WARWICK
THE UNIVERSITY OF WARWICK


# Computational Methods for Studying Proteins: Differentiable Lattice Proteins and Predicting PROTACs

## CS913 Dissertation Report

**University ID: 1637919**
**Khaashif Riaz**
**Supervisor: Dr. Fayyaz Minhas**

**Department of Computer Science**
**University of Warwick**

**September 2021**

# Contents

# Abstract

This research project is a study on using computational methods to better understand proteins. The project is divided into two parts, the first which covers the build, implementation, and analysis of a differentiable learning model for protein structure prediction. The second covers the exploration of machine learning models for predicting PROTAC-protein binding pairs.

Differentiable Learning (DL) is a generalized form of the way in which neural networks learn and in recent years there has been an increase in DL-based libraries capable of modelling physical systems. These physical systems (e.g., particle systems) are governed by some sort of energy function, DL uses gradient descent-based methods to optimize over the energy function and find optimal states of the physical systems. In this paper, we build a differentiable learning model for lattice protein structure prediction as no such model yet exists. The intention here is to provide a foundational differentiable lattice model capable of handling all 3 of the protein folding research problems.

A DL model was built using Python library DiffTaichi, however in the end, the DL model although implemented correctly was not successful in predicting lattice protein structures. The decision was made to move on to another area of study as the DL approach was deemed unsuccessful.

The second part of this research project was inspired by existing machine learning methods capable of predicting drug compound-protein pairs in the human body, these methods (Neural Network and Support Vector Machine) were applied to PROTAC-protein compounds. PROTACs are alternatives to drugs, capable of binding to previously undruggable target proteins and highlighting them for disposal to the body's natural protein degradation mechanism – the proteasome.

The best performing model was then used to make predictions on the human proteome against all known PROTAC compounds. The intention for this model is to aid biologists in reducing the number of experiments that must be carried out in the lab to determine whether a PROTAC and protein bind.

**Keywords:** protein folding, PROTACs, differentiable learning, neural network, support vector machine

# Acknowledgements

I would like to take this opportunity to thank my supervisor, Dr. Fayyaz Minhas for his continuous guidance and education throughout this journey. My hope was for a research project in the realm of the natural sciences and Fayyaz has given me that opportunity. I have learned a great deal and will take forward the skills and knowledge that I have gained throughout this project.

I would like to thank my family for their support throughout academic life. I would also like to thank Nero Aziz, one of my closest friends, for his advice and aid throughout this research project.

# List of Figures and Tables

# 1. Introduction

Computational methods have been used to gain a better understanding of proteins for some time now. Gaining a deeper understanding of proteins and their interactions are beneficial to the field of biology, this knowledge can be used for example to design new types of drugs or even design proteins for specific tasks, e.g., a protein to break down plastics that are harmful to the environment. This paper will explore two different applications of computational methods applied to proteins, both of which are in the realm of machine learning.

**(1) Differentiable Learning of Lattice Protein Structures**; the first application involves creating a differentiable learning model to predict protein structures from their amino acid sequences. This framework aims to tackle all 3 research problems associated with protein folding, they are listed as follows, (1) protein structure prediction, (2) amino acid sequence design, and (3) the design of an energy function governing the protein folding process.

**(2) Predicting PROTACs**; the second application is based on a slightly different problem relating to drug discovery. It involves using machine learning to predict which PROTAC compounds will bind to a target protein to reduce experimental costs for biologists. PROTACs aim to harness the body's natural cellular waste disposal mechanism, the proteasome, and target redundant proteins that the body fails to remove independently.

## 1.1 Differentiable Learning of Protein Structures

Proteins are found in all cells and carry out essential functions that govern the operation of the human body. A protein's function is determined by its structure [1]. There are many types of proteins, each of a different shape and size, hence, each carrying out a unique function, e.g., Insulin, which regulates blood sugar levels, or Hemoglobin, which is involved in transporting oxygen around the body. Proteins do exist elsewhere in nature, however for the purpose of this research project, we shall focus on those that are known to be at least in the human body.

Proteins are made up of amino acids, which are organic compounds that bond with another to create a chain-like structure, this is referred to as a polypeptide chain. Amino acid chains are synthesized using genetic information, the chain initially exists in an unstable state, and eventually the amino acids interact with one another to form a folded protein structure, this process is referred to as protein folding. A polypeptide chain will fold into the structure that requires the least energy to maintain, this is determined by the forces acting between the amino acids. It is in the minimum energy structure that a protein can carry out its function.

Figure 1 shows a computer-generated model of a chain of amino acids (left) and the corresponding folded protein structure (right).

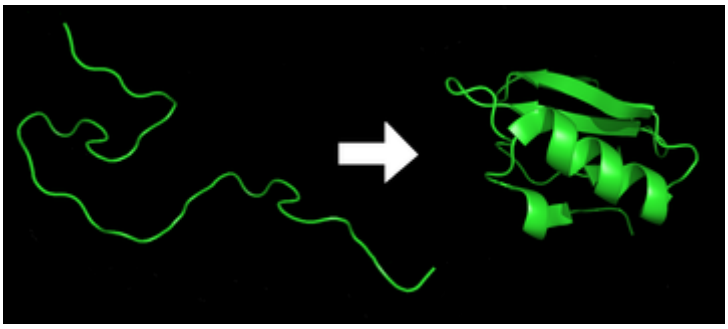

*Figure 1. Protein Folding*

Computationally, we can generate a 2D or 3D folded configuration of an N long chain of amino acids. According to Levinthal's Paradox, there is an astronomical number of configurations the chain may occupy [2], however in nature, the chain will fold into the configuration which has the minimum energy. This makes protein folding an optimization problem, hence the need for computational models to predict protein structures efficiently and accurately, design amino acid sequences and model folding pathways. The goal of the optimization problem can be visually understood by Figure 2, all possible states (configurations) exist on the horizontal axis, we wish to find the state with the lowest energy (global minimum). Fig. 2 is an oversimplification of the energy function; the true function is in fact unknown, making the optimization of it to be a computational challenge.

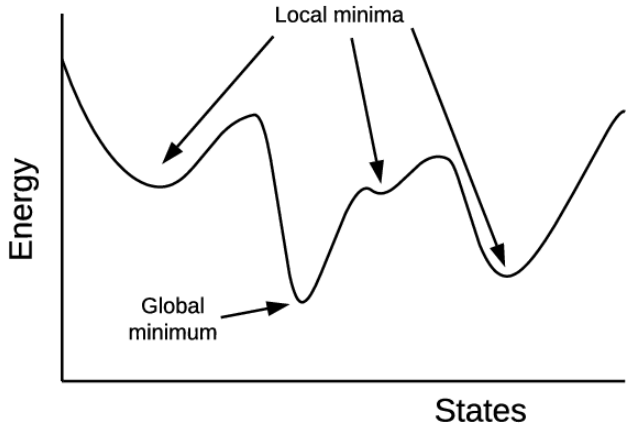


*Figure 2. Energy Function Minimization*

Many successful protein structure prediction methods exist and shall be expanded on in **2.1.** We shall take a Molecular Dynamics based approach (MD), this is a computer simulation of the physical and chemical processes that occur at atomic and molecular levels, typically it involves finding the optimal state of a system. The optimal state can be found by an optimization technique called Differentiable Learning, popularly used in MD applications, and for our purpose, will be used to

find the minimum energy state of a protein structure. This will be done by modelling the way in which amino acids interact with one another such that they eventually fold into a protein structure, this is discussed in **3.1** and **3.2**. Differentiable learning is an area within machine learning and is a more general form of deep learning, providing greater freedom over the function we wish to optimize.

This research will tackle three problems simultaneously, they are, (1) structure prediction; what structure will a chain of amino acids fold into, (2) amino acid sequence design; what combination of amino acids is required given the desired protein structure and (3) determining the energy function; how best to model the forces governing amino acid interaction. The same differentiable model can be used to tackle all 3 problems, which has the potential to simplify protein folding research. No such differentiable approach has been taken on *lattice* proteins to date, so the aim of this project is to build a working foundational differentiable model for lattice proteins, which can then be expanded in future works to lattice protein interactions and even real-world proteins.

## 1.2 PROTACs

PROTACs, or PROteolysis TArgeting Chimeras, are chemical compounds inserted into the body to help the degradation of redundant or dangerous proteins. PROTACs are even capable of targeting proteins that were previously not druggable [3].

Within the body there exists a mechanism called protein degradation; this is where a protein is broken down into amino acids [4]. The purpose of this is to remove any unwanted or potentially dangerous proteins from the body. One of the ways in which intracellular degradation (breaking down proteins within cells) can occur is by the ubiquitin-proteasome system, responsible for the process of ubiquitination. During this process, unwanted proteins are first tagged by a ubiquitin (a small regulatory protein), then the proteasome (a protein complex) breaks down the peptide chain bonds within the tagged protein, thus, degrading the protein.

A PROTAC consists of two ligands, one which binds to the target protein (warhead) and one which binds to an E3 ubiquitin ligase - there is a chemical linker to join the two. We shall focus on the former, the warhead, as this is the ligand responsible for binding with the target protein. Figure 3 shows the two ligands attached by a linker and how the ligands bind to the shapes of the target protein and E3 ligase.

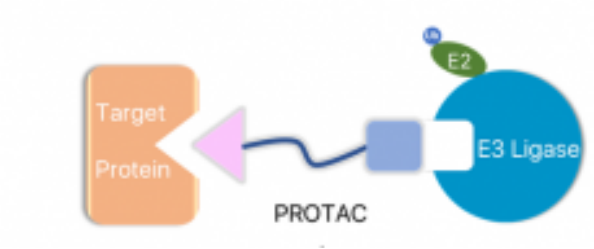


*Figure 3. PROTAC Diagram*

PROTACs can highlight unwanted proteins that the body's ubiquitination process fails to do so independently. They first must bind onto the target protein to be able to carry out ubiquitination, then the ubiquitin ligase will attach ubiquitin on to the surface of the target protein, highlighting to the proteasome for degradation. Given the large number of possible chemical and protein structures that exist, it is difficult to determine whether a PROTAC-protein pair will bind without carrying out an unfeasible amount of lab experiments between a target protein and many proposed PROTAC compounds.

Machine learning methods may be used to predict whether a PROTAC compound and target protein will bind or not as the research problem can be modelled as binary classification. An accurate PROTAC-target binding predictor has the potential to reduce workloads for biologists as they can first use the predictor on PROTAC-protein pairs, then select the pairs most likely to bind to carry out wet lab experimental validation of the model's prediction.

The aim of this research project is to build a binary classifier that will predict whether a PROTAC compound in SMILEs format binds with a protein sequence string. The classifier was trained with the human dataset from Tsubaki et al. [5]. The dataset consists of experimentally understood compound-protein interactions within the human body. The classifier was then tested against a dataset of known PROTAC-protein interactions, more on this will be explained in chapter **4**.

# 2. Literature Review

## 2.1 Differentiable Learning of Protein Structures

There is a wide variety of literature on computational models for protein folding. Over the years there has been a move towards deep learning methods as computational power has increased compared to the earlier statistical analysis methods used [6]. Proteins can be represented computationally at varying levels of complexity, starting at simple representations such as lattice proteins, right up to complex all-atom models. There have been no differentiable learning methods to date that tackle the lattice protein folding optimization problem. This literature review will mainly focus on the differentiable learning methods on proteins and existing optimization methods on lattice proteins.

