Using Machine Learning to Aid Programmability in Systems of Self-Assembly

Newcastle University

Thomas Sturgeon

2019

Degree: Computing Science
Supervisor: Dr. Natalio Krasnogor
Word Count: 12,246

Abstract

Self-Assembly is an autopoietic phenomenon in which complex structures 'emerge' from a system of autonomous entities without a master plan. Research into such systems often faces the forward, backward and yield problems. These are usually solved through simulations and the use of extremely complicated heuristics. This research explores the plausibility of using modern machine learning techniques to achieve a functional approximation of a sophisticated Kinetic Monte Carlo self-assembly simulator in order to produce a data-driven solution to the forwards and backwards problem

Declaration

I declare that the material presented in this Undergraduate Dissertation is my own original work, except where explicitly stated otherwise.

Student Number: 150375245

Acknowledgments

Thank you to my supervisor, Dr. Natalio Krasnogor, for introducing me to the fascinating world of Self-Assembly.

Contents

List of Figures

List of Tables

List of Abbreviations and Nomenclature

Abbv.	Explanation
SA	Self-Assembly
DSA	Dynamic Self-Assembly
ML	Machine Learning
AI	Artificial Intelligence
ANN	Artificial Neural Network
KMCS	Kinetic Monte Carlo Simulator
RNN	Recurrent Neural Network
GPU	Graphical Processing Unit
NCD	Normalized Compression Distance
KC	Kolmogorov Complexity
IaaS	Infrastructure as a Service
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine
PDT	Photodynamic Therapy
NCD	Normalised Compression Distance
KMC	Kinetic Monte Carlo Simulator
TAM	Tile Assembly Model
UAT	Universal Approximation Theorem
MSE	Mean Squared Error
CL	Constrastive Loss
ReLU	Rectified Linear Unit
Adam	Adaptive Moment Estimation
GA	Genetic Algorithm
GAN	Generative Adversarial Network
DQL	Deep Q Learning
GCP	Google Cloud Platform
VMs	Virtual Machines
LSTM	Long Short Term Memory
GRU	Gated Recurrent Unit

Table 1: List of abbreviations used throughout the document

1 Self-Assembly

Self-Assembly (SA) is an autopoietic natural phenomenon in which complex structures 'emerge' from a system of autonomous entities with no central control mechanism or master plan, based solely on a set of rules governing the interactions between entities and the environment. Self-Assembly is seen in systems at the micro and macroscopic level. The act of amino acid polymerization is one such microscopic example: amino acids will bind dependent on their environment (temperature, other aminos) to form Polypeptides which in turn self-assemble through folding to produce complex proteins. It can be seen further at the macroscopic level, where the microscopic conformations build the resultant macroscopic conformation. The brain is an example of amassed self-assembly in the form of neuronal connections. Nature is highly dependent on the use of self-assembly and it is ubiquitous therein. There is no master plan in the Universe and biological life as we know it has built itself - as John A. Pelesko aptly points out in his pivotal book on self-assembly: 'No one put you together'[12]

Research into Self-Assembly is promising due to its distributed nature and sometimes non-synchronous genesis. The ability to create organised structures with little to no interference could change the the modern world as we enter into the technological epoch. The benefits promised by nanoscale self-assembly in areas such as medicine, food, electronics and technology are staggering.

With this in mind, it is important to define what is meant by the term 'Self-Assembly' and thus make a categorical distinction between self-assembly and a natural aggregation process. This paper will henceforth adopt John A Peleskos definition of Self-Assembly, which in turn borrows from the literature itself:

> "Self-Assembly refers to the spontaneous formation of organized structures through a stochastic process that involves pre-existing components, is reversible, and can be controlled by proper design of the components, the environment and the driving force"[12].

This creates a useful distinction between self-assembly and aggregation in that self-assemblies are spontaneous, reversible and controlled by design. The use of the word 'design' is intentional: it highlights the notion of a programmable self-assembly in order to produce stable, pre-ordained geometries in a system outside of any equilibrium.

1.1 Programming Systems of Self Assembly

One of the main goals involved in any self-assembly problem domain is to achieve a design 'interface' for assembling intricate, pre-ordained geometries. This concept of designing, engineering or 'programming' a self-assembly is hereby defined as the extent to which an agent, a user or a computer program, can control the output of the self-assembly process. That is, knowing the tile set 'design' for producing pre-ordained geometries. The level of complexity involved with producing pre-ordained conformations is NP-Hard in the vast majority of cases. Although some self-assembly 'programming' is being applied in the present day chemical engineering industry, if we can explore further the world of programming self-assemblies by 'learn[ing] to build small in the way that nature builds small'[12] we will unlock previously impossible technologies: amorphous computing, self-assembled electronic circuits, nano-robotics and advanced computational methods for solving hard problems through large-scale distribution.

Programming a self-assembly is non-trivial when the rules governing interactions in systems of autonomous entities are naturally rudimentary. Due to the complex stochasticity of interactions under such rule systems: 'it is strongly suggested that exact polynomial

time deterministic algorithms do not exist [for finding the optimal design of a tile set].'[22]. With this in mind, current research into self-assembly and by way of that, engineering self-assemblies, pivots around three problems: 'The Forwards Problem', 'The Backwards Problem' and the 'Yield Problem'.

1.1.1 Forward, Backward and Yield Problem

The forwards problem is concerned with predicting the outcome of a self-assembly: given the environmental parameters (the set of entities, temperature, binding force) how does one predict the final conformation? The backwards problem, on the other hand, is the design problem: how does one design the set of entities and the environmental parameters such that the produced conformation is a pre-defined one? The yield problem is concerned with controlling and predicting the amount of self-assembled structures 'yielded' from a given self-assembly process.[1] The forwards problem is often solved through the application of simulations and models using probabilistic approaches such as Monte Carlo methods. The backward problem, however, is more complex. It is the NP-Hard nature of the backwards problem, due to the astronomically large search spaces required even for non-complex self assemblies that means it is only solved via 'very sophisticated heuristics'[22] which lack optimality and are often out of reach.

Discussion of the problems involved with self-assembly research serves as a segue into a discussion of modern machine learning and its efficacy as a toolkit for learning functional approximations of complex problems with a high levels of variability, such as the forwards and backwards problems.

[1] This paper will intentionally neglect discussion of the Yield problem as it lays outside the scope of this research

2 Machine Learning

Machine Learning (ML) is a subset of Artificial Intelligence (AI) made up of algorithms that form the basis for classification, regression and control solutions in various business, scientific and artistic endeavours without the need for explicit programming. There are three main types of machine learning: supervised, unsupervised and reinforcement. With the right amount of data these learning types can be used to solve many different categories of problems.

- Supervised Learning
  - Uses labeled data to achieve a functional approximation from a feature set to a label
- Unsupervised Learning
  - Uses non-labeled data to detect patterns in data and perform a clustering analysis
- Reinforcement Learning
  - Attempts to optimise a reward by taking actions in a given environment based on the current state and previously learned states. This type of learning is often used to produce solutions to complex control problems

Perhaps the most effective form of undertaking the three types of learning is Deep Learning through Artificial Neural Networks (ANN). When applied to the most common form of machine learning, supervised learning, these biologically inspired models attempt to approximate some function that maps a set of features X to a set of labels y. This is done through through stochastic gradient descent (SGD) with back-propagation to update connection weights between interconnected 'neurons', based on an error value. ANNs have become very efficient at solving problems in many different domains through variable architectures, neurons and activation functions. Like many other ML models, ANNs are loosely inspired by the functionality of a biological brain:

> "It is best to think of [neural] networks as function approximation machines that are designed to achieve statistical generalization, occasionally drawing some insights from what we know about the brain, rather than models of brain function"[5]

Although AI and more specifically, machine learning, may seem relatively new in the collective consciousness, it is a discipline with a rich history in Mathematics and Computer Science. Early developments in AI took place before the invention of modern computers and emerged from probabilistic models such as Bayes Theorem and Markov Chains in the early 19th and 20th century. The first mathematical model of a neuron was produced by McCullough and Pitts in 1943. It consists of two internal functions that act as a binary switch dependent on a set of inputs $I1, I2...Im$, a set of weights $W1, W2...W3$ and a linear step that produces an output y such that $y \in 0, 1$:

$$g = \sum_{i=1}^{N} IiWi \tag{1}$$

$$y = f(g) \tag{2}$$

Where W is the set of weights and I is the set of inputs

Figure 1: The two functions of making up the McCullough and Pitts mathematical model of a Neuron

This was developed further in 1949 by Hebbian learning, a formula that increases the connection strengths between neurons that often fire together. Interestingly, Hebbian Learning has seen a modern resurgence and is used in 'Differential Plasticity'[11] for modern cutting edge machine learning using 'Hebbian Plastic Connections'[11]. Unfortunately, ANNs saw a marked declined after the initial research accomplishments of the 1950s due to the *AI Winter* and were mostly replaced by a more statistical, knowledge based approach to weak AI. The 1990s, however, begot a juxtapositional shift from knowledge based statistical approaches to more data based approaches. A vast amount of data was easily accessible due to the introduction of *ARPANET* and the World Wide Web. The mathematics for neural networks was now feasible with advanced modern hardware and neural networks became effective at solving real, data-driven problems. In the early 2000s, Deep Learning with back-propagation through stochastic gradient descent was shown to be an extremely efficient and effective way for performing learning. This revelation has propagated an inordinate amount of interest in machine learning, capturing the attention of the general public and the scientific community (Figure 2).

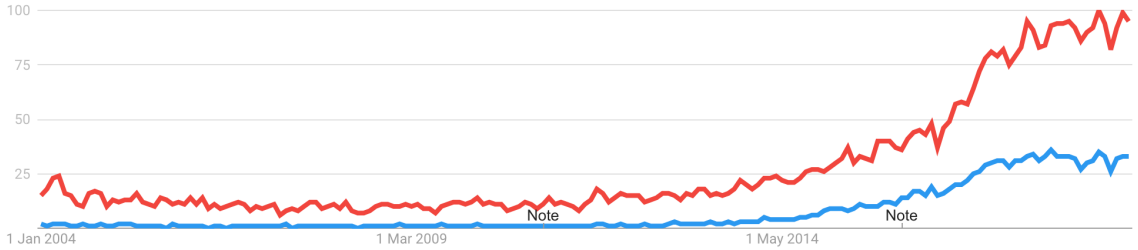


Figure 2: A Google Trends graph demonstrating the increased interest in Machine Learning (Red) and Deep Learning (Blue)

Deep Learning advancements through increased hardware power from Graphical Processing Units (GPUs) occurred in parallel with the advent of public access Infrastructure as a Service (IaaS) notably with Amazon and Google launching affordable cloud services between 2006 and 2008. Further, the release of cutting edge machine learning languages with accessible front-ends such as Tensorflow, Keras and CAFFE perpetuated the already keen interest in machine learning. The capabilities of deep learning were revealed worldwide in 2016 with DeepMind beating the world champion Lee Sedol in a public match of Go. Deep Learning is being applied to problems with huge feature sets that are unreasonable for classical data analysis or human reasoning. This is paving the way for bleeding edge feats in many areas of science such as cancer diagnosis, protein folding predictions, complicated control problems, natural language processing and Generative Adversarial Networks (GANs) that can generate unique, never before seen data. However, despite the extremely promising advancements in deep learning, training a model requires a substantial amount of data. A trivial problem such as hand written digit classification can require upwards of 60,000 instances for training and 10,000 for validation. This limits some domains from taking full advantage of machine learning.

