CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Autonomous Drug Design with Reinforcement Learning

Automatization and decision making within a machine learning framework for drug discovery

Master's thesis in Computer science and engineering

Filip Edvinsson
Victor Jonsson

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2023

Master's thesis 2023

Autonomous Drug Design with Reinforcement Learning

Automatization and decision making within a machine learning framework for drug discovery and selection

Filip Edvinsson

Victor Jonsson

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2023

Autonomous Drug Design with Reinforcement Learning
Automatization and decision making within a machine learning framework for drug discovery
Filip Edvinsson
Victor Jonsson

Supervisor: Morteza Haghir Chehreghani, Department of Computer Science and Engineering, Chalmers University of Technology
Advisor: Hampus Gummesson Svensson, AstraZeneca and Department of Computer Science and Engineering, Chalmers University of Technology
Examiner: Alexander Schliep, Department of Data Science and AI, University of Gothenburg

Master's Thesis 2023
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2023

Autonomous Drug Design with Reinforcement Learning
Automatization and decision making within a machine learning framework for drug discovery
Filip Edvinsson
Victor Jonsson
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

The drug design process is currently one of manual trial and error, where potential drug candidates are proposed by chemists, synthesized in laboratories, and then tested and analyzed for properties and efficacy. This process, also called the Design-Make-Test-Analyze (DMTA) cycle, is repeated until a satisfying drug candidate is reached. Statistical models to sample the chemical space and generate potential molecules, combined with automated laboratories and machine learning allows for the automatization of the DMTA-cycle. However, there is still a need for improvement and this is where our project comes in.

One way to improve the automatization of the DMTA-cycle is to reduce the number of cycles needed, and our aim was to achieve this by improving the selection of compounds. To do this, we developed two deep reinforcement learning algorithms, Deep-Q Network (DQN) and Double Deep-Q Network (DDQN), and compared these to two baseline selection algorithms. This approach was chosen as it translates well into the drug development field. Reinforcement learning in drug discovery works by exploring the proposed molecules to find potential candidates and selecting the most promising ones based on molecular similarity to some predetermined properties.

Ultimately, the project was unsuccessful. The baseline selection algorithms using random and greedy selection approaches proved more efficient and accurate than the two algorithms we developed. The involvement of reinforcement learning agents when selecting compounds seemed to cloud the generative model's understanding of what constitutes a good molecule, and thereby reduced the quality of proposed molecules for both the implemented selection algorithms. However, we found that the DQN algorithm shows some signs of promise and can, with some fine-tuning, potentially be brought up to par with the baseline selection algorithms, and perhaps even surpass them.

Keywords: Drug discovery, drug design, design-make-test-analyze cycle, dmta-cycle, machine learning, deep reinforcement learning, deep Q-learning

Acknowledgements

We would like to give a special thanks to AstraZeneca for providing us with the equipment and computational resources needed to complete this project, and for allowing us to use their on-site facilities in Mölndal.

Further, the biggest of thanks goes to Hampus Gummesson Svensson for your unwavering support and encouragement. Without you, none of this would have been possible.

We would also like to extend a big thank you to Morteza Haghir Chehreghani for always making sure we were on track and fulfilling the academic purposes of this project.

Lastly, thank you to Alexander Schliep for all your constructive feedback, and especially for your understanding when we ran into some issues early in the project.

Filip Edvinsson, Victor Jonsson, Gothenburg, January 2023

Contents

List of Figures

List of Tables

1
Introduction

1.1 Background

The drug design process within the pharmaceutical industry is, both historically and presently, one of trial and error, and for which a Design-Make-Test-Analyze (DMTA) cycle can be used as a development approach [1]. In the context of drug discovery, the design-step means proposing one or several molecules, which are then selected in the selection process to be synthesized in the make-step. Next, properties and efficacy are tested in a series of biological assays, after which results are analyzed and improvements made before starting the next iteration. This cycle is repeated until reaching a satisfying drug candidate, called a lead, that can safely be used for its intended purpose. This lead will then go through optimization and, further down the line, clinical trials resulting in an eventual end product [2]. While there have been some improvements to the DMTA-cycle's efficiency over time, the procedure is still both costly and time consuming. Costs are often estimated between $500 million and $2 billion, with an estimated average time from the start of clinical testing to regulatory approval of 7.2 years [3–6].

Two newly emerged paradigms within the pharmaceutical industry show great promise in speeding up the drug design process: AI-augmented molecular design and automation of the DMTA-cycle. AI-augmented molecular design uses generative models to sample the chemical space, containing upwards to 10^{60} drug-like molecules, in order to select possible molecules [7, 8]. A generative model is a statistical model of the joint probability distribution of some given data, and can generate new data instances by utilizing this distribution. Speeding up the DMTA-cycle through automation uses a combination of automated laboratories and machine learning to design, make, test and analyze molecules without the need for human action [9, 10].

This project will primarily focus on the development of a framework for automation of the DMTA-cycle through use of reinforcement learning (RL). RL is an area of machine learning where desirable behaviors are rewarded and undesirable ones are punished. In general, a reinforcement learning agent learns useful actions through trial and error by receiving rewards when interacting with its environment. Applying this concept to drug design, the agent could reward the use of a promising molecule while punishing the selection of a less desirable one. Before going more in-depth with reinforcement learning, let's take a closer look at what happens during the individual stages of the DMTA-cycle, visualized in Figure 1.1, and how they are

affected when mixed with AI-augmented molecular design and automation.

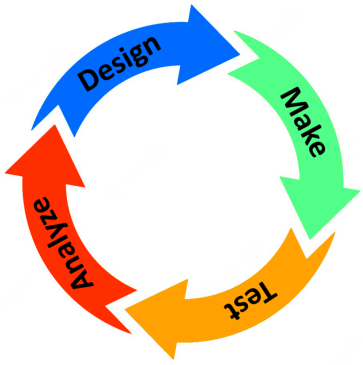

Figure 1.1: A visualization of the Design-Make-Test-Analyze (DMTA) cycle.

