

UNIVERSITY OF SOUTHAMPTON
Faculty of Engineering and Physical Sciences
School of Electronics and Computer Science

Evolution of Molecular Representations

by

Oliver Simmonds

September, 2024

Supervisor: Dr Jo Grundy

Second Examiner: Prof Richard Watson

A dissertation submitted in partial fulfilment of the degree

MSc Artificial Intelligence

Abstract

The molecule representation problem entails encoding a molecule’s information as concisely as possible while preserving its key details in a format that a machine learning model can interpret and use to predict molecular features. Being able to effectively predict features of a molecule can aid in drug discovery making the task of improving molecule representation methods an important one. This paper presents a novel technique to do this, using evolutionary algorithms to evolve networks which produce representations. This work shows that such a method can be effective with the model presented being the best-performing representation method out of the methods tested for some regression tasks. This paper then encourages future work to tune this novel approach so that it can be improved and become a strong performing method for a larger variety of tasks.

Statement of Originality

- I have read and understood the [ECS Academic Integrity](#) information and the University's [Academic Integrity Guidance for Students](#).
- I am aware that failure to act in accordance with the [Regulations Governing Academic Integrity](#) may lead to the imposition of penalties which, for the most serious cases, may include termination of programme.
- I consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to verify whether my work contains plagiarised material, and for quality assurance purposes.

You must change the statements in the boxes if you do not agree with them.

We expect you to acknowledge all sources of information (e.g. ideas, algorithms, data) using citations. You must also put quotation marks around any sections of text that you have copied without paraphrasing. If any figures or tables have been taken or modified from another source, you must explain this in the caption and cite the original source.

I have acknowledged all sources, and identified any content taken from elsewhere.

If you have used any code (e.g. open-source code), reference designs, or similar resources that have been produced by anyone else, you must list them in the box below. In the report, you must explain what was used and how it relates to the work you have done.

I have not used any resources produced by anyone else.

You can consult with module teaching staff/demonstrators, but you should not show anyone else your work (this includes uploading your work to publicly-accessible repositories e.g. Github, unless expressly permitted by the module leader), or help them to do theirs. For individual assignments, we expect you to work on your own. For group assignments, we expect that you work only with your allocated group. You must get permission in writing from the module teaching staff before you seek outside assistance, e.g. a proofreading service, and declare it here.

I did all the work myself, or with my allocated group, and have not helped anyone else.

We expect that you have not fabricated, modified or distorted any data, evidence, references, experimental results, or other material used or presented in the report. You must clearly describe your experiments and how the results were obtained, and include all data, source code and/or designs (either in the report, or submitted as a separate file) so that your results could be reproduced.

The material in the report is genuine, and I have included all my data/code/designs.

We expect that you have not previously submitted any part of this work for another assessment. You must get permission in writing from the module teaching staff before re-using any of your previously submitted work for this assessment.

I have not submitted any part of this work for another assessment.

If your work involved research/studies (including surveys) on human participants, their cells or data, or on animals, you must have been granted ethical approval before the work was carried out, and any experiments must have followed these requirements. You must give details of this in the report, and list the ethical approval reference number(s) in the box below.

My work did not involve human participants, their cells or data, or animals.

Contents

1	Introduction	7
2	Background	9
2.1	Evolutionary Algorithms	9
2.2	Molecular Fingerprints	11
2.3	One-hot-encoding and dimension reduction	12
3	Methods	13
3.1	Hardware and software	13
3.2	Data	13
3.2.1	AqSolDB	14
3.2.2	Tox-21	15
3.3	Mol2Vec	16
3.4	NeuroEvolution of Augmenting Topologies	17
3.5	Bayesian Hyperparameter Optimisation	19
3.6	Prediction Model	22
3.7	Co-Optimisation Model Structure	25
4	Results	27
4.1	Preprocessing	27
4.1.1	Vocab size	27
4.1.2	Output size	29
4.1.3	Generations	31
4.2	Final evaluation	32
4.2.1	Regression	32
4.2.2	Classification	36
5	Conclusion	40
5.1	Final Remarks	40
5.2	Future Work	41

List of Figures

1	An image to illustrate the complexity of life. Through the process of evolution, non-living matter collaborates in the confines of the human body to produce the remarkable human mind. [6]	9
2	Examples of two molecules and their corresponding SMILES strings.	11
3	An image to show the process of one-hot encoding. In this image, we see how benzene is represented as a SMILES string which is in turn represented as a one-hot encoding. [14]	13
4	The distribution of the solubility values from the AqSolDB dataset. Note that the minimum and maximum recorded solubility values were -13.2 and $+2.14$ respectively.	14
5	The distribution of data for each of the 12 features from the Tox21 dataset.	15
6	An image which shows how Mol2Vec first creates the initial vocabulary and corpus of molecules followed by how it trains a Word2Vec inspired model to create the final representations. [15]	18
7	An image which shows the basic structure of the networks evolved by NEAT [12].	19
8	This figure illustrates the BHO process in the Python library <code>bayes_opt</code> [7]. This is an example where only 1 hyperparameter x is considered and its impact on the fitness function $f(x)$. The true impact is shown by the solid blue line. This image is taken from the 9 th iteration of BHO and hence there are 9 data points shown in red. BHO uses all of these data points to build a probabilistic model of how the hyperparameter x influences $f(x)$. BHO will then use the acquisition function to select a new hyperparameter and will continue this process iteratively until some maximum number of iterations.	21
9	This figure shows an example of the confusion matrix. Confusion matrices are used to calculate ROC AUC scores. [11]	24
10	Considering a probability threshold of 0.75, the TPR and FPR values are calculated to plot on the ROC curve. The value 0.75 indicates that if the prediction model deems a point to have a 75% chance of being in class 1, then that's where it classifies it. Otherwise, the point is classified as 0. [11]	25
11	This figure shows an example of a made-up classification task being completed by three different classifiers. The respective ROC AUC scores for each classifier are shown in the legend of this plot. The higher the ROC AUC score the better.	26
12	This image shows the test loss curves for different radii of Morgan identifiers as the NEAT algorithm is run using an XGBoost model on the AqSolDb dataset.	28
13	This image shows the test loss curves when using different vocab reduction methods for radius 1 Morgan identifiers as the NEAT algorithm is run using an XGBoost model on the AqSolDb dataset.	29

14	This image shows the validation loss curves when using different representation lengths using an XGBoost model on the AqSolDb dataset. Output lengths of 25, 50, 100, 200, and 500 were tested.	30
15	This image shows the training and validation loss curves over time when using an XGBoost model on the AqSolDb dataset. Also displayed is the moving average of both of these curves.	31
16	This image shows the training and test loss curves for the final run when using an XGBoost model on the AqSolDb dataset. Also displayed is the moving average of both of these curves.	33
17	This image shows the training and test ROC AUC scores for the final run when using an XGBoost model on the Tox21 dataset. Also displayed is the moving average of both of these curves.	38

List of Tables

1	Relevant Python Modules and Their Versions	13
2	A table showing the quartiles of the properties considered in testing from the AqSolDB dataset.	15
3	A table showing what each feature from the Tox21 dataset represents [26].	16
4	A table showing the hyperparameters and their possible values tuned over when running the NEAT algorithm using an XGBoost model on the AqSolDb dataset. Note that <code>learning_rate</code> , <code>subsample</code> , and <code>colsample_bytree</code> , can take the value of floats whereas <code>max_depth</code> and <code>n_estimators</code> must be integers.	23
5	A table showing the final hyperparameters found when running the NEAT algorithm using an XGBoost model on the AqSolDb dataset.	33
6	Quality of different representations evaluated by applying the XGBoost regression model on 4 properties from the AqSolDb dataset.	34
7	A table showing the final hyperparameters found when running the NEAT algorithm using an XGBoost model on the Tox21 dataset.	36
8	Quality of different representations evaluated by applying the XGBoost classification model on 12 properties from the Tox21 dataset.	39

1 Introduction

This project is concerned with the task of representing molecules in a manner such that machine learning algorithms can efficiently and effectively predict properties of the molecule such as solubility. Solubility prediction is important since it can be used to quickly help identify a new potential drug’s bioavailability [27]. Many methods to represent a molecule numerically, so that they can be understood by machine learning algorithms, already exist. For example, MACCS and Avalon representations follow simple, quick and effective procedures to represent the chemical information of a molecule but they aren’t perfect, no representation method is perfect. Some struggle with generalisation across different prediction tasks, some struggle with compute time, some struggle with achieving good accuracy scores in classification problems, etc. [22]. In addition, many representations overlook the 3D structure of a molecule, leading to a loss of critical information about isomerism and chirality. These properties are often essential in drug discovery.

In this project, a novel approach to tackling this problem is presented by evolving neural networks to represent a given molecule. This approach is complemented by utilising co-optimisation to tune machine learning models to the evolved representations. This is a new approach to dealing with the representation problem since other representation methods, such as MACCS and Avalon, typically produce the representations of the molecules independent of the dataset and the prediction task at hand. This makes these representations versatile but the approach described in this paper lays the framework for a model which has much more freedom to tune representations for a specific task. In this paper, the novel approach of evolving representations has been utilised on a molecular representation task but could be easily extended further afield to protein representation or word processing tasks etc.

Evolving representations has not been done before and this paper lays down a framework showing how to implement such a method. To start this paper Section 2 will motivate the decision to use evolutionary algorithms to conduct this task, explaining their utility and potential. This section also looks at what other types of molecule representations can be used so that the quality of the evolved representations can be assessed by means of comparison.

In Section 3, the datasets used in this paper are described, identifying the distribution of data and how it will be utilised to build molecular representations. Discussed in detail is an additional molecular fingerprint method Mol2Vec since this model gave inspiration to much of the structure of the evolved model put forward in this paper. Next, the evolutionary algorithm NEAT is introduced since it is the algorithm used to evolve the representations in this project using neural networks. NEAT forms one of the key components of the co-optimisation method, with the other key part being Bayesian Hyperparameter Optimisation, hence, in Section 3 Bayesian Hyperparameter Optimisation is described in detail. Subsequently, to train and test the evolution model, the prediction models

used are described. Finally, a high-level overview of how the whole evolution model works is presented at the end of Section 3.

Section 4 discusses the results found in this paper. First, the results seen while building and tuning the evolution model are presented and analysed. Subsequently, the final results seen from the evolved representation models’ performance on both a regression and a classification task are discussed rigorously.

The method put forward in this paper is unique and novel and has shown promising results. Hence, this paper hopes to motivate the utility of using evolutionary algorithms for tasks similar to molecule representations and offer a framework on which could be built upon in the future.

2 Background

2.1 Evolutionary Algorithms

Evolutionary Algorithms (EAs) are a subset of artificial intelligence (AI) inspired by the principles of biological evolution. They seek to develop AI models that replicate the complexity and effectiveness observed in natural life by means of evolution. Over a very short period of time, AI has improved massively and has started to become integrated into everyone's daily life. From medical applications to space travel, AI has solidified itself as being some of the most advanced technology humans have to offer. However, despite all of the progress seen in the field of AI, this technology still pales in complexity when compared to even relatively simple biological life.