### 2.1.1 Lattice Protein Optimization

HP Model:
The HP model is a simplified representation of real-world proteins, there are two amino acids, (H); hydrophobic and (P) hydrophilic [7]. The protein is represented on a lattice structure of 2 or 3 dimensions, an example can be seen in Figure 4. The differentiable learning algorithm built in this research project uses the HP model to represent proteins as it provides a foundation upon which to expand the model, such that it can optimize for more detailed protein representations.

Existing Lattice Structure Prediction Models:
The HP model presents an optimization problem, i.e., finding the minimum energy lattice structure. Several optimization algorithms have shown success in lattice structure prediction such as, simulated annealing, quantum annealing, Monte Carlo simulations, large neighbourhood search, and other methods [8], [9], [10], [11], [12]. While all these methods show success in lattice structure prediction, they lack the ability to model folding pathways that are representative of how proteins fold in nature. In these methods, the folded structure changes randomly at each iteration. This is unlike our proposed differentiable model which utilizes gradient descent with respect to the energy function, so with this model, we can observe how the structure changes at each iteration with respect to the energy interactions between amino acids.

### 2.1.2 Differentiable Learning

While the optimization methods mentioned are successful in predicting protein sequences and some also in designing amino acid sequences, they lack the ability to model folding pathways in a physics-based fashion. Previously mentioned methods are discrete optimization techniques, the differentiable learning approach is a continuous form of optimization, utilizing gradient descent to minimize the energy function. The continuous form of optimization allows us to

model folding pathways and observe the evolution of the amino acid chain into a folded protein structure.

Libraries for Differentiable Programming:
Differentiable learning utilizes differentiable programming to learn the optimal state of some system. Differentiable programming can calculate gradients of some programmatically defined function in an automated fashion. Differentiable learning can therefore be used to minimize the energy of a physical system using gradient descent. There have been libraries developed in Python to handle such physical systems, a couple popular ones are DiffTaichi or Jax MD. DiffTaichi is a library utilizing differentiable programming techniques for physical simulations [13]. Jax MD is similar to DiffTaichi, only it offers more built-in energy minimization functions for pre-defined physical systems [14].

Differentiable Learning for Protein Folding:
A couple differentiable approaches have been taken in tackling the protein structure prediction problem. End to end learning of protein structure using a differentiable approach achieved state of the art performance in predicting novel protein structures [15]. It has also been used to learn the optimal parameters that govern the energy function of the protein folding process, the parameters can then be used in predicting protein structures from sequences , the model was successful in matching experimentally observed results on small proteins [16].
While both methods are successful, they are computationally demanding as they run on "real-world" proteins, i.e., proteins modelled at a fine level of detail - multiple atoms per amino acid. When looking to explore other areas of proteins e.g., protein-protein interactions, starting with such computationally demanding models is not feasible. For this, we propose a differentiable model for simpler, less computationally demanding lattice proteins. This would provide a foundation for differentiable models of lattice proteins, especially useful when modelling large protein sequence folding and even other more complex phenomena, such as protein-protein interactions.

Alphafold 2:
It is worth mentioning Google DeepMind's recent success in solving the protein structure prediction problem, in line with experimentally understood structures [17]. Alphafold 2 is a type of neural network trained on protein sequence and known structures and is optimized over some loss function. While the results are a true milestone in the world of computational biology, the approach taken does not optimize over the energy function representing the physics of the protein folding process. This means the network is not quite capable of modelling folding pathways and observing the polypeptide chain's folding process. Our approach will differ in that it will optimize over the energy function and thus, model the folding pathways. Our model will also use simple lattice structures rather than ribbon representations.

## 2.2 PROTACs

Successful machine learning (ML) models exist for predicting whether a drug compound and target protein bind, however no such ML models exist for PROTAC compounds. There is plenty of motivation to build a predictor for PROTAC-protein compounds. PROTACs can target drug-resistant proteins and even have shown promise in tackling cancer diseases, immune disorders, viral infections, and neurodegenerative diseases [18]. It is also intuitive to assume a model can be adapted for a PROTAC compound given the existence of drug compound models.

The PROTAC-DB houses data on known PROTACs with their corresponding target proteins that have been experimentally obtained [19]. Datasets can be split into the various parts of a PROTAC, e.g., warheads, E-3 ligands, and linkers. These datasets contain chemical structure as well as various biological properties of the PROTACs, however, we are only interested in the chemical compounds of warheads, and the corresponding target protein sequences to which the warheads bind.

Machine learning has been applied to compound-protein interactions in the past, successful examples such as classical SVM [20] and even more recent deep neural networks [21] which have shown high accuracy (98%) on a large number of examples (4 million). Deep learning methods have gone as far as to combine GNNs and CNNs using end to end representation learning for compound-protein interaction prediction [5]. No such machine learning models have yet been applied to PROTAC-protein data. The high accuracy levels achieved with the referenced machine learning models provide motivation to apply similar models on PROTAC-protein data - this will be the focus of this research project. The human dataset consisting of compound-protein interactions in the human body from Tsubaki et al. [5] has been taken and is to be used to train our model.

A recent method of identifying the likelihood of a PROTAC binding with a target protein has been developed [22]. This method analyses current literature on a desired target protein and then based experimental variables derived from the literature, a binding confidence score is assigned. Although this project shares the same goal of supporting researchers in discovering PROTACs, the method by which this is done is different. So, it remains that no machine learning methods have yet been tested on PROTAC-protein data.

# 3. Differentiable Learning of Protein Structures

This chapter explores the capability of a differentiable model to solve all 3 research problems associated with proteins, protein structure prediction, amino acid sequence design and modelling the folding pathways. The model is developed in Taichi, and its performance is compared with a simulated annealing model for its known success in lattice protein structure prediction and amino acid sequence design.

This chapter will begin by outlining the theory behind core concepts, protein representation, simulated annealing and finally the focus, the differentiable learning model.

## 3.1 Theory

### 3.1.1 Lattice Proteins

Protein representation is of importance, generally the more detailed the protein representation, the more computationally demanding the model. The goal of this project is to assess whether differentiable learning is an ideal method for predicting protein structures. An ideal way to go about this is by starting with simple models of proteins and then, if successful, building up to more detailed ones. Our differentiable model utilizes simplified versions of real-world proteins, lattice proteins, essentially a chain of multiple amino acids on 2-dimensional lattice space.

Lattice proteins are typically on the HP model, that is, modelling a protein with two types of amino acids, Hydrophobic (H) and Polar (P). These amino acids replicate the hydrophobic effect, that is, where water-repelling molecules will stay close together in a water-based solution. The hydrophobic effect can be seen as the driving force behind the protein folding process in the HP lattice model. An example of a HP lattice model can be seen in Figure 4.

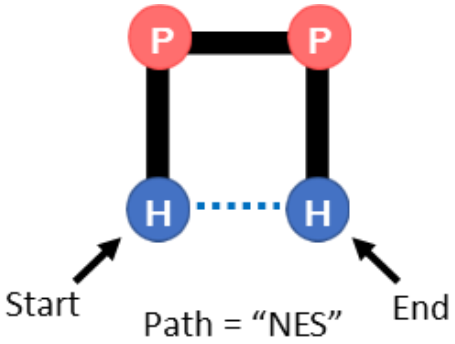


*Figure 4. Lattice Protein Structure*

Figure 4 shows a chain of amino acids, "HPPH", folded into its native (minimum energy) state, that is, the shape which maximizes H-H bonds. The bond has been drawn on as a dotted line between the H-H non-linked amino acids. On larger structures, H amino acids interact with one another in such a manner as to group at the core of the protein, hence, are the main factor influencing the shape into which a polypeptide chain folds.

The energy equation, H, is used to calculate the total energy of a HP lattice structure, it can be seen below:

$$H = \sum_{\{i,j=1\ldots n, i<j\}} E_{p_i,p_j}\,\delta(r_i - r_j) \qquad (1)$$

Values $r_i$ and $r_j$ are positional values for a pair of amino acids, $p_i$ and $p_j$ and the following constraints apply:

$$\delta(r_i - r_j) = 1 \text{ if } |r_i - r_j| \leq 1,$$
$$\delta(r_i - r_j) = 0 \text{ otherwise}$$

Energy potentials between specific types of amino acids have been chosen as follows:

$$E_{HH} = -3$$
$$E_{PP} = E_{PH} = E_{HP} = 0$$

Given that $E_{HH}$ is the only energy potential greater than 0, it can be assumed that H-H bonds are the only ones of interest in lattice models. The intuition behind this formula is to count the number of H-H bonds, the more H-H bonds, the lower the energy score of the protein structure. From a computational perspective, the goal is to take some sequence string of H and P amino acids and then find the structure that yields the most H-H bonds and thus, the minimum energy state.

### 3.1.2 Simulated Annealing (SA)

The simulated annealing method will be used to optimize over the energy functions for a set of sequences to find their global minima, the results obtained from this method will be used to benchmark against results from the differentiable model. It was chosen for its proven success and simplicity in implementation [8].

SA is a probabilistic method of optimization; it can approximate the global optima of some function and is ideal in scenarios where the search space is discrete, large and there is an incomplete understanding of the shape of the function [23]. In our case, considering lattice models, there are many possible protein structures that can be made from an N-long chain of amino acids. The shape of the energy function is also unknown, and a protein structure is defined by a set of discrete coordinates, lattice protein structure prediction fits the criteria for simulated annealing optimization.

The SA algorithm begins by generating a random solution, then at each iteration will consider a random neighbouring solution, if it is an improvement, the algorithm accepts it and repeats, otherwise the algorithm will accept it with some probability, $p = e^{\left\{-\frac{|\Delta energy|}{temp}\right\}}$, where $p$ is dependent on the temperature of the system. There is a cooling constant associated with the algorithm, this decreases the temperature of the system at each iteration, such as to slowly decrease the likelihood that the algorithm will accept worse solutions as the space is searched through, aiding convergence to the global minimum.

Figure 2 shows the global minimum we wish to find with SA, although the energy landscape is likely to be a discrete function, rather than the continuous one shown. Figure 2 is merely an abstraction to aid the reader's understanding of the problem.

### 3.1.3 Differentiable Learning

Differentiable Learning is a method of optimization based on automatic differentiation. Automatic differentiation is self-explanatory, it involves using the computer's ability to calculate derivatives to compute gradients of some predefined function quickly and efficiently [24]. The ease of derivative calculation allows us to harness gradient descent as a method of optimization for our research problem.

Neural networks use differentiable learning to minimize some loss function during the learning process. Neural networks learn the ideal mathematical parameters to map inputs to outputs [25]. These parameters can then be used to predict outputs of some unseen inputs, this is useful for predicting outcomes, hence why neural networks are often used for tasks such as classification of inputs or predicting future states of some system.

Our model is a variant of neural networks, it is similar in that it utilizes differentiable learning to find the optimal parameters of a function. The only

difference is in the function itself, instead of learning weights and biases of some network to minimize a loss function, our model will learn protein structures to minimize an energy function. Figure 5 graphically shows the gradient descent process, the function J(w) in the case of a neural network would be the loss function, that is, the difference between predicted and target values. In the case of our differentiable learning model, J(w) represents the energy function and $J_{min}(w)$ is the native (minimum energy) protein structure. Of course, the energy function has been oversimplified to a convex function for understanding purposes.