AI-augmented molecular design is used to generate molecules in the design-step. The most promising ones are then synthesized (in silico[1]) in the make-step, which will be simulated in a simplified scenario where we assume everything can be synthesized and has equal cost. The aim of this project is to speed up the DMTA-cycle by improving the selection of molecules passed to the make-step. In the test-step, the suggested molecules are evaluated based on what is believed to constitute an efficient compound. This data is then fed to the analyze-step where the newly acquired data is used to update the generative model with the hope of steering it towards a promising area of the chemical space. Hopefully, the autonomous drug design framework will minimize the required number of DMTA-cycles, and thereby enabling a more efficient way of finding drug candidates, ultimately providing new drugs for unmet medical needs faster to the benefit of patients worldwide.

1.2 DMTA-cycle

Before properly introducing the selection process, it is worthwhile to go into more detail in what happens during the different stages of the DMTA-cycle.

1.2.1 Design

Traditionally, the design-step consists of chemists manually designing and selecting compounds that seem to be of interest as a potential candidate for a given purpose [10, 11]. The idea behind an in-silico creation stems from the recent advancements made in computational speed and machine learning, allowing computer models to generate large amounts of potential molecules. These models are called generative models [12, 13]. A generative model is a statistical model of a joint probability distribution of a given data set that can generate new data instances [14].

The chemical space is vast, and although virtual screening libraries are becoming

[1] In silico is a term for something being conducted by computer simulation

enormous, they correspond to a minuscule proportion of chemical space. By generating compounds in a directed manner using de novo design, computational practitioners hope to traverse chemical space more effectively, reaching optimal chemical solutions while considering fewer molecules than allowed by brute-force screening of large chemical libraries [7]. Figure 1.2 is an example, showing the difference in traversal of chemical space between virtual screening of large pre-existing chemical libraries and by an effective de novo molecular program. The color of the plane shows the quality of the molecular property landscape, with yellow being poor and purple being good. A dot signifies a molecule being considered.

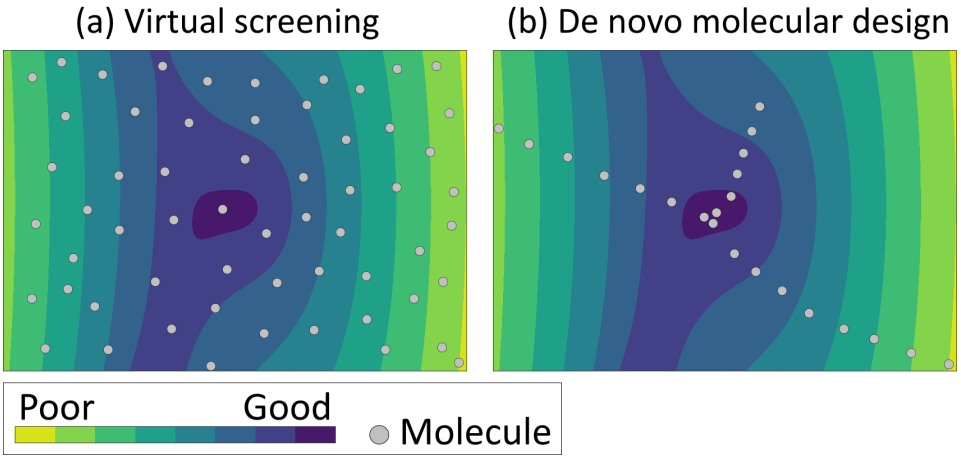

Figure 1.2: A visual example of what the difference in traversal of chemical space between virtual screening of large pre-existing chemical libraries **(a)** and by an effective de novo molecular program **(b)** could look like.

De novo molecular design has recently been called generative chemistry, due to the increased use of generative models in AI. The process of automatically proposing novel chemical structures to optimally satisfy a desired molecular profile is known as de novo molecular design. This is often done in drug discovery to create a molecule that will provoke a desirable biological response while also possessing acceptable pharmacokinetic properties [7]. REINVENT is a well performing example [15] of a reinforcement learning AI tool for de novo drug design which utilizes generative models to create and present a large number of compounds [16].

1.2.2 Make

The make-step is where the molecules suggested in the design-step are synthesized in laboratories, a process requiring expertise, equipment, time and money [4]. Still, an issue lies with the uncertainty of efficacy in the created compounds. There might be a large number of proposed compounds that one could want to test, so in order to reduce time and costs, a few compounds are usually tested in a batch and iterated across a several steps [17]. The knowledge and data obtained from these

iterative small batch tests are then used to improve the process in the following DMTA-cycles.

Simulation can hopefully improve the make-step by reducing the effort wasted on creating ultimately unsuccessful drug candidates. Simulating the drug design process can hopefully result in a better understanding of which molecules should later be synthesized in a real laboratory [10]. In this work, the most promising molecules are simulated in a simplified scenario where we assume all molecules can be synthesized and have an equal cost. It could be useful to consider the molecules sizes in order to better simulate which molecules show enough promise to be considered for real-life testing.

Small molecule drugs are synthetic chemicals that mimic, or are inspired by, natural products. These drugs usually target specific biochemical processes to diagnose, treat, or prevent diseases. In contrast, biologics are molecules that are derived from living organisms. These proteins are often similar to human proteins, making them very effective at treating various conditions [18]. Figure 1.3 is a visual example of the differences between some regular compounds and a polymer, showing the difference between small and large molecules. A polymer is a compound consisting of very large molecules, composed of many repeating sub groups. The large molecule in Figure 1.3 is a biopolymer, which is a natural polymer that is produced from cells of living organisms. To be more specific this biopolymer, is a polypeptide, which consists of chains of amino acids, which are organic compounds that contain amino and carboxylic acid functional groups.