From the building blocks of DNA to the systems that govern our bodies, a remarkable amount of independent complex features work collaboratively in the confines of our skin to produce the incredible complexity of the human being. This complexity is illustrated in Figure 1. Humans are the product of the uninformed, random process of evolution by natural selection. This algorithm produced the human brain which is able to outperform the very best AI has to offer many areas of intelligence. This begs the question, if we understand the process that brought about the complexity of biological life and we have the means and resources to reproduce such an algorithm artificially, why have we not yet seen AI surpass humans in every metric of intelligence?

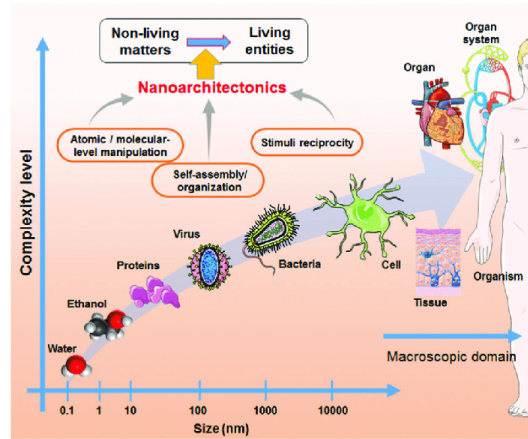


Figure 1: An image to illustrate the complexity of life. Through the process of evolution, non-living matter collaborates in the confines of the human body to produce the remarkable human mind. [6]

The field of EA aims to better understand the process of evolution by trying to evolve AI. EAs have been able to find solutions to tough optimisation prob-

lems where conventual reinforcement methods have fallen short [25] and hence this project aims to present a novel approach, applying EAs to the molecular representation problem.

EAs are a form of optimisation which tries to find a genome (a solution) that best fits some problem. Optimisation techniques fall under different categories. Some, such as random or grid search, simply explore different possibilities and return the best solution stumbled across. Some, such as Bayesian Hyperparameter Optimisation, use probability to inform the direction of search. EAs perhaps fall somewhere in between these two categories since the assessment of genomes has an impact on the direction a population travels in the genome space but does so indirectly maintaining the freedom for members of a population to explore different regions and to start unique species. This freedom helps EAs in tasks where a fitness landscape is rough and it is hard for an optimisation algorithm to inform itself on the best direction of travel.

Starting with an initially random population of some pre-specified size, each genome is assessed on its fitness as a solution to some task which impacts its probability of passing on its genes to the next generation of the algorithm. If a genome is successful at passing on genomes to the generation, these genomes will be mutated according to some pre-specified mutation rule, potentially including a form of crossover of two or more genomes. This process will continue for some pre-specified number of generations or until the fitness of some genome reaches some fitness threshold. A simplistic EA is displayed in Algorithm 1.

Algorithm 1: Simplistic Evolutionary Algorithm [8]

Input : Population size N , Number of generations G , Crossover probability p_c , Mutation probability p_m

Output: Best individual

Initialise population P with N random individuals;

for *generation* $g \leftarrow 1$ **to** G **do**

 Evaluate fitness of each individual in P ;

 Select individuals from P based on fitness to form mating pool;

for *each pair of individuals in the mating pool* **do**

 | With probability p_c , perform crossover to produce offspring;

end

for *each offspring* **do**

 | With probability p_m , perform mutation;

end

 Evaluate fitness of offspring;

 Select next generation P from current individuals and offspring based on fitness;

end

return best individual from P ;

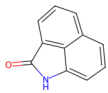
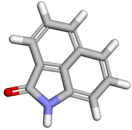
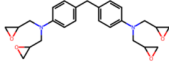
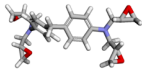
Molecule	Model	SMILES String
Benzo[cd]indol-2(1H)-one 		<chem>O=C1Nc2cccc3cccc1c23</chem>
4-({4-[bis(oxiran-2-ylmethyl)amino]phenyl)methyl}-N,N-bis(oxiran-2-ylmethyl)aniline 		<chem>C1OC1CN(CC2CO2)c3ccc(Cc4ccc(cc4)N(CC5CO5)CC6CO6)cc3</chem>

Figure 2: Examples of two molecules and their corresponding SMILES strings.

As is discussed in Section 3.4, the EA used in the co-optimisation model is more complex than what is presented above with Section 2.1 only intending to motivate the use of EAs and give a brief description of how they work. In summary, EAs use a trial-and-error approach to hone in on some solution to a problem in a similar manner to how evolution honed in on complex life. Humans aren’t the optimal solution to the problem of life, but they are nonetheless a rather remarkable attempt at a solution. Humans, and all other complex life, are the product of a random process which we are yet able to replicate from scratch. Hence, investigating EAs is a fruitful endeavour since it yields both the possibility of producing elegant unseen solutions to complex black box problems and also the possibility of helping us discover something new about the algorithm that built all complex life on earth.

2.2 Molecular Fingerprints

Molecules can be represented by SMILES (Simplified Molecular Input Line Entry System) strings, which offer a simplistic way of encoding the chemical information of a string into a standardised format [21]. For example, the SMILES string of 4-chlorobenzaldehyde is given by Clc1ccc(C=O)cc1. More examples are shown in Figure 2. SMILES strings can be made unique for a given algorithm by considering their canonical form which is further discussed in Section 3.3. SMILES strings form the starting point for molecular fingerprints which like SMILES string, encode the chemical information of a molecule but typically do so numerically in a way that a computer can easily read them.

Molecular fingerprints are commonly used to represent molecules in machine learning tasks. Consequently, several popular fingerprints are employed as benchmarks in the results section of this project. The three most commonly used methods to convert SMILES strings into molecular fingerprints are Morgan, MACCS, and Avalon representations.

Morgan representations are created by identifying ‘the structures of a molecule within a certain radius of organic molecule bonds’ [20]. This means they can

be programmed to consider only the structure of each component of a molecule or they can be extended to consider also the interactions between components of radius 1 or 2 away from each other. Morgan identifiers are a core concept in Mol2Vec representations, another molecular fingerprint, which is discussed in Section 3.3.

MACCS representations are comprised of binary bits which indicate the presence or absence of certain features in a molecule, typically comprised of a 166-bit 2D structure [19]. Avalon representations use similar concepts but consider a wider range of structural features [3].

This project will also consider one-hot encoding representations which, like Mol2Vec will be discussed in more detail later on. Ultimately, the EA presented in this project is designed to produce molecular fingerprints in a novel and unique way which offers certain advantages over the other ones mentioned in this section.

2.3 One-hot-encoding and dimension reduction

The final representations presented in this project are initially built from and subsequently compared against a simple one-hot encoding of the Morgan identifiers. The representations produced by the EA act as a form of dimension reduction since they will take as input the one-hot encodings and produce a representation which is much smaller. Hence a one-hot encoding of the Morgan identifiers that has had the dimension reduction technique of principle component analysis (PCA) applied to it is also considered in the results section for comparison [13].

One-hot encoding representations of a molecular component create a bit string that indicates the presence or absence of each possible component (e.g., all components observed in the training data, plus an additional ‘UNK’ component for unseen elements). Specifically, each component is represented by a string of all 0s with a single 1 in the position corresponding to that component. The one-hot encoding of a molecule is then the sum of the one-hot encoded representations of each of its components. An example of a simple one-hot encoding is shown in Figure 3

PCA is a dimension reduction technique that is applied to the one-hot encoded representations in the results section for comparison. PCA takes as input the data set of molecule representations and calculates for which dimensions the greatest amount of variance occurs. PCA is based on the idea that the dimensions with the greatest variance contain the most information about the data point. The one-hot encodings are then reduced to return the n dimensions with the largest inter-dataset variance. The value n is chosen to be equal to the output vector chosen for the evolution model when testing its performance in comparison [13].

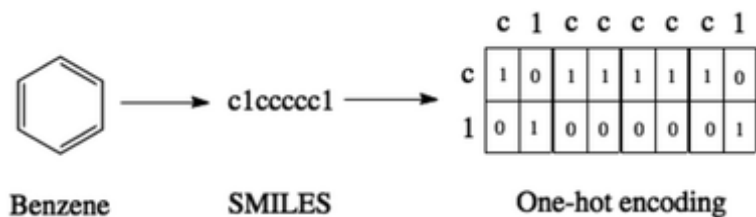


Figure 3: An image to show the process of one-hot encoding. In this image, we see how benzene is represented as a SMILES string which is in turn represented as a one-hot encoding. [14]

3 Methods

3.1 Hardware and software

Due to the nature of EAs, much of the work conducted in this project was computationally expensive. Hence, the majority of the code ran in the building of models and all the final tests were conducted on the University of Southampton’s Iridis 5 HPC. Python version 3.9 was used to run the code where the random seed was set to be 42. The list of Python modules used and their versions can be found in Table 1.

Module	Version
pandas	2.2.2
rdkit	2023.9.5
numpy	1.26.4
mol2vec	0.2.2
scikit-learn	1.5.1
neat-python	0.92
xgboost	2.1.0
bayesian-optimization	1.5.1
joblib	1.4.2
tensorflow	2.17.0
gensim	4.3.3

Table 1: Relevant Python Modules and Their Versions

3.2 Data

This project focuses on two datasets, one to do regression tasks and one to do classification tasks. Both datasets were shuffled using Seed 42 before being used.

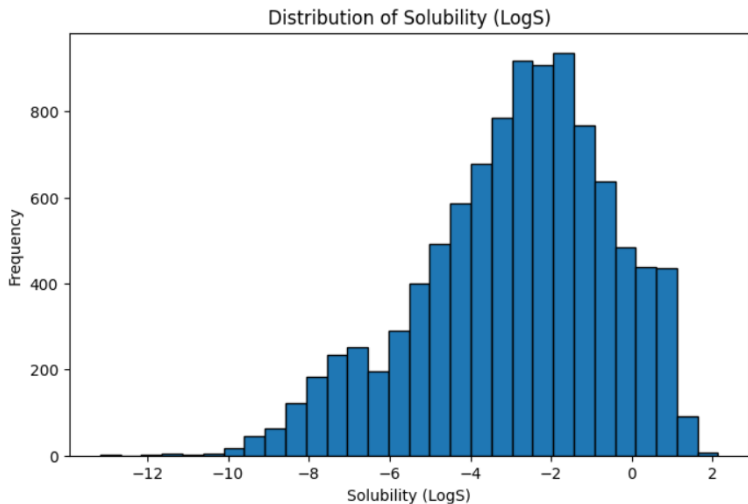


Figure 4: The distribution of the solubility values from the AqSolDB dataset. Note that the minimum and maximum recorded solubility values were -13.2 and $+2.14$ respectively.

3.2.1 AqSolDB

The AqSolDB dataset, otherwise known as the aqueous solubility dataset, was used for a large proportion of this project. This dataset consists of 9,982 unique compounds drawn from 9 different datasets. From this dataset, the main focus was on using the SMILES strings and their solubility value to conduct regression prediction tasks. The solubility value was recorded as $\log(S)$ where S was measured in 10^3 mol/L . This distribution of the solubility values is shown in Figure 4 [23].