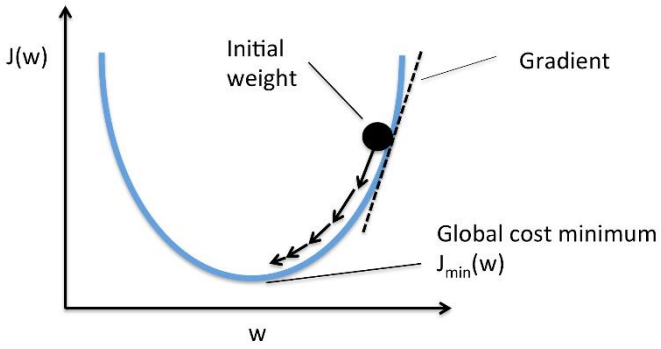


*Figure 5. Gradient Descent*

If the model finds the global optimum of an amino acid sequence's energy landscape, then it has found the native state into which that sequence will fold - we can then say it has predicted an ideal protein structure. The goal of the model is as it is for SA, find the global optimum of the energy function, only here we are using differentiable programming based gradient descent to do so.

Suppose at some timestep $t$, we have the protein structure $x_t$, then at timestep $t + 1$, we would now have the protein structure at t+1:

$$x_{t+1} = x_t + \alpha \frac{dE(x_t)}{dx_t} \tag{2}$$

where $E(x_t)$ is the energy function and $\alpha$ is some constant controlling the rate at which the algorithm moves along a curve. The algorithm repeats steps until we reach some time step $t_*$ where $x_{t_*}$ corresponds to the minimum energy structure.

Once the algorithm has been implemented, the intention is to adapt it so it can also solve the inverse problem. The inverse problem is the second research problem associated with protein folding, amino acid sequence design. This can be done by simply by reversing the inputs and outputs of the model, inputting some protein structure, and then outputting some prediction of the ideal amino acid sequence. The third problem can also be considered, modelling the folding pathways, the model will record each set of coordinates at each time step of the algorithm. The coordinates can then be plotted in a continuous fashion, and one would be able to

visually observe the folding pathways caused by the interactive nature of the H amino acids, thus, handling all three protein research problems with one model.

# 3.2 Methodology

## 3.2.1 Programming Languages

The model was built in Python, this was chosen for its widely accessible libraries on differentiable programming. Libraries such as Jax MD, DiffTaichi and TensorFlow all showed promise for building a differentiable model. Other libraries were used for computation, especially for the simulated annealing problem, such as Numpy for mathematical calculations and data manipulation. Matplotlib was used to plot lattice protein structures onto.

## 3.2.2 Lattice Protein Representation

There are two main components for programmatically representing a lattice protein structure:

- The sequence of amino acids to determine the order of H and P amino acid arrangement.
- A set of coordinates to plot onto some grid to represent the lattice protein structure.

The sequence is simply stored as a string of n "H" and "P" characters, e.g. "HPPH" and the coordinates would be stored as a Numpy array of 2D arrays, e.g. ((0,0), (1,0), (1,1), (0,1)). Note, the order of the coordinates is important as each point corresponds to the respective character within the sequence string.

Now that it is understood how the proteins are represented, we can move onto the more technical aspects of the SA algorithm.

## 3.2.3 Simulated Annealing

The input for the SA algorithm is the sequence string and the output is some set of coordinates corresponding to the predicted native protein structure.

The main functions created and involved in the algorithm are defined as follows:

---

**randomFold()**
input = amino acid sequence string
output = randomly folded protein coordinates

**drawFold()**
input = randomly folded protein coordinates
output = 2-D matrix representing H and P positioning

**energy()**
input = 2-D matrix representing H and P positioning and sequence
output = energy score (integer)

**randomFoldChange()**
input = protein coordinates
output = protein coordinates after small random change

---

The pseudocode for the simulated annealing algorithm is described below.

**Simulated Annealing**

```
1. Initialize protein structure path using randomFold(), drawFold()
2. initial_energy = energy(initial_path)
2. for all iterations:
5.      temperature = temperature * cooling_rate
6.      new_path = randomFoldChange(initial_path)
7.      new_energy = energy(new_path)
8.      if new_energy < initial_energy:
9.              initial_path = new_path
10.     initial_energy = new_energy
11.     else:
12.             prob = exp(-|initial_energy – new_energy|/ temperature)
13.             if prob > random.uniform(0,1):
14.                     initial_energy, initial_path = new_energy, new_path
```

The algorithm first generates some initial path, for example, for a 4 long sequence "HHPH" it may generate "NENW". Then for each iteration the algorithm slightly changes this path, say to "NEEN", and compares the corresponding energy values. If the new path scores a lower energy, then the path and energy value are updated. Otherwise, the path and energy value are only updated if the probability $p = e^{\left\{-\frac{|\Delta energy|}{temp}\right\}}$ is greater than some random value between 0 and 1.

The temperature also will decrease across iterations because the cooling rate is a value below 1, this will cause the algorithm to converge the energy to some minimum value. Coordinates and energy values are stored at each iteration, once the algorithm has carried out all iterations, we then take the coordinate set that yielded the minimum energy score and use this as the algorithm's prediction for the native energy structure.

The appropriate values for starting temperature and cooling rate were unknown, so parameter tuning had to be carried out to figure out the best values to use for these. Values between 10 and 250 (increments of 10) were trialled for temperature and 0.95 and 0.999 (increments of 0.003).

A similar method was followed for solving the inverse problem, that is, predicting the optimal amino acid sequence given a protein structure such that it will yield

minimize the energy score. For this problem, functions randomFold(), drawFold(), and randomFoldChange() are changed to carry out the same process but on sequences, not folds. Note: *the terms path and fold are used interchangeably*. The pseudocode follows similarly as for the protein structure prediction problem, only this time we are trying to predict amino acid sequences. Parameter tuning was also carried out here, testing the same range of values for temperature and cooling rate.

### 3.2.4 Differentiable Learning Model

The model was built in Python; libraries JaxMD and DiffTaichi were considered as they had differentiable programming capabilities, making them ideal for building a differentiable learning model. An attempt at building the model was first made using JaxMD, after its failure, DiffTaichi was eventually used. This section will explain in more detail reasoning behind library choice as well as explaining the technicalities of the model's implementation.

Given that this model is intended to be used as a foundation upon which advanced differentiable protein folding models may be built, its accuracy is of great concern. Accuracy is crucial especially if lab experiments are to rely on computationally simulated protein tests before carrying out physical biological tests. Of course, the model is far from that stage however the goal here is to build a strong foundation upon which to develop advanced iterations of the model, capable of handling real-world proteins. To test our model's accuracy, its results are to be compared with the simulated annealing model's results.

The model must also be scalable as protein sequences can span up to great lengths, e.g., the protein Titin, estimated to be around 35,000 amino acids long according to the UniProt database [26]. With scalability being of significance, computational efficiency must be considered carefully when designing the model.

#### 3.2.4.1 Jax MD Implementation:

Jax MD is the Molecular Dynamics (MD) specialized version of the Jax library. It has in built functions capable of modelling particle systems and has energy minimization functions available to use that are built on automatic differentiation principles. Protein structure prediction is essentially an energy minimization problem, thus, it made sense to explore the built-in energy functions that Jax MD had to offer.

The Jax MD energy minimization function had two inputs: particle coordinates and an energy function. The energy minimization function is defined below:

**energy_minimization( )**
input = particle coordinates, **energy( )**
output = minimum energy coordinates

The role of **energy( )** was to calculate the energy value of the given arrangement of particle coordinates. There was limited freedom over the design of the energy function, the minimization function required **energy( )** to be in a specific format, this format could only be achieved by using other pre-existing Jax MD functions. None of these pre-existing functions had capabilities to handle a lattice like structure, nor the sequential nature of the coordinates of amino acids (particles). This made it difficult to use any of the pre-existing functions to design our energy function. Attempts were made to design our own version of the pre-existing functions, however it meant we had to dig too deep into source code to make this work which proved to be technically challenging and time consuming. Instead, the decision was made to explore another library, DiffTaichi, which offered much more freedom in particle system representations and energy function design.

Jax MD is an ideal library to use where particle systems are free flowing, such as in liquids or gasses, or even particle systems such as densely packed solid structures. The lattice like structure of the protein seemed too intricate to be represented with Jax MD's library. Perhaps this is a misrepresentation of the Jax MD library and if there were more time, we could explore more in depth how to model our lattice structures successfully.

#### 3.4.2.2 DiffTaichi Implementation

DiffTaichi is a recent differentiable programming library aimed at building efficient differentiable physical simulators. The library offered a simple explanation on how to carry out gradient descent which was used to build our differentiable model. In comparison to Jax MD, DiffTaichi gave a lot more freedom over the design of how we wish to represent our particle system (lattice protein) as well as how we design our energy function.

There are three main areas to consider for the DiffTaichi implementation, (1) how we represent the lattice proteins for the model, (2) the design of the energy function, and finally (3) the differentiable learning algorithm itself. All three areas will be explored in this section.

1. **Lattice Protein Representation**

The protein is represented still as a lattice structure; however, the model will optimize not over coordinates of the structure directly, but rather over the path that the amino acid chain takes. The protein can be represented as a 2-D self-avoiding walk and so has an associated path. Directions North, East, South and West are used to describe the path. We will find the optimal path using gradient descent over the sequence's corresponding energy function. We cannot perform gradient descent over a string character path, e.g., "NEENWS", so instead we use numeric angles to represent the path, rather than discrete string characters.

The angles representing each of the directions are as follows:

- North = 0°
- East = 90°
- South = 180°
- West = 270°

Suppose we have a protein path such as "NWNEESS", we would then represent this with the following angular array: {0, 270, 0, 90, 90, 180, 180}. Now, we can perform gradient descent over the energy function with respect to the angular array, we could not do this with respect to the string character path.

In Figure 6 below, we have an example of a lattice protein structure with its corresponding string path and angular array. The path starts at the bottom left with the blue arrows indicating the direction of the path.

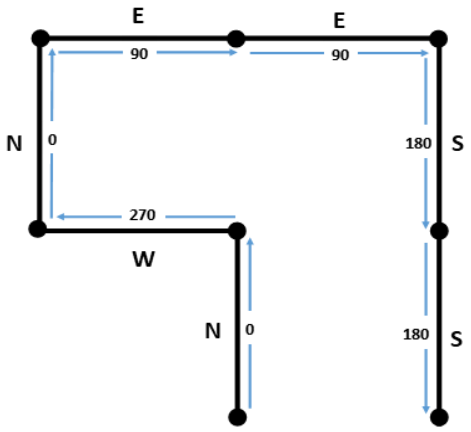


*Figure 6. Angular Array for Lattice Protein Path*

The Taichi language requires information to be stored as vector fields rather than more commonly used Numpy arrays. The angular path is therefore stored as a 1-D vector field, where each vector within this field is an angle, $\theta_i$. The angular array can be represented as
$\theta = [\theta_1, \theta_2, \ldots, \theta_n]$, where n is the number of directions of the path, so in the case of Fig. 5, we have $n = 7$.

### 2. Energy Function

The energy function is the most important aspect of the differentiable learning process. It is necessary to correctly model the energy interactions in a lattice protein so that the model can correctly predict protein structures.

Energy is calculated in a different way in the differentiable model than it is in SA model. The intuition is the same, calculate the number of H-H bonds of interest, however the Taichi library requires us to abide by the differentiable style of programming.