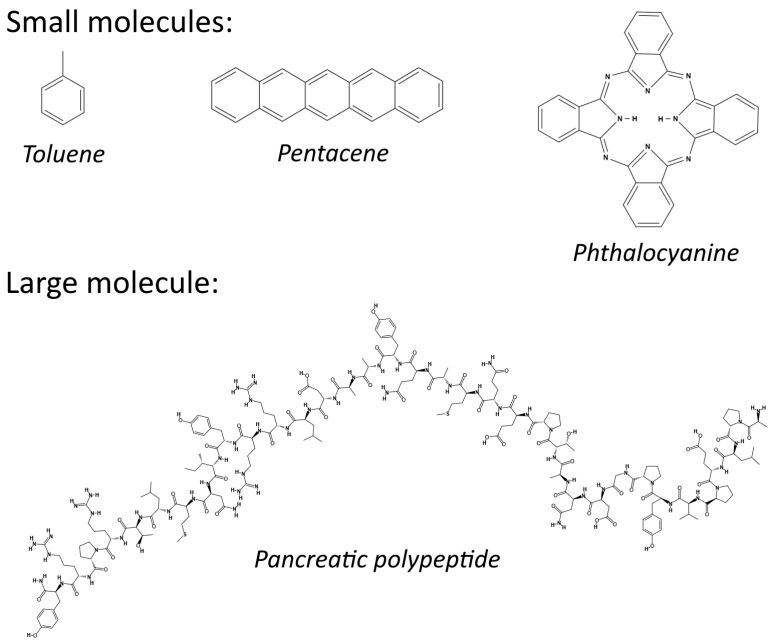

Figure 1.3: Three regular compounds and a amino acid chain to visualize the difference between small and large molecules.

While we have the potential to create, and simulate, large molecules, we will focus on small molecules as they tend to have attributes and abilities generally sought-after in the field of drug discovery.

1.2.3 Test

In the testing step, the compounds created in the make stage get their attributes and abilities tested. The testing of compounds is typically done through the process of laboratory tests and experiments. The cost of this stage is highly dependent on how many compounds are tested, and subsequently on how many DMTA-cycles are needed. In a real world setting, this step is usually costly, taking up approximately 63% of the over US$ 100 billion that the pharmaceutical industry spends yearly on drug development [4].

The key objective of the test-step is to deliver relevant data in a timely manner for all specific project hypotheses [11]. In terms of time investment within the DMTA-cycle, the test-step is generally the one that can be improved the most [3]. The design and analyze steps tend to be fast, while the make-step tends to be slow, often taking weeks to complete the synthesis of the molecules assuming they are sufficiently novel and complex. The test-step can be optimized and streamlined to be fast and predictable, in particular within areas such as vitro data generation, including potency, selectivity and profiling [3].

In the simulated test-step, created molecules are compared to the ground-truth. The ground-truth can be viewed as a set of rules that dictate the quality of any given molecule. It can also be a reference drug used for comparison. This can be as simplistic as a binary or continuous value based on the chemical composition of the molecule, or as complex as an entire mathematical model that tries to predict the physicochemical, biological and environmental properties of a molecule based on its chemical structure [19].

1.2.4 Analyze

The analyze-step is where the results and data gathered in the previous steps is used to optimize the next iteration of the DMTA-cycle. In order for the DMTA-cycle to work most efficiently, data must be rapidly turned into knowledge. The right analysis is essential for learning and making rational decisions that lead back to the design-step and the statement of a new hypothesis. The analyze-step is thus an essential and key part of the DMTA-cycle [11].

The information from the test-step is a valuable insight into which molecules worked well, and which did not. This allows for more informed decisions in the next iteration's design-step. If the DMTA-cycle is performed in-silico, this means feeding the generative model with this information with the hopes that it will output better molecules in the future.

1.3 Selection

The selection process is a sub-step within the DMTA-cycle, taking place in-between the design- and make-steps. During this stage, the selection of which molecules attained from the design-step are to be brought forward to the make-step is made. In the past, this was usually done by identifying molecules similar to those already tested and having shown promise. Due to both time and cost limitations, only a smaller amount of molecules could be selected this way, and there was therefore a large reliance on previous test data [4, 20]. Most of the disadvantages from the past still apply today, albeit they usually surface later in the drug discovery process.

In today's modern world, we have the ability to simulate this stage. This can be done in multiple ways, with varying complexity. In the most basic form, an algorithm that simply simulates random choice selects molecules to bring forward at random. In a more advanced version, an algorithm that ranks data from the test-step in the previous DMTA-cycle and selects molecules based on that data could be used. An even more advanced selection algorithm could utilize machine learning to have the machine learning agent learn which molecules to select based on results from previous choices and test results. This is the part we have decided to be the major focal point of our project. We plan to implement two new selectors based on reinforcement learning (RL).

1.4 Aim

The goal of the project is to improve the automation of the DMTA-cycle. In more detail, the goal of the project is to improve the selection of compounds proposed by the design-step, which are later fed to the make-step, through the use of reinforcement learning. Even more specifically, we aim to implement selectors based on Deep Q-Network (DQN) and Double Deep Q-Network (DDQN), both utilizing Deep Q-learning to improve on the basic form of RL[2]. In particular, we will investigate the following questions:

- Is the DQN and/or the DDQN selection method an improvement over the baseline methods?
- How does the implemented methods fare in terms of speed when it comes to producing a sufficient result?
- Does the amount of selected compounds have an effect on which selection method performs better?
- Will a stricter classification of what a novel compound entails affect which selection method performs better?

[2]These concepts will be introduced and explained in detail in Section 2.1

1.5 Ethical Considerations

The main goal of this project is to improve upon the automatization and decision making process of a machine learning framework used for drug discovery at AstraZeneca. The framework is still in an early enough stage that it is unlikely that our contributions will affect drugs that reach consumers in the near future. The framework is also not intended to replace the entire development process, but rather speed it up. Before any drugs reach the market, they would have to undergo rigorous testing, analysis and clinical trials, all greatly reducing the risk of us suggesting drugs with adverse side effects.

Another ethical consideration worth noting is that this creation might not only be used for its original stated intention. Here, the aim is to improve and speed up the drug development process by automatization. A fully automated system for drug discovery could theoretically be used without any chemical knowledge. If these improvements were to fall into the hands of a malicious user, they could instead be used to facilitate the development of hazardous chemicals that could potentially be used as chemical weapons. This is a case of dual-use dilemma and is not something we wish to see occur, and not something we deem likely to occur either. We intend to make sure our contributions made by this project follow the *Ethics Guidelines for Trustworthy AI*, published by the European Commission in 2019, in order to lessen the chance of this happening [21].

