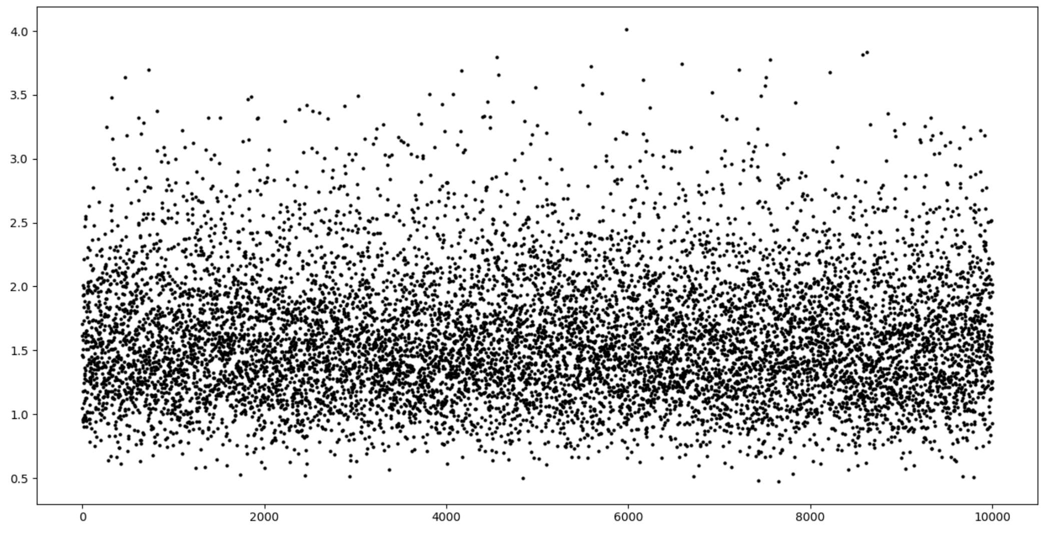


Figure 16: Distribution of the distance around the equilibrium.

The diagram shows that most of the distance lie within the range $(0, 3.5)$. This means in the equilibrium region for WildType, the distance from RefState can go up to 3.5 but very unlikely further. Therefore, we will pick the threshold value to be 3.5

The meaning of mutation significance is then formulated below.

Definition 5.2. A mutant $C \in \mathsf{MUTANTS}^*$ is declared **significant** if it causes noticeable structural change in the native state, indicated by

$$\mathsf{AD}\left(\mathsf{RefState}_C, \mathsf{RefState}\right) > 3.5$$

With this definition, we proceed to build our significance classifier in the next part.

5.7.2 Model Structure

For the classifier, we will use the values for the distance predicted by the four models in Section 5.5, and produce the predictive classification based on Definition 5.2. The structure is visualized in Figure 17.

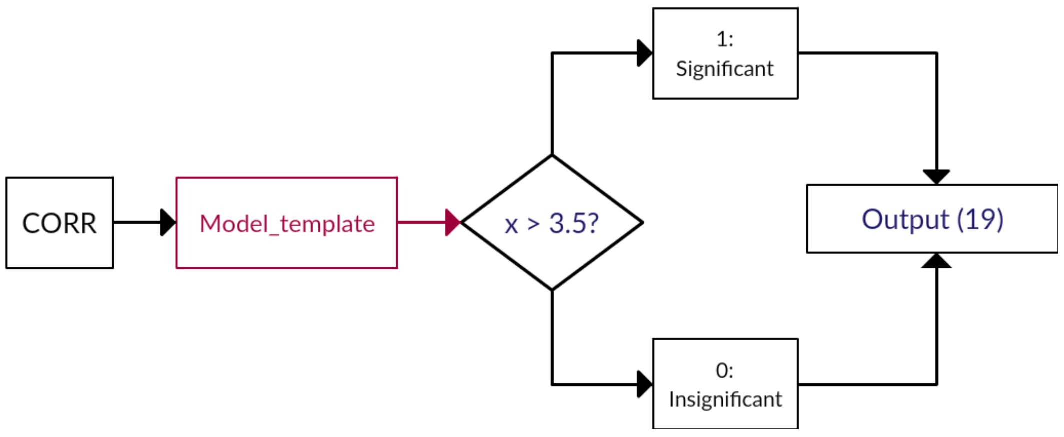


Figure 17: Structure of the significance classifier.