The pseudocode for our energy function can be seen below:

**Differentiable Energy Function**

```
1. input = θ      # input = theta
2. coords[0] = [0,0]
3. for i in range(1,N):
4.       coords[i] = coords [i-1] + [ti.cos(θ[i-1]), ti.sin(θ[i-1])]
5. for i,j in ti.ndrange(N,N):
6.       if i!=j:
7.             displacement = displacement between coord i and coord j
8.             if displacement = 1:
9.                   if i and j are HH bond of interest:
10.                        U -= 3
11.            elif 0 < displacement < 1: # amino acids too close
12.                  U += too_close_penalty
13.            elif displacement = 0:
14.                  U += overlap_penalty
15. for i in θ:
16.      U += β*min((θ[i]-0)^2, (θ[i]-90)^2, (θ[i]-180)^2, (θ[i]-270)^2)
```

In the pseudocode above, there are 3 main parts, the first for loop (line 3), responsible for converting $\theta$ into 2-D coordinates. The second for loop (line 5), which assesses all pairs of coordinates and either awards or penalizes the energy score. The final loop (line 15) penalizes the energy score if the path deviates too far from a lattice structure. Each of these for loops is broken down into more detail for the remainder of this section. Any functions from the Taichi library used start with "ti."

Looking at lines 34, the goal is to convert our angular path $\theta$ into a set of 2-D coordinates, such that they are compatible with the Taichi language. The first coordinate is always set to [0,0]. In line 4, each angle $\theta_i$ is passed through [ti.cos($\theta_i$), ti.sin($\theta_i$)] and added onto the last coordinate because the distance between each coordinate (amino acid) in the chain is always 1, so each coordinate is dependent upon the preceding coordinate in the chain.

Looking at lines 5-6, we can see that we loop through each pair of coordinates using ti.ndrange(N,N) where N is the length of the set of coordinates. Displacement between each pair of coordinates is calculated and used to determine how the amino acids at the coordinates interact with one another. There are then 3 main conditional if-elif statements to determine the nature of the bind, the first being the H-H bond of interest and awarding -3 to the energy score. The second being displacement between 0 and 1, if the amino acids are too close to one another, a penalty is applied to the energy score. The third is if the displacement is equal to 0, then we assume the lattice structure is not self-avoiding as amino acids are overlapping and apply a penalty for this.

Lines 15-16 loop through the angular array $\theta$, and apply a penalty based on the angular path's deviation from a typical lattice structure (i.e., values 0, 90, 180 and

270). The differences between $\theta_i$ and each of the angular directions is squared and then the minimum of all of these is taken as penalty. Before applying the penalty to the energy score, it is scaled by parameter $\beta$.

There are several parameters associated with the energy function, too_close_penalty, overlap_penalty and $\beta$. Parameter tuning is carried out on these amongst others used in the model which are discussed in the next section.

## 3. Algorithm

Now that lattice protein representations and the energy function are established, we shall delve into how the differentiable learning algorithm works. There will first be pseudocode followed by an explanation and reasoning behind design choices.

Taichi provides examples of how to implement a simple differentiable learning algorithm, this is essentially gradient descent to find some local optima. We wished to minimize the energy function; therefore, the goal was to find the minimum point on the energy curve for some protein sequence, this will give the native structure which that sequence will fold into.

First, we take some random angular path and calculate its energy score. We then proceed to update the existing angular path using gradient descent such that the new path is $\theta_{new} = \theta_{old} + \alpha \frac{dE}{d\theta_{old}}$ where $E$ is the energy function. We then repeat the process and calculate the new angular path's energy score and so on. We do this for a certain number of iterations, enough to allow the model to both explore the space well enough before convergence.

The pseudocode for the differentiable learning algorithm is below:

**Differentiable Protein Structure Prediction**

1. initialize seq = "HPPHP"
2. initialize angular_array = $\theta$ # input = theta
2. $\theta$_list = []
3. E_list = []
4. for k iterations:
5. E = compute_energy($\theta$, seq)
6. $\theta = \theta$ - $\alpha$*E.grad($\theta$)
7. store $\theta$ and E at kth iteration
8. prediction = $\theta_{min}$ = $\theta$ at index of min(E_list)

The pseudocode above shows a simple implementation of gradient descent. The parameters iterations and $\alpha$ were predefined. Parameter tuning was carried out for $\alpha$ to determine the optimal value, this parameter controls the speed of the gradient descent, that is, the rate at which it moves along the energy curve at each iteration. 2000 iterations were deemed enough for the model for convergence to some minima.

In lines 1 and 2, the input values are initialized, first a random angular path, $\theta$, is generated for the sequence, seq.

Line 5 computes the energy score for path $\theta$. Line 6, the important gradient descent step is carried out and $\theta$ is updated. Line 7 stores the values $\theta_k$ $and$ $E_k$ in $\theta$_list and E_list at each iteration k. Finally, line 8 shows the algorithm's prediction of an energy structure, or angular path, is obtained by taking the value of $\theta$ which had the minimum corresponding energy score.

It was initially trialled on small sequences, up to size 6 to assess its success in predicting energy scores. There was also a version of the algorithm that incorporated a momentum variable in the gradient descent step, the motivation behind this being to avoid getting stuck in local minima. Results are compared with the SA algorithm in **3.3.**

# 3.3 Results

## 3.3.1 Simulated Annealing

A model for structure prediction was built using the SA algorithm. On a protein sequence of length 6, "HPHHPH", the algorithm predicts the energy structure in Figure 7 with an energy of -6, the corresponding directional path is "ESWSE", starting at coordinate (0,0). The blue amino acids represent hydrophobic amino acids. The two bonds can be seen clearly on the diagram as green dashed lines. As can be seen in Figure 8, the model converged at just over 100 epochs. The parameters that yield this result were: temperature = 100, cooling rate = 0.95.

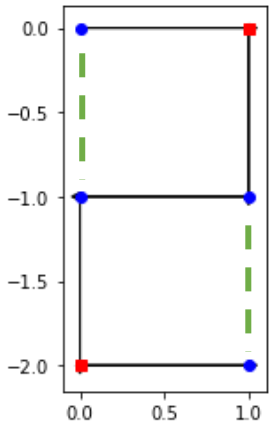


*Figure 7. Optimized Lattice Structure (Short)*

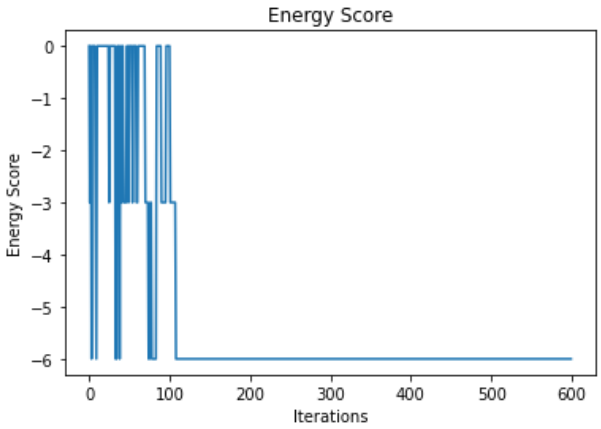


*Figure 8. Energy Score Plot (Short)*

The same model was used to predict a longer sequence of length 20. The sequence is as follows "HHPPHHPHHPPHHPHHPPHH", and the corresponding protein structure can be seen on Figure 9. The directional path is "WWNEEESEESWWSWWWNEE" The corresponding energy score is -30. The parameters that yield this result were: temperature = 130, cooling rate = 0.974.

Figure 9 shows the protein structure, we can see the hydrophobic amino acids group together, at the core of the protein, supporting the theory of protein folding, where water-repelling molecules tend to stick together, at the core of the structure as proteins are usually in some aqueous solution.

Figure 10 shows the change in energy score across iterations. We can see it takes around 440 iterations for the model to converge, more iterations than for the short sequence, this is likely because it is a large sequence and there is a larger solution space for the model to explore. The abrupt jumps in the first 100 epochs can be attributed to the probabilistic nature of the algorithm, it moves to a different conformation with some probability and does not converge until later iterations, as can be seen by the step like line beyond 200 iterations.

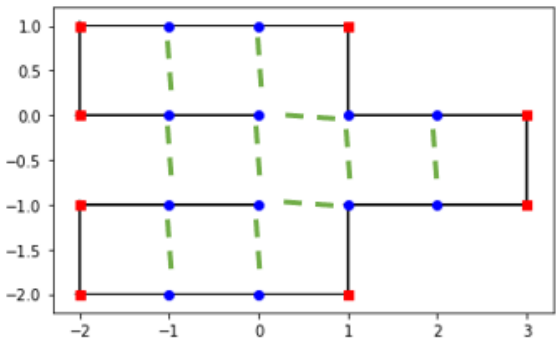


*Figure 9. Optimized Lattice Structure (Long)*

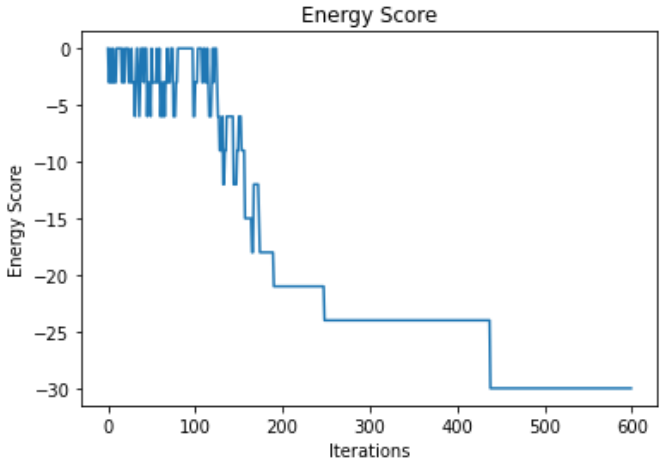


*Figure 10. Energy Score Plot (Long)*

The simulated annealing algorithm was also expanded to carry out sequence design on a given protein structure. The structure "NNNESSENN" was given and predicted a sequence of "HHHPHHHPHH", this yielded an energy score of -12. Figure 11 shows the given protein structure with the predicted amino acid sequence. Through simple observation of the structure, it can be identified that 4 bonds can be created, as highlighted on the diagram. The model correctly predicted these bonds, and the sequence it predicted is dominated by the H amino acids. There is nothing in place to prohibit the model from predicting an all-H amino acid chain e.g., "HHHHHHHHHH", easily minimising the energy score of any given structure. If the model were to be expanded, research would have to be done into whether an all-H sequence is valid for HP models of proteins.