Trustworthy AI have three components, lawful, ethical and robust. The lawful part controls whether the AI is complying with all applicable laws and regulations, and the ethical part ensures adherence to ethical principles and values. Finally the robust part, which has both technical and social perspectives, tries to ensure that the previous two parts are complied with since AI, even with good intentions, can cause unintentional harm [21]. A sincere attempt to follow these guidelines should form enough of a barrier to hopefully prevent our contributions to the project ending up being misused.

Then there are also the usual considerations when working with ML models, where bias can occur if the data set is not collected correctly and with care. With our previous experiences, we have some experience dealing with this issue. This, combined with the fact that we are aiming to follow the *Ethical Guidelines for Trustworthy AI*, should hopefully be enough to minimize the risk of this being an issue.

2
Theory

2.1 Reinforcement Learning

The goal of the autonomous drug design framework is to minimize the required number of DMTA-cycles needed to find drug candidates. Several generative models for de novo molecular design exist, capable of proposing large numbers of molecules based on some initial critera, one being REINVENT, as introduced in Section 2.5 [16, 22]. Given the large number of small molecules these models are capable of producing, it is highly important to effectively select the most promising ones and machine learning has shown great promise in this area [23]. As utilizing machine learning for automatization and decision making is a fairly novel concept in the pharmaceutical field, there is a lot of room for exploration in regards to the choice of models [24].

We are looking to investigate the use of reinforcement learning (RL) for drug design. RL is a machine learning technique comprised of an agent and an environment, as seen in Figure 2.1. The agent is the learner and decision maker, and the environment is everything outside the agent itself, with different possible configurations of the environment referred to as states. As seen in Figure 2.1, the agent and environment interact at discrete time-steps, $t = 0, 1, 2, ...$[1]. With every time-step, the environment presents the agent with a state, S_t, and the agent selects an action, A_t, based on its current knowledge of the given state. The mapping of states to the probability of selecting each possible action in the given state is called a policy, often denoted by π.

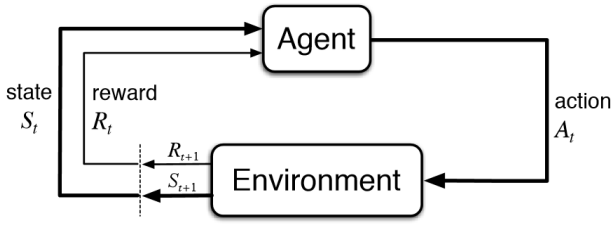

Figure 2.1: Overview of the reinforcement learning cycle[26].

At the next time-step, $t+1$, the agent receives a reward, R_{t+1}, from some unknown distribution based on the action, A_t, selected in the previous state, S_t [25]. The

[1]RL also supports continuous-time cases, but that is omitted as it is not relevant for this application [25]

reward is a numeric value that signifies whether the action taken was positive or negative. The sum of all rewards is known as the return, $G_t = R_{t+1}+R_{t+2}+...+R_{t+T}$, with T being the final time-step. It is important to note that a reward may not be a direct result of the agent's latest action as it may have been a consequence of a series of actions taken previously [27]. Further, all rewards are not necessarily valued equally. A discount factor, γ, can be used to determine how much the reinforcement learning agent values immediate versus future rewards. The ultimate goal of the agent is to find the optimal policy, i.e., maximizing the cumulative reward [28]. Sutton and Barto describe RL as "learning what to do - how to map situations to actions - so as to maximize a numerical reward signal" [25]. They are careful to note how actions may not only affect immediate rewards, but also the situation itself, and thereby changing subsequent rewards [25].

2.1.1 Exploration & Exploitation

As the reinforcement learning algorithm starts off with no knowledge of its environment, exploration is a very important aspect of every RL implementation. Exploration is achieved by selecting random actions to gain information. All states have what is called a Q-value, $Q(s, a)$ (also known as action value or state-action value), which is a measure of the expected reward when the agent is in state s and selects action a. $Q(s, a)^\pi$ is the expected return starting in state s and selecting actions a according to policy π, seen in Equation 2.1.

$$Q^\pi(s, a) = E_\pi\{G_t|s_t = s, a_t = a\} \tag{2.1}$$

Once the algorithm has a better understanding of which actions are beneficial in certain states, it can start using this information to make educated choices on which action to select at a given time. This is called exploitation. The balancing of exploration and exploitation is vital both in terms of model accuracy and efficiency. If the model does not explore enough it might not find the most suitable action, which in drug discovery would be the most promising molecules. If the model explores too much, the model might be too slow and therefore unusable under time-constraints. A common approach to this balancing act is called ε-greedy, showcased in Equation 2.2. An ε-greedy exploration algorithm takes a uniformly random action with probability ε and selects the best action given the current information with probability $1 - \varepsilon$ [25, 28], that is

$$\text{action at time(t)} = \begin{cases} \max Q_t(s, a) & \text{with probability } 1 - \varepsilon \\ \text{any action}^2\text{(a)} & \text{with probability } \varepsilon. \end{cases} \tag{2.2}$$

Exploration becomes less important as the model learns more about its environment, and it is therefore common to use a decaying ε to shift emphasis towards exploitation. Otherwise, the actions taken will be too random and the behavioural policy risks not gaining enough experience with states and actions near the optimal

[2]Picked uniformly at random from the set of possible actions A_t

policy. Furthermore, ε has to be decaying in order for the policy to converge to the optimal policy [28, 29].

Low values of ε allow for deeper exploration, namely exploration that not only considers the immediate information gain but also the consequences of an action for future learning [30]. A high ε, on the other hand, helps prevent overspecialization [31].