The metrics used for testing this classifier will be `accuracy` [10], measured by

$$\mathsf{ACC}(\text{Pred}, \text{Label}) = \frac{100}{n} \sum_{i=1}^{n} \mathbb{1}_{\{\text{Pred}[i]=\text{Label}[i]\}}$$

6 Implementation and Results

6.1 Implementation and Runtime

We implement our model in `Keras`, an increasingly popular Python package for Deep Learning [14], with uses `Tensorflow` backend [15].
For the train/test splitting, we use the API from `scikit-learn` [16], another popular Python package for scientific computing and machine learning [13].
Each model is train on 1000 epochs. As mentioned in Section 5.1, the training, validation and testing set have sizes 44290, 10573, 13216 respectively. Each batch is the

whole training set. We can afford such number of epochs and batch size thanks to the input reduction in Section 5.4, which reduce tremendously the time and memory consumption of the new models. In fact, the following table shows the training time and memory consumption for each model on a 24-core, 96GB-RAM NSCC node, which makes the new models the practical choice compared to the vanilla model.

Model	Time	Memory
`Model_combined`	≈ 2:15:00	≈ 17.4 GB
All other models	≈ 1:00:00	≈ 8.5 GB

6.2 Performance on Training and Validation

Let us consider `Model_linear_sigmoid` first. The training curves for this model, and the other three, look almost identical due to the first few epochs have large values that greatly scale up the y-axis. Following the idea in Vanilla Convolution Network, we only plot the curves for epochs from 500 only. The plots for `Model_linear_sigmoid` are shown in Figure 18.

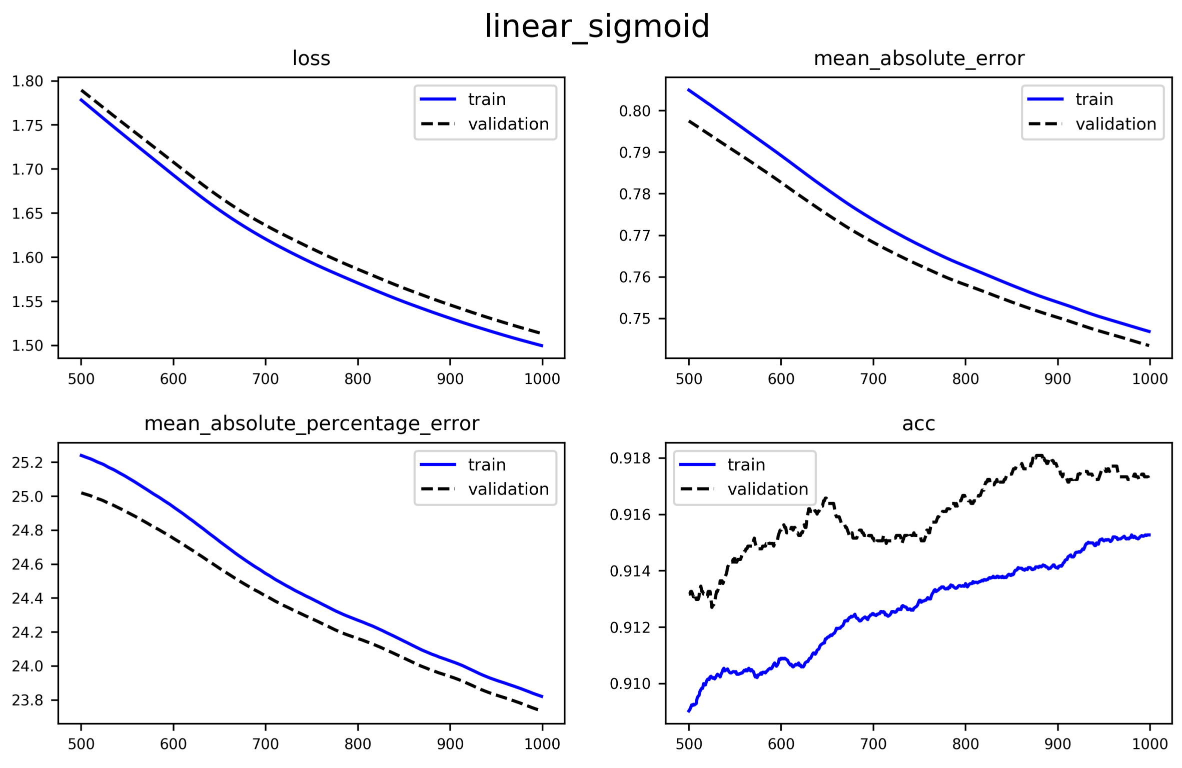


Figure 18: Training curves of `Model_linear_sigmoid` from epoch 500

The gradually decreasing loss and errors indicate that we can even train more epochs without over-fitting, but for our project the current loss is acceptable.
We are interested in two statistics: the loss and the accuracy on the validation set. Let us plot these statistics for all four models and make comparisons. The starting epoch

is again 500. The graphs are shown in Figure 19 and Figure 20.