It is important to evaluate the ability of the evolved representations to generalise by using them to predict other properties of the molecule. In order to do that, the values of MolWt, MolLogP, and MolMR are also considered in this investigation. MolWt represents the molecular weight. MolLogP represents the octanol-water partition coefficient which is the ‘distribution of an organic compound between octanol and water phases, quantifying the substance’s lipophilic and hydrophilic properties’ [2]. Finally, MolMR, molar refractivity is the ‘measure of the total polarizability of a mole of a substance’ [1].

The quartiles of the properties considered from the AqSolDB dataset are shown in Table 2. As this table shows, these properties have very different distributions and hence will help give a good indication of a representation’s ability to generalise during the testing.

Quantiles	Solubility	MolWt	MolLogP	MolMR
Min	-13.2	9.01	-40.9	0
25%	-4.33	162	0.62	40.6
50%	-2.62	229	1.95	58.6
75%	-1.21	320	3.24	81.9
Max	2.14	5300	68.5	1420

Table 2: A table showing the quartiles of the properties considered in testing from the AqSolDB dataset.

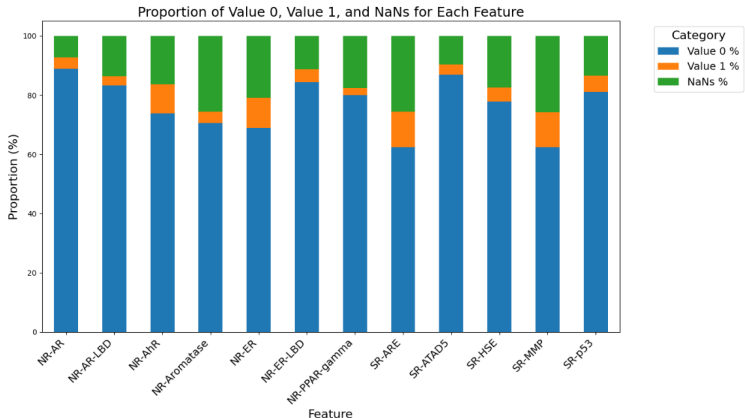


Figure 5: The distribution of data for each of the 12 features from the Tox21 dataset.

3.2.2 Tox-21

The AqSolDB dataset is useful for investigating regression problems. To test the ability of the evolution model on classification tasks, the Tox21 dataset is utilised [26]. The Tox21 dataset contained 7,832 samples which recorded the binary classification of 12 different toxicological experiments. There were some missing values in this dataset which required consideration. The evolution model was trained on the feature NR-AR so all rows which had missing NR-AR values, of which there were 567, were removed. When testing the generalising ability of the model on the other features, rows which had missing data for the feature being tested on were temporarily ignored. Figure 5 shows, for each feature investigated, the distribution of 0s, 1s and NaNs. Figure 5 shows that the majority of values for each feature is 0. This has to be considered when choosing an evaluation metric in Section 3.6. To see what each of these features represents see Table 3.

Feature	Description
NR-AR	Androgen Receptor, Full
NR-AR-LBD	Androgen Receptor, Ligand Binding Domain
NR-AhR	Aryl Hydrocarbon Receptor
NR-Aromatase	Aromatase
NR-ER	Estrogen Receptor, Full
NR-ER-LBD	Estrogen Receptor, Ligand Binding Domain
NR-PPAR-gamma	Peroxisome Proliferator-Activated Receptor Gamma
SR-ARE	Antioxidant Response Element
SR-ATAD5	ATAD5
SR-HSE	Heat Shock Element
SR-MMP	Matrix Metalloproteinases
SR-p53	p53

Table 3: A table showing what each feature from the Tox21 dataset represents [26].

3.3 Mol2Vec

Mol2Vec is a novel approach to representing molecules which was presented in the Journal of Chemical Information and Modelling in 2017 [15]. Mol2Vec uses Morgan identifiers to create a one-hot encoding of each molecule, which is then fed into a Word2Vec-inspired model to generate the molecular representations. This method was shown to be very effective in some classification and regression tasks [15]. The representations are not directly calculated from the SMILES string but were instead trained using the whole dataset to help encapsulate its structure. This approach serves as the foundation for this project, where the evolved representations aim to uncover hidden aspects of the dataset that traditional machine learning methods might miss. Simultaneously, the model seeks to co-optimize the prediction model, ensuring that the representations not only capture the underlying structure of the data but also align with the specific prediction task.

The model presented in the project uses Mol2Vec as a starting point which in turn uses Word2Vec methods to pre-process a dataset. Both the evolutionary model and Mol2Vec use Morgan identifiers to formulate a unique numerical vector representation of each molecule. These can be seen as being equivalent to the sentences in a Word2Vec corpus. Each unique Morgan identifier found in the training data set is hence treated as a unique word which is then one-hot-encoded.

A 80/20 train/test split was employed when building the evolution model. The training set was utilised to build a corpus, consisting of all molecules in the training set represented by their Morgan identifiers, and to create a vocabulary, which includes all Morgan identifiers that appear at least once in the corpus. As mentioned in the Mol2Vec paper, it is important that Canonical SMILES strings are used to produce the Morgan fingerprint since Canonical SMILES

strings follow a procedure to ensure that the order of components of the SMILES strings is always consistent.

As is discussed in Section 4.1 different methods of modifying the corpus and vocabulary were experimented with to enhance the performance of the evolved representations. Firstly, Morgan identifiers can be programmed to consider the interactions of components in a molecule of radius 0, 1, or 2. By increasing to radius 2 you add a lot more information to the Morgan fingerprint which in turn massively increases the size of the vocabulary. More information about a molecule can be useful for machine learning algorithms but an overly large vocabulary can hinder the evolution models’ ability to find effective representations by dramatically increasing the size of the genome space. On the other hand, a radius of 0 may lose valuable information since it only considers the individual components of the molecule and not how they interact with each other. The Mol2Vec model put forward in [15] uses a radius of 1.

Similarly, it is common to remove infrequent and stop words from Word2Vec models since the model doesn’t have the ability to learn representations of infrequent words and stop words harbour very little information about a sentence. Consequently, experiments were conducted in which infrequent and extremely frequent Morgan identifiers (treated as stop words) were removed from the vocabulary. In these experiments, the removed identifiers were replaced with the string ‘UNK’ whenever they appeared in the corpus. Mol2Vec removed words which appeared less than 3 times.

After generating the corpus and vocabulary, each word in the vocabulary is one-hot encoded. The corpus is then used to produce a representation for this encoding that is effective in machine learning prediction tasks. The model creates the molecule’s representation by summing the representations of each identifier, similar to the Word2Vec approach. Mol2Vec here uses a Word2Vec model to generate the representations for each word/identifier whereas the model in this project evolved the representation and hence it here departs from the work of Mol2Vec. Figure 6 is from the original Mol2Vec paper and illustrates how the model works.

In order to compare results, the Mol2Vec model is implemented in full for the results section of this paper. To keep results consistent with the original Mol2Vec paper, a window size of 5 is used to implement the Word2Vec model. The resulting representations were then used on all the tasks given to the evolved representations.

3.4 NeuroEvolution of Augmenting Topologies

As outlined above, the aim of the EA here is map one-hot encodings of Morgan identifiers to a lower-dimensional space, creating informative representations of these identifiers. These lower-dimensional representations can then be combined to form molecule representations. These representations will be then used to train chemical property prediction models. Mol2Vec uses a Word2Vec inspired

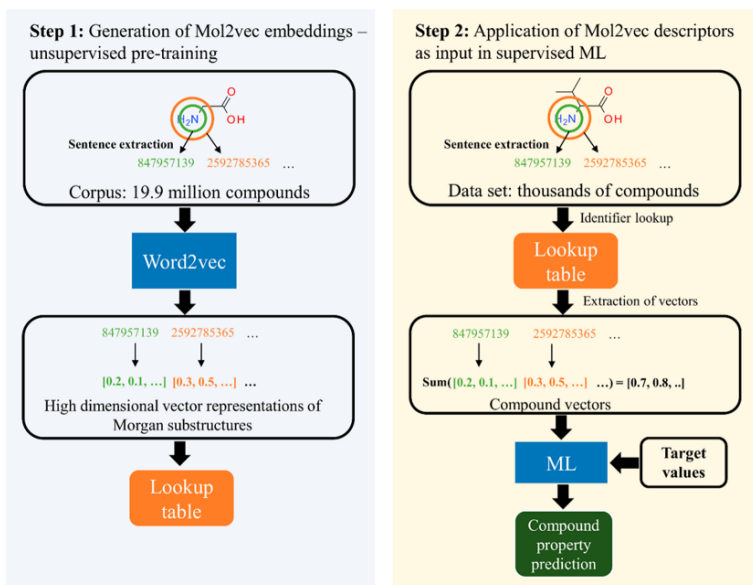


Figure 6: An image which shows how Mol2Vec first creates the initial vocabulary and corpus of molecules followed by how it trains a Word2Vec inspired model to create the final representations. [15]

model to produce the identifier representations. However, in this project, a neural network is used which takes as input the one-hot-encoded identifiers and outputs a lower dimensional space output vector.

Unlike conventional neural networks, the weights of the connections between nodes inside the network won’t be learned using methods such as backpropagation. Instead, a population of neural networks will be produced, they will be assessed on their ability to produce representations of molecules and then evolved using an EA. Such an approach is novel to word processing and could potentially be used to represent words in natural language processing problems.

Therefore, an EA capable of evolving neural networks is required and hence the NeuroEvolution of Augmenting Topologies (NEAT) algorithm is introduced here. NEAT was first presented in the MIT Press in 2002 [24] and is regarded as one of the strongest methods of evolving neural networks. NEAT requires an evaluation function which assesses the capabilities of a network and uses this information to evolve them. The output from the NEAT algorithm is the best neural network in the population after the last generation according to the evaluation function.

The NEAT algorithm is given the freedom to evolve the internal structure of the neural networks including the weights between nodes and also the number of hidden nodes inside the network. Hence, the networks in the NEAT population

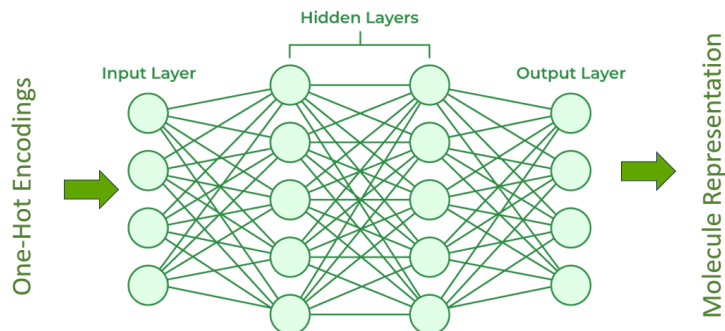


Figure 7: An image which shows the basic structure of the networks evolved by NEAT [12].

only need to have a fixed input length and a fixed output length, the rest of the network is free to be evolved in any other manner. The input length is the length of the one-hot-encoded words or Morgan identifiers returned after the vocabulary and corpus are built. The output length is a hyperparameter. The larger the output parameter, the harder it is for NEAT to train an effective network due to the increased number of connections between the hidden nodes and the output node. However, a larger output vector allows more information about a molecule to be encapsulated. In fact, as shown in Section 4.2, the one-hot encodings performed very well in some tasks without any dimension reduction since no information had to be lost from the initial molecule representation.