2.1.2 Q-learning

Q-learning is a model free, off-policy reinforcement learning algorithm with the purpose of learning the value of an action in a given state [25]. Model free means that the algorithm learns by taking actions that are outside the current policy, such as taking a random action. Off-policy means the Q-values are updated using the next state S' and the greedy action a', essentially acting as if the algorithm would follow a greedy policy [25]. Q-learning is an abbreviation of Quality-learning, where Quality signifies the quality of a specific action given a specified state. In practice, it works by initializing a table with n rows and m columns as seen in Table 2.1, representing each possible state and action, respectively.

$$\begin{array}{c} \\ s_0 \\ s_1 \\ ... \\ s_n \end{array} \begin{array}{c} \begin{array}{cccc} a_0 & a_1 & ... & a_m \end{array} \\ \begin{pmatrix} 0.8 & 0.2 & ... & 0.2 \\ 0.3 & 0.4 & ... & 0.9 \\ ... & ... & ... & 0.0 \\ 0.1 & 0.8 & 0.4 & 0.6 \end{pmatrix} \end{array}$$

Table 2.1: Example of a Q-table with n states and m actions

Every state-action pair is given a certain value signifying how beneficial it is to take an action in a specific state. The Q-table is updated, as per Equation 2.3, as the algorithm learns more about the environment by either exploring or exploiting.

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max Q(s_{t+1}, a)}_{\text{estimate of optimal value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right) \tag{2.3}$$

In order to determine how much the model values the newly gained information compared to the old, it makes use of something called learning rate, α. This is a parameter that controls how much the model should change when presented with the estimated error $(r_t + \gamma \cdot \max Q(s_{t+1}, a) - Q(s_t, a_t))$ [25]. The algorithm also makes use of the discount factor, γ, to balance immediate and future rewards. A high discount factor means future rewards are highly valued, while a low discount factor values immediate rewards higher, as shown in the following equation: $\gamma \cdot \max Q(s_{t+1}, a) - Q(s_t, a_t)$. Both these concepts are used in Equation 2.3 when updating the Q-table. Additionally, $\gamma < 1$ ensures the rewards are finite [25, 27].

2.1.3 Deep Q-learning

One issue with tabular Q-learning is that the Q-table scales with both the state and action spaces, meaning that the table can become unfeasibly large for complex systems. One solution, first successfully realized by DeepMind, is to introduce a neural network to the reinforcement learning algorithm [32]. Instead of a Q-table, a neural network is used to approximate the Q-value function. The neural network takes a state as input and outputs the Q-value of all possible actions. This is known as deep reinforcement learning (DRL) [33–35].

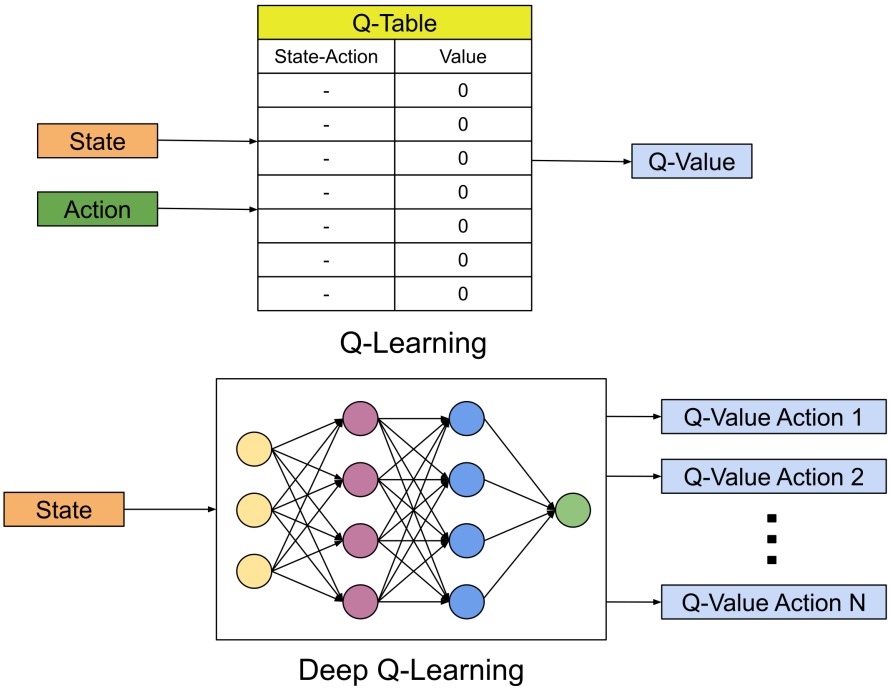

Figure 2.2: Illustrates the difference between Q-learning and deep Q-learning [36].

To determine the error between the observed value and the expected value, a loss function is used. The mean squared error (MSE) is often used as loss function for deep Q-learning. It is defined as $\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$, where N is the number of data points, y is the value of the observed variable, and $\hat{y}$ is the expected value.
One drawback with deep Q-learning is how the loss function can only be approximated by randomly sampling the distribution. This would imply that samples should be independent and identically distributed. However, when gathering state transition online, meaning that the algorithm uses data as soon as it is available, samples are correlated. E.g., $(s_t, a_t, r_{t+1}, s_{t+1})$ is directly followed by $(s_{t+1}, a_{t+1}, r_{t+2}, s_{t+2})$, and the reward r_{t+2} is a direct result of action a_t in state s_t, as explained in Section 2.1. This correlation will cause the deep neural network to overfit as it will learn from the very specific cases it encounters, and it therefore risks converging to a local minimum. If the model was instead able to learn from disconnected samples, the knowledge acquisition would be averaged out over many previous states. This would make it more likely to find solutions to problems it has not necessarily encountered

replicas of before, allowing it to possibly converge to a global maximum more often [32].

The problem with correlated samples stems from when the neural network is fed transitions as they are gathered. A solution to this is utilizing a replay buffer where transitions are stored, rather than continuously fed to the network [32]. The neural network can then be trained by randomly sampling from the transitions stored in this buffer using a function approximator, such as stochastic gradient descent (SGD)[3]. The larger the replay buffer, the less likely sample elements will be correlated, but this also increases memory usage and may slow training [32, 37].