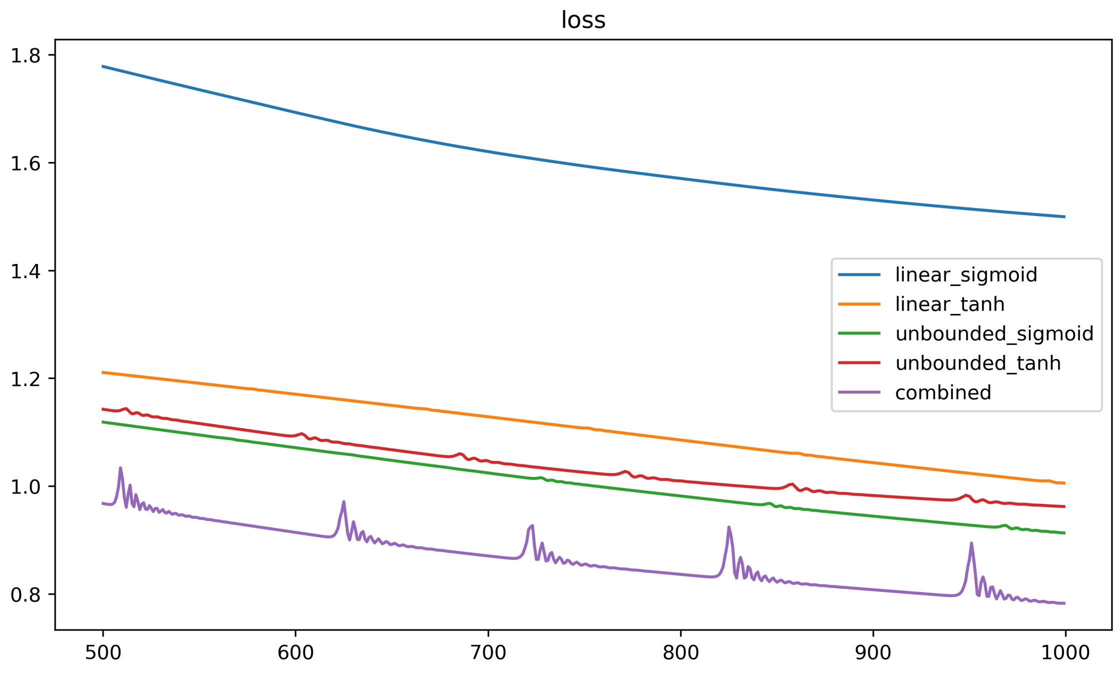


Figure 19: Validation losses of all models

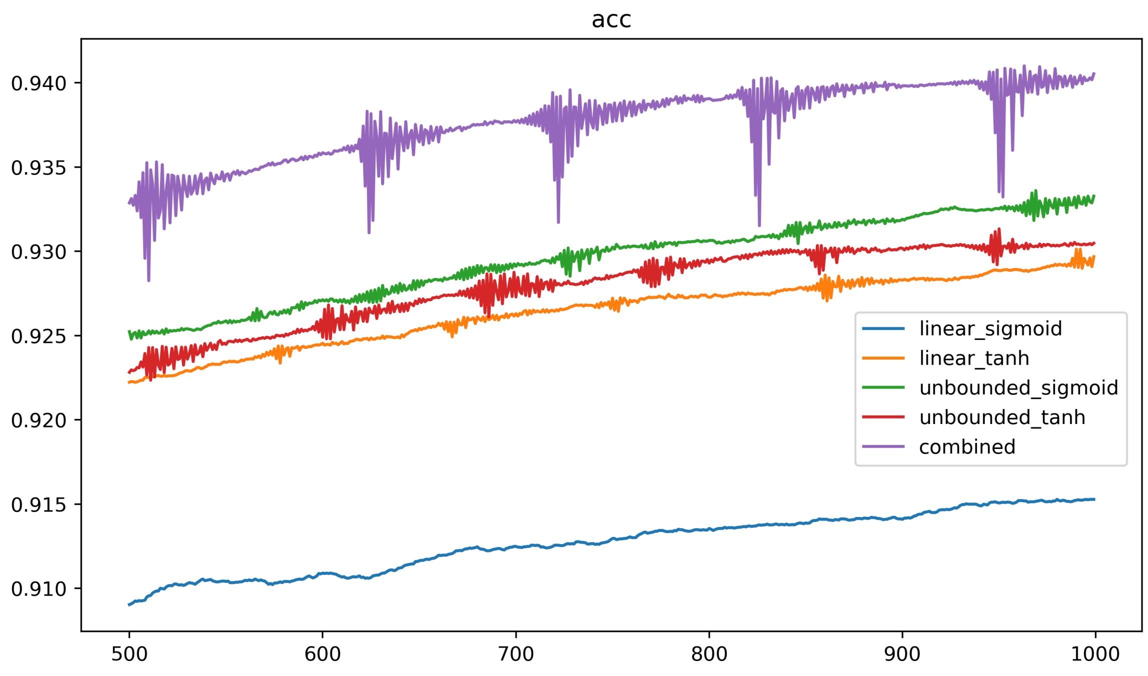


Figure 20: Validation accuracy of all models

From these figure, we deduce that:

1. `Model_unbounded_sigmoid` has the best average performance among the four variations.

2. `Model_combined` has the best overall performance, which matches our expectation. However, its loss and accuracy curves has periodic fluctuations. This could be due to its large number of trainable parameters: ≈ 56000, four times as many as one of the variations, ≈ 14000.

3. In terms of accuracy, while a 1%, e.g. $0.925 \to 0.935$ increase may seem negligible, since our data set has 66000 samples, it corresponds to more than 600 more mutants correctly classified. Hence it is reasonable to declare `Model_combined` the most accurate among all models considered.

To verify the hypothesis that `Model_combined` is the best among the models, we will look at their performance on the test set, which consists of ≈ 20000 data points.

6.3 Performance on Test

In Table 3 we show the performance on the test set:

Model	MSE	MAE	MAPE	Accuracy
`Model_linear_sigmoid`	1.484	0.743	23.908	91.745%
`Model_linear_tanh`	1.031	0.625	20.599	93.357%
`Model_unbounded_tanh`	1.008	0.624	20.596	93.523%
`Model_unbounded_sigmoid`	0.981	0.612	20.310	93.705%
`Model_combined`	0.925	0.597	19.780	93.833%

Table 3: Performances of models on the test data.

Again, the combined model achieves the best performance, i.e. lowest errors and highest accuracy, in all metrics. We conclude that this model's advantage is robust, and it is effective both for estimating the distance between the mutant's native state and the original RefState, and for distinguishing the significant mutations from the insignificant ones.