The evaluation function defined for the NEAT algorithm takes as input a representation neural network from the population and uses it to produce representations for every molecule in the corpus. These representations are then split into a train and test set and used to train a prediction model. The fitness then reflects how effectively the model performed on some prediction task. For example, for regression tasks, this function would return the negative of the residual square mean error (RSME). It returns the negative since the NEAT algorithm tries to maximise the fitness and hence will return the smallest RSME when configured this way.

Figure 7 shows an illustration of the basic structure of the neural networks evolved by NEAT. As input the network takes the one-hot encoding of a molecule’s identifiers and returns a lower dimensional representation.

3.5 Bayesian Hyperparameter Optimisation

Bayesian Hyperparameter Optimisation (BHO), similar to EAs, is used as an optimisation technique. It is an iterative technique which can be seen as an informed random search [17]. BHO is well-suited for black-box problems where the objective is to improve the output target or fitness by optimizing the selection

of hyperparameters. For example, it can be used to tune the hyperparameters for a regression model given some representations.

BHO first runs for some prespecified number of initial iterations which randomly try different sets of hyperparameters to explore the parameter space. Subsequently, BHO runs some prespecified number of optimisation iterations. In each of these iterations, BHO models the fitness landscape over the parameter space and uses this to assign priorities to sets of hyperparameters. That is, BHO builds a probability model of the fitness function and selects hyperparameters which are likely to yield good results given that model. The BHO used in the co-optimisation section of the evolution model ran 20 initial iterations and 80 optimisation iterations.

The above is a high-level explanation of BHO. Hyperparameter optimisation in general is used to find the set of hyperparameters which maximise a fitness function. That is,

$$x^* = \arg \max_{x \in X} f(x),$$

where x is a combination of hyperparameters belonging to the set of all possible hyperparameter combinations X . Hence, the aim is to find the combination of hyperparameters x^* which maximises the objective function $f(x)$ [18].

BHO optimises $f(x)$ by remembering the performance of past combinations of hyperparameters and building a probabilistic model known as the surrogate. A simplistic form of the surrogate could be represented as,

$$\mathbb{P}(\text{fitness} \mid \text{hyperparameters}).$$

This project used the `bayes_opt` library of the `BayesianOptimization` Python class to conduct the BHO which models the surrogate as a Gaussian Process Model and updates it after every iteration using data augmentation [7].

In each iteration of optimisation, BHO chooses the next set of hyperparameters to test by using an acquisition function. The `bayes_opt` library by default uses the Expected Improvement (EI) acquisition function. The EI function aims to find the combination of hyperparameters, given the surrogate, that will see the biggest improvement, with respect to the fitness function, on the previous best-seen combination of hyperparameters. The EI function is given by

$$\text{EI}(x) = \mathbb{E} [\max (f(x) - f(x^+), 0)],$$

where $f(x)$ is the predicted mean value of the fitness value when using the combination of hyperparameters x and $f(x^+)$ is the best previously seen fitness score [7].

So to summarise this project's implementation of BHO, 20 iterations of random search were run which builds a base probabilistic model of the fitness function given a set of hyperparameters. Then BHO spends 80 iterations updating this probabilistic function to guide the direction of search. The first 20 random iterations are important so that BHO has the possibility to explore lots of different

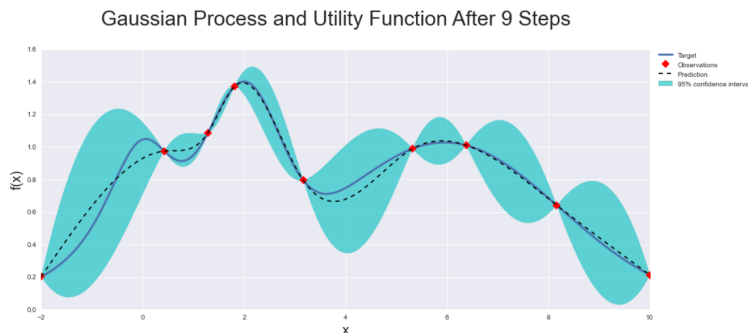


Figure 8: This figure illustrates the BHO process in the Python library `bayes_opt` [7]. This is an example where only 1 hyperparameter x is considered and its impact on the fitness function $f(x)$. The true impact is shown by the solid blue line. This image is taken from the 9th iteration of BHO and hence there are 9 data points shown in red. BHO uses all of these data points to build a probabilistic model of how the hyperparameter x influences $f(x)$. BHO will then use the acquisition function to select a new hyperparameter and will continue this process iteratively until some maximum number of iterations.

regions of the probability space. After the last iteration of BHO, the best-found hyperparameters seen at any point in the BHO process are returned.

BHO is used to tune the prediction models every 30 generations of the EA. This is done to encourage co-optimisation of both the representations and the prediction models. The value 30 was chosen somewhat arbitrarily and could be adjusted in future work. However, it was considered important not to select too small a value, as BHO is highly time-consuming, and applying it too frequently would significantly slow down the model’s runtime. To initiate the BHO, the strongest genome from the NEAT population is selected. Specifically, this refers to the neural network that has been most successful in generating molecule representations, as measured by the accuracy of the prediction model’s outcomes. The best NEAT network is used to produce a representation for every molecule in the training set.

Once the representations have been produced, the 20 initial iterations followed by the 80 optimisation iterations are conducted. Typically, around one-fifth of the total BHO iterations are used in the initial stage, with the remaining iterations applied during the optimisation stage. The total of 100 iterations was chosen because this is generally sufficient for effective BHO implementation while still allowing the evolution model to reach its potential within the allocated time. The performance of a set of hyperparameters is assessed using an evaluation function. In this project’s BHO, the evaluation function takes the molecule representations, splits them further up into a new train and test set using an 80/20 split and then builds a prediction model given the hyperparameters being tested in this iteration. The fitness of the parameters is assessed by

the prediction model’s performance on the new test sets.

After the BHO has concluded the best-found hyperparameters are returned and used in the NEAT algorithm until the next round of BHO is called where they will be updated given the new representations.

3.6 Prediction Model

Most of the testing conducted in this project was done using the AqSolDb dataset focusing on solubility prediction. Hence, this prediction task requires a regression model for which XGBoost was selected. XGBoost was selected since it is an effective regression model which has the scope to tune several hyperparameters. Hence, it was believed it would be a good model to use for the co-optimisation model.

XGBoost stands for extreme gradient boosting. In turn, gradient boosting refers to the ensemble method of boosting. This is where decision trees are added to a model sequentially to fix errors that occur with the current model. This is done by using a gradient descent method on a loss function, hence the name gradient boosting [9]. XGBoost is a simple open-source implementation of gradient boosting which has been shown to be effective at regression tasks.

Explicitly, XGBoost aims to minimise the XGBoost objective function which is given by,

$$\mathcal{L}(\theta) = \sum_{i=1}^n \ell(y_i, \hat{y}_i) + \Omega(\theta).$$

The term $\ell(y_i, \hat{y}_i)$ represents the loss between the prediction \hat{y}_i and its true value y_i . Hence $\sum_{i=1}^n \ell(y_i, \hat{y}_i)$ is the sum of loss across all data points in the dataset. Finally, $\Omega(\theta)$ acts as a regularisation function which punishes model complexity to prevent overfitting and is given by the formula,

$$\Omega(\theta) = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2.$$

Here γ and λ are regularisation parameters. T is the number of leaves in a tree and w_j^2 is the weight of the weight of the j^{th} leaf.

For the XGBoost model used in this project, a loss function of squared error was selected. This was partly because it was desired to evaluate the final model using Root Mean Squared Error (RMSE) since previous work [28] used this to evaluate the performance of regression models on the AqSolDb dataset as well. Using the same metric of evaluation allowed for validation of results when building the prediction models. Furthermore, RMSE gives an effective view of how well a prediction model performs. Taking the square error penalises answers far away from the truth harshly. Taking the root of the square error gives an easily interpretable indication of how the model performs. For example, if a

model has a RMSE of 1 on a log solubility regression task, then on average it is 1 away from the true log solubility values.

XGBoost models also have several hyperparameters which are tuned for using BHO in the final mode. Namely, `learning_rate`, `max_depth`, `subsample`, `colsample_bytree`, and `n_estimators` [4] are tuned for over the runtime of the NEAT algorithm.

In the attempt to prevent overfitting, `learning_rate` is used to determine the step size shrinkage used in the gradient descent method. This means that as time progresses, the XGBoost model becomes more conservative with its exploration and it increases its conservatism at the rate of the `learning_rate`. The maximum depth of the decision trees used in the ensemble is determined by `max_depth`. The larger the `max_depth`, the higher the risk of overfitting. Likewise, small `max_depth` risks underfitting.

The hyperparameter `subsample` is used at the start of every boosting iteration to split the training set randomly, only using some subsection of the training set to inform the building of the decision tree. This again helps reduce the risk of overfitting. Similarly, with the aim of reducing the risk of overfitting, `colsample_bytree` is used to select the proportion of features used to build each decision tree. Finally, `n_estimators` determines the number of trees used in the ensemble.

Table 4 shows what values each hyperparameter was tuned for when conducting the BHO within the NEAT algorithm. The values for `learning_rate`, `subsample`, and `colsample_bytree` can take the value of floats rounded to 2 decimal places however, `max_depth` and `n_estimators` must be integers.

Hyperparameters	Possible Values
<code>learning_rate</code>	0.01 - 1.0
<code>max_depth</code>	2 - 16
<code>subsample</code>	0.01 - 1.0
<code>colsample_bytree</code>	0.01 - 1.0
<code>n_estimators</code>	100 - 300

Table 4: A table showing the hyperparameters and their possible values tuned over when running the NEAT algorithm using an XGBoost model on the Aq-SolDb dataset. Note that `learning_rate`, `subsample`, and `colsample_bytree`, can take the value of floats whereas `max_depth` and `n_estimators` must be integers.

XGBoost can also be utilised for classification tasks and was used for the Tox21 dataset. This was decided partly so that the comparison of the evolved representations’ performance on regression and classification tasks is easier and more standardised. Also, sticking with XGBoost for classification meant that the architecture of the EA model didn’t have to be significantly altered.

	Predicted 0	Predicted 1
Actual 0	TN	FP
Actual 1	FN	TP

Figure 9: This figure shows an example of the confusion matrix. Confusion matrices are used to calculate ROC AUC scores. [11]

The only change to the prediction model required so that it could be used for the classification task was to change the evaluation metric. The XGBoost model for classification in this project utilised the logloss evaluation metric. The performance of the evolved representations was calculated by considering the ROC AUC score and logloss acts as a simplified version of the ROC AUC score. Logloss is given by

$$\text{LogLoss} = -\frac{1}{N} \sum_{i=1}^N [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)],$$

where n is the number of data points being evaluated, y_i is the true classification of data point i and p_i is the probability of the class of data point i being 1 assigned by the prediction model. Hence, logloss rewards the model, not only for making the correct classification but also for being confident with correct choices [10]. This accumulates with logloss being an effective evaluation metric for classification tasks.