In some environments, DQN performs quite poorly due to its tendency to overestimate action values [38]. This occurs as it selects the maximum action value as an approximation of the expected action value [39]. One approach to counter this is Double DQN (DDQN), where two function approximators are trained on different samples and are then asked to evaluate the same action. Since they have seen different data, it is unlikely they will overestimate the same action [38].

Deep reinforcement learning has achieved many impressive results in the last decade, with one of the most prominent being the state-of-the-art Atari scores achieved by DeepMind in 2015 [40]. These results, however, required tens of thousands of episodes of experience per game [40, 41]. In real-life applications, there is an importance of producing results and reaching conclusions quickly as resources are almost always limited. As the ultimate purpose of this framework is to aid in speeding up autonomous drug development, we will have to optimize our approach to reach the best score within some finite time and budget constraints. These constraints could be simulated by only allowing our models to train for a minimal set of episodes and epochs. An epoch refers to one cycle through the full data available for training.

2.1.4 Optimizers

Many problems in the fields of science and engineering can be viewed as the optimization of some parameterized objective function, requiring maximization or minimization of its parameters. A function with differentiable parameters is often optimized using gradient descent as "the computation of first-order partial derivatives w.r.t. all the parameters is of the same computational complexity as just evaluating function" [42]. Therefore, the stochastic gradient-based optimization is of high importance in many areas [42]. Many objective functions are comprised of subfunctions evaluated with different data, and in these cases optimization can performed by stochastic gradient descent (SGD). Instead of computing the exact gradient, each iteration estimates the gradient based on a single randomly chosen example [42, 43].

Another method for stochastic optimization is Adam, an adaptive optimization algorithm requiring only first-order gradients with low memory usage [42, 44]. It was

[3]Introduced in Section 2.1.4

largely developed to handle the issue of difficulties tuning the learning rate hyperparameter of SGD, stemming from widely varying magnitues of parameters and adjustment requirements during training [44]. Adam proposes adapting learning rates of different parameters automatically, based on estimates of first and second gradients. Although often converging faster and simplifying the tuning of hyperparameters, Adam tends to generalize significantly worse than SGD in certain scenarios [42, 44].

2.2 Molecular Encoding

2.2.1 SMILES

In order to formalize chemistry, an unambiguous and reproducible notation is needed for naming chemical compounds. Such systems were swiftly established only years after the birth of structural chemistry in 1861 [45, 46]. Molecular configurations were identified in text by linearly specifying symbols for molecular segments as connected [46]. However, with computers and chemical knowledge now at a point where enormous amounts of chemical information can be stored and used, a more efficient system is needed for providing relevant information. One of these systems is the Simplified Molecular-Input Line-Entry System (SMILES) [45, 47]. The SMILES format is a specification that represents the 2D chemical structures of molecules in the form of text strings specifically designed for computer use [45, 48]. Figure 2.3 showcases an example of how chemical structure relates to SMILES strings.

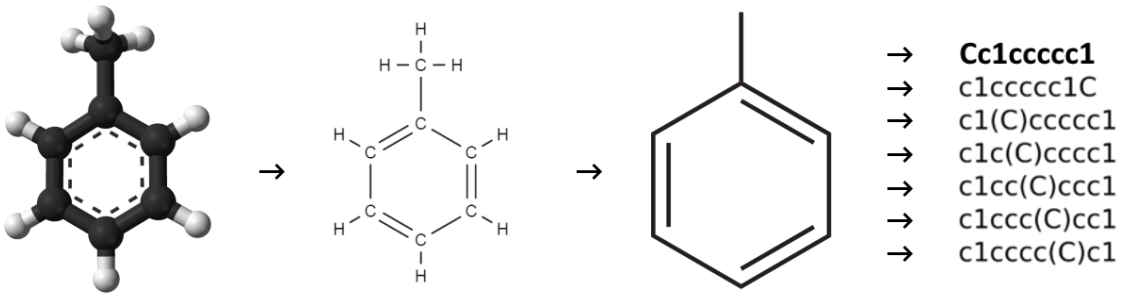

Figure 2.3: We illustrate different ways to represent the chemical structure of toluene (C_7H_8), starting with a ball-and-stick model. This model is simplified to a detailed skeleton structural formula, and then simplified further to a simplified structural formula. Lastly, it is transformed into seven SMILES enumerations, where all strings represent the same molecule.

This example illuminates a potential issue with the SMILES notation: a single molecule can be represented with multiple different SMILES strings. Toluene (Figure 2.3), with seven carbon atoms, has seven possible SMILES enumerations. Efforts made to consolidate how SMILES are generated resulted in the definition of a canonical SMILES, guaranteeing that a molecule corresponds to a single SMILES string [48, 49]. In Figure 2.3, the top SMILES string (Cc1ccccc1) is the canonical one. From this point onwards, SMILES and canonical SMILES will be used interchangeably unless otherwise stated.

2.2.2 Molecular Fingerprints

One of the most common molecular abstractions is called molecular fingerprints, where a molecule is converted into a bit vector. This allows for easy comparisons [50]. Fingerprints have been used for a long time in drug discovery, as the representation of molecules in the form of mathematical objects allows for their application in both statistical analysis and machine learning. Their ease of use and the speed at which searches can be performed using them adds to their popularity [51].
There are several types of molecular fingerprints, each depending on how a molecule was converted into a bit string. Most methods use the 2D molecular graph (As seen in Figure 2.4), and are therefore called 2D fingerprints. These are the types of molecular fingerprints covered in this section.

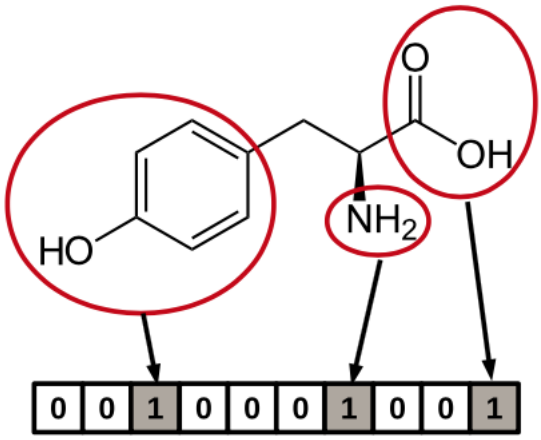

Figure 2.4: Representation of a 10-bit substructure fingerprint. The three bits set represent the substructures present in the molecule (circled) [51].