6.4 Randomized Testing Scheme and Results

As a final test of the models' robustness, we uniformly sample 2300 mutants from MUTANTS (set of all possible mutants from charges $\{0, \pm Q\}$), each of which is of the form $[q_1, q_2, \cdots, q_{30}]$, where

$$q_i = \begin{cases} 0 & \text{with probability } 1/3 \\ -4 & \text{with probability } 1/3 \\ +4 & \text{with probability } 1/3 \end{cases} \quad \text{for} \quad i = 1, 2, \cdots, 30$$

We give this test set a name: TestMutants. Again, we compute the aligned distance AD (TestMutants) using the long simulation (Algorithm 2.1) as labels, and correlation matrices CORR (TestMutants) using the short simulation (Algorithm 2.4) as input for this set. Finally, we apply the operations in Section 5.4 to get the reduced diagonal input data. The metrics used for evaluation are the same as those in the previous test. Table 4 shows the performance on TestMutants.

Model	MSE	MAE	MAPE	Accuracy
`Model_linear_sigmoid`	0.942	0.534	18.488	93.522%
`Model_linear_tanh`	0.726	0.495	17.626	93.739%
`Model_unbounded_sigmoid`	0.741	0.503	17.823	93.739%
`Model_unbounded_tanh`	0.764	0.512	18.278	93.565%
`Model_combined`	0.745	0.497	17.669	94.739%

Table 4: Performances of models on the randomized test set.

Interestingly, all the models perform well, even achieving less loss and more accuracy than on the designated test set. This implies MUTANTS* is a good representative of all the mutants in MUTANTS, which means our method of data generation (Section 4) achieves sufficient coverage of all the possible mutants.
`Model_linear_tanh` achieving the least loss this time, rather than `Model_combined`. However, in terms of classification accuracy, `Model_combined` exceeds all the others by at least 1%, hence we still declare it the practically best model.