'Machine learning' will be used interchangeably throughout this document to refer to artificial neural networks and will act synonymously with 'deep learning' unless explicitly stated.

3 Aims and Objectives

This dissertation will seek to explore the symbiotic relationship between self-assembly research and modern deep learning techniques. An Artificial Neural Network (ANN) will perform supervised learning over data produced from a Kinetic Monte Carlo Simulator (KMCS)[22] for simulating Porphyrin molecular self-assembly. The aim of this learning is to produce two machine learning models that will reduce a 'black box' simulator to interconnected layers of neurons and act as solutions to the forward and backward problems. The term 'black box' is employed here as the models will have no knowledge of the intricacies or semantics of the KMCS and will therefore have to construct a set of weights for 'driving $f(x)$ to equal $f'(x)$'[5] using data alone. This will seek to demonstrate the effectiveness of using machine learning to bypass the simulations and complex heuristics required for the forward and backward problems and instead replace them with an implicit function approximation.

3.1 The Kinetic Monte Carlo Simulator

The data from this project was generated by a Kinetic Monte Carlo Simulator developed in [22]. The use of Monte Carlo methods provides an accurate approximation of the behaviour of Porphyrin molecules in nature without having to solve the complex spatio-temporal equations that govern the natural system. Thus, the simulator acts as a probabilistic solution to the forwards problem for hetero-functionalised Porphyrin molecular self-assembly on a gold substrate. A Porphyrin is an umbrella term for a group of organic molecules that have many applications in modern science and medicine, for example, in the use of Photodynamic Therapy (PDT) which is a form of non-invasive cancer treatment.

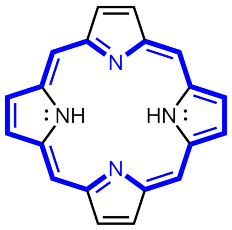


Figure 3: A Porphyrin Molecule

The functionality of the simulator is loosely based on a formal system of Wang Tiles. A Wang Tile system is made up of squares on a Euclidean plane where each square has colors that can tile by connecting with other matching colors. The use of Wang Tiles in self-assembly models is inspired by Rothemund's observation that 'Self-Assembly and computation are linked by the study of mathematical tiling'[14] leading to 'the crucial insight of Winfree...to realize that the colors on Wang Tiles could be replaced by specific binding rules...[meaning] that tile based self-assembly could *compute*'[12]. The mathematics governing these binding rules from [22] are presented: Let $\sum$ encode the 'glue' type associated with each edge or λ where the edge doesn't have a glue. The set of tiles is $\tau = \{t \mid t = (x_0, x_1, x_2, x_3)\}$ where x_i is a glue type or 'colour'. Thus it is possible to associate $x0, x1, x2, x3$ with the north, west, south and east edges of each tile. After a collision event, the two tiles involved, t_i, t_j, will assemble relative to their edges e_i, e_j if the colours and strengths in τ are compatible. Dependent on this strength, they will either combine or bounce off. The tiles here evidently represent the Porphyrin molecules (abstractly known as 'Porphyrin Tiles') and the glue types or colours are an abstraction of the bonding potentials between two independent molecules. Throughout this research,

the term Porphyrin, or Porphyrin molecule, will be used synonymously with Porphyrin Tile. As an input, the KMCS takes a state vector (Figure 4) where E_{11} and E_{22} are the binding energies between the same functional groups, E_{12} is the binding strength between different functional groups and E_s is the binding energy between the Porphyrin Tiles and the substrate. The output of the KMCS is a self-assembled Porphyrin Tile conformation as seen in Figure 5. The outputs produced by the KMCS for state vectors ranging from $\vec{S} = [0.5, 0.1, 0.1, 0.3]$ to $\vec{S} = [1.0, 1.0, 1.0, 1.0]$ provide the perfect dataset for supervised learning using an ANN.

$$\vec{S} = \begin{bmatrix} E_{11} \\ E_{22} \\ E_{12} \\ E_s \end{bmatrix} \tag{3}$$

Figure 4: The input state vector for the Kinetic Monte Carlo Simulator

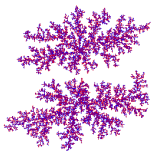


Figure 5: The output of the simulator showing one specific conformation of Porphyin Tiles

3.2 Aims and Objectives

This exploration into machine learning and self-assembly will hopefully show the capability for an ANN to functionally approximate the simulator, offering a unique, data-driven solution to the forward *and* backward problems. Thus, the main thesis of this research is proposed:

> 'Can a complex self-assembly simulator based on Kinetic Monte Carlo Methods be reduced via a neural network function approximation in order to solve the forwards and backwards problem and to aid in simplifying the process of 'programming' a self-assembly?'

The initial aims and objects have been radically changed since the inception and proposal of this research. Initially, the intention was to use two neural networks to solve the backwards and forwards problem in *unison* with the KMCS and use human verification or reinforcement learning to solve the forwards and backwards problem. This has changed to conducting a machine learning function approximation on the data since access to the simulator itself was unavailable but historical simulation data was available. In hindsight, even with the KMCS available for use, human verification over every instance in the dataset would have proved extremely slow and the computational power and replay memory required for Deep Q reinforcement learning was not available. The revised aims and objectives are presented:

3.2.1 Aim

1. To reduce the complexities of a Porphyrin self-assembly simulator based on Kinetic Monte Carlo Methods into an artificial neural network function approximation, through data alone, in order to solve the forwards and backwards problem

3.2.2 Objectives

1. Perform data analysis on the simulator data using Kolmogorov Complexity[10] and Normalised Compression Distance[4]
2. Develop a neural network model to functionally reduce the Kinetic Monte Carlo Simulator in the forwards direction
3. Develop a neural network model to functionally reduce the Kinetic Monte Carlo Simulator in the backwards direction
4. Tune and optimise both models on a representative subset of the data
5. Train both optimal models across the whole dataset
6. Perform further data analysis on both trained models using an augmented dataset to assess accuracy at solving the forward and backward problems.

Achieving the objectives defined here required extensive research into data science, machine learning frameworks, becoming highly competent in Python and the complex modules used for data manipulation.

4 Structure

4.0.1 Literature Review

This section seeks to present a brief review of self-assembly literature and where possible, links to machine learning. This dissertation identifies a gap in the literature and proposes this research so as to fill the gap.

4.0.2 Methodology

The methodology section provides high-level experimental details. It explains how data analysis was carried out on the Porphyrin conformations dataset including a Kolmogorov Complexity analysis. It then explains how two boilerplate ANNs were optimised and tuned for solving the forwards and backwards problem on a smaller, representative subset of data through meticulous data collection and evaluation. An explanation is given of the data augmentation techniques used on the original dataset to address initial problems identified through data analysis and boilerplate testing.

4.0.3 Results

The results from both networks are presented and discussed at length while contemplating the standard of the results in terms of solving both the forwards and backwards problem using a functional approximation.

4.0.4 Conclusion

The dissertation concludes with a broad discussion of the whole research project - examining the aims and objectives, considering what was achieved and what remains unanswered before offering a final summary, looking to the future and proposing further research.

5 Literature Review

Self-Assembly research pervades modern scientific endeavour with the goal of creating an algorithmic mathematical model for self-assembly. Winfree's pivotal observation that self assembling DNA tiles on a lattice are capable of performing universal computation and the astounding observation that 'Self-Assembly and computation are linked through the study of mathematical tiling'[14] has changed the modern approach to solving the forwards and backwards problem at molecular and supra-molecular scales. Further investigation into this astounding link between self-assembly and computation will prove essential in the scientific investigation of biological systems, physical sciences and controlled self-assembly.

Self-Assembling synthetic DNA nano-technology designed using thermodynamic models of nucleic acid hybridization [13] routinely achieves complex three-dimensional self-assembly paradigms. The ability to engineer interactions in these systems thanks to Watson-Crick complementarity has allowed for the construction of origami[15] and conformations that rival the complexities of structural hierarchies found in nature. However, DNA self-assembly is not without its limitations. DNA has no useful physical properties and is highly perishable outside of rigorously controlled conditions. Although research in [13] has been carried out into the capabilities of 'grafting' DNA to colloidal nano-particles to 'piggyback' properties ranging from electrical, optical to structural qualities onto DNA self-assemblies - these advancements, albeit extremely groundbreaking, still remain at the ransom of the stringent temperatures and PH levels that cater for DNA self-assembly.

The problems associated with moving away from DNA into non-DNA molecular self-assembly centre around lacking the required quantitative understanding of interactions between entities and the inability to correctly engineer the process in order to create precise, pre-ordained geometries. This distinct lack of quantitative understanding of self-assembly interactions and the limitations of DNA is perhaps what motivates modern research into engineering this process at all scales. That is, the meticulous process of synthetic DNA grafting on colloidal particles is unrealistic and 'we anticipate that in the near future, as the number of applications for self assembly (and their complexity) increases, a point will be reached where humans cannot design the set of components and their interactions'[21]. Thus, developing a computational theory of self-assembly, as Adleman posits:

> "promises to provide a new conduit by which results and methods of theoretical computer science might be applied to problems of interest in biology and the physical sciences."[2]

Moving away, but inspired by, branched DNA self-assembly, [16] questions the extent to which intrinsic computational mechanisms pervade outside of the biological world. An investigation is presented into the link between self-assembly and computation inherent in the program size complexity of pseudo-crystalline self-assembly of squares. This landmark research lays the foundations of computational analysis of the Tile Assembly Model (TAM). The TAM is a Turing complete model that uses computational Wang Tiles with edge 'glues' and an initial tile 'seed'. Winfree remarked that systems at $\tau = 2$, a measure of 'cooperativity', are capable of universal computation. This is especially appealing considering that Rothemund theorises that $\tau = 2$ is plausible in the physical, lateral capillary force self-assembly, meaning that universal computation without the use of DNA may indeed pervade the physical domain. [2] introduces a method for achieving optimal results in building the square assemblies. Initial constructions of $n\ x\ n$ squares into the binary counter in [16] required an asymptotic time complexity of $O(n \log n)$ and program size $O(\log n)$. [2] instead presents a new construction for $n\ x\ n$ which uses optimal time $O(n)$ and program size $O(\log n / \log \log n)$ by using a self-assembled base converter

to build the binary counter from the seed. This is the lowest bound achievable as dictated by Kolmogorov Complexity.

Moving further away from DNA self-assembly, [22] presents a solution to the forwards problem using the tile assembly model for Porphyrin molecular self-assembly with a Kinetic Monte Carlo Simulator that closely matches the behaviour of Porphyrin Molecules on a simulated gold substrate. Porphyrin molecules are different to previously researched DNA self-assembly in that they are 2D and therefore reduce the complexity for investigation. They can self-assemble under conditions adverse to the stringent environment required for DNA. Evidently, this bodes well for embedding versatile computational abilities into the self-assembly and specifically for investigating methods to aid in the programmability of non-DNA based molecular systems.