Computational advances have allowed for virtual screening, an in-silico method of searching large databases for bioactive molecules that largely reduces the number of compounds a researcher has to test experimentally [52]. A common approach for screening databases is the ligand-based method, retrieving compounds similar in some way to a single known bioactive molecule used as a starting point [53]. Examining the similarity between two known, complex compounds can be quite challenging. To facilitate this process, some simplification or abstraction is often necessary [51].

2.2.3 Fingerprint Types

Substructure keys-based fingerprints set bits in the bit vector depending on the presence or absence of substructures or specified structural keys in a molecule. This approach can be seen in Figure 2.4, where the bit vector has three set bits as the substructures or structural keys they represent are present in the molecule. This approach is highly useful when molecules are likely to be covered by some known keys [51].

Topological fingerprints instead work by examining all different fragments by following a path making up the molecule up to a specified number of bonds. The possible paths are then hashed to create the fingerprint, which allows for any molecule to be meaningfully simplified into a fingerprint [51, 54].

Circular fingerprints are a type of topological hashed fingerprints, but instead of examining molecular paths, each atom's environment is examined up to a certain radius. Circular fingerprints are not suitable for substructure searching as they do not capture explicit connectivity. Instead, they are used for full structure similarity queries [51, 54, 55]. The standard circular fingerprints version is the Extended-Connectivity Fingerprints (ECFPs), which is based on the Morgan algorithm [56]. ECFPs represents "circular atom neighborhoods", which produce fingerprints of varying length with a commonly used diameter of 4 or 6, referred to as ECFP4 and ECFP6, respectively [51, 54].

2.3 Molecular Novelty

The purpose of our automated framework is to suggest potential molecules for drug development, and it is therefore important to create a diverse set of candidate compounds. To achieve this, the framework has to ensure suggested compounds differ somewhat from those suggested in previous cycles. In this section, molecular novelty will be defined using the language of set theory [57]. A molecule will be considered novel if dissimilar enough to the previously generated molecules. A commonly used metric for determining the similarity between sets is the Jaccard index, J, defined as $\mathrm{J}(A,B) = \frac{|A \cap B|}{|A \cup B|}$. However, as molecular novelty will be based on set dissimilarity rather than similarity, it will instead be computed using the Jaccard distance, J_δ. It is a non-similarity measurement between data sets derived from the Jaccard index [58], defined as $\mathrm{J}_\delta(A,B) = 1 - \mathrm{J}(A,B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}$.

The molecular novelty for each molecule in a set is defined as its average Jaccard distance to all previously created molecules, with the denominator being the total number of created molecules. This is defined by $\mathrm{novelty}(A,B) = \frac{\mathrm{J}_\delta(A,B)}{\mathrm{size}(B)}$, where A is the set of compounds to be evaluated and B is the entire set of created molecules. This will return a list where each element corresponds to the average Jaccard distance of a molecule in A, to all molecules in set B.

2.4 QSAR

Quantitative structure-activity relationship (QSAR) is a mathematical model for displaying relationships between structural properties and biological activities. Generally, variations in structural properties result in differing biological activities, and these models can therefore be used to predict the physicochemical, biological, and environmental properties of compounds from the knowledge of their chemical structure [59]. QSAR models are useful for in-silico drug discovery as they provide a

mechanism for prioritizing large quantities of chemicals based on their biological activities, thus alleviating the need for manual testing [59].

QSAR models can be either regression or classification models. A regression model relates a predictor value to the potency of a response variable, and a classification model relates a predictor value to a categorical value of the response variable. A predictor is a physicochemical property of a compound, and the QSAR response variable is some biological activity of the chemicals [60].

2.5 REINVENT

The design-step of the DMTA-cycle is traditionally performed by chemists manually designing and selecting interesting compounds (Section 1.2.1). In order to create an automated in-silico version of the DMTA-cycle, molecules have to be generated automatically. REINVENT is a de novo design tool that has shown promising performance in this area [15, 61].

2.5.1 Overview

REINVENT 3.0 is an open source AI tool for de novo drug design and will be used as our generative model for molecule creation [16]. REINVENT is trained on datasets derived from ChEMBL, a manually curated database of molecules with drug-like properties [62], and is capable of generating compounds in the SMILES format. It is trained by "randomizing" the SMILES representation of the input data. This randomization of the compounds' representations uses multiple SMILES encodings for the same compound to ensure that the model will learn the grammar rather than memorizing specific strings or substrings. The resulting model will thus show a significantly improved generalization potential and achieve a validity above 99% for the produced SMILES strings. The generative model can be fed with initial information, influencing the type of molecules it generates and thus producing the best molecules based on these criteria.

Most de novo drug design tools have three main components: search space (SS), search algorithm, and search objective. A generative model, or search space, typically consists of either distribution learning or goal-oriented generation. Distribution learning mostly focuses on generating ideas resembling a particular set of molecules, while goal-oriented generation typically uses search algorithms to suggest molecules that satisfies the specific requirements without having to traverse the entire search space. Both cases make use of a scoring function (search objective) to filter results [63]. REINVENT covers both distribution-learning and goal-oriented approaches. A generative model is used as the search space in the goal-directed approach, with reinforcement learning used as search algorithm, and a flexible scoring function is used to form rewards.

2.5.2 Diversity Filter

The scores metioned in Section sec: Overview can also be regulated by a diversity filter (DF) that punishes redundancy and encourages exploration by rewarding solution diversity. As explained in Section 2.3, novelty is highly important for the framework to perform its task effectively, and REINVENT can aid in achieving molecular novelty by providing access to what is called diversity filters. "DF can be regarded as a collection of buckets used for keeping track of all generated scaffolds and the compounds that share those scaffolds" [16]. These filters allow REINVENT to generate solutions that are different from a set of reference values, allowing it to find solutions that differ from previous ones, thereby achieving exploration of the search space [16, 64].