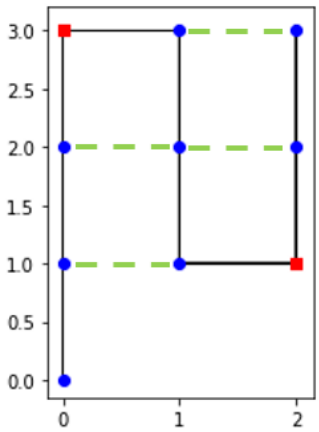


*Figure 11. Optimized Sequence*

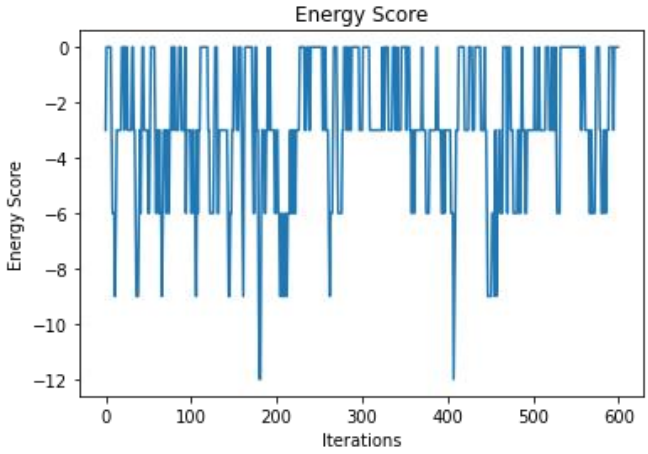


*Figure 12. Energy Score Plot (Sequence Design)*

Figure 12 shows no convergence to the global minima, unlike is the case with the protein structure prediction problem. To compensate for this, the minimum energy score and corresponding protein structure were taken. Again, if this model were to be expanded, further investigation would have to be done into why no convergence is seen.

### 3.3.2 Differentiable Learning

The sequences used in the DL model was given the same 6-long sequence as the SA model, "HPHHPH". Suppose we have a random initial structure as seen in Figure 13 (middle), the model's prediction is then the structure on the right, not the optimal state. The optimal state expected for the given sequence is Figure 7. Upon observation of all outputs generated by the model at various initial states, for all possible hyperparameter combinations trialled, the model failed to overcome local minima. Figure 13 (left) shows the algorithm gets stuck around a certain local minimum and fails to find the global optimum. Please note, the amino acids have not been colour coordinated on to the diagrams in this section.

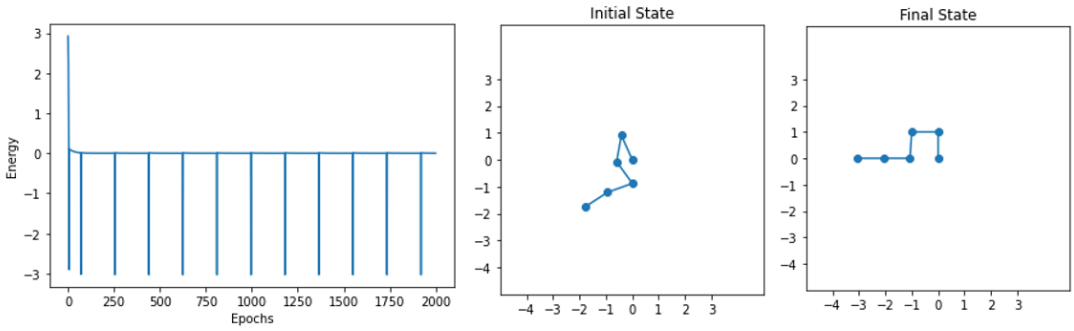


*Figure 13. Differentiable Results (Short 1)*

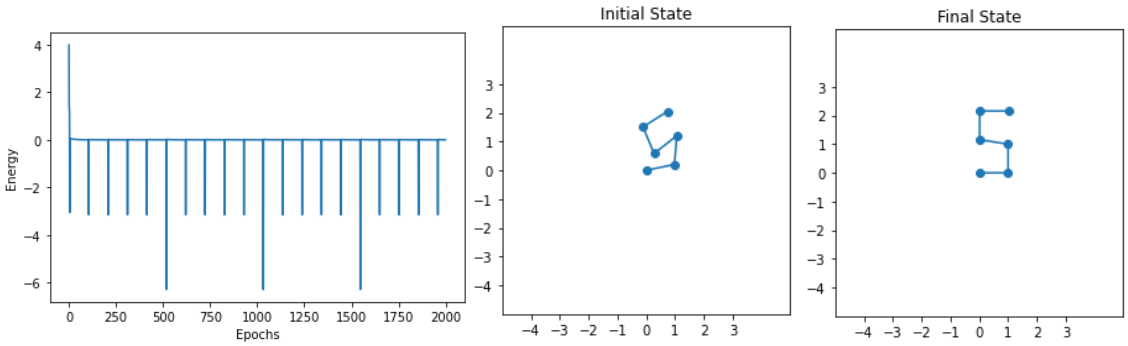


*Figure 14. Differentiable Results (Short 2)*

The only instance the model achieves the desired minimum energy state of -6 was when the initial state was already close to the global minimum. The initial state can be seen in Figure 14 (middle), and the final state can be seen on the right, it can be seen there is not much difference between the initial and final states. The energy score plot in on the left shows how the model reaches the local minimum and hovers around there.

As expected, the model did not perform well on the 20-long amino acid chain, the initial and final state can be seen in Figure 15 (middle and right). The predicted energy score is 0. Again, the algorithm kept getting stuck in local minima as can be seen on the left. Due to the gradient descent nature of the algorithm, it tends to get stuck and hover around these local minima. Parameters used here are not relevant because any reasonable combination of parameters fails to achieve optimization to the global minima.

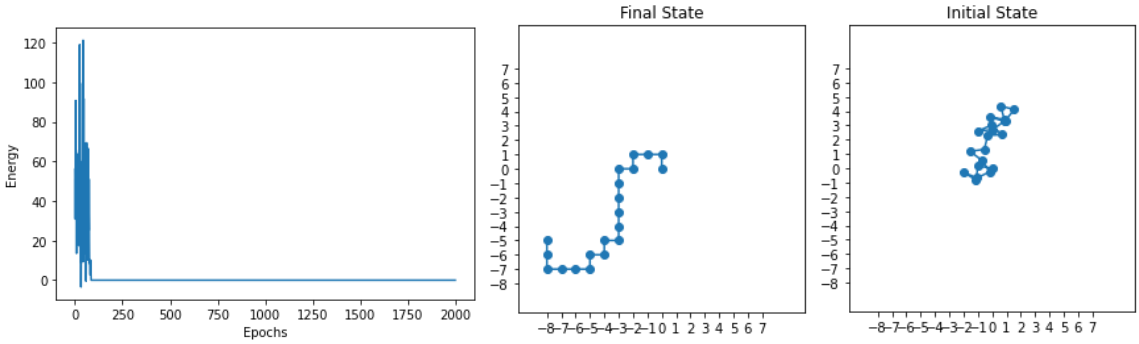


*Figure 15. Differentiable Results (Long)*

Given the model's failure on short sequence prediction, it was not expanded to solve for amino acid sequence design.

### 3.3.3 Comparison

It is clear the simulated annealing model was superior in structure prediction. It correctly predicted the protein structure given a sequence of amino acids and was also able to scale up to longer chains, useful as protein sequences in nature are often long. It was also able to tackle the sequence design problem, something the

differentiable model was not able to progress onto as it did not handle the structure prediction problem well enough. The problems are the inverse of one another so we can assume if it performs poorly in one, it is likely to perform poorly in the other. The simulated annealing algorithm did not get stuck in local minima as the algorithm would probabilistically move to another point on the energy curve, whereas the differentiable learning model did get stuck in local minima. The only positive about the differentiable model is that it was successfully implemented and can carry out gradient descent to the nearest minimum of some random initial angular path, $\theta$.

## 3.4 Evaluation

The differentiable learning algorithm likely failed as the energy function was too complex to optimize over with simple gradient descent. There are methods that exist that may overcome this problem, these are however dependent on the nature of the curve to be optimized over. One such method that was trialled was momentum, that is, remembering the change in $\theta$ between iterations. This is done by calculating the next $\theta$ as a linear combination of the previous $\theta$ and the gradient. This can be mathematically defined as follows:

$$\theta = \theta - \alpha \frac{dE}{d\theta} + \gamma \Delta\theta \tag{3}$$

where $\gamma$ is some decay factor between 0 and 1, determining the contribution of the current gradient and earlier gradients. $\alpha$ is the same learning rate parameter as before and $\Delta\theta$ is the change in values between iterations.

Given that the momentum implementation showed no improvement, results were not included.

A systematic approach was taken to the differentiable learning model, trialling multiple libraries, and carrying out rigorous tuning of parameters, none of which were able to overcome the assumed complexity of the energy function. Considering the many ways in which 6 coordinates can be arranged in relation to one another, it is likely there will be many local minima. There do exist other variants of gradient descent that may overcome local minima, however without knowing what the energy function landscape looks like it is difficult to determine which is likely to succeed. Perhaps this is an area to research in the future.

It is also worth noting that continuous coordinates were being forced onto a discrete lattice structure and then being optimized over in a continuous fashion (gradient descent). Perhaps if the discrete lattice nature of the structure were to be relaxed, a differentiable learning algorithm may yield different results, possibly another route to explore in the future.

# 4. PROTACs

This chapter is about building a model to predict PROTAC-protein binding pairs, the problem is defined in **1.2**. A Support Vector Machine (SVM) and a Neural Network (NN) were built in Python programming language, using popular machine learning libraries Scikit Learn and Keras. This section first outlines the theory behind SVMs and NNs and then explains the method taken to build each of the classifiers, and finally compares their results. The successful iterations of the models were used to give prediction scores for the human proteome against all known PROTAC compounds, with the intention here for this to be used as validation against other PROTAC-protein binding literature, or even as machine learning supported measure of "PROTACtability" [22] for biology researchers to use to reduce experimental fees.

## 4.1 Theory

### 4.1.1 Support Vector Machine (SVM)

An SVM is a method of supervised learning capable of carrying out binary classification [27]. An SVM will take training examples as input and map them to points in space so that the distance in space is maximized between the two classes. The distance in space is labelled as the margin in Figure 16 below.

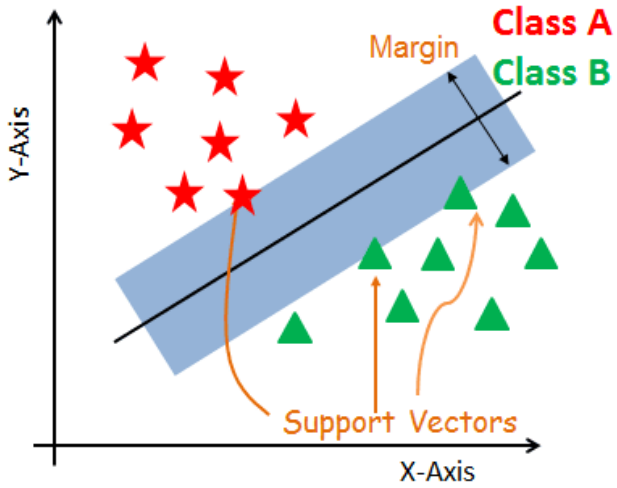


*Figure 16. SVM Diagram*

An SVM uses a kernel function to find a decision boundary in higher dimensional space relative to the input space. The SVM uses "the kernel trick" to calculate higher dimensional inner products between feature points without transforming the data to a higher dimension as this would be computationally infeasible.