Thus, it is only reasonable, armed with the computational research into the TAM concerning time and size complexities, to question, in the same way synthetic DNA is used to organise a self-assembly: 'what is the optimal tile set (program size complexity) required for self assembly of a pre-ordained complex shape?' [1], presents this as the combinatorial optimization problem. It is shown that finding an answer to this question (the program size complexity) is NP-Hard in most cases and that 'exact polynomial time deterministic algorithms do not exist for these problems'[22]. This may be disheartening in the endeavour to develop a quantitative understanding of self-assembly as current researchers 'believe that good solutions to these problems are important for developing a mature, algorithmic theory of self assembly and for applying this theory to real-life scenarios.'[1] Regardless,[21] makes positive light of this NP-Hardness and posits that 'NP-Hardness results have not, in the past, deterred the advance of other branches of science and engineering'. Examples are cited involving quasi-solutions to the *Traveling Salesman Problem* abundant in modern science and engineering. Further, [21], interrogates the use of a Genetic Algorithm (GA) to circumvent this NP-Hardness and impossibility of polynomially pre-ordaining optimal tile sets. Some promising results are found with this automated approach to minimizing the limitations around programming self-assemblies. This shows the effective use of evolutionary algorithms to aid in replacing the unreliable, complex heuristics often involved with solving the backwards problem. Genetic Algorithms, however, are slow, and do not have the benefit of learning from pre-existing data unlike machine learning architectures. It is further reasonable, then, to question if a data-driven approach through machine learning will serve as a more effective medium for offering solutions to the backwards problem in the future.

One of the major hurdles involves developing suitable frameworks so that machines can learn about the subtleties of Self-Assembly thus removing the unreasonable requirement to 'design the set of components and their interactions'[21]. Applying machine learning in this area will hopefully reveal valuable insights into self-assembly and validate the provocative position of [24] that 'Nanotechnology...is the 21st Century's great leap forward in scientific knowledge.' Some research, [8], has been conducted into the use of machine learning for aiding programmability of self-assembly, notably on the capabilities of ML on multi-scale functional self-assembly in simulated surfactant molecules. That is, applying ML to micro-scale interactions in order to predict macro-scale conformations. Promising results were found in training on the initial molecular data and predicting the macro-scale self-assembly using a Random Forest performing supervised regression. The research finds an innate symbiosis between self-assembly in simulation and machine learning, positing that 'machine learning can be applied to multi-scale systems'[8].

Neural networks have been shown to model and provide solutions for complex problems that supersede the mental capacities of human beings and even classical algorithms in problems of NP-Hardness. State of the art deep learning models have already been adopted in industry, offering functional approximations, clustering and control solutions that aid in

computational research, medicine[7], farming and economics[25] to name just a few. This is especially provocative considering the Universal Approximation Theorem (UAT) of neural networks proves there is no mathematical bound on the functional approximation abilities of a neural network with enough data and learning. As Goodfellow succinctly posits:

> "A feed-forward network with a single layer is sufficient to represent any function, but the layer may be infeasibly large and may fail to learn and generalize correctly."[5]

With the UAT in mind, Machine learning is already disturbing the world of huge multivariate problems - solutions to which were considered, by some researchers, to be impossible ten years ago. [19] showed, famously, that using a neural network to perform deep learning alongside heuristic Monte Carlo search that the game Go, with a search space of 2^{170} legal board positions, was conquerable and achieved a 99.8% win rate against famous games including beating Lee Sedol, the world champion. Neural networks are evidently powerful in contributing to the solution or mitigation of problems with huge feature sets that consist of dimensional inter-relations and subtle reactionary trends in data over time, space and semantics. The development of a quantitative understanding of self-assembly outside of equilibrium will hopefully benefit from the recent breakthroughs in deep learning to achieve clever, data-driven solutions to the backwards and forwards problem inherent in understanding how to engineer pre-ordained, self-assembled geometries.

Other than [8], the literature is heavily lacking in terms of the actual application of modern deep learning techniques to perform functional approximations on the complex processes involved in self-assembly. With this lack of literature in mind, this research seeks to offer a humble contribution to this multidisciplinary area and highlight the exciting use of modern deep learning techniques to problems of self-assembly where 'humans cannot design the set of components and their interactions.'[21]. Deep learning will perhaps present a way to avoid devising human pioneered, complex, heuristic solutions to the forwards and backwards problem and instead approximate an accurate solution for them based on natural data.

6 Methodology

6.1 Data Scraping

Training data was scraped from the Nottingham University website containing supporting evidence from [22]. The data consisted of labeled images showing Porphyrin conformations for a given state vector. The data came in the form of 5941 *512x512* RGB Portable Network Graphics (PNG) of Porphyrin conformations (Figure 6) The filenames of the images included each of the state vector variables, E_{11} E_{22} E_{12} E_s, respectively.

Figure 6: A random subset of the dataset

6.2 Complexity Analysis

Kolmogorov Complexity[10], as defined by Andrey Kolmogorov, is a measure of the complexity of an object by calculating the length of the shortest computer program on a Universal Turing Machine required for the reconstruction of the object. This is an entirely theoretical metric as one cannot calculate with any level of mathematical precision exactly how many symbols are needed for computational reconstruction. In practice, estimations or approximations are used to compute Kolmogorov Complexity through the potential for reconstruction. One such tool for calculating this potential for reconstruction is lossless compression. Lossless compression algorithms function by minimizing the amount of represented data while preserving the capability for reconstruction on inflation. Thus, the extent to which an object can be compressed is telling of its 'complexity': if the compression rate is high, the object has structure and a certain level of order or repeatability. On the other hand, if the compression rate is low, the object must be of a less complex, more stochastic structure.

An estimated Kolmogorov Complexity analysis was conducted to analyse the complexity of the dataset. This would also highlight the heavily ordered assemblies of Porphyrins and demonstrate where data had to be later augmented and segmented[23] to improve the influence of 'rare'[23] instances on learning. An algorithm was written to traverse the dataset and perform lossless compression using the DEFLATE compression algorithm. For each instance, the Kolmogorov Complexity was plotted to produce a complexity analysis of the whole dataset (Figure 7). This analysis shows that dataset instances with very high Kolmogorov Complexity are extremely sparse and instead the dataset consists of mostly stochastic assemblies that have a very low Kolmogorov Complexity rating. This was initially worrying as the stochasticity would heavily impact the ability for machine learning algorithms to learn. Attempts to mitigate this and improve the learning ability are presented later in this section.

Figure 8a shows an ordered, complex formation of Porphyrin molecules west of the centre of the substrate. This particular assembly is from the state vector $\vec{S} = [0.5, 0.1, 0.1, 0.3]$. On closer inspection through a 200% zoom of this particular assembly (Figure 9a), it is evident that the molecules take an ordered form with structural repetition. On graphical cross-referencing, an extremely high Kolmogorov Complexity estimation of *6.4* is reported. This is one of the highest values in the dataset and reflects the ordered, repeatability of

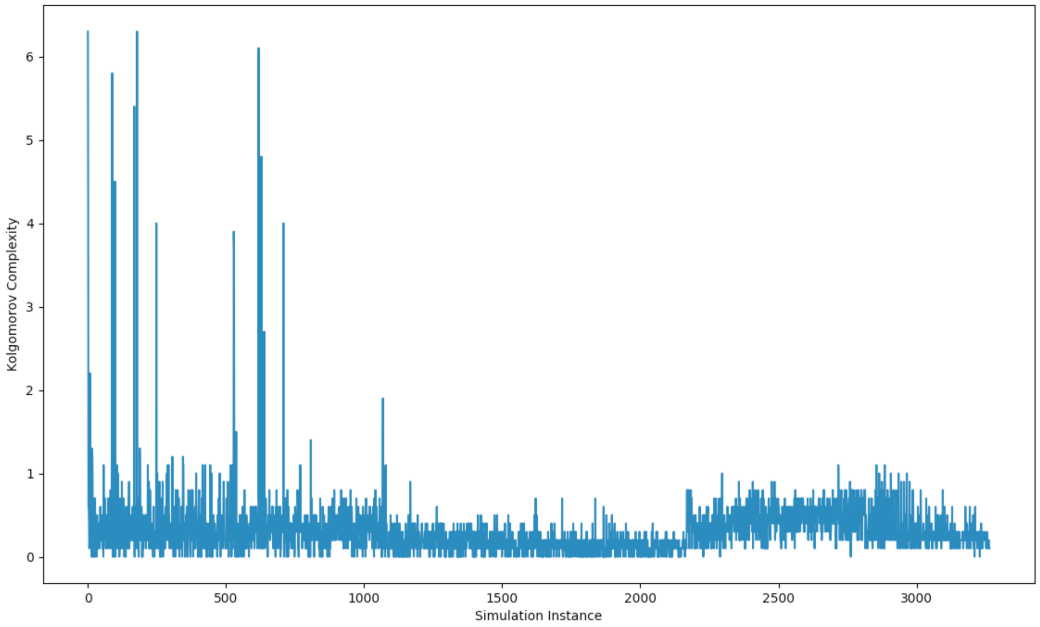


Figure 7: The Kolmogorov Complexity across all simulated images in the dataset

the self-assembly for this state vector. Compare this with a stochastic diffusion of Porphyrin molecules on the substrate with a high energy state vector $\vec{S} = [1.0, 1.0, 1.0, 1.0]$ as shown at a zoom of 200% in Figure 9b, revealing the stochastic nature of the conformation. For this conformation, a small compression rating is recorded of just *0.2* as the stochastic nature renders it difficult to produce a smaller, compression based representation. As Kolmogorov defines: the length of the computer program required to reconstruct 8b would be much longer than the program required for reconstructing 8a. It was thus evident that (high) low Kolmogorov Complexity was indeed related to the (complexity) randomness of the instance.