The choice of using a ROC AUC score to assess the evolved representations was for two reasons. Firstly, as was the case with RSME, ROC AUC was used to test the performance of representations on classification tasks using the Tox21 data in previous work [28]. Hence, using the ROC AUC score allowed for results to be validated. Furthermore, ROC AUC scores allow for fair and effective evaluation of classification models on data sets with an uneven distribution, like the ones seen in the Tox21 dataset.

To understand how ROC AUC scores are calculated first consider Figure 9 which shows a confusion matrix. Here TN represents true negatives, where the model correctly identifies a data point belonging to the class 0 for some feature. Similarly, TP, true positive, refers to when a model correctly places a data point in the class of 1. FN, false negatives occur when a data point is falsely classified as belonging to class 0 and FP, false positive occurs when a data point is falsely classified as belonging to class 1. Using these 4 values one can produce the TPR,

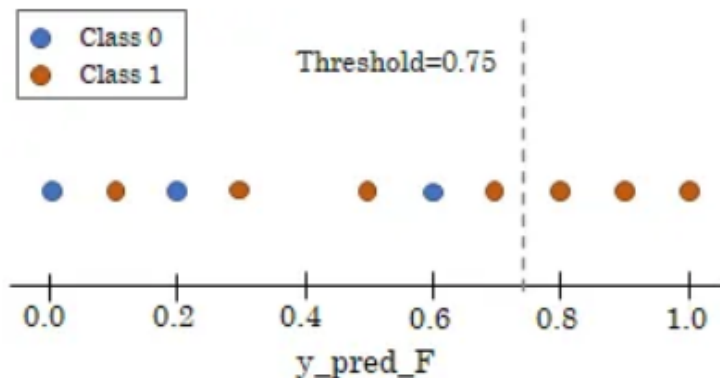


Figure 10: Considering a probability threshold of 0.75, the TPR and FPR values are calculated to plot on the ROC curve. The value 0.75 indicates that if the prediction model deems a point to have a 75% chance of being in class 1, then that’s where it classifies it. Otherwise, the point is classified as 0. [11]

true positive rate and FPR, false positive rate, by using the following formulas,

$$TPR = \frac{TP}{TP + FN},$$

$$FPR = \frac{FP}{FP + TN}.$$

TPR is the percentage of data points which actually belong to class 1 that have been correctly classified and FPR is the percentage of points belonging to class 0 that have been incorrectly classified. When TPR is plotted against FPR we get the Receiver Operator Characteristic (ROC) graph [11]. For each probability threshold used to classify a point as belonging to class 1, the True Positive Rate (TPR) and False Positive Rate (FPR) are calculated. The corresponding point (TPR, FPR) is then plotted on the graph. Figure 10 shows the start of an example of calculating the point (TPR, FPR) using a probability threshold of 0.75. AUC stands for area under curve and hence the ROC AUC score is simply the area under the ROC curve. The higher the ROC AUC score the better with 1 being the maximum possible score and 0 being the minimum. Figure 11 shows a toy example of some different classifiers and their ROC AUC scores.

3.7 Co-Optimisation Model Structure

With the key components of the evolution model established above, the complete model can now be presented in full. The evolution model initialises a NEAT population of size n . Then for x NEAT generations, it attempts to optimise the representations of molecules with respect to the fitness of the prediction model.

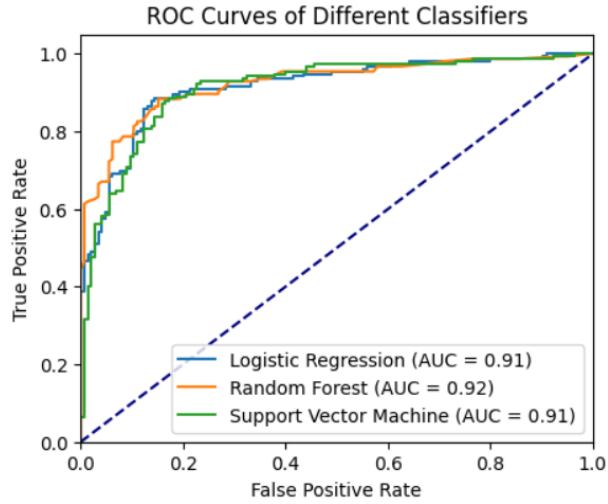


Figure 11: This figure shows an example of a made-up classification task being completed by three different classifiers. The respective ROC AUC scores for each classifier are shown in the legend of this plot. The higher the ROC AUC score the better.

Every 30 generations the model tunes the hyperparameters of the prediction model using BHO. Every 25 generations the model tests the quality of the representation network and the parameters by running the model on the test set. This is done only to evaluate the model's generalised performance while it runs and is not used to influence either the representation network or the hyperparameters. After the x generations, the model downloads the architecture of the representation network and displays the tuned hyperparameter ready for testing. The model is then evaluated on the test set to assess its performance in comparison to other representation techniques.

4 Results

4.1 Preprocessing

4.1.1 Vocab size

As discussed in Section 3.3, the data sets were split into train and test sets using an 80/20 split. The test set was not used at any point to inform the training process but was used every 25 generations of the NEAT algorithm to assess the current model’s performance on unseen data.

Also discussed in Section 3.3 was how in Word2Vec models, infrequent words and stop words can be removed from the corpus. It was of interest to see how applying some similar methods of pre-processing would influence the performance of the evolved representations. Similarly, Mol2Vec has the freedom to choose what radius the Morgan identifiers should consider and hence the performances for different radii were conducted also.

Altering the vocabulary by deciding on what frequency the words should be and how large of a radius to use had a massive impact on the vocabulary size and hence the input size for the networks in the NEAT models. The AqSolDb dataset and the XGBoost regression model were used to assess the performance of these different preprocessing steps.

The performance of using a radius of 0, 1, and 2 for the Morgan identifiers was tested. Subsequently, for radius 1, the impact of keeping the whole vocabulary, then removing all identifiers seen less than 3 times and then removing all identifiers seen less than 3 times and those seen more than 10,000 times was tested. Removing identifiers seen more than 10,000 times removed the 6 most frequent identifiers and was done to simulate the equivalent of removing stop words from a Word2Vec model.

Using radii 0, 1, and 2 for the Morgan identifiers yielded a vocab size, (and hence an input size of the NEAT networks) of 130, 3,814, and 22,470. This illustrates effectively how drastic an impact the radius size has on the network size with an input size of 22,470 being very large for such a model and an input size of 130 losing a lot of valuable information.

Figure 12 shows the test performance recorded on every 25th generation of the NEAT algorithm when the model was run using radii 0, 1, and 2 Morgan identifiers. The EA was run for 5,000 generations unless 48 hours of compute time elapsed while running the model on the Iridis 5 HPC. As discussed in 3.6, to assess the performance of the XGBoost model on the AqSolDb dataset, RSME error is used.

Figure 12 clearly illustrates how varying the radii of Morgan identifiers impacts the computation time of the NEAT algorithm. Using radius 2 only allowed time for a little under 1,000 generations in 48 hours of compute time on the Iridis 5 HPC. Radius 0 completed the 5,000 generations with 8 hours to spare. Using

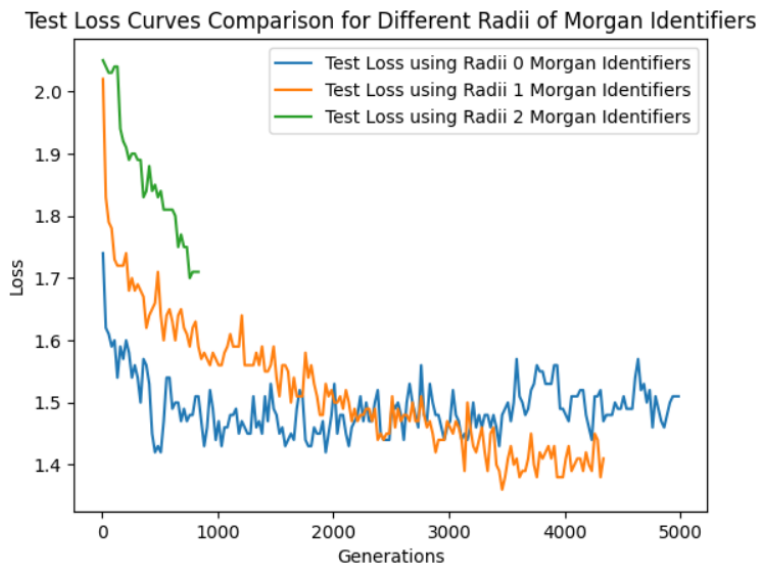


Figure 12: This image shows the test loss curves for different radii of Morgan identifiers as the NEAT algorithm is run using an XGBoost model on the Aq-SolDb dataset.

radius 1 saw a little more than 4,300 generations completed in the allocated time.

Figure 12 shows that all models quickly reduced their loss in the first few generations before plateauing. However, the smaller the radius the smaller the loss was for the initial few generations. This makes sense since the smaller the input to the NEAT networks, the easier it is for the models to learn. Using radius 2 had the opposite problem where it didn’t have time to yield strong results. Radius 1 was shown to be the strongest performer in this initial test since, like radius 0 it had time to get down to some good loss scores, but unlike radius 0, it was able to utilise extra information about the molecule. The Mol2Vec paper used radius 1 identifiers and hence the evolution model presented here also uses radius 1.

Figure 13 shows the same investigation as described above but this time only tested radius 1 Morgan identifiers and instead experimented with either removing just infrequent identifiers, removing both infrequent and overly frequent identifiers, and removing no identifiers from the vocab. This resulted in vocab sizes of 1,697, 1,691, and 3,814 identifiers respectively.

Figure 13 tells a similar story to Figure 12 with all models quickly reducing their loss before plateauing. Note that the graph for the test loss with no vocabulary reduction is the same as the graph for radius 1 in Figure 12. With the use of the vocab reduction techniques, the models completed the 5,000 generations within

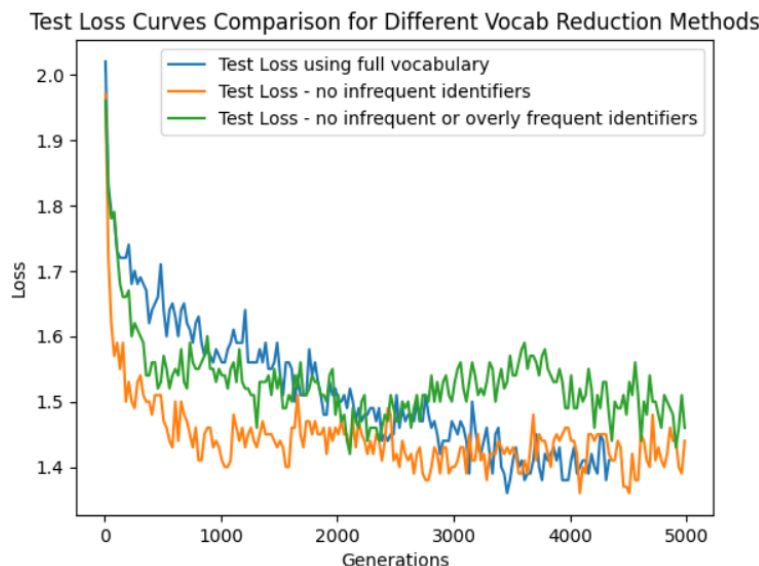


Figure 13: This image shows the test loss curves when using different vocab reduction methods for radius 1 Morgan identifiers as the NEAT algorithm is run using an XGBoost model on the AqSolDb dataset.

the last hour of the allocated compute time.