An SVM gives a decision function as:

$$f(x) = sign(\frac{1}{2}\sum_{i=0}^{n} \alpha_i K(x_i, x) + b) \quad (4)$$

Where $\alpha_i$ is the parameter to be trained for to find the optimal decision boundary and K is the kernel function, $n$ is the number of dimensions of the feature vector. $\alpha_i$ is optimized by maximising the following expressions, where $y_i$ is the true class label:

$$min\,(\sum_{i=0}^{n} \alpha_i - \frac{1}{2}\sum_{i=0}^{n}\sum_{j=0}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x)) \quad (5)$$

$$s.t. \sum_{i=0}^{n} y_i \alpha_i = 0, \alpha_i \geq 0$$

The kernel function K used in our model is the popularly used Radial Basis Function [28], defined as follows:

$$K(x_m, x_n) = exp\,(-\frac{||x_m - x_n||^2}{2\sigma^2}) \quad (6)$$

### 4.1.2 Binary Neural Network

A neural network is a system of nodes, categorized into layers that hold and move information throughout the network [25]. The network is based on the way in which the human brain processes information using a neuronal system, information arriving at each node is summed and passed through an activation function, if it reaches a certain threshold, the activation function allows the information to be sent on to the connected nodes in the next layer. This is analogous to the way in which neurons in the brain will only "fire" and send an electrical signal on to connected nodes if the signal exceeds some threshold.

The diagram in Figure 17 below shows an example of a simple neural network. A brief introduction to neural networks is given in this section with the aid of Figure 17. In section **4.2** our neural network's architecture will be explained.

Suppose we were to have two input features in numerical form, for example, one number for a compound and one number for a protein, respectively $x_1$ and $x_2$. The neural network multiplies these inputs features with some weights, $w_1$ and $w_2$, then adds on a bias term, $b$, term to obtain the variable z:

$$z = x.w + b \quad (7)$$

$z$ is passed through a non-linear sigmoid activation function $\sigma(x)$ to give the prediction $\hat{y}$:

$$\hat{y} = \sigma(z) = \frac{1}{1+ e^{-z}} \tag{8}$$

$\hat{y}$ is the probability of the input vector $(x_1, x_2)$ belonging to the positive class, in our case this is the probability that the input compound $x_1$ and protein $x_2$ bind to one another.

The estimated probability is then fed into a loss function that calculates a score for how close the prediction is to the true class label. This loss function is then used in a process called backpropagation to adjust the weights and bias of the network until the prediction is improved and the loss is minimized.

The optimization problem is then as follows, we wish to find weights and bias parameters $w$ and $b$ that minimize the following binary cross entropy loss function:

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{n} y_i.log\ log\ (\hat{y}_i) + (1 - y_i).log\ (1 - \hat{y}_i) \tag{9}$$

Where $y_i$ is the true class label and $\hat{y}$ is the predicted class label and $N$ is the number of classes and $n$ is the number of features in the sample.

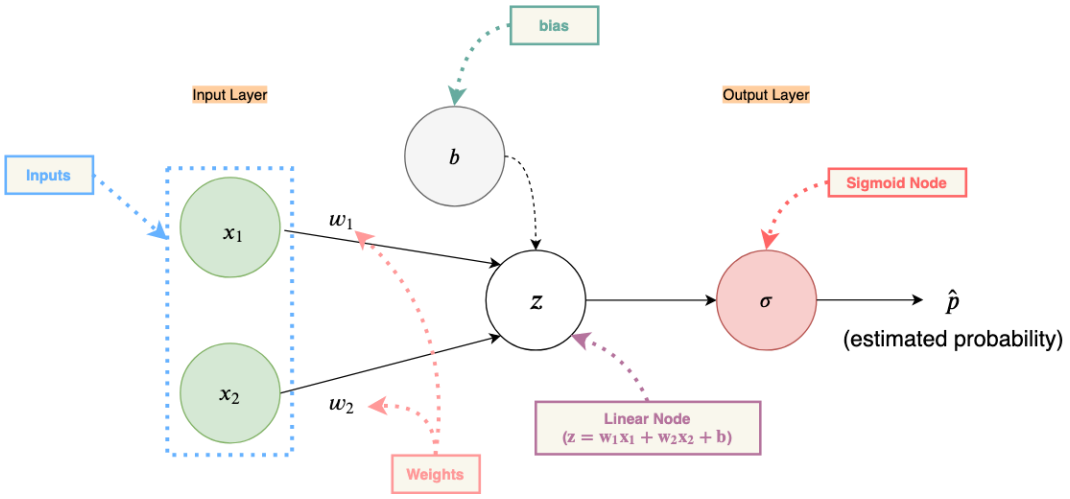


*Figure 17. Neural Network Diagram*

## 4.2 Methodology

### 4.2.1 Data

Three datasets were used to build the models on, various combinations of the datasets were used to create the train and test sets for the models, they are explained below.

**Dataset 1: Drug-Protein Interaction Data Dataset**
The first dataset contains information about whether a drug compound and target protein bind or not. There are three columns:

**Protein Sequence String | Compound SMILES String | Interaction (0 or 1)**

This dataset was taken from Tsubaki et al. [5] and is used as training data for the machine learning models. The data is on experimentally observed drug compound-protein interactions occurring in the human body. There are 6738 examples, with 3369 positive examples and 3369 negatives. There are 852 proteins and 1052 compounds in the dataset. The model will learn on proteins and their binding compounds which it can then use to predict binding PROTAC compounds for unseen proteins.

**PROTAC Warhead-Protein Interaction Dataset**
The second dataset is similar to the first, only instead of drug compounds, it contains PROTAC compounds. The dataset also includes information about the nature of the interactions; however, we only extract the protein sequence and PROTAC (warhead) compound columns.
The other difference is that we only have positive examples in this dataset - there are 957 rows of data, all positive PROTAC-protein binding pairs. The PROTAC column refers to the warhead ligand of the PROTAC as this is the only part of the PROTAC that physically interacts with the target protein. The dataset was taken from the PROTAC-DB database [19].

**FDA Approved Compounds**
The third is a dataset consisting of 3992 drug compounds that have been approved by the U.S. Food and Drug Association [29]. This dataset was used to create negative PROTAC-protein to test the positive examples from dataset 2 against. These negative examples were used to compute a Rank of First Positive Prediction (performance metric) against, this is detailed in **4.2.4**.

### 4.2.2 Feature Extraction

Protein sequence and compound structure data is in string format and needs to be converted into numerical data so that our machine learning model can accept it as input.

The protein sequence strings were converted into arrays using k-mer counts to describe amino acid frequency composition where k=1,2,3, these are then combined to create a 412-dimensional feature vector. This representation encapsulates the physicochemical properties of varying lengths of amino acid sequence subsets within the full sequence; a similar technique is used in Gull et al. [30].

Compound structure strings (a.k.a. SMILES) were converted into Morgan Fingerprints (Extended Connectivity Fingerprints) [31] using Python library rdkit's function GetMorganFingerprintAsBitVect. This gives us a 1024-dimensional array representation of the compound structure in the form of 0s and 1s.

### 4.2.2 SVM

Before data is input directly into the neural network, two processes are first carried out. The first being scaling the data, we apply the Scikit Learn StandardScaler function to fit the training data, then transform the test data accordingly. This is done to normalize input features, i.e., have all features on the same scale, this improves optimization of the loss function and in turn improves model performance [32].

The SVM was implemented with the popular machine learning Python library, Scikit Learn. The model was initialized using parameters penalty C=1 and class_weight = balanced.

Before the model is trained, we pass the data through the kernel function to improve the separation between classes in some higher dimensional space to improve classification performance. The kernel used was the radial basis function (RBF) as is defined in **4.1.1.**

Taking data from dataset 1, we first pass protein column $x_p$ and compound column $x_c$ through the RBF, defined as K. Training data is processed as follows:

$$K_{train} = \left(K(x_p) + K(x_c)\right)^2 \tag{10}$$

The model is then trained on $K_{Train}$. As for the test dataset, K is applied in a similar fashion to the test dataset, only this time, both training ($x$ ) and test data ($x\_test$ ) together is passed through the RBF.

$$K_{test} = \left(K(x_p, x\_test_p) + K(x_c, x\_test_c)\right)^2 \tag{11}$$

No hyperparameter tuning was carried out in the interest of time.

### 4.2.3 NN

As mentioned in **4.2.2**, before training the model, the data is scaled using the StandardScaler function. Then feature arrays are "flattened" and combined transforming 2 sets of features x_p and x_c into one array of features. Each feature is then a 1436-dimensional vector. Each node of the input layer of a neural network requires a single numerical value, hence, flattening features is necessary.

The NN was implemented in Python using Keras, the popular machine learning library. The architecture of the neural network is defined as follows:

**Neural Network Architecture**

```
1. model initialization
2. model.add(Dense(2*xlen, input_dim=xlen, activation='relu')
3. model.add(Dropout(0.3))
4. model.add(Dense(512,activation='relu')
5. model.add(Dropout(0.3))
6. model.add(Dense(128,activation='relu')
7. model.add(Dropout(0.3))
8. model.add(Dense(64, activation ='relu')
9. model.add(Dropout(0.3))
10. model.add(Dense(1,activation='sigmoid')
```

This simple architecture was developed through trial and error, adjustments were made to layers until results improved. xlen defines the dimension of input feature vectors, 1436. A ReLU activation function was used on each dense layer, this function essentially outputs its input directly if positive, otherwise outputs 0.

Regularization was incorporated into the model to reduce overfitting of the training data. Overfitting is when the model learns the intricacies of the training data too well and fails to generalize to unseen data. Dropout layers were incorporated at 0.3 parameter values, meaning 30% of all neurons from the previous layer are dropped at random, this is to reduce overfitting. L2 regularization was added to the loss function at parameter value 0.0001, this was done by specifying the kernel_regularizer parameter within each dense layer. Adding regularization terms to the loss function can help to reduce the complexities of the neural network and reduce overfitting on training data, improving model generalization.

The neural network was only allowed to run for one epoch, more epochs were not explored in fears of overfitting and as well as having time constraints. Once the neural network was defined, it was trained on the scaled data and tested on the scaled test data, like the SVM, of course, no kernel functions were applied here. Again, rigorous hyperparameter tuning was carried out in the interest of time, reasonable parameters for the model were gauged through a few trial-and-error iterations.

### 4.2.4 Performance Evaluation - Rank of First Positive Prediction (RFPP)

When assessing the capability of a classifier, in the context of biology, accuracy is not the most useful metric for a researcher. While accuracy is a useful metric in assessing model performance, it tells us nothing about the experimental implications of the results. Accuracy itself, labelling an interaction as 0 or 1 may not be useful to a biologist, because a PROTAC that scores 0.52 likely to be labelled as 1, is largely different from a PROTAC that scores 0.98 and is also labelled as 1. The biologist will still need to carry out physical experiments on all of the PROTAC compounds labelled as 1, even if they are weak predictions. Instead, we have the RFPP which is coined in ul Amir Afsar Minhas et al. [33]. The RFPP ranks a single positive interaction amongst many negative interactions based on the probability of it being a binding interaction.

Each target protein and its positive interactions from the PROTAC dataset is combined with all the compounds from the FDA approved dataset to generate 3992 negative interactions. We combine the positive and negative interactions, then input this test set into model to obtain binding likelihood predictions, we then sort these interactions by prediction with the highest probabilities at the top of the list. We then take the rank of the first positive prediction and use that as our RFPP score for that specific target protein. We do the same for all target proteins in the test PROTACs dataset and obtain a list of 253 RFPP scores.

It can then be derived that a score closer to 1 is better than a score closer to 4001. RFPP can be analysed at percentiles, e.g., RFPP(50%) = 4 would imply the top half of the predictions achieve an RFPP of 4. The ideal is to have an RFPP(100%)=1, implying all the true positives are correctly identified by the model amongst many negative interactions.