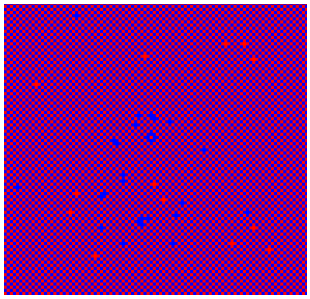
(a) An ordered assembly

(b) An unordered assembly

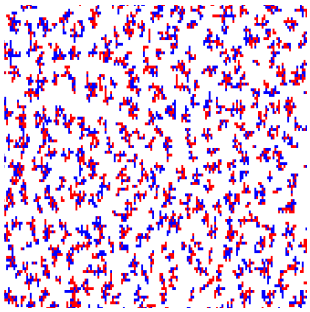
Figure 8: Varying diffusions of Porphyrin Molecules on the substrate

6.3 Normalised Compression Distance

Estimated Kolmogorov Complexity through compression and specifically its use in Normalized Compression Distance (NCD)[4] is a useful metric in many forms of scientific research due to it being a general measure of similarity. Traditional methods for calculating similarity such as histogram bins in music analysis[4] and alignment in genomic phylogeny require 'specific and detailed knowledge of the problem area, since one needs



(a) Micro-structure of 8a



(b) Micro-structure of 8b

Figure 9: The micro-structures of an ordered Porphyrin conformation and a stochastic Porphyrin conformation

to know what features to look for'[4]. Instead NCD captures 'every effective metric: [an] effective version of Hamming Distance, Euclidean Distance, Lempel-Ziv Distance and so on'[4]. Normalized Compressions Distance using a compressor C and two objects x and y is defined in [4] as:

$$NCD(x,y) = \frac{C(xy) - min\{C(x), C(y)\}}{max\{C(x), C(y)\}} \tag{4}$$

(a) (b) (c) (d)

Figure 10: Varying assemblies of Porphyrin Tiles

A measure of similarity that is separate from the esoteric complexities of the self-assembly process means that analysis can easily be carried out on outputs from the functional approximation in the results section for the forwards model. Firstly, however, an NCD algorithm taken from [4] was used to highlight the effectiveness of using NCD on this dataset. Comparing Figure 10a to Figure 10b resulted in a very low Normalised Compression Distance of only *0.9773*. This is evidently because the assemblies have a very different conformation, Figure 10a being highly stochastic arrangement and Figure 10b being an extremely ordered, repeated sequence of Porphyrins assembled into a square-like geometry. However, comparing Figure 10c to Figure 10d produced a lower Normalized Compression Distance of *0.9604* due to the fact they both share a similarly positioned, dendritic formation, thus increasing the similarity score. Moreover, on comparing Figure 10c to itself, an NCD value of *0.0053*[2] was produced. This value is expected since they are the same image, thus highlighting the effectiveness of Normalized Compression Distance as a similarity metric. NCD can analyse the similarity of Porphyrin conformations without knowledge of the process involved in assembling them. Not only will Normalised Compression Distance prove imperative in comparing predictions from the forwards neural network to the actual conformations from the KMCS dataset, it will also be used to prove

[2]This value will never be exactly 0 due to the nature of compression algorithms

that augmenting a dataset for verification of the backwards network creates a 'quasi-new' dataset.

6.4 Data Preprocessing

Before the data could be used in the deep learning models, data preprocessing had to be applied to the instances in the dataset. Preprocessing is the act of transforming a dataset in order to prepare it for use with a machine learning model and the mathematics of learning, namely Stochastic Gradient Descent (SGD). A data processing module was developed in Python for manipulating and transforming Porphyrin images. The module contained functions for scraping filenames to create label data-frames, writing label files, normalizing and manipulating images, data storage through *pickle* and computing normalised compression distance.

The filename of each instance in the dataset was scraped for the state vector values and saved to a CSV file ready for use by the popular data analysis module *pandas*. The dataset labels were loaded and any null values were removed so as to avoid interference in the learning process. Throughout this paper, X will refer to the current training set and y will refer to the labels. X and y were alignment shuffled using scikitlearn to avoid bias. That is, bias through predicting based on the linear progression of the state vector as opposed to the actual function approximation $f'(X) = y$ that maps the two. The size of each image was reduced from *512x512* to *120x120* and re-encoded as gray-scale images. The data integrity of the gray-scale images was verified by ensuring that: $\forall P \mid P \in \{x, y, z\}$. Where P is every pixel, x represents one Porphyrin class, y represents the other class and z represents the substrate, respectively. The images were then normalized so pixel values were between 0 and 1. Normalisation massively reduces the training time of a model without affecting the accuracy or integrity of the learning. The preprocessed images were then converted to Numpy arrays. Numpy is a research grade scientific computation library for Python that allows high level manipulation of multi-dimensional data structures at extremely fast speeds. As Python is interpreted, Numpy passes the high level Python structures to a C back-end for fast, low level computation - this bodes extremely well for the machine learning modules being used as the backbone for this project. The Numpy arrays were saved using *pickle* for reuse convenience. Figure 11 shows that integrity of the image remains even after preprocessing - it is possible to identify the two different classes of Porphyrin tiles and the geometry, be it with randomness or order.

Figure 11: A Porphyrin conformation after pre-processing

6.5 Machine Learning Technology Stack

Before any training or network design could take place, hardware and software had to be procured from an IaaS provider. This subsection documents the technologies used for building, verifying and training the machine learning models.

6.5.1 Tensorflow, Tensorboard and Keras

Tensorflow is an open source Machine Learning library developed by the Google Brain team that allows developers to easily develop and deploy machine learning algorithms. It is a large library consisting of numerous machine learning tools, models and algorithms for creating effective machine learning solutions at any scale. The industry standard language for interfacing with Tensorflow is Python. Despite Python being an interpreted language, Tensorflow uses Python as an accessible means for developers to interact with a complex C++ library written to take full advantage of modern hardware. Tensorflow is used in this project as a the mathematical backbone, along with another open source library, Keras (that runs on top of Tensorflow). Keras is an higher level library, geared towards the rapid development and experimentation of machine learning architectures. Tensorflow and Keras are extensively used in commercial industry by companies such as a Dropbox, Airbus and IBM. Both are also used in many fields of research, notably computer science, physics, biology and chemistry for cutting edge natural language processing, translation, simulations, protein folding predictions and speech, to name a few. Tensorflow and Keras will be used to build both machine learning models and specialise them for the forwards and backwards problem.

Tensorboard is a graphical suite for analysing the training efficacy of machine learning models. It allows developers to easily graph results from both the training and validation processes. Tensorboard will be used to analyse the data from both models over numerous epochs to aid in the discovery of optimal architectures and hyper-parameters.

6.5.2 Google Cloud Platform

Due to the computationally intensive nature of training a Deep Learning model, this project makes use of a Virtual Machine hosted by Google as part of the Google Cloud Platform (GCP). GCP was first released in 2008, offering *IaaS* computing environments for various domains such as business, technology and research. Specifically, it offers a 'Compute Engine' service consisting of customisable Virtual Machines (VMs) specifically designed for machine learning. These employ high end, customizable hardware and software solutions for running machine learning models on the cloud without the need to invest in expensive infrastructure. It functions on a pay-as-you-go scheme in which users pay for the amount of CPU time they are using. The most attractive asset of Google Compute Engine is the ability to apply for GPUs and TPUs. TPU, or Tensor Processing Unit, are Google's bespoke computing chip for machine learning in Tensorflow. Based on my application, access was granted to an extremely powerful GPU that would vastly improve training time. Unfortunately, access to a TPU was not possible for this project as there was no budget available. This project would not have been possible without access to this service due to the methodical approach taken to tuning and training complex machine learning models. The specifications of the hardware and software used by the cloud VM can be found in Table 2.

6.6 Deep Feed-Forward Networks

After much research, it was decided that the problem of reducing the KMCS in the forwards and backwards direction would be best solved using a densely connected, deep feed-forward regression network. At their core, deep feed-forward networks with supervised learning approximate some function mapping a vector of features X to an output y. This consists of 'driving $f(x)$ to equal $f'(x)$'[5]. They are known as feed-forward networks as this is

Hardware	Software
4 vCPUs	Debian 8.9
15GB RAM	Tensorflow 1.13.1
1 x NVIDIA Tesla P100	Keras 2.2.4
50GB Solid State Drive	Nvidia CUDA 10.0
	Python 3.6.5

Table 2: Summary of the Hardware and Software used

the directional flow of data through the network inputs to the outputs. In this type of architecture, the machine learning algorithm must compute how to use the hidden layers, specifically the weight values between interconnected nodes, to achieve an accurate function approximation $f'(x)$ based on repeated loss evaluations and stochastic gradient descent. Since the data from the KMCS is non-linear, the interconnected nature of a deep feed-forward network is required over more traditional machine learning models to learn the subtleties of the mapping between state vector and Porphyrin conformation.

Designing a machine learning model, especially a neural network, should be based on prior research done in that domain and empirically based, centering around methodical changes and meticulous evaluation of the training process. Empirical investigation is especially difficult to conduct if the data is heavily stochastic as learning may be subtle. Due to the complexity of the dataset as identified in Figure 7, it was practical to implement two boilerplate networks, specialised for the backwards and forwards problem and fine-tune them on a representative subset of the data. A representative subset of the data allows for methodical experimentation in order to identify optimal architectures and hyper-parameters. This meant two optimal models could be developed without wasting expensive computational time on the cloud. Without the representative subset, evaluating a small change in each model would require hundreds of passes over the full dataset - evidently not feasible with the time and budget allowance.

A subset of 300 images was created through a representative sampling of the original dataset. The size of the subset was decided as a midway point between overfitting and underfitting. Overfitting is a phenomenon in machine learning when a model learns a dataset too well and therefore cannot generalise to the problem domain. Underfitting occurs when the data is too complex to learn a function approximation from. Too little data would render the boilerplate experiments useless due to overfitting but too much would reduce the amount of effective tuning and optimisation that could be undertaken. 300 images allowed for a balance between effective learning of the KMCS function approximation, while also permitting rigorous testing. The main aim of the boilerplate models was to determine suitable loss functions, learning rates and optimization techniques. Thus, numerous essential design questions for both models were proposed:

1. What is the optimal network architecture?
   (a) Neurons per layer?
   (b) Number of hidden layers?
2. What are the optimal loss functions?
3. What is the optimal learning rate?
4. What are suitable techniques for optimization and regularization

Two feed-forward boilerplate networks were written in Python using Tensorflow and Keras. A pipeline was created to propagate data between the data processing module and the machine learning models. The models were then individually specialised for the forwards and backwards problem. Methodological investigation was carried out by creating an object oriented tuning module in Python to cycle through layers, neurons, learning rates, optimisers and regularization / optimisation techniques in the boilerplate models. The module was uploaded to the cloud and each architectural / hyper-parameter change was executed on the compute engine for 100 epochs[3]. 100 epochs was chosen as it would sufficiently show the (in)effectiveness of each model while avoiding extensive training times. The results from the 100 epochs were graphed using Tensorboard. In this section each experiment is presented alongside an in-depth discussion. Visual summaries of the optimal architectures are then presented (Figures 21a, 21b) and a final summary of the tuned hyper-parameters can be found in Table 3.

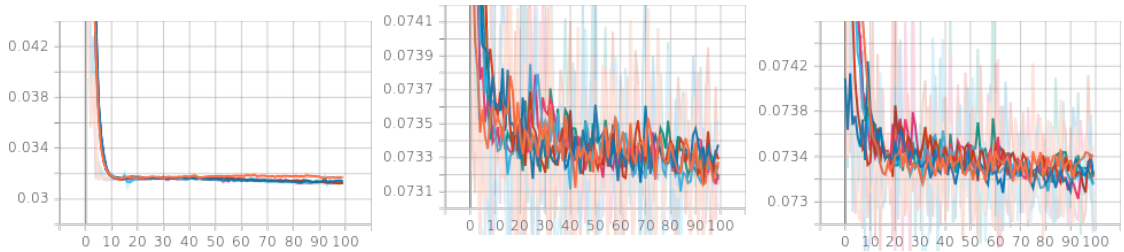
6.6.1 Architecture

The input and output layers of both networks had to be in a specific format regardless of the hidden inner layer structure of the networks. This was dependent on the input and output data dimensions. Thus, the experiments into the architecture focused on optimising the amount of hidden layers between input and output layers.