2.5.3 Scoring Functions

Scoring functions are used by REINVENT to guide compound generation towards the desired area of chemical space. They can be tuned by the user to accommodate any given drug discovery project. The user defines which molecular components are of interest, and these are then combined into a composite scoring function, with each component corresponding to a single target property. The feedback from the scoring function is then used in REINVENT's reinforcement learning loop [64].

2.5.4 Reinforcement Learning in REINVENT

Reinforcement learning (RL, Section 2.1) is used to direct the generative model towards areas of the chemical space containing interesting compounds. The user can define a set of requirements reflecting the most important features of the desired compounds, and the reinforcement learning scenario is used to maximize the outcome of a scoring function with relevant parameters [16].

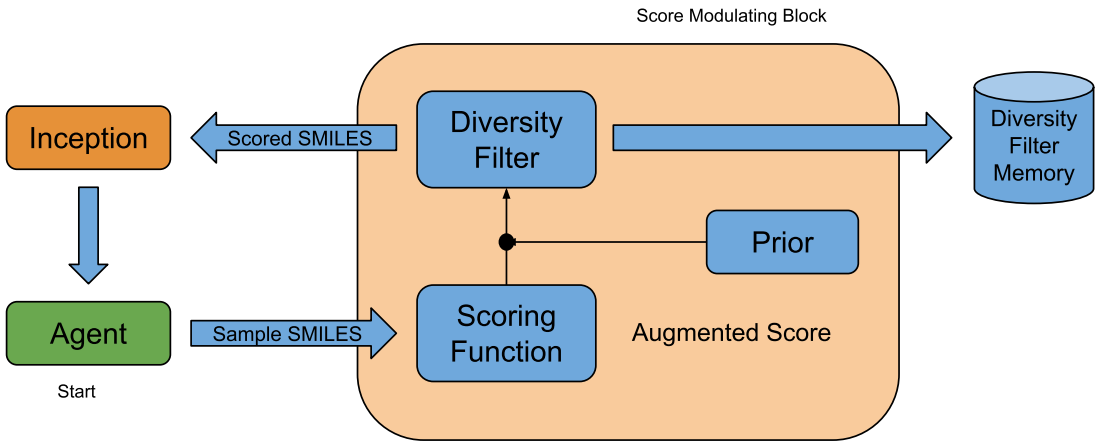

Figure 2.5: The REINVENT reinforcement learning loop [16].

A reinforcement learning structure consists of an actor and an environment. The actor takes an action and receives a reward indicating the quality of the action

taken [25]. In REINVENT, actions are the steps taken for building token sequences, translating to SMILES strings. The environment is shown as *Score Modulating Block* in Figure 2.5. The prior used in the environment is a generative model, with identical architecture and vocabulary as the agent, and it samples compounds from a vast area of the chemical space. It acts as a reference point for the likelihood of sampling any given SMILES. The last component in the RL loop is inception, which keeps track of compounds that have scored well and randomly exposes some of these to the agent to help guide its learning[16].

3
Methods

3.1 DMTA-cycle in-silico: Framework

The developed framework is an in-silico simulation of the DMTA-cycle. The DMTA-cycle consists of four main steps: the design, make, test and analyze steps. There is also a sub-step of interest, the selection step. Figure 3.1 is a visualization of the DMTA-cycle with the selection sub-step included.

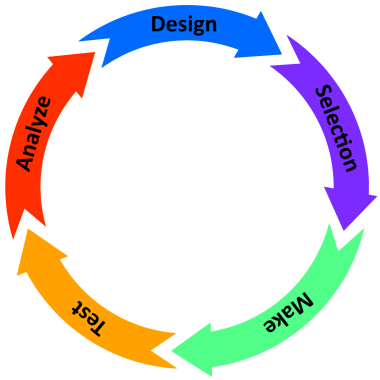

Figure 3.1: A visualization of the Design-Make-Test-Analyze (DMTA) cycle with the selection sub-step included.

3.1.1 Design

Molecular design is traditionally performed by chemists manually designing compounds that seem to be of interest (Section 1.2.1). However, as we aim to automate the drug design process, a generative model will be used to imitate this step. The generative model of choice is REINVENT (Section 2.5) and it will be run for each simulated DMTA-cycle to create a set of molecules. In order to select how many molecules to make available to the next stage in the simulated DMTA-cycle, an integer variable called n_design is available to the user, which the framework uses to tell REINVENT to regulate how many of the generated molecules to present [16].

REINVENT samples molecules in reinforcement learning epochs, 128 per epoch. The user can specify a variable called n_steps to determine the least number of epochs to run [16]. For all following runs, n_steps is set to 500.

Molecular diversity is important when trying to produce useful molecules (Section 2.3), and REINVENT has a boolean setting called Diversity Filter (Section 2.5.2) that makes it prioritize diversification of the generated molecules [16]. This option is activated in every run.

The settings used by REINVENT to generate molecules in the different runs are presented in Table 3.1.

REINVENT settings		
Setting	Run 1	Run 2
n_design	1000	10000
n_steps	500	500
Diversity filter	True	True

Table 3.1: REINVENT settings used in different framework evaluation runs.

3.1.2 Selection

The purpose of the selection process is to choose the 100 most promising compounds from the large pool of compounds created by the design step and presented for use in the subsequent DMTA-cycle stages. It is important to note that the same compound cannot be selected multiple times. The hope is that a reduced number of compounds advancing from the design-step to the make-, test-, and analyze-steps will speed up the DMTA-cycle time (Section 1.3). The selection is done using deep reinforcement learning (Section 2.1). Deep RL was chosen as the design-step is capable of generating very large numbers of molecules, and storing these in a Q-table, as would have been the case with tabular Q-learning, is unfeasible. Instead, a deep Q-network (DQN) is used (Section 2.1.3). As DQN:s can suffer from overestimation of action values in some environments, both DQN and double DQN (DDQN)[1] will be tested and evaluated. Details on the reinforcement learning implementation used for selection is available in Section 3.2.3.

3.1.3 Make

The make-step of the DMTA-cycle is the stage where molecules are normally synthesized in a laboratory (Section