## 4.3 Results

### 4.3.1 Neural Network (NN) vs Support Vector Machine (SVM)

The first set of results compares two machine learning methods, the SVM and the NN. The focus is not so much on which performs better, more so on the fact that several machine learning methods can predict PROTACs for target proteins. The data has been trained on the drug compound-protein interactions (dataset 1) and tested on the PROTAC-protein interactions (dataset 2) using the RFPP metric. Dataset 1 and dataset 2 share 106 proteins with above 95% similarity and share 2 proteins above 80% similarity. The 106 proteins above 95% have been removed from the training dataset to ensure results obtained are novel.

Figure 18 shows a comparison of RFPP scores at various percentiles. The NN is better at recognizing a true PROTAC-protein binding pair amongst many non-binding PROTAC-protein pairs as its RFPP percentile curve is lower than the SVM's.

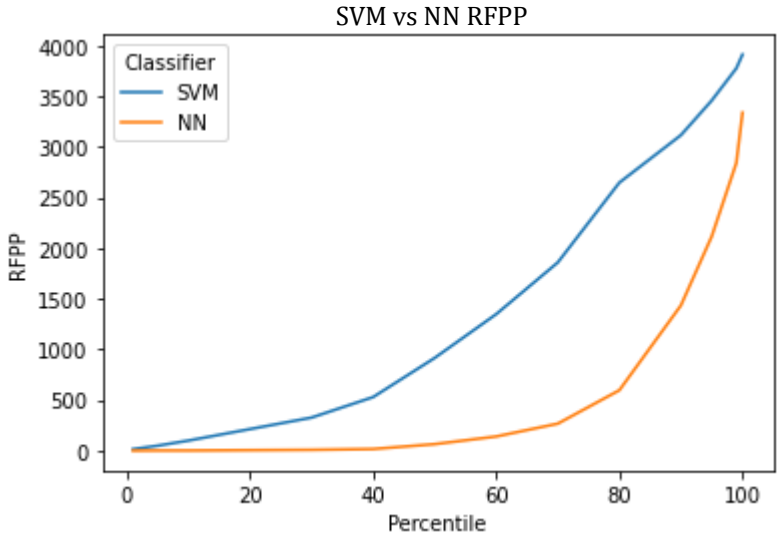


*Figure 18. SVM vs NN RFPP Plot*

Given that the NN outperforms the SVM on training on dataset 1 and testing on dataset 2, we will continue the rest of the results with the NN.

### 4.3.2 Adding PROTACs to Training Data

The next set of results assesses how performance is affected once PROTAC compounds are added into the training data. Here, the PROTAC-protein dataset is split into 5 folds, 4 of which are added onto the original training dataset and then the final fold is used for testing. Cross fold validation is performed here, and average RFPP scores are taken. The folds were based on the number of unique proteins in the dataset, i.e., there are 957 examples but only 253 unique proteins. It is these unique proteins that were split into 5 folds, such that the training data and the testing data do not contain any of the same proteins.

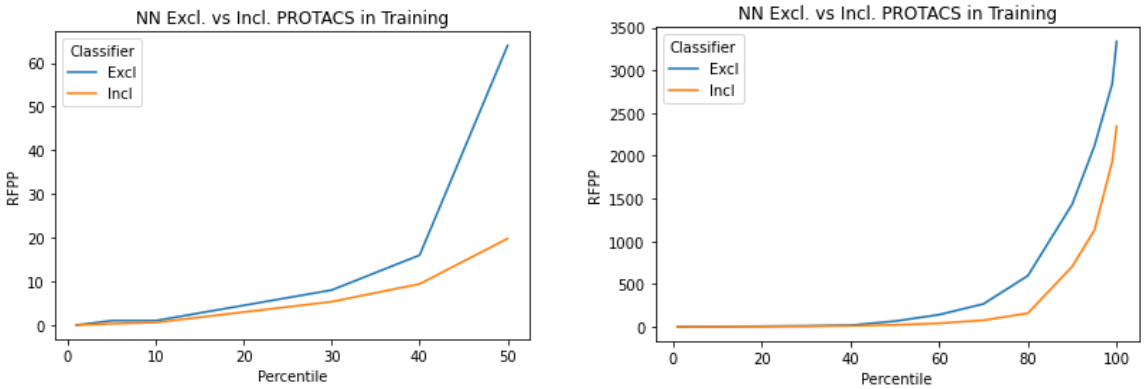


*Figure 19. Excl. vs Inclusion of PROTACs in Training, Top 50 Percentiles (Left), All 100 Percentiles (Right)*

As can be seen in Figure 19, the orange and blue curves represent inclusion and exclusion respectively of PROTAC interactions in the training data. It is clear to see that performance improves as the RFPP curve is lower once we include PROTACs in the training process. The plot on the left shows percentiles 0 to 50 while the plot on the right shows percentiles 0 to 100.

### 4.3.3 Regenerate Variable

In the data preparation portion of the code, before the model is trained, an *if* statement exists, this block of code shall be referred to as the regenerate (Regen) block. If set to true, it generates negative examples for all positive examples in the dataset, matching protein and compound pairs that do not exist in the original dataset (dataset 1). The intention here is to increase the ratio in favour of negative examples, e.g., 5:1 negative to positive examples in the training set. This is so the model is better able to identify positive examples among many negative examples. Results in Figure 20 are from when the model is trained on dataset 1 and tested on dataset 2, with varying regenerate values.

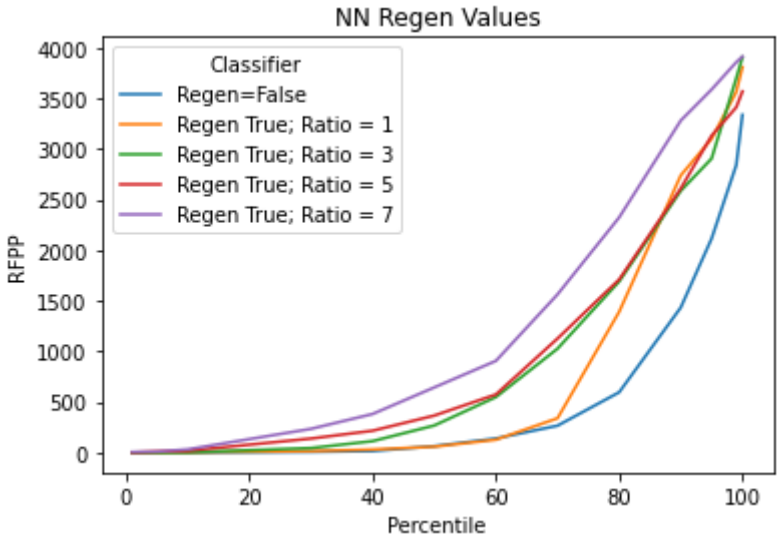


*Figure 20. NN Regenerate = True and Varying Ratios*

Figure 20 shows how utilizing the regenerate block of code and regenerate is set to true with a ratio of 1, i.e., 1:1 positive to negative training examples, the RFPP scores are similar compared to when regenerate set to false for the top 60$^{th}$ percentile. When regenerate is set to true and the ratio of negative to positive examples is increased, performance worsens. This implies the model performs better when tested on a balanced dataset, i.e., 1:1 negative to positive examples.

**Table of Results**

A table of all variants of models tested, excluding regenerate ratios greater than 1, have been included for comprehensive purposes. The results discussed in previous sections of **4.3** are also included in this table.

*Table 1. All Model Variant Results*

| Model | Train | Test | Regenerate; Ratio | RFPP_1 | RFPP_10 | RFPP_50 | RFPP_100 |
|---|---|---|---|---|---|---|---|
| SVM | Original | PROTACs | False | 14.52 | 99.2 | 915.0 | 3910.0 |
| SVM | Original | PROTACs | True; 1 | 17.12 | 106.2 | 468 | 3969.0 |
| SVM | Original + ⅘ PROTACs | ⅕ PROTACs | False | 0 | 0.6 | 7.6 | 3536.4 |
| NN | Original | PROTACs | False | 0 | 1.0 | 64.0 | 3336.0 |
| NN | Original | PROTACs | True; 1 | 0.52 | 4.0 | 59.0 | 3807.0 |
| NN | Original + ⅘ PROTACs | ⅕ PROTACs | False | 0 | 0.58 | 19.8 | 2344.8 |

"Original" refers to dataset 1, containing drug-protein interactions. PROTACs refers to dataset 2, containing PROTAC-protein interactions. All the plots discussed earlier have been included in the results except for those where Regen=True; Ratio=3,5,7. It is also worth noting, an RFPP score of 0 means the target protein was ranked at the top of the list (ranks are taken by index), so for example RFPP(100%) = 0 is a perfect score.

# 4.4 PROTACtability of the human proteome

Two variations of the NN model were used to predict binding probabilities for target proteins from the human proteome against PROTACs from dataset 2. The intention here being that a biologist can filter the csv file for a desired protein and select the top PROTAC binding candidates, i.e., those with the highest prediction score. The prediction score will be treated as a "PROTACtability" measure for target proteins, taking inspiration from Schneider et al. [22].

The first variation is where the NN is trained on dataset 1 and validated on PROTACs from dataset 2. The model was optimized for accuracy on the validation set and achieved a validation accuracy of 93%. The script to generate this csv file has been labelled nn-incl-106_human_proteome.py.

The second variation is where the NN is trained on the original dataset plus ⅘ folds of the PROTAC dataset, then validated on the remaining PROTAC fold, validation accuracy here was 95%, the script to generate this dataset has been labelled nn-incl-106-4-5-folds-human_proteome.py.

The 106 shared proteins were kept in the training data to improve data availability for the model and improve prediction scores on the human proteome.

## 4.5 Evaluation

Our main goal with this project was to highlight that machine learning methods used for drug-protein interaction predictors can predict PROTAC-protein interactions. When the models were tested on dataset 1 and tested on PROTAC-protein interactions, the median RFPP score (RFPP(50%)) was 915 and 64 for the SVM and NN respectively. It is clear that the NN performed better than the SVM, however given that no hyperparameter tuning was carried out on the SVM at all and only slightly on the NN, it is not entirely a fair comparison to make. However, given that the RFPP score was out of approximately 4000 interactions, both models did show promise in predicting true PROTAC-protein compounds. One means of improving this in the future would be to carry out rigorous hyperparameter tuning and testing of various neural network architectures to improve performance, this was not done in this project in the interest of time.

Variations of training and test data were used by the models. It can be concluded that adding positive PROTAC examples into the training data improves RFPP scores, median scores for the SVM and NN were 7.6 and 19.8 respectively, however RFPP(100%) was better for the neural network at 2344.8 compared to the SVM's 3536.4 as shown in the table. Including PROTACs in testing is expected to improve results as the model has true examples of PROTAC-protein interactions to learn from and so can better recognize them in unseen datasets. If the model is to be used in the labs, it is ideal to train it on at least some portion of the existing experimental PROTAC data available to improve model performance.

The *regenerate* code block did not seem to improve performance when higher ratios of negative examples were generated, the model seemed to perform better on a balanced dataset. It is not clear why this was the case, perhaps adjusting model parameters to account for imbalanced training data on higher ratios may improve performance, this is a possible avenue to explore in the future.