The forwards model only made use of dense layers but the backwards model was unique in that it used 2D Convolutional layers with max pooling to deal with the Porphyrin image data as input. A convolutional layer performs a dot product on the inputs it receives and passes this to the next layer of the network. A convolutional filter with specified dimensions traverses the data searching for 'features'. Features may become more pronounced as data moves through convolutional layers. For example, if a convolutional layer was being used for classification of handwritten digits, the first convolutions may learn to detect the lines that make up digits, the second layer may detect the ovals that form the numbers 6,8,9 and so forth. In the context of Porphyrin tile self-assembly, the convolutional layers may learn to detect a dendritic conformation or a dense conformation. These convolutions over the data form what is known as a 'feature map'. The max pooling layer then reduces the feature map into a smaller representation by taking the maximum value in each filter position. Max Pooling in this manner has been shown to improve training time and accuracy. After the convolutional and pooling layers, the output is flattened and enters into a variable amount of Dense layers. Thus three experiments were conducted here involving both convolutional and dense layers in the backwards model and dense layers in the forwards model. A supernumerary hidden layer was added each at each iteration of the Python module and the validation loss of each configuration was graphed in Tensorboard (Figure 19).

The backwards model validation loss was extremely erratic with extreme oscillations - this speaks to the stochasticity of the dataset. After smoothing the graphed data, it is noted in Figures 12b and 12c that the backwards model performed similarly with different combinations of dense and convolutional layers. This could be because the regression in the backwards direction has a smaller dimensionality than the regression in the forwards direction. This was beneficial as it means the network could be of a reduced complexity through using less convolutional and dense layers and would thus reduce the training time of the final optimised model over the whole dataset. The forwards network on the other hand experiences a very slight improvement in reducing the loss function when using 6 hidden Dense layers between the input and output. This wasn't too detrimental to training times since Dense layers are less computationally intensive than convolutional layers.

[3]It is important to note that further testing would be desirable but was restricted by computational time allowances on GCP



(a) The forwards model hidden dense layers over 100 epochs

(b) The backwards model hidden convolutional layer over 100 epochs

(c) The backwards model hidden dense layers over 100 epochs

Figure 12: Graphs showing the architectural optimisation experiments for both models

6.6.2 Activation Functions

Rectified Linear (ReLU) activation functions were used for the convolutional layers in the backwards network as this has been shown to vastly speed up training times and accuracy with convolutional layers. The dense layers in both models used a sigmoid activation function to reflect the normalized image data and the values in the state vector as sigmoid produces outputs between 0 and 1.

$$S = \frac{1}{1 + e^{-}x} \tag{5}$$

Figure 13: The Sigmoid Function

$$f(x) = max(0, x) \tag{6}$$

Figure 14: The ReLU Function

6.6.3 Neurons

(a) Neurons per hidden layer in the forwards model

(b) Neurons per hidden layer in the backwards model

Figure 15: Graphs showing the validation loss of the boilerplate networks using varying amounts of neurons in the hidden layers: 16 (Orange), 32 (Dark Blue), 64 (Red), 128 (Light Blue)

The number of neurons per each hidden layer determines the amount of complexity presented to the model in order to perform a functional approximation. That is, if there are too many neurons, the model may have trouble learning as it will be too complicated

to successfully update weights between them to reflect a concise approximation. Too few neurons will reduce the model‘s ability to learn the deeper complexities involved in the KMCS data.

It seems that for the forwards model, 64 neurons per hidden layer was optimal in reducing the validation loss. This may seem rather high when compared to the forwards network optimal of 16 neurons per hidden layer. However, this is logical: there are 14400 output dimensions in the forwards direction and therefore a larger dimensional space in the hidden layers is desirable - evidenced further by the fact that the second most optimal configuration is 128 neurons per layer. The theory holds in the other direction too: the backwards model is attempting to ‘scale’ down from Porphyrin conformation to state vector and therefore a smaller dimensionality in the hidden layers performs better.

6.6.4 Loss Functions

A loss function in a feed-forward network determines the ‘error’ of the models predictions during training. That is, a value of how incorrect the model is over the current batch of the dataset. Using an effective loss function in a model is critical as it determines how well it will train. For the forwards model, it was determined that the best hypothetical loss function would be a minimization of Normalised Compression Distance between the target assembly and the predicted assembly. Unfortunately, this technique is computationally expensive due to the concatenation and compression of two separate objects meaning it was not practical in this case. With enough computational power, NCD would perhaps be an extremely useful metric for a loss function in this domain. Instead, two loss functions were compared, namely Mean Squared Error (MSE) and a distance based loss function known as Constrastive Loss (CL). Constrastive loss is found in Siamese networks used to detect image similarity. In this case it is used in a slightly different capacity to how it is used in Siamese networks for input image comparison. Instead it acts by comparing the predicted conformation to the actual conformation for that feature set. It utilises a classic Euclidean distance algorithm but applies a margin to punish bad predictions: ‘contrastive loss function is employed to learn...in such a way that neighbors are pulled together and non-neighbors are pushed apart.’[6] Mean squared Error and Constrastive Euclidean based loss are presented in Equations 7 and 8, respectively.

$$L(W, Y, X_1, X_2) = (1 - Y)\frac{1}{2}(D_w)^2 + Y\frac{1}{2}\{max(0, m - D_w)\}^2 \tag{7}$$

Where D_w is a distance function and X_1, X_2 are vectors

Figure 16: The Constrastive Loss[6] function used in the forwards model

$$1/n \sum_{i=1}^{N} (a_i - y_i)^2 \tag{8}$$

Where n is the dataset, a is the prediction by the model and y is the actual prediction

Figure 17: The Mean Squared Error loss function used in the backwards model

In Figure 18 the euclidean based loss function unexpectedly failed to reduce the validation loss. MSE, however, reduced the validation loss and reached a respectable validation

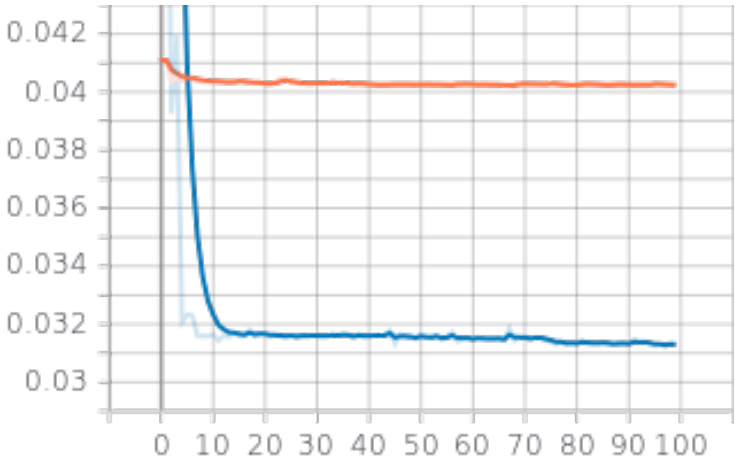


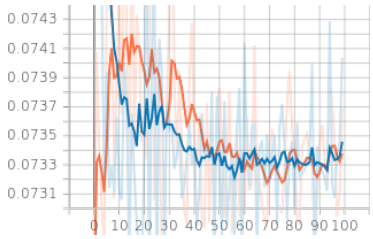
Figure 18: Mean Squared Error (Blue) vs Constrastive Loss (Orange) in the forwards model

loss of just *0.031*. This could be due to the fact MSE handles outliers better than a Euclidean distance based algorithm. That is, it is possible that the stochasticity of the subset was having a negative effect on the euclidean based formula and causing it to not learn effectively. However, it is extremely strange that constrastive loss did not reduce the validation loss at all and perhaps highlights an underlying problem with its implementation in this context.

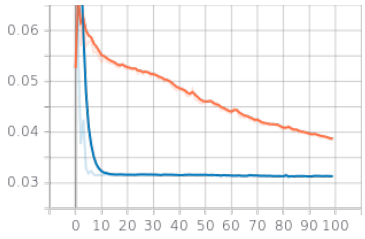
6.6.5 Optimizers

The optimizer in an ANN controls the gradient descent of the network by working with the loss function to move towards a global optimum based on weight updates between interconnected neurons. Choosing the correct optimization technique is essential for ensuring the model can learn quickly and confidently. Although there are a few standards for optimization techniques in certain scenarios, empiricism and testing is the best method to identify an effective optimization technique for a given model. For both models, Adaptive Moment Estimation (Adam) and RMSProp were compared. Adam is a method for stochastic optimization introduced by [9]. It is hugely popular in the Machine Learning community and has been rising in popularity ever since its introduction. Adam is an adaptation of classic stochastic gradient descent but uses 'momentum' to move towards an average of the gradient instead of the actual gradient and updates learning rates *automatically* relative to the descent. RMSProp is a unique optimizer proposed by Geoffrey Hinton and famously never published. It is similar to Adam in that it moves with momentum but utilises a moving average to normalize the gradient which avoids the vanishing and exploding gradient problem. It is most notably used by Google Deep Mind in reinforcement learning models for playing Atari games.

In both models, the performance of Adam was optimal in terms of initial learning rate and the correct descent direction towards a minimum. RMSProp starts with a high learning rate that causes a regression in validation loss before gaining momentum in the correct descent direction. It is also evident that in terms of this dataset, Adam is more stable and predictable at reducing the loss function - compared with the erratic nature of RMSProp. Furthermore, Adam achieves a lower loss than RMSProp for most of the 100 epochs. Although RMSProp is slightly faster over the 100 epochs (9 seconds versus 12), it seems the obvious choice for both models is Adam.



(a) Adam vs RMSProp in the forwards model



(b) Adam vs RMSProp in the backwards model

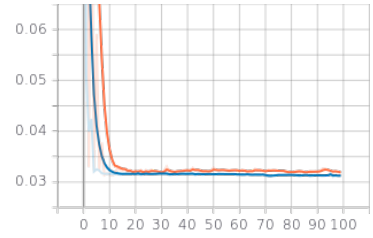
Figure 19: Graphs showing the validation loss of the boilerplate models using optimizers Adam (Blue) and RMSProp (Orange)

6.6.6 Learning Rates

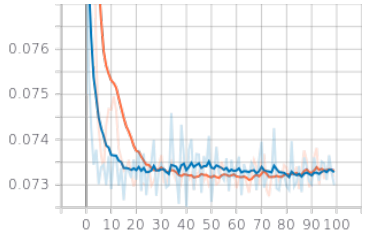
Learning Rates can have a huge impact on how the model performs during training. A high learning rate means that the gradient descent is more pronounced whereas a low learning rate means movements towards the gradient are less pronounced. High learning rate will train a model faster but risks not finding the global optimum due to high levels of oscillation around the minima. A small learning rate will normally reach the global minimum with reduced oscillatory behaviour but it may take many more epochs to achieve this. Evidently, there exists a optimal learning rate between these two extremes that provides fast *and* accurate training. Fortunately, Adaptive Moment Estimation solves this dichotomy through momentum and automatic changes of the learning rate.

6.6.7 Layer-Based Optimisation and Regularisation

Optimisation and regularisation are also important concepts to consider when fine tuning a neural network architecture. Batch Normalization (BN) is a poorly understood optimization technique for improving the speed and stability of a network by scaling the activation outputs. Researchers believe this scaling achieves a smoothing of the objective function[17]. Dropout[20] is a regularization technique to reduce overfitting and introduce noise to the model. It functions by dropping a neuron from a layer with a given probability. This forces the model to re-balance to account for the omitted neuron and therefore it slightly changes its learned representation of the dataset, reducing overfitting and increasing the ability to generalise. Batch Normalization and Dropout were added to both models and compared to the normal behaviour of the models without optimisation and regularization techniques.