This plot shows that although removing frequent identifiers improves initial performance in compression to doing nothing at all, it could be hypothesised that this is mainly driven by the improvements caused by removing infrequent identifiers. The model which removed infrequent identifiers outperformed the model that removed infrequent and frequent identifiers for the entirety of the NEAT algorithm’s run cycle. The model that didn’t remove any identifiers did eventually catch in performance to the model that did remove infrequent identifiers but it was slower and never significantly outperformed the model which removed infrequent identifiers. This complements the findings found in the Mol2Vec paper [15] which chose to remove all identifiers which appeared less than 3 times in the training corpus. For the rest of this project, radius 1 Morgan identifiers were used with identifiers which appeared less than 3 times removed.

4.1.2 Output size

Next, the output vector length was tested on a validation set from the training set. That is, the training set is further divided using an 80/20 split to create a new training set and a validation set. The validation set is used in this case because, unlike preprocessing decisions, there were no existing resources to guide the choice of output vector length. As a result, the validation data was employed

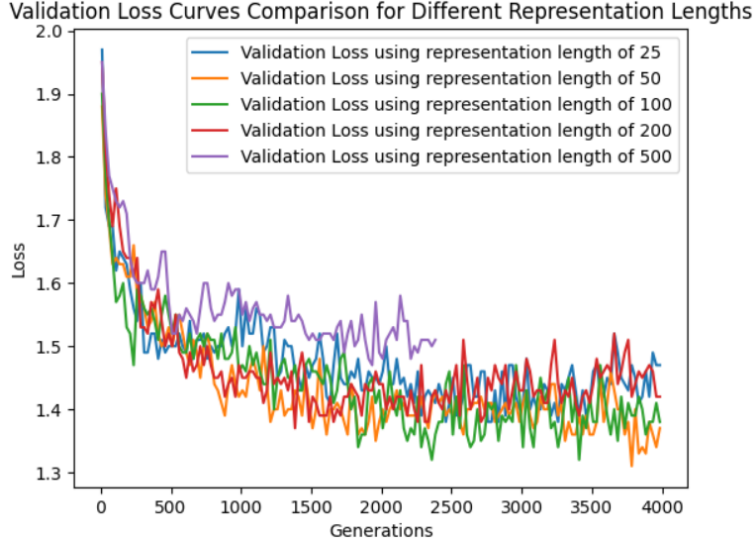


Figure 14: This image shows the validation loss curves when using different representation lengths using an XGBoost model on the AqSolDb dataset. Output lengths of 25, 50, 100, 200, and 500 were tested.

to refine the parameters of the final model. Discussed in Section 3.4 was how the output length can influence the performance of the model. The results from this investigation are shown in Figure 14.

Figure 14 tells a different story to that of Figure 13 and Figure 12 where it seems the hyperparameter doesn't seem to impact performance significantly. All loss curves drop quickly at the start and plateau at the same rate. The training time for an output length of 25 was just under 14 hours, the training times for output lengths 50, 100 and 200 was about 18 hours, and for output length 500, the NEAT algorithm was only able to run for a little under 2,500 generations in the allocated 48 hours. Note that the training times in repeat tests fluctuate significantly so giving a specific training time at this point isn't more informative than a rough guide.

It was decided to move forward with an output length of 100 for future tests. This is because it doesn't seem to matter too much for the evolved representation what length is chosen but this will allow for easier comparison with other representations which use similar output lengths. Also testing for radius size and vocab reductions were done using an output length of 100. It is possible that other combinations of these hyperparameters may yield better results but since training times for each of these models are so long, a sophisticated method of hyperparameter optimisation is unlikely to work effectively.

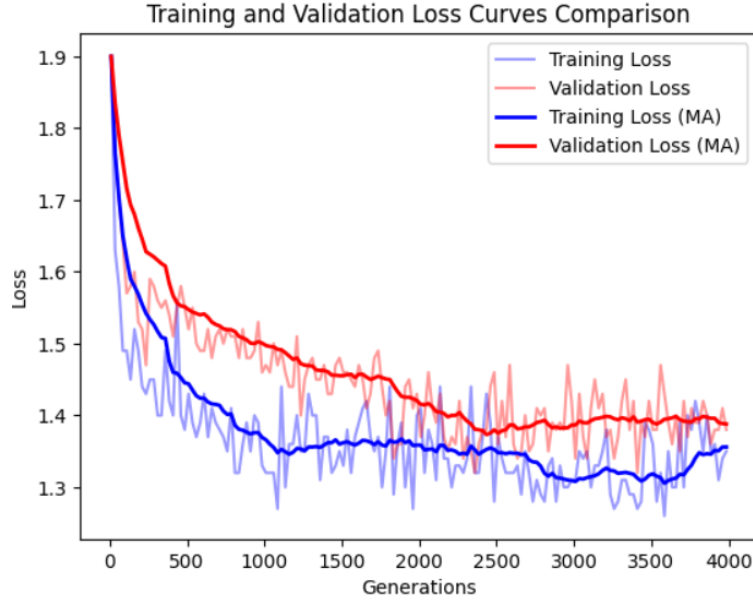


Figure 15: This image shows the training and validation loss curves over time when using an XGBoost model on the AqSolDb dataset. Also displayed is the moving average of both of these curves.

4.1.3 Generations

Another hyperparameter to be considered is how many generations the NEAT algorithm is run for. Figure 14 shows that the graph for an output length of 200 starts to rise after about 2,000 generations suggesting that it started to overfit. A typical approach to determining how many generations to run involves examining the training and validation loss curves. It is typical to see the training and validation loss curves fall at similar rates, then plateau and then see the curves diverge [5]. It is desirable to cut off the training process before this divergence.

Figure 15 shows the training and validation losses over time using the hyperparameters named above for the XGBoost model on the AqSolDb dataset. In this plot, the moving average is displayed as well as the original of both curves for ease of interpretation. This plot shows that there is little improvement after 2,500 generations however the validation curve doesn't appear to significantly increase after this time point either. Hence, to be safe it was chosen to run the NEAT algorithm for 3,000 generations.

4.2 Final evaluation

To evaluate the final performance of the evolved representations, 10 representation networks and the best set of hyperparameters returned at the termination of the NEAT algorithm were taken and their performances on the test set were compared to other representation methods. The 10 representation networks were selected by running an additional 10 generations of the NEAT algorithm and returning the best network from each of these generations. From these 10 networks, the best was selected for comparison with the other tested representations

Similar to how a Word2Vec model is assessed, the vocabulary and word representations found in training were used to form the representations of the molecules found in the test set. This test set was not used at any point to inform any process of the network training. Any components of the molecules in the test set which were not seen 3 or more times in the training corpus are represented with the string ‘UNK’. A molecule representation was tested by building the prediction model on the training set using the given representations and then returning its performance on the test set.

For each chemical property prediction, the evolved network was compared against the following representations; Morgan, MACCS, AVALON, one-hot encoded, PCA applied to one-hot encodings, and Mol2Vec. For the Morgan representation, a radius of 1 and 2048 bits was used. The MACCS representation used an output vector of length 167. Avalon used an output vector length of 512. The output length for both Mol2Vec and the PCA reduction was set to be the same as the output length from the evolution model. The hyperparameters for the representations above were chosen to be the same as what was put forward in the previous work [28] and the paper [24] as a means to validate the results obtained in testing.

4.2.1 Regression

First, consider the results from the XGBoost model trained on the AqSolDb dataset. This model took 19 hours and 37 minutes to train on the Iridis super-cluster. Table 7 shows the final hyperparameters found in the co-optimisation process for the XGBoost regression model. These values are discussed in detail when compared to the hyperparameters chosen by the classification model in Section 4.2.2.

Figure 16 shows the training and test loss curves found when running the NEAT model on the regression task. These curves illustrate what would be expected given the results shown in Section 4.1. That is, they show a sharp drop in loss followed by plateauing. It doesn’t appear that this model overfitted since the test curve didn’t at any point significantly increase. Having said this, the model could have stooped about 500 generations earlier and seen similar results.

Table 6 shows the performance of the NEAT model compared to other pop-

Hyperparameters	Tuned Value
learning_rate	0.15
max_depth	6
subsample	0.46
colsample_bytree	0.62
n_estimators	173

Table 5: A table showing the final hyperparameters found when running the NEAT algorithm using an XGBoost model on the AqSolDb dataset.

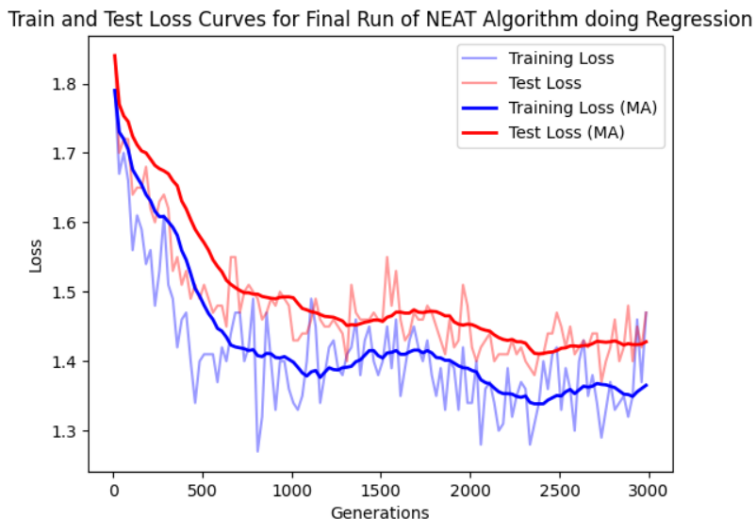


Figure 16: This image shows the training and test loss curves for the final run when using an XGBoost model on the AqSolDb dataset. Also displayed is the moving average of both of these curves.

ular representation methods. Note that the NEAT model was trained trying to predict solubility data. The other networks are uninformed representation methods with respect to the prediction task. All of the values in Table 6 come from building representations from the test set which was at no point used to inform the training of either the Mol2Vec model or the NEAT model. For each property predicted, the best three performing representations have their loss value underlined with the very best value also being written in **bold**.