7 Conclusion

In this section, we provide a brief summary of this paper.
In Section 1 (page 4), we provided some background about Protein chains and their folding process, including the definition of native states. We then describe our purpose and objectives in Section 1.2.1 (page 4), Section 1.2.2 (page 4), before quickly summarizing our solution in Section 1.3 (page 5).
In Section 2 (page 5), we described our model of the protein chain (Section 2.2, page 6) and the folding process as a long simulation (Section 2.3, page 8), followed by the definition and model of the short-term dynamics as the correlation matrices of the short simulations (Section 2.4, page 9).
After defining the models, in Section 3 (page 12) we formulated and express our problem as a regression task, which gives the short-term correlation matrices and asks for the aligned distance (minimum root-mean-squared distance) between the native states of the original and the mutated chain. We also defined the set of mutants this project

concerns in Section 3.1 (page 12), and discussed the choices of model parameters that signifies the influence of these mutants in Section 3.3 (page 15).
Next, we described our method to generate a sufficient dataset for training and testing our prediction models in Section 4 (page 24), including how to generate the mutants to best represent the whole mutant set, and provided justifications for it in Section 4.2 (page 25).
We then proceeded to describe our prediction models, which are CNNs with various configurations in Section 5 (page 29). We covered our training and testing scheme in Section 5.1 (page 29), our output rescaling methods to match the actual label range in Section 5.2 (page 30), our first-attempt network and its scalability problems in Section 5.3.1 (page 32), how we fixed them by reducing and refining the input in Section 5.4 (page 36). We then redesigned the network to exploit the new input, and introduced four variations in Section 5.5 (page 38). We also went further by combining these variations into one in Section 5.6 (page 40), and extending the models to help them infer the mutation significance by interpreting their output in Section 5.7 (page 41).
Finally, in Section 6 (page 43), we tested these new models, both on the designated test set and a randomized test set, and showed that the combined model has the best performance in most metrics. Hence, to conclude, we have proposed a neural network model that can predict the impact of the mutation on a protein chain of 30 atoms with high precision, while also telling whether the mutation causes significant structural changes with high accuracy. This model also trains fast on large datasets, with low memory consumption, thus is scalable into projects with more sophisticated proteins, more parameters and higher data volume.

A Appendix

A.1 Correlation Coefficient

In this section we define the correlation coefficient of two variables x and y. Equivalent definitions can be found in [11]. Suppose we have a dataset of n records. Let x and y be two fields of the records, where their per-record values are:

$$\mathsf{Val}(x) = \{x_1, x_2, \cdots, x_n\} \quad \text{and} \quad \mathsf{Val}(y) = \{y_1, y_2, \cdots y_n\}$$

Let the **mean** of x and y respectively be:

$$\langle x \rangle = \frac{1}{n} \sum_{i=1}^{n} x_i \quad \text{and} \quad \langle y \rangle = \frac{1}{n} \sum_{i=1}^{n} y_i$$

For any function of at most two inputs and at most one output $F(x, y)$, we treat $F(x, y)$ a new composite variable with the values

$$\mathsf{Val}(F(x,y)) = \{F(x_1,y_1), F(x_2,y_2), \cdots, F(x_n,y_n)\}$$

then the concepts of mean also applies analogously to $F(x, y)$.
Now we can define the **correlation coefficient** between x and y as:

$$\mathsf{corr}\,(x,y) = \frac{\langle (x-\langle x\rangle)(y-\langle y\rangle)\rangle}{\sqrt{\langle (x-\langle x\rangle)^2\rangle}\sqrt{\langle (y-\langle y\rangle)^2\rangle}} = \frac{\langle xy\rangle - \langle x\rangle\langle y\rangle}{\sqrt{\langle x^2\rangle - (\langle x\rangle)^2}\sqrt{\langle y^2\rangle - (\langle y\rangle)^2}}$$

This coefficient is intended to measure the strength of the relationship between x and y. For example, if $y = Ax + B$, i.e. a linear function of x, we have $\mathsf{corr}\,(x, y) = 1$.

A.2 The centroid method

We briefly describe the centroid method, which we apply to the samples from the equilibrium region to obtain the reference state. The goal of this method is to find the sample among them whose **sum of squared aligned distances** to the other is the minimum.