(a) Batch Normalization and Dropout vs No Batch Normalization and Dropout in the forwards model



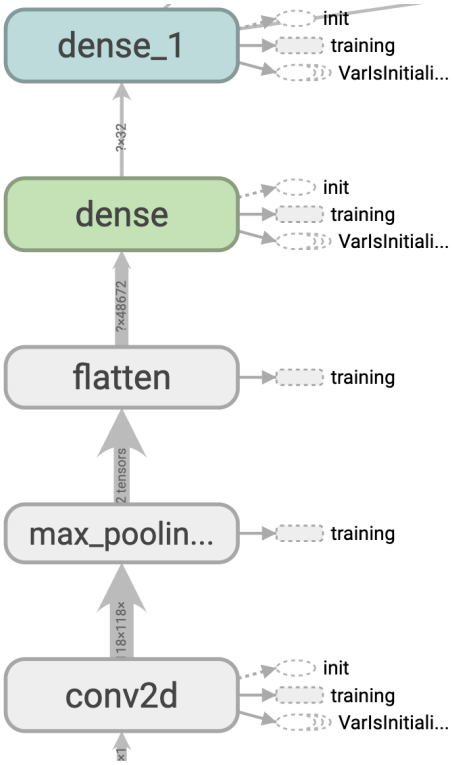
(b) Batch Normalization and Dropout vs No Batch Normalization and Dropout in the backwards model

Figure 20: Graphs showing the validation loss of the boilerplate models using batch normalization and dropout (Orange) versus without batch normalization and dropout (Blue)

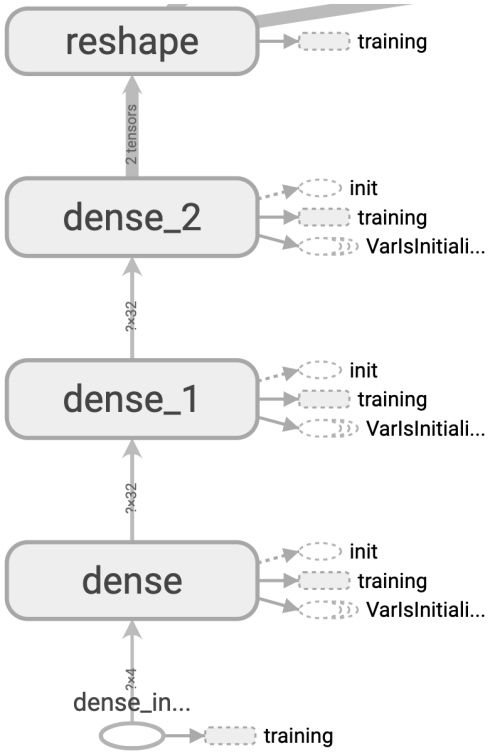
Figures 20 show that, surprisingly, batch normalization and dropout reduced optimality in both models by almost doubling the time taken to achieve a smaller reduction in validation loss. This could be explained through the complexity of the dataset. Dropout was forcing the refitting of the model with missing neurons but due to the stochastic nature of most instances in the dataset, the added level of model complexity lead it less towards global minima exploration and more towards the exploitation of local minima. Since they were used together, Batch Normalization possibly didn't improve stability due to the deficits created by the dropout.

6.7 Summary

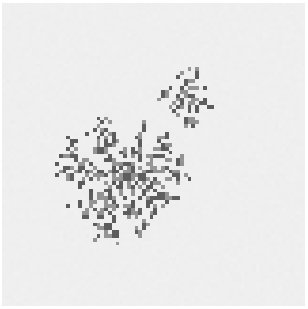
After conducting the optimisation experiments on the representative subset, two neural network models were created using the reported optimal configurations. The optimal forwards model was trained again on the smaller subset to verify that the predictions were working as expected and learning properly. The results are presented in Figure 22.



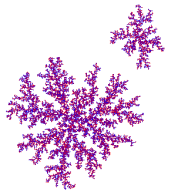
(a) The final architecture of the backwards model



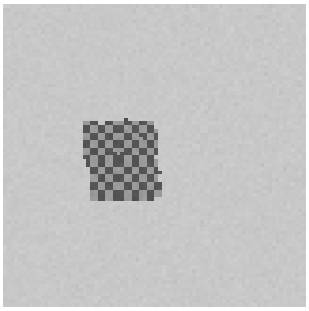
(b) The final architecture of the forwards model



(a) $[0.5, 0.1, 0.1, 0.7]$



(b) $[0.5, 0.1, 0.1, 0.7]$



(c) $[0.5, 0.1, 0.1, 0.3]$

(d) $[0.5, 0.1, 0.1, 0.3]$

Figure 22: Comparisons between the optimal forwards network and the actual diffusions as found in the dataset

	Forwards Model	Backwards Model
Neurons P/L	6	1 Convolutional, 1 Dense
Activations	Sigmoid	ReLU, Sigmoid
Loss functions	Mean Squared Error	Mean Squared Error
Optimisers	Adaptive Moment Estimation	Adaptive Moment Estimation

Table 3: A summation of the optimal neuronal and hyper-parameter combinations of both models

It was extremely promising to see that 22c was producing the same organised structure from the actual Porphyrin assembly. This proved the model was learning to reproduce assemblies with a high level of accuracy. However, this was merely a proof of concept that the models could perform how they were intended and a much larger dataset was required for meaningful predictions. Furthermore, the predictions were suffering heavily from interference (through tending to the mean), due to the stochastic imbalance in the representative subset that were also manifest in the main dataset. This is extremely

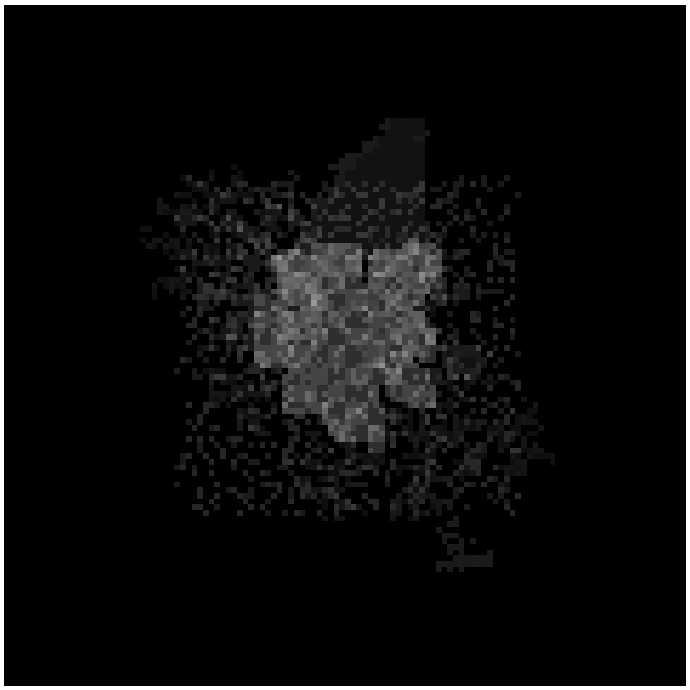


Figure 23: Clear interference in the prediction of a Porphyrin assembly

evident in Figure 23 where the pixels have been inverted for convenience; notice how the predicted assembly contains noisy interference from many other conformations in the background.

6.7.1 Data Imbalances and Random Noise

A possible explanation of the interference is found in the data analysis. The vast majority of Porphyrin Tile conformations in the dataset are assembled stochastically on the substrate i.e. they have very low Kolmogorov Complexity as in Figure 8b. Moreover, as aforementioned, instances of high Kolmogorov Complexity comprise a very small segment of this dataset. In fact, there are so few instances of Kolmogorov Complexity > 1 such that these instances would have very little impact on the overall functional approximation of the KMCS. When training Deep Learning models, it is important to have a balanced dataset otherwise the algorithm will learn a bias towards the majority instance. As the majority of y corresponds to stochastic Porphyrin assemblies that take a very similar form to Figure 8b, the algorithm finds itself at a local minimum and will learn to predict randomly each time, thus learning a bias. This is a problem of instance rarity[23] and stochasticity, both of which cause extreme problems in machine learning models. To prevent this tendency towards the dataset mean of stochasticity, a solution for this problem was attempted through data repetition, segmentation[23] and noisy augmentation. It was found in [18] that adding small random noise to a dataset can improve the accuracy and training process of machine learning models substantially. This essentially forces the neurons to create stronger connections that do not rely on a homogeneous representation of one instance in the dataset and thus makes the network more generalisable. Dataset Augmentation is the process of slightly manipulating the pixel representation of an image. Augmentor[3], was used to simplify the augmentation process and add small random noise through random distortions using complex mathematical operations. Augmentor is also pipeline based meaning it can be easily integrated with Tensorflow and Keras. Random distortions of a medium magnitude were applied to one randomly selected Porphyrin assembly and NCD analysis was carried out to show the effectiveness of random noise

through augmented distortions.

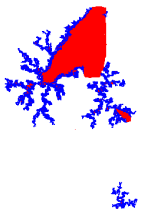
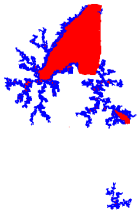
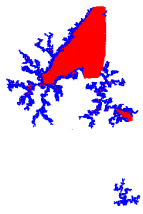


Figure 24: The original Porphyrin conformation to be augmented



(a) 0.9732334047109208



(b) 0.969532100108814

Figure 25: Examples of some randomly distorted Porphyrin assemblies

The random distortions in 25a and 25b are undetectable to the naked eye. However, on calculating the NCD between the distorted and the original assemblies, the difference is significantly high. For Figure 25a the NCD from the original conformation is *0.973* and for Figure 25b the NCD was *0.9695* making them both around 97% different to the original. These images, despite appearing the same, have an extremely different pixel makeup which is being detected by the NCD algorithm. Essentially, new data has been created here in the eyes of the machine learning model but the geometric conformation has remained unchanged. Random noise was added to every instance in the dataset with $P = 1.0$ to improve training effectiveness and significantly reduce the effects of data imbalances. Further, a data segmentation technique similar to that found in [23] was used to address the problem of rarity[23] - the first third of the full dataset was distorted and repeated to account for the lack of representation of assemblies with high Kolmogorov Complexity in an attempt to reduce the bias towards stochastic assemblies. By augmenting and repeating, it was ensured that 'rare' instances in the dataset comprised a more significant proportion so as not to effect the inductive bias and 'the ability to learn rare cases and rare classes'[23].

6.7.2 Training the Networks Across the Whole Dataset

The feasibility networks were applied over the whole augmented dataset and trained over 500 epochs. A random 20% split of the dataset was used as testing validation in both networks. The next section reports the results of the training and the predictions of both networks.