It is first worth noting that the NEAT model performed poorly on the solubility prediction task beating only the Morgan representation. The NEAT model was built using Morgan identifiers which, since the Morgan representation performed poorly as well, could have been hypothesised as the cause of this poor performance. The Morgan representation was actually the worst representation in 3 of the 4 prediction tasks, narrowly beating the NEAT model in the Mol-

Property	Morgan	MACCS	Avalon	one-hot-encoding	PCA	Mol2Vec	NEAT
Solubility	1.4243	1.2198	<u>1.2671</u>	<u>1.2322</u>	1.3643	1.2932	1.4002
MolWt	143.9801	123.0533	124.1369	<u>77.3676</u>	<u>81.2064</u>	90.8559	61.0406
MolLogP	2.0090	1.9605	1.9966	1.2981	<u>1.8340</u>	<u>1.6576</u>	2.0768
MolMR	35.8574	32.5847	35.0489	<u>20.4544</u>	<u>21.7071</u>	21.8868	17.5025

Table 6: Quality of different representations evaluated by applying the XGBoost regression model on 4 properties from the AqSoDb dataset.

LogP task where it was the second worst representation. However, the Mol2Vec representations, for example, were also built using a base of Morgan identifiers and did not suffer as bad a feat in the solubility task.

The best representation method for the solubility task was the MACCS representation, however, this representation wasn’t in the top three for any of the other prediction tasks. This suggests that the MACCS representation was well-suited for the solubility task. Similarly, the Avalon representation was third best on the solubility prediction task but did not perform well on the other tasks. In fact, all three of the traditional molecule representation methods, Morgan, MACCS, and Avalon, performed poorly on all other properties. This is especially disappointing since these three networks were given licence to have larger output representations which should have improved their performance. They were allowed larger output lengths so that the typically used values for these representations were used in this project but all three of the PCA, Mol2Vec and NEAT representation models were limited to an output length of just 100.

It seems that solubility was a unique property with regard to the results with the results from MolWt, MolLogP and MolMR being in much stronger agreement with what are the better representation methods. As is discussed in Section 5.2, it would be interesting to see the performance of the NEAT model if trained on one of these other properties instead of solubility.

The second-best representation network on the solubility task was the one-hot encoded representation of the Morgan identifiers. The one-hot encoding representations were the second best for two properties and best for two properties but this is to be expected. All other representations had to perform some form of dimension reduction on their input data. This is a key aspect of the representation problem where one tries to summarise a molecule as succinctly as possible, losing as little information as possible. One-hot encoding makes no attempt at this and would hence be impractical for larger molecule machine learning models, such as ones used to aid drug discovery. The one-hot encoding representation was only included in the results table as a means to compare against, with any representation beating the one-hot encoding for a property being very impressive.

It is worth noting that if a chemist has the means to be able to afford to use one-hot encodings then it is the representation that holds the most information but this is a trivial statement. The PCA, Mol2Vec and NEAT representation

models are all effectively dimension reduction methods applied to the one-hot encoding and on the solubility task, they all didn’t perform very well.

However, these dimension reduction techniques did better on the other three regression tasks with the NEAT model actually beating the one-hot encodings on the MolWt and MolMR tasks. This is a remarkable feat since none of the other dimension reduction representations were able to beat the one-hot encodings on any of the other tasks. The fact that the NEAT model beat the one-hot encoding method for the MolWt and MolMR tasks means that the evolved representation was able to encapsulate the information of a molecule in a manner which is more useful to the XGBoost model than having all the information as ones and zeros.

However, despite the very impressive performance of the NEAT model on the MolWt and MolMR tasks, it was the worst model for MolLogP prediction. The contrast in performance of the NEAT model between properties is strange with it being the best for 2 tasks and then one of the worst for the other two. This suggests that for some tasks evolving representations may be the optimal method. This acts as proof that evolving representations has the potential, with modifications to the model, to be a really strong performing form of representing molecules.

It is worth briefly mentioning the trade-off between time to find the representation vs the quality of the representation. The NEAT model took a long time to run in order to find the representations and hence, perhaps one would be happy to use the PCA model instead since this was quick. However, once trained the NEAT model can be applied to new chemicals instantly so this may not be a problem. Having said this, one of the key benefits of the NEAT model over other forms of representations is that the representations are tuned for the specific dataset and prediction task. Hence the NEAT model would perhaps be best if applied to a dataset which is going to be used a lot. The model can be trained once and then the network can be repeatedly used on the dataset from then onwards.

The final key result found from these tests was how impressive the PCA method was. This isn’t a traditional method of representing molecules but did much better than the Morgan, MACCS, and Avalon methods in three of the four tasks. It was better than the Mol2Vec and the NEAT models in two of the four tasks. PCA is quick and easy to implement and hence perhaps should be the go-to method for representing molecules when completing regression tasks. The only drawback of the PCA method is it requires the whole dataset to learn how best to reduce the dimensions and it therefore cannot be applied to a stand-alone molecule.

To summarise, the NEAT model performed incredibly well in two tasks and poorly in two. This acts as proof of concept and should encourage the reader that with some fine-tuning, the NEAT model could be an effective representation method for regression tasks.

4.2.2 Classification

Next, the results from the classification version of the XGBoost model, trained on the Tox21 dataset are considered. The run time for the NEAT algorithm to complete the same amount of generations as the regression model was only 7 hours and 37 minutes. This is almost a third of the time that it took the regression model to run. This is surprising since the AqSoDb dataset was only slightly larger than the Tox21 dataset, 9,982 unique values compared to 7,832. This decrease in runtime could be due to many factors; time to run regression vs classification, time to calculate RSME vs ROC AUC, or the most likely possibility is that increasing the size of the dataset has an exponential impact on the run time of the model which is compounded over the 3000 generations.

Table 7 shows the final hyperparameters found in the co-optimisation process for the XGBoost classification model.

Hyperparameters	Tuned Value
learning_rate	0.08
max_depth	7
subsample	0.36
colsample_bytree	0.89
n_estimators	266

Table 7: A table showing the final hyperparameters found when running the NEAT algorithm using an XGBoost model on the Tox21 dataset.

Comparing the hyperparameters from the regression model to those selected for the classification model yields some interesting insights. Firstly, the **learning_rate** selected for the classification model was almost half what was selected for by the regression model, 0.08 compared to 0.15. This suggests that the classification model required more freedom to explore different tree structures. Potentially since the classes were so imbalanced for the classification task, it was much harder to find trees which were strong performers at classifying this dataset than it was to find trees which could make strong solubility predictions on the AqSoDb dataset, leading to the requirement for XGBoost to explore more tree structures for classification.

Likewise, the classification model selected a larger value for the hyperparameter **n_estimators**, 266 compared to 173. This again suggests that the classification model required more time to explore tree structures. It could also be possible that the classification model was less prone to overfitting and hence could afford to have an increased value for **n_estimators**.

The values **subsample** and **colsample_bytree** are both used to help prevent the risk of overfitting, see Section 3.6, by reducing the amount of data used when building trees. The regression model selected 0.46 and 0.62 for these values respectively, whereas the classification model selected the values 0.36 and 0.89 respectively. It is interesting to note that although both models were given the

freedom to choose any value between 0.01 and 1 for both these hyperparameters, both models selected a smaller value for `subsample` than `colsample_bytree`.

In the XGBoost documentation, it is suggested that `subsample` should be chosen to be larger than 0.5 but both the models here selected values less than 0.5 [4]. This perhaps suggests that both models struggled with overfitting which means that the classification models choice for a larger `n_estimators` was not due to not having to worry about overfitting but more likely due to the need to explore more tree structures to be able to deal with the difficult data distributions given by the Tox21 dataset. The value selected by each of the models for `colsample_bytree` was more in line with what is recommended by the literature around XGBoost [16].

Both models selected a similar size value for the hyperparameter `max_depth` with the regression model using a value of 6 and the classification model, a value of 7. Upon examination of the hyperparameter selection from both models, it seems the contrast of selections is likely due to the contrasting structures of the dataset as opposed to the actual task of regression vs classification. To confirm or refute this suspicion would require more models to be trained on different datasets for both the regression and classification tasks which is discussed in Section 5.2.

Figure 17 shows the training and test ROC AUC score curves found when running the NEAT model on the classification task. First note that a higher ROC AUC score is better which is the opposite of RSME, hence the ROC AUC curves are concave instead of convex. The ROC AUC score curves are much less steep than the RSME loss curves and also the data seems to be more noisy, but this could just be due to the nature of the score. Also shown is that the training ROC AUC score started to drop significantly after 1,500 generations. This highlights the fact that the preprocessing steps to determining the hyperparameters for the NEAT model were conducted on the regression task. Maybe if they were tuned on the classification model, a smaller number of generations would have been chosen.

Table 8 is the classification version of Table 6. It shows the performance of the different representation techniques at classifying the 12 properties found in the Tox21 dataset. Again, for each property predicted, the best three performing representations have their ROC AUC score underlined with the very best score also being written in **bold**.

As was mentioned above, the NEAT model was trained on the NR-AR property. The vocab and corpus for the Mol2Vec and NEAT models were built using the training set. Despite the fact that the NEAT model was built using the NR-AR property, it performed poorly on this classification task. This is similar to how the NEAT model performed poorly on the solubility prediction task in Section 4.2.1. However, unlike what was seen in Section 4.2.1, the NEAT model was not in the top three for any of the classification tasks. In fact, the NEAT model was the worst-performing representation model for all of the properties.

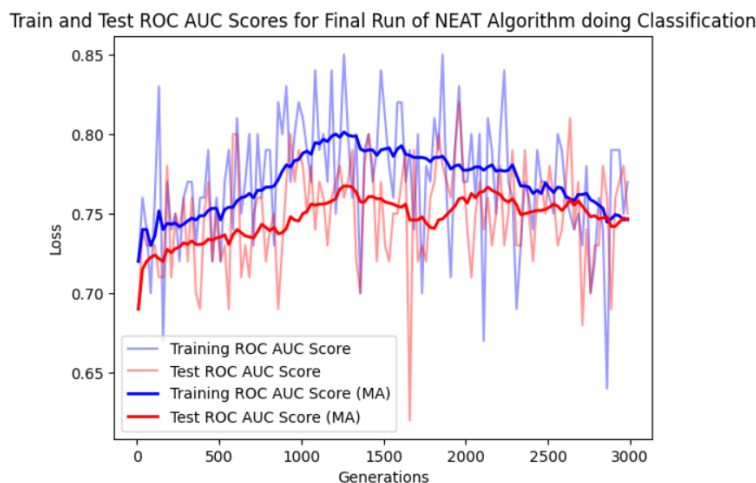


Figure 17: This image shows the training and test ROC AUC scores for the final run when using an XGBoost model on the Tox21 dataset. Also displayed is the moving average of both of these curves.

The Tox21 dataset is a hard dataset for classification problems with the large imbalances in size of the classes and it seems the NEAT model wasn't well suited for this. It is worth noting that the hyperparameters for the NEAT model were chosen using the regression model. However, it wasn't just the NEAT model which saw a difference in performance when doing the classification task instead of the regression task. The PCA model was very effective at the regression task where it was a top three representation method in 75% of the problems, whereas it was only a top three representation method in 25% of the classification problems. The opposite is true of the MACCS and Avalon representations which were poor performers in the regression tasks, being top three representations in only 25% of the tasks. In comparison, they both were a top three representation in 75% of the classification tasks.