Definition A.1 (Aligned Distance)**.** Let $C = \{(x_i, y_i, z_i)\}_{i=1}^n$ and $C' = \{(x_i', y_i', z_i')\}_{i=1}^n$ be two atom chains, define the *spatial distance* between 2 atom chains as the root-mean-squared deviation (RMSD) of them:

$$d_s(P,P') = \sqrt{\frac{1}{n}\sum_{i=1}^{30}[(x_i-x_i')^2+(y_i-y_i')^2+(z_i-z_i')^2]}$$

Define the *aligned distance* between 2 atom chains as the minimum distance between 2 translate-rotation of them:

$$d(P,P') = \min\left\{d_s\left(R(P)+\vec{v}, R'(P')+\vec{v'}\right) \,\middle|\, R, R' \in \mathsf{Rot}(3), \vec{v}\in\mathbb{R}^3, \vec{v'}\in\mathbb{R}^3\right\}$$

where $\mathsf{Rot}(3)$ is the set of rotations in 3 dimensions.

Given a set of atom chains, called the $\mathsf{Samples}$, our goal is to find $C \in \mathsf{Samples}$ such that the sum of aligned distances from C to other samples are minimum among $\mathsf{Samples}$. The algorithm is described in steps below.

1. For any pairs of chains C, C' in $\mathsf{Samples}$, compute $d(C, C')$ using the RMSD minimization algorithm from Kabsch [12].

2. Find $C^* = \text{argmin}\left\{\sum_{C' \in \text{samples}} d(C, C') \mid C \in \mathsf{Samples}\right\}$.
3. Return C^*

The resulting C^* is the reference state we need to find.

References

Unpublished Reports

[1] K. Shiina, "Data analytics on non-equilibrium relaxation of proteins," 2017.

Protein and Folding Process

[2] D. Nelson, A. Lehninger, and M. Cox, *Lehninger Principles of Biochemistry*, ser. Lehninger Principles of Biochemistry. W. H. Freeman, 2008, ISBN: 9780716771081. [Online]. Available: https://books.google.com.sg/books?id=5Ek9J4p3NfkC.

[3] D. Whitford, *Proteins: Structure and Function*, 1st. Wiley, 2013, ISBN: 9780471498940.

[4] D. Martin, Ed., *Protein Synthesis: Methods and Protocols*, ser. Methods in Molecular Biology. 1998, vol. 77, ISBN: 9781592595631.

[5] J. D. Bryngelson, J. Nelson Onuchic, N. D. Socci, and P. Wolynes, "Funnels, pathways and the energy landscape of protein folding: A synthesis," vol. 21, pp. 167–95, Mar. 1995.

[6] Z. Li and H. A. Scheraga, "Monte carlo-minimization approach to the multiple-minima problem in protein folding," *Proceedings of the National Academy of Sciences*, vol. 84, no. 19, pp. 6611–6615, 1987. eprint: http://www.pnas.org/content/84/19/6611.full.pdf. [Online]. Available: http://www.pnas.org/content/84/19/6611.abstract.

Machine Learning

[7] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. arXiv: 1412.6980. [Online]. Available: http://arxiv.org/abs/1412.6980.

[8] "Mean squared error," in *Encyclopedia of Machine Learning*, C. Sammut and G. I. Webb, Eds. Boston, MA: Springer US, 2010, pp. 653–653, ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_528. [Online]. Available: https://doi.org/10.1007/978-0-387-30164-8_528.

[9] "Mean absolute error," in *Encyclopedia of Machine Learning*, C. Sammut and G. I. Webb, Eds. Boston, MA: Springer US, 2010, pp. 652–652, ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_525. [Online]. Available: https://doi.org/10.1007/978-0-387-30164-8_525.

[10] "Accuracy," in *Encyclopedia of Machine Learning*, C. Sammut and G. I. Webb, Eds. Boston, MA: Springer US, 2010, pp. 9–10, ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_3. [Online]. Available: https://doi.org/10.1007/978-0-387-30164-8_3.

[11] C. Zaiontz, *Real statistics using excel: Correlation: Basic concepts*. [Online]. Available: http://www.real-statistics.com/correlation/basic-concepts-correlation/ (visited on 11/08/2017).

[12] W. Kabsch, "A solution for the best rotation to relate two sets of vectors," *Acta Crystallographica Section A*, vol. 32, no. 5, pp. 922–923, Sep. 1976. DOI: 10.1107/S0567739476001873. [Online]. Available: https://doi.org/10.1107/S0567739476001873.

Python Libraries

[13] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[14] F. Chollet *et al.*, *Keras*, https://github.com/fchollet/keras, 2015.

[15] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 265–283. [Online]. Available: https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf.

[16] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "API design for machine learning software: Experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, pp. 108–122.