For future improvements, to get a more accurate measure than simple plots or percentile RFPP scores, it may be worth calculating the area under the RFPP percentile curves. The closer the area is to 0, the better the model at predicting PROTAC-protein interactions, plus this method would also account for all RFPP scores in the test prediction, rather than just percentile intervals. If there was more time, this could have been implemented.

For future works, it may be worth exploring other factors available on PROTAC-protein interactions in case they contribute to the classification of binding interactions. Factors for example from the original PROTAC-protein dataset include exact compound mass or heavy atom count, among others. Please see the warhead csv file from the PROTAC-DB for more [19].

Although the PROTACtability of the human proteome scored well on the validation dataset, it is worth cross validating results with the database from Schneider et al. [22] to determine whether our predictions for PROTAC-protein pairs support current literature. This is a possible avenue to explore in the future.

# 5. Conclusion

## 5.1 Differentiable Learning of Protein Structures

The goal of this project was to build a differentiable model capable of learning protein structures from a given sequence of amino acids. Had this been successful, it would have been expanded to tackle the inverse problem, amino acid sequence design as well as modelling of the folding pathways. The intention was to build a model capable of tackling all 3 protein folding research problems, and use this as a foundation for differentiable models to learn more complex phenomena, for example, more detailed protein representations or even protein-protein interactions.

Python was used to build a simulated annealing model capable of solving the structure prediction and sequence design problems, it was also to be used as a benchmark against the differentiable model's results. The Python library DiffTaichi was used to design the energy function that did successfully calculate lattice structure energy scores, however, optimization over the energy landscape to find the global minimum was unsuccessful. Gradient descent was not an ideal optimization technique given the nature of the energy landscape, there were many local minima in which the algorithm kept getting stuck. Momentum based gradient descent was also tested but to no avail. The outcome of the differentiable model meant the project was shorter than expected, so a decision was made to move on to a different research problem associated with proteins.

## 5.2 Predicting PROTACs

The goal of this project was to assess whether machine learning methods can predict PROTAC-protein binding pairs. An accurate model would be useful for biology researchers as it would reduce the number of experiments necessary to determine the best PROTAC candidates for a target protein.

Data on drug-protein pairs was used to train the model and PROTAC-protein pairs were then tested using the RFPP metric. SVM and NN models were tested and the NN outperformed the SVM in terms of the RFPP score. The model further improved in performance when PROTAC data was added into training, the model was then tested on unseen PROTAC data and results improved considerably over the first iteration.The human proteome was taken and two of the model variants were used to generate binding prediction scores for all proteins against all PROTAC compounds. The model variants used were (1) excluding PROTAC data from training and (2) including PROTAC data in training. The intention here being to validate the prediction scores against known literature. Assuming the prediction scores are correct, they can be used by biologists to narrow down PROTAC candidates for a target protein.

# 6. Project Management

## 6.1 Change of Plan

The original intention was to implement a differentiable model of lattice proteins and then expand it to handle not only larger protein sequences, but also tackle the amino acid sequence design problem and finally model visually the folding pathways. After carrying this out, the plan was to expand the model to handle more complicated protein structures, e.g., 3-D and even model lattice protein-protein interactions. Given that the model was not successful, the remaining couple of months were dedicated to the PROTACs research, a different problem but still in the realm of applying machine learning methods to proteins.

## 6.2 Supervisor Communication

For the majority of the project, at least once a week a Microsoft Teams meeting would be held with the supervisor, Dr. Fayyaz Minhas, and relevant concepts, research progress and next steps were discussed. A plan was also agreed upon as can be seen in the interim report, however, has changed due to the first project ending sooner than expected.

## 6.3 Code Management

Python scripts were stored locally and each time a new variant of a model was built, the change was stored, and versions were labelled accordingly with comments and READ_ME files to aid understanding. Code is also stored in a private Github repository.

# References


[1] 'Protein Structure and Function - Gregory Petsko, Dagmar Ringe - Oxford University Press'. https://global.oup.com/academic/product/protein-structure-and-function-9780199556847?cc=za&lang=es (accessed Sep. 05, 2021).

[2] 'Levinthal's Paradox', May 23, 2011. https://web.archive.org/web/20110523080407/http://www-miller.ch.cam.ac.uk/levinthal/levinthal.html (accessed Sep. 05, 2021).

[3] 'PROTACs'. https://www.astrazeneca.com/r-d/next-generation-therapeutics/protacs.html (accessed Sep. 06, 2021).

[4] R. J. Mayer, A. J. Ciechanover, and M. Rechsteiner, *The Ubiquitin-Proteasome System and Disease, Volume 4*. John Wiley & Sons, 2007.

[5] M. Tsubaki, K. Tomii, and J. Sese, 'Compound-protein interaction prediction with end-to-end learning of neural networks for graphs and sequences', *Bioinforma. Oxf. Engl.*, vol. 35, no. 2, pp. 309–318, Jan. 2019, doi: 10.1093/bioinformatics/bty535.

[6] H. Li, C. Tang, and N. S. Wingreen, 'Nature of Driving Force for Protein Folding: A Result From Analyzing the Statistical Potential', *Phys. Rev. Lett.*, vol. 79, no. 4, pp. 765–768, Jul. 1997, doi: 10.1103/PhysRevLett.79.765.

[7] K. F. Lau and K. A. Dill, 'A lattice statistical mechanics model of the conformational and sequence spaces of proteins', *Macromolecules*, vol. 22, no. 10, pp. 3986–3997, Oct. 1989, doi: 10.1021/ma00200a030.

[8] L. Zhang, H. Ma, W. Qian, and H. Li, 'Protein structure optimization using improved simulated annealing algorithm on a three-dimensional AB off-lattice model', *Comput. Biol. Chem.*, vol. 85, p. 107237, Apr. 2020, doi: 10.1016/j.compbiolchem.2020.107237.

[9] 'Finding low-energy conformations of lattice protein models by quantum annealing | Scientific Reports'. https://www.nature.com/articles/srep00571. (accessed Sep. 05, 2021).

[10] E. Shakhnovich, G. Farztdinov, A. M. Gutin, and M. Karplus, 'Protein folding bottlenecks: A lattice Monte Carlo simulation', *Phys. Rev. Lett.*, vol. 67, no. 12, pp. 1665–1668, Sep. 1991, doi: 10.1103/PhysRevLett.67.1665.

[11] I. Dotu, M. Cebrian, P. Van Hentenryck, and P. Clote, 'On Lattice Protein Structure Prediction Revisited', *IEEE/ACM Trans. Comput. Biol. Bioinform.*, vol. 8, no. 6, pp. 1620–1632, Nov. 2011, doi: 10.1109/TCBB.2011.41.

[12] S. P. N. Dubey, N. G. Kini, S. Balaji, and M. S. Kumar, 'A Review of Protein Structure Prediction Using Lattice Model', *Crit. Rev. Biomed. Eng.*, vol. 46, no. 2, 2018, doi: 10.1615/CritRevBiomedEng.2018026093.

[13] Y. Hu *et al.*, 'DiffTaichi: Differentiable Programming for Physical Simulation', *ArXiv191000935 Phys. Stat*, Feb. 2020, Accessed: Aug. 25, 2021. [Online]. Available: http://arxiv.org/abs/1910.00935

[14] S. S. Schoenholz and E. D. Cubuk, 'JAX MD: End-to-End Differentiable, Hardware Accelerated, Molecular Dynamics in Pure Python', Sep. 2019, Accessed: Aug. 25, 2021. [Online]. Available: https://openreview.net/forum?id=r1xMnCNYvB

[15] M. AlQuraishi, 'End-to-End Differentiable Learning of Protein Structure', *Cell Syst.*, vol. 8, no. 4, pp. 292-301.e3, Apr. 2019, doi: 10.1016/j.cels.2019.03.006.

[16] J. G. Greener and D. T. Jones, 'Differentiable molecular simulation can learn all the parameters in a coarse-grained force field for proteins', Feb. 2021. doi: 10.1101/2021.02.05.429941.

[17] J. Jumper *et al.*, 'Highly accurate protein structure prediction with AlphaFold', *Nature*, pp. 1–11, Jul. 2021, doi: 10.1038/s41586-021-03819-2.

[18] X. Sun *et al.*, 'PROTACs: great opportunities for academia and industry', *Signal Transduct. Target. Ther.*, vol. 4, no. 1, pp. 1–33, Dec. 2019, doi: 10.1038/s41392-019-0101-6.
[19] G. Weng *et al.*, 'PROTAC-DB: an online database of PROTACs', *Nucleic Acids Res.*, vol. 49, no. D1, pp. D1381–D1387, Jan. 2021, doi: 10.1093/nar/gkaa807.
[20] F. Cheng, Y. Zhou, J. Li, W. Li, G. Liu, and Y. Tang, 'Prediction of chemical– protein interactions: multitarget-QSAR versus computational chemogenomic methods', *Mol. Biosyst.*, vol. 8, no. 9, pp. 2373–2384, 2012, doi: 10.1039/C2MB25110H.
[21] M. Hamanaka *et al.*, 'CGBVS-DNN: Prediction of Compound-protein Interactions Based on Deep Learning', *Mol. Inform.*, vol. 36, no. 1–2, p. 1600045, 2017, doi: 10.1002/minf.201600045.
[22] M. Schneider *et al.*, 'The PROTACtable genome', *Nat. Rev. Drug Discov.*, pp. 1–9, Jul. 2021, doi: 10.1038/s41573-021-00245-x.
[23] P. J. van Laarhoven and E. H. Aarts, *Simulated Annealing: Theory and Applications*. Springer Science & Business Media, 1987.
[24] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, 'Automatic Differentiation in Machine Learning: a Survey', *J. Mach. Learn. Res.*, vol. 18, no. 153, pp. 1–43, 2018.
[25] K. Gurney, 'Introduction to Neural Networks'. Taylor & Francis, Oxford.
[26] 'UniProt'. https://www.uniprot.org/ (accessed Sep. 08, 2021).
[27] C. Cortes and V. Vapnik, 'Support-vector networks', *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, Sep. 1995, doi: 10.1007/BF00994018.
[28] M. D. of the M. P. I. for B. C. in T. G. P. B. Scholkopf, S. Istrail, and P. A. Pevzner, *Kernel Methods in Computational Biology*. MIT Press, 2004.
[29] V. B. Siramshetty *et al.*, 'SuperDRUG2: a one stop resource for approved/marketed drugs', *Nucleic Acids Res.*, vol. 46, no. D1, pp. D1137–D1143, Jan. 2018, doi: 10.1093/nar/gkx1088.
[30] S. Gull, N. Shamim, and F. Minhas, 'AMAP: Hierarchical multi-label prediction of biologically active and antimicrobial peptides', *Comput. Biol. Med.*, vol. 107, pp. 172–181, Apr. 2019, doi: 10.1016/j.compbiomed.2019.02.018.
[31] D. Rogers and M. Hahn, 'Extended-connectivity fingerprints', *J. Chem. Inf. Model.*, vol. 50, no. 5, pp. 742–754, May 2010, doi: 10.1021/ci100050t.
[32] 'sklearn.preprocessing.StandardScaler — scikit-learn 0.24.2 documentation'. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html (accessed Sep. 07, 2021).
[33] F. ul Amir Afsar Minhas, B. J. Geiss, and A. Ben-Hur, 'PAIRpred: Partner-specific prediction of interacting residues from sequence and structure', *Proteins*, vol. 82, no. 7, pp. 1142–1155, Jul. 2014, doi: 10.1002/prot.24479.