As was discussed in Section 4.2.1, the one-hot encoding representation method would be the default guess for what representation should be best at each task since it has such a large representation size and is allowed to keep all of the available information. However, the one-hot encoding method was only the best method in 58% of the classification tasks, even being beaten by the Mol2Vec model in two of the tasks.

In general, the dimension reduction techniques did not do as well at classification as they did at regression. The opposite is true with the traditional methods such as MACCS and Avalon. It's worth noting that the Morgan method was poor at both classification and regression which again reinforces the supposition that it may be worthwhile using a different base to the NEAT model instead of using Morgan identifiers.

Property	Morgan	MACCS	Avalon	one-hot-encoding	PCA	Mol2Vec	NEAT
NR-AR	<u>0.7992</u>	<u>0.7990</u>	<u>0.7918</u>	0.7435	0.7141	0.7701	0.7350
NR-AR-LBD	<u>0.8553</u>	<u>0.8610</u>	<u>0.8616</u>	0.8537	0.8286	0.8404	0.7420
NR-AhR	0.8819	<u>0.8942</u>	<u>0.9014</u>	<u>0.8965</u>	0.8625	0.8810	0.7695
NR-Aromatase	0.7967	0.8427	0.8263	<u>0.8707</u>	<u>0.8554</u>	<u>0.8866</u>	0.7330
NR-ER	0.7230	<u>0.7416</u>	0.7109	<u>0.7624</u>	0.7275	<u>0.7305</u>	0.6415
NR-ER-LBD	0.7956	0.8160	0.7881	<u>0.8705</u>	<u>0.8335</u>	<u>0.8313</u>	0.6948
NR-PPAR-gamma	0.6914	0.7768	0.7740	<u>0.8554</u>	<u>0.8374</u>	<u>0.7960</u>	0.6014
SR-ARE	0.8053	<u>0.8215</u>	<u>0.8248</u>	<u>0.8257</u>	0.8006	0.7992	0.6698
SR-ATAD5	0.8118	<u>0.8564</u>	<u>0.8248</u>	<u>0.8506</u>	0.8099	0.7994	0.6367
SR-HSE	0.7613	<u>0.7944</u>	<u>0.7820</u>	<u>0.8438</u>	0.7697	0.7429	0.6267
SR-MMP	0.8584	<u>0.9061</u>	<u>0.8995</u>	<u>0.9138</u>	0.8963	0.8852	0.7802
SR-p53	0.8002	0.8501	<u>0.8586</u>	<u>0.8915</u>	<u>0.8752</u>	0.8559	0.7032

Table 8: Quality of different representations evaluated by applying the XGBoost classification model on 12 properties from the Tox21 dataset.

Although evidence was found that the NEAT model was effective at some regression tasks, no evidence has been found to say the same about classification.

5 Conclusion

5.1 Final Remarks

In this paper, a novel method for representing molecules was presented. This model used an EA to build a network to represent molecules which has not been done before. This was inspired by the work done with Mol2Vec which took the one-hot encodings of the Morgan identifiers and formed representations from them. There are many ways EAs could be utilised for this problem and the model presented here shows how they can be effective.

The evolved representations outperformed all other representations tested in this project in 50% of the regression tasks. That is, the evolved representations outperformed the traditionally used techniques like MACCS and Avalon. It outperformed the Mol2Vec model which inspired the evolution model. It even outperformed the one-hot encodings meaning the evolved representations were able to not only reduce the size of the one-hot encoding vector but do so in a way that improved its effectiveness in regression prediction tasks.

It was also discovered that for some tasks, using a simple dimension reduction method on the one-hot encoding, such as PCA, yielded rather impressive results in the regression task. As was to be expected, the one-hot encodings themselves were effective representations of the molecules in most tasks. The Morgan representations, which form the foundation of the NEAT model, were shown to be poor in most tasks.

The best representation varied a lot between different tasks and datasets suggesting that there isn’t likely to be one best representation and different tasks may require different methods to represent the molecules. Having said that, if there was to be a method that was good for all tasks, it would likely be a model which uses the dataset and prediction task to inform the representations. This suggests that an improved version of the model presented in this paper could potentially act as a representation method which is good at all tasks.

The model presented here was only trained using one property and on two datasets and hence it is reasonable to believe that there is scope for a dataset to be found in which the evolution model is genuinely the best representation method. Even in the small sample size of datasets and prediction tasks tested here, it was found that the evolution model was the best at 50% of the regression tasks in the AqSoDb dataset.

Although the evolution model did do relatively well on the AqSoDb dataset, as was noted in Section 4.2.1, one needs to consider the trade-off between training time and the effectiveness of a representation model. The model presented here took several hours to train on the Iridis supercluster. Hence, this model would likely need improvements before this could be justified to be used in industry. However, this paper has acted as proof of concept and displayed the possibility that evolved representations could be very effective. The model presented in

this paper had some floors but also displayed some impressive results. There is room to build upon this model, some ways are highlighted in Section 5.2, but this novel method of representing molecules has definitely displayed its potential to be a very strong method for building molecular representations.

5.2 Future Work

The EA presented in this paper displayed some impressive results but as discussed in Section 5.1, there is room for improvement and avenues to explore for future work. Firstly, each model was trained on just one property from each dataset. It would be interesting to see whether if the model is trained on different features it would yield different results. Also, it would be interesting to expand this investigation to more datasets. Only two were investigated here and as was seen in Section 4.2.2, the EA struggled on the Tox21 dataset. In a similar vein, the only prediction model tested in this paper was the XGBoost model. Testing other prediction models could find a model which works better with the evolution model.

As was noted in Section 5.1, the Morgan representation was consistently ineffective and the evolution model in this paper used Morgan identifiers as its foundation. It would be of interest to see if other ways of forming the initial representation input for the NEAT algorithm would yield better results. In previous work, [28], a one-hot encoding was formed by using the SMILES string directly and resulted in some effective representations. Maybe using the one-hot encoding of the SMILES string directly would see an improvement in the model’s performance. This could be taken even further by using the graphs of molecules as input instead of SMILES strings. By using graphs, the representation can account for the molecule’s 3D structure, thereby enhancing the information captured in the resulting representations.

On top of this, different methods of evolving molecules should be explored. This paper has shown that EAs can be effective at representing molecules and it would be interesting to see, if a different EA structure is used, maybe without using NEAT, could better representations be obtained.

Finally, this model has only been applied to molecule representations but could be used in other fields. Similar methods could be used in word processing, protein representations, social network representations, financial market representations, etc. It would be interesting to see this model’s performance when applied to other similar problems.

However, as discussed in Section 5.1, although there is scope for improvement on the EA presented, it nonetheless performed very strongly in a couple of regression tasks, and I hope that the search for these improvements is taken up in future work.

References

- [1] Molar refractivity and interactions in solutions - chemical papers. https://www.chempap.org/file_access.php?file=434a489.pdf.
- [2] Octanol-water partition coefficient - an overview — sciencedirect topics. <https://www.sciencedirect.com/topics/chemistry/octanol-water-partition-coefficient>.
- [3] . Fingerprints in the rdkit, n.d.
- [4] . Xgboost parameters. XGBoost Parameters - xgboost 2.1.1 documentation, n.d.
- [5] S. Afaq and S. Rao. Significance of epochs on training a neural network. Significance Of Epochs On Training A Neural Network, n.d.
- [6] Katsuhiko Ariga. Nanoarchitectonics: A navigator from materials to life. *Mater. Chem. Front.*, 1, 02 2017.
- [7] Bayesian-Optimization. Bayesian-optimization/bayesianoptimization: A python implementation of global optimization with gaussian processes. <https://github.com/bayesian-optimization/BayesianOptimization>, n.d.
- [8] Thomas Bäck and Hans-Paul Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, 1(1):1–23, 1993.
- [9] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. *arXiv.org*, June 10 2016.
- [10] G. Dembla. Intuition behind log-loss score. *Medium*, April 27 2024.
- [11] G. Dembla. Intuition behind roc-auc score. *Medium*, July 20 2024.
- [12] GeeksforGeeks. Artificial neural networks and its applications, June 2 2023.
- [13] Michael Greenacre, Patrick J. F. Groenen, Trevor Hastie, Annibale I. D’Enza, Andria Markos, and Elizaveta Tuzhilina. Principal component analysis. *Nature News*, December 22 2022.
- [14] Ryan-Rhys Griffiths and José Miguel Hernández-Lobato. Constrained bayesian optimization for automatic chemical design using variational autoencoders. *Chemical Science*, November 18 2019.
- [15] Sabrina Jaeger, Simone Fulle, and Samo Turk. Mol2vec: Unsupervised machine learning approach with chemical intuition. *Journal of Chemical Information and Modeling*, 58(1):27–35, 2018. PMID: 29268609.
- [16] Amit Jain. Xgboost parameters tuning: A complete guide with python codes, July 19 2024.
- [17] Timu Theckel Joy, Santu Rana, Sunil Gupta, and Svetha Venkatesh. Hyperparameter tuning for big data using bayesian optimisation. In *2016 23rd*

- International Conference on Pattern Recognition (ICPR)*, pages 2574–2579, 2016.
- [18] W Koehrsen. A conceptual explanation of bayesian hyperparameter optimization for machine learning. *Medium*, July 2018.
 - [19] Hiroshi Kuwahara and Xin Gao. Analysis of the effects of related fingerprints on molecular similarity using an eigenvalue entropy approach. *Journal of Cheminformatics*, March 23 2021.
 - [20] Darko Medin. Data science for drug discovery research - morgan fingerprints in python. *Medium*, February 9 2024.
 - [21] Mid-Continent Ecology Division. Smiles tutorial. EPA, January 23 2007.
 - [22] Mizuki Nakajima and Takashi Nemoto. Machine learning enabling prediction of the bond dissociation enthalpy of hypervalent iodine from smiles. *Nature News*, October 12 2021.
 - [23] M. C. Sorkun. Aqsolddb: A curated aqueous solubility dataset. <https://www.kaggle.com/datasets/sorkun/aqsolddb-a-curated-aqueous-solubility-dataset>, November 21 2019.
 - [24] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.
 - [25] E. T. Evolutionary algorithm outperforms deep-learning machines at video games. *MIT Technology Review*, April 2 2020.
 - [26] U.S. Department of Health and Human Services. National institutes of health. <https://tripod.nih.gov/tox21/challenge/>.
 - [27] Jun Wang and Tingjun Hou. Recent advances on aqueous solubility prediction. *Latest TOC RSS*, January 1 1970.
 - [28] Chia Cherng Xi. *Chemical Properties Prediction Using Machine Learning and Different Molecular Representations*. Meng thesis, University of Southampton, Faculty of Engineering and Physical Sciences, School of Electronics and Computer Science, April 29 2024. A project report submitted for the award of MEng Electrical and Electronic Engineering.