7 Results

Exploring the training performance and verifying a machine learning model is essential to surmising its effectiveness in its domain. Analysing the effectiveness of a model hinges around two principles: a) checking that the model is training effectively and b) validating the predictions of the network on novel data. In most cases this is trivial: new data can be gathered that has not been used in training. In an image classification problem, for example, data outside of the training and testing dataset can be obtained through numerous means for verification purposes. Unfortunately, in this scenario, there was no access to the KMCS meaning that no novel data could be generated. Even though the model was trained with a 20% validation split, it is imperative to investigate its predictive quality on new data. In this section, one technique is proposed to circumvent this limitation and provide 'quasi-unseen' data. Results from training both networks on the Google Compute Engine VM for 500 epochs are presented and a prediction analysis of each dataset is given and discussed followed by a summary of the results quality and how the results of this research coincide with the original defined objectives.

7.1 Model Training

7.1.1 The Backwards Model

The backwards model was the slowest to train due to the added complexity of convolutional and pooling layers processing images with 14400 dimensions. Despite the slow training progress, the optimised backwards network learned how to minimize the mean squared error of the state vector predictions over the augmented dataset (Figure 27). The model trains rapidly following initialisation and learns in a short amount of time. Although the model only reduces the loss by a small margin, the model does converge on a minimum at epoch 81, achieving a validation loss of *0.048*. The rise in loss after this point could be due to numerous factors, for example, a tendency to overfit on the data by moving towards the mean in the dataset - aggravated by the stochasticity. The model is indeed training - and learning - from the dataset. Whether this training is valuable and correct will require further graphical analysis.

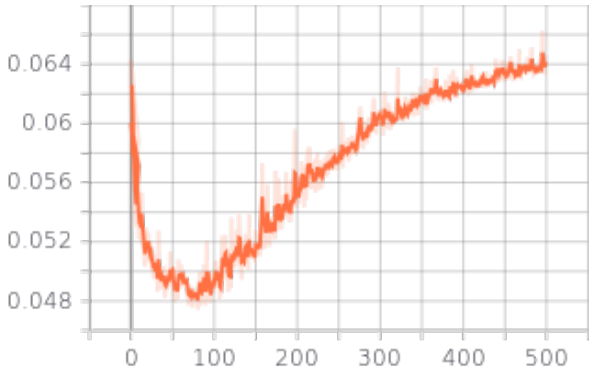


Figure 26: The backwards model validation loss function over 500 epochs

7.1.2 The Forwards Model

The forwards network learns slowly despite the use of adaptive moment estimation. Unfortunately, it appears that the network architecture isn't very successful at learning from the full dataset. This is perhaps due to the 14400 dimensionality of the prediction. With every dimension to predict, there is a higher probability of producing an erroneous prediction. More training time would be necessary to examine the extent of learning. However,

judging from the slow reduction in validation loss, it would take an inordinate amount of time to produce valuable results and would perhaps result in drastic overfitting.

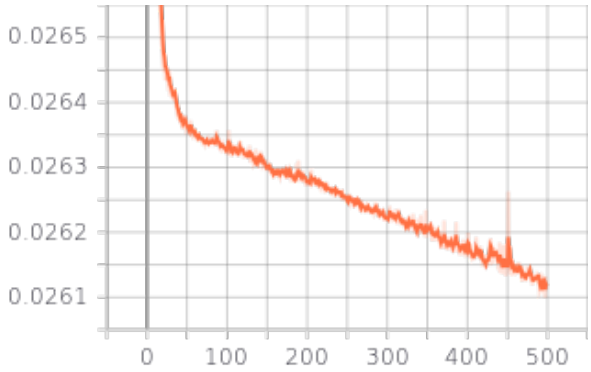


Figure 27: The forwards model validation loss function over 500 epochs

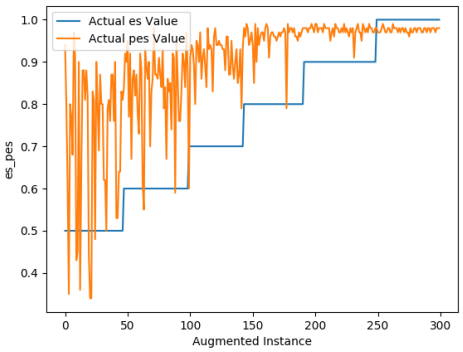
7.2 Prediction Validation

7.2.1 The Backwards Model

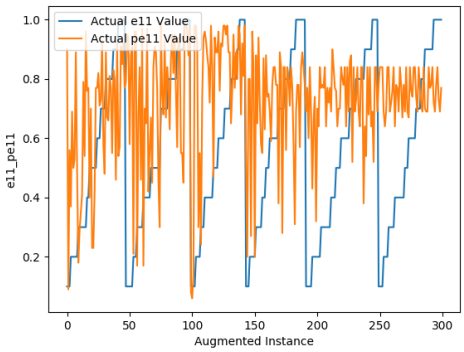
The proposed technique for backwards model verification uses data augmentation in order to create a 'quasi-new' dataset based on a representative sample of the the main dataset. As evidenced from the prior section, augmenting the data creates a 'quasi-new' set of images in the eyes of normalised compression distance and the machine learning model. A fresh dataset of 300 images was created using Augmentor with the same random distortion techniques used to reduce bias in the dataset. The trained model with the lowest validation loss (epoch 81) was applied to this augmented dataset, predictions were written to file and graphed using matplotlib. (Figures 28a, 28b, 28c, 28d)

Figures 28a, 28b, 28c and 28d are extremely promising, showing that the predictions move with the trend of the true value.[4] The machine learning algorithm has learned about each individual parameter in the state vector and the effect it has on the overall conformation of Porphyrin tiles. The model encounters problems and falls into erroneous prediction, oscillating between 0.75 and 0.8 when the energy values in the state vector are higher. This could be due to data imbalances but is likely due to the increased stochasticity of the Porphyrin assemblies in a high energy environment. Despite this, it seems that the backwards model has learned a function approximation of the KMCS for individual parameters in the state vector with an unexpectedly high level of success. However, the same quality of results are not produced when the model predicts all four elements from the state vector. Instead, the accuracy drops substantially. Evaluating the model across the whole augmented subset produces a validation loss of *0.0922*. This is much higher than the validation loss on the test set during training of *0.048*. Evidently, the model has overfit on the training dataset and the augmented dataset has reduced its accuracy. Despite this drop in accuracy and generalisability, the produced predictions are definitely beyond random chance. This shows the network has learned a successful function approximation of the KMCS to a *reasonable* extent and can act as a surprisingly effective substitute to the backwards problem.

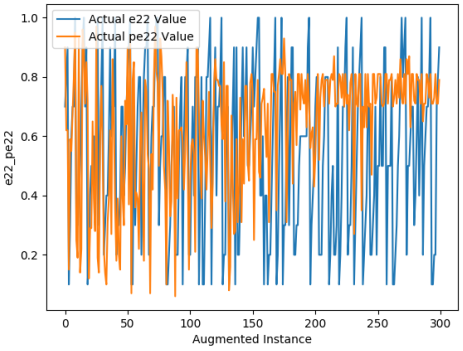
[4]It is important to reiterate that the networks were both trained on alignment-shuffled data and any correspondence in the data is solely due to a functional approximation of the KMCS



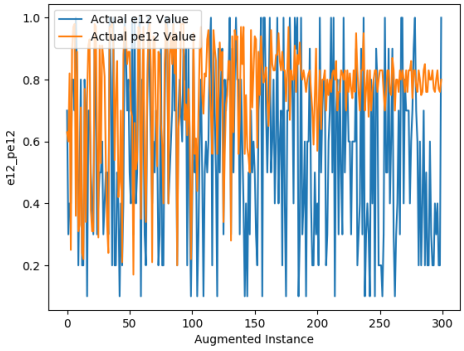
(a) Actual es compared with predicted es



(b) Actual e11 compared with predicted e11



(c) Actual e22 compared with predicted e22



(d) Actual e12 compared with predicted e12

Figure 28: Model predictions for each element of the KMCS state vector

7.2.2 The Forwards Model

Prediction validation across novel data in the backwards direction proved impossible without access to the simulator. Having the ability to simulate new parameters would render verification trivial: the model would predict the conformation for novel parameters, the simulator would simulate those parameters and a normalised compression distance analysis would be applied to both. Instead, an NCD analysis (Figure 29) was carried out across forwards predictions for every possible state vector. This prediction analysis took an extensible amount of time to execute, proving that NCD would not be an effective metric as a loss function with the standard of hardware available. Further, for completeness, a module was written that utilises the trained forwards model to predict Porphyrin conformations over the whole range of the state vector. Some examples of the predicted conformations compared with the actual conformations from the dataset are presented in Figure 30 and an NCD analysis is carried out on them.

Figure 29 shows that the forwards model becomes better at predicting assemblies as the state vector moves from low energy to high energy. Again, this is probably due to the stochasticity of high energy conformations: the model has learned to predict random pixels in order to reduce its loss function. Logically, this holds: if a high energy conformation consists of 14400 individual pixels representing white space and Porphyrins, placing Porphyrins randomly will eventually produce a net reduction in loss. The tendency to predict randomly also effects accuracy in other areas, serving to reduce the similarity between predicted and actual conformations. This is extremely noticeable in Figure 30h.

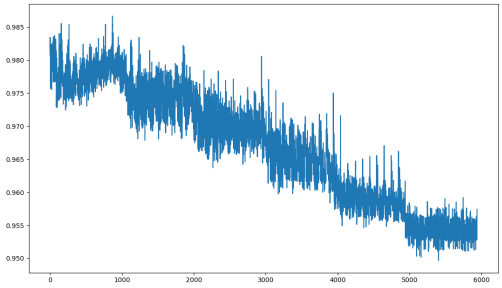
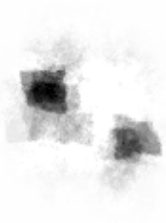


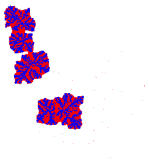
Figure 29: NCD predictions across the whole dataset

Unfortunately, this analysis is perhaps the extent that this research can achieve without having access to the simulator itself. Thus, the question remains open as to whether the model has reached the relevant objective. It is clear from the sub-figures in Figure 30 that the model can accurately predict some Porphyrin conformations with a promising level of success but most of the predictions are extremely noisy. When the forwards model does predict a similar geometry to that of the actual conformation, it performs relatively well and in Figure 30f manages to achieve a similar geometry with an NCD score of *0.9717*. However, it is disappointing to note that 30h has a similar NCD despite the fact it is heavily perturbed by random noise. Thus, it is accepted that the model does not act as a reasonable and reliable substitute for solving the forwards problem. This is perhaps due to the chosen model architecture and an evaluation of other methods for regression in the forwards direction is recommended.

(a) [05010204]



(b) [05010204]
NCD = 0.9794034090909091

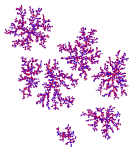


(c) [07010201]

(d) [07010201]
NCD = 0.9814126394052045

(e) [05020601]

(f) [05020601]
NCD = 0.9717925386715196



(g) [06020707]

(h) [06020707]
NCD = 0.9733990147783251

Figure 30: Examples of forwards model predictions over numerous state vectors compared to the actual conformation for that state vector with an NCD analysis between the two

7.3 Summary

A recurring theme in the results is interference due to the stochasticity of the dataset. This caused problems when training both models, requiring extensive augmentation, experiments and meticulous data analysis to produce any form of valuable results. However, this stochasticity is also a fact of nature that cannot be ignored. It is the randomness of nature that makes solution to these problems NP-hard and usually unachievable. Further investigation would be beneficial using heuristic methods for reducing the impact of these stochastic assemblies on the inductive bias of the models and to properly investigate using a data-driven functional approximation as a replacement to the complex simulations and heuristics involved in the current forwards and backwards solutions. Furthermore, judging from the presented results, more training is necessary to fully examine the effectiveness of the forwards model. This was impossible given the circumstances as computational time

allowances were highly limited due to IaaS expenses.

Disappointingly, it was evident from the beginning of this research that the huge stochasticity of the dataset would play a role in undermining the effectiveness of both networks. Although the results do not fully convince an observer that machine learning can or should be used as solutions to the forward and backwards problems, it is interesting and promising to see that there is at least *some* value in applying the techniques. Specifically, as will be discussed further, the results from the backwards network performed beyond expectation.

8 Conclusion

Concluding this research project entails considering the aims and objectives of the project and discussing the extent to which they have been met:

8.1 Aim

1. To reduce the complexities of a Porphyrin self-assembly simulator based on Kinetic Monte Carlo Methods into an artificial neural network function approximation, through data alone, in order to solve the forwards and backwards problem

8.2 Objectives

1. Perform data analysis on the simulator data using Kolmogorov Complexity[10] and Normalised Compression Distance[4]
2. Develop a neural network model to functionally reduce the Kinetic Monte Carlo Simulator in the forwards direction
3. Develop a neural network model to functionally reduce the Kinetic Monte Carlo Simulator in the backwards direction
4. Tune and optimise both models on a representative subset of the data
5. Train both optimal models across the whole dataset
6. Perform further data analysis on both trained models using an augmented dataset to assess accuracy at solving the forward and backward problems.

All of the objectives have been met through the methodological process of this research. A Kolmogorov Complexity analysis was conducted over the whole dataset and a demonstration of the effectiveness of normalised compression distance was given. This data analysis highlighted the essential modifications that were required to handle data imbalances and lack of representation for 'rare' instances. Two models were tuned and optimised using a small subset of the training data to find suitable architectures and hyperparameters. The optimal networks were then developed and trained successfully through the reduction of validation loss functions. These models were able to make predictions in both the backwards and forwards direction. NCD analysis was used to analyse the predictions of the forwards model over the original dataset and for comparing predicted conformations against actual conformations. The efficacy of the backwards model in predicting each state vector value and the whole state vector were graphed and analysed.

The results showed that it is somewhat possible to achieve the aim of reducing the complicated 'black box' KMCS to a conglomeration of weighted neuronal connections and thus build a creative solution to the forwards and backwards problem using self-assembly data alone. The backwards network is especially relevant to the main thesis: namely that the functional approximation simplifies the programmability of self-assemblies since it allows for backwards simulation - an zundoing' of sorts that temporally reverses interactions between entities to predict the parameters for a given molecular self-assembly. The quality of the results are negatively affected due to the high levels of stochasticity present in the data meaning that the models suffered heavily in terms of their accuracies across the board. Further, no access to the KMCS meant verifying the backwards model on augmented data and being unable to verify the forwards model on novel data - severely denigrating the quality of the results and the ability to surmise concrete conclusions.

Undertaking this research project has been immensely engaging, challenging and rewarding. It has enlightened me to the fascinating worlds of molecular self-assembly, biological computation and machine learning. The reward of which is the competent knowledge of many different programming languages, research techniques, frameworks and scientific techniques for conducting data preprocessing, analysis and learning. If given the opportunity to conduct further research on the matter I would investigate the disappointing performance of the forwards model. This could be done through applying Deep Q Learning (DQL), a type of reinforcement learning such that an agent learns to perform actions in an environment depending on the state to receive a reward or punishment. In the context of the Porphyrin self-assembly, the agent would be the input to the simulator, an action would be changing the state vector, the state would be the output of the simulator and the reward would be the minimization of the NCD between a target conformation of Porphyrins and the actual conformation. This was not possible in this piece of research as DQL requires a substantial 'replay memory'[19] which includes millions of pre-encountered states. Furthermore, it may be interesting to investigate the use of Recurrent Neural Networks (RNNs) for both problems using novel layer designs such as Long Short Term Memory (LSTM) or Gated Recurrent Units (GRU) that help networks remember what they have seen before - this retrospective information could prove useful in learning subtleties in Porphyrin self-assembly.

Cracking the self-assembly code, I feel, will act as a remedy to problems in many areas of modern society. However, as shown from this humble research project, there are extremely big scientific hurdles to jump before organised, industrial self-assembly becomes a reality. Perhaps the main way to cross the conceptual hurdles is to find clever solutions to the backwards problem, therefore allowing the sophisticated 'design' of self-assembly processes. Due to the NP-Hard nature of finding these solutions, researchers will need to be intelligent in searching for them. This is a highly multi-disciplinary endeavour that will only be achieved through the application of clever computation such as machine learning. Machine Learning has provided practical solutions to difficult problems in the recent past including reliable cancer diagnoses, fraud protection, wildlife preservation and many others. It is clear that machine learning has earned its place in the scientific arsenal to aid in cutting edge, world changing research. Personally, I feel the most promising capability of machine learning is the ability to learn what the human brain cannot. A human being with no access to the KMCS would not have the capability to inspect the data produced from the 'black box' and create a function approximation from said data in the way that a machine learning algorithm can. The KMCS simulator was developed over many years using inspiration from a vast amount of prior research. On applying a neural network to it, a very basic functional approximation was created with no awareness regarding the internal intricacies of the KMCS. As humans enter into the new technological era, we will be faced with problems that challenge our understanding - problems that will present themselves with more and more variability as we venture into the age of big data. Scientific endeavour will meet similar 'black boxes' many times and I feel machine learning can open those boxes for us and provide a glimpse of what is inside.

I am extremely grateful to have been given the academic freedom to undertake this project and hope that this research has presented a humble investigation into the fascinating capabilities of machine learning in the self-assembly domain.

References

[1] Len Adleman et al. "Combinatorial Optimization Problems in Self-assembly". In: *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*. STOC '02. Montreal, Quebec, Canada: ACM, 2002, pp. 23–32. ISBN: 1-58113-495-9.

DOI: 10.1145/509907.509913. URL: http://doi.acm.org/10.1145/509907.509913.

[2] Leonard Adleman et al. "Running time and program size for self-assembled squares". In: *Proceedings of the thirty-third annual ACM symposium on Theory of computing - STOC 01* (2001). DOI: 10.1145/380752.380881.

[3] Marcus D. Bloice, Christof Stocker, and Andreas Holzinger. "Augmentor: An Image Augmentation Library for Machine Learning". In: *CoRR* abs/1708.04680 (2017). arXiv: 1708.04680. URL: http://arxiv.org/abs/1708.04680.

[4] Rudi Cilibrasi and Paul M. B. Vitányi. "Clustering by compression". In: *CoRR* cs.CV/0312044 (2003). URL: http://arxiv.org/abs/cs.CV/0312044.

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. The MIT Press, 2017.

[6] R. Hadsell, S. Chopra, and Y. Lecun. "Dimensionality Reduction by Learning an Invariant Mapping". In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2 (CVPR06)* (2006). DOI: 10.1109/cvpr.2006.100.

[7] Lawrence Hunter. "Opportunities and obstacles for deep learning in biology and medicine." In: *F1000 - Post-publication peer review of the biomedical literature* (2018). DOI: 10.3410/f.732001658.793542077.

[8] Takuya Inokuchi et al. "Multiscale prediction of functional self-assembled materials using machine learning: high-performance surfactant molecules". In: *Nanoscale* 10.34 (2018), pp. 16013–16021. DOI: 10.1039/c8nr03332c.

[9] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980 (2015).

[10] A. N. Kolmogorov. "Three approaches to the quantitative definition of information". In: *International Journal of Computer Mathematics* 2.1-4 (1968), pp. 157–168. DOI: 10.1080/00207166808803030.

[11] Thomas Miconi, Jeff Clune, and Kenneth O. Stanley. "Differentiable plasticity: training plastic neural networks with backpropagation". In: *CoRR* abs/1804.02464 (2018). arXiv: 1804.02464. URL: http://arxiv.org/abs/1804.02464.

[12] John A. Pelesko. *Self-Assembly: The Science of Things That Put Themselves Together*. Chapman and Hall CRC, 2007.

[13] W. Benjamin Rogers, William M. Shih, and Vinothan N. Manoharan. "Using DNA to program the self-assembly of colloidal nanoparticles and microparticles". In: *Nature Reviews Materials* 1.3 (2016). DOI: 10.1038/natrevmats.2016.8.

[14] P. W. K. Rothemund. "Using lateral capillary forces to compute by self-assembly". In: *Proceedings of the National Academy of Sciences* 97.3 (2000), pp. 984–989. DOI: 10.1073/pnas.97.3.984.

[15] Paul W. K. Rothemund. "Folding DNA to create nanoscale shapes and patterns". In: *Nature* 440.7082 (2006), pp. 297–302. DOI: 10.1038/nature04586.

[16] Paul W. K. Rothemund and Erik Winfree. "The program-size complexity of self-assembled squares (extended abstract)". In: *Proceedings of the thirty-second annual*

ACM symposium on Theory of computing - STOC 00 (2000). DOI: 10.1145/335305.335358.

[17] Shibani Santurkar et al. *How Does Batch Normalization Help Optimization? (No, It Is Not About Internal Covariate Shift)*. May 2018.

[18] Jocelyn Sietsma and Robert J. F. Dow. "Creating Artificial Neural Networks That Generalize". In: *Neural Netw.* 4.1 (1991), pp. 67–79. ISSN: 0893-6080. DOI: 10.1016/0893-6080(91)90033-2. URL: http://dx.doi.org/10.1016/0893-6080(91)90033-2.

[19] David Silver et al. "Mastering the game of Go without human knowledge". In: *Nature* 550.7676 (2017), pp. 354–359. DOI: 10.1038/nature24270.

[20] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[21] G. Terrazas et al. "Automated Tile Design for Self-Assembly Conformations". In: *2005 IEEE Congress on Evolutionary Computation* (2005). DOI: 10.1109/cec.2005.1554907.

[22] Germán Terrazas, Hector Zenil, and Natalio Krasnogor. "Exploring programmable self-assembly in non-DNA based molecular computing". In: *Natural Computing* 12.4 (2013), pp. 499–515. DOI: 10.1007/s11047-013-9397-2.

[23] Gary M. Weiss. "Mining with rarity". In: *ACM SIGKDD Explorations Newsletter* 6.1 (2004), p. 7. DOI: 10.1145/1007730.1007734.

[24] Michael Wilson et al. *Nanotechnology: basic science and emerging technologies*. Univesity of New South Wales Press, 2002.

[25] Yang Zhao, Jianping Li, and Lean Yu. "A deep learning ensemble approach for crude oil price forecasting". In: *Energy Economics* 66 (2017), pp. 9–16. DOI: 10.1016/j.eneco.2017.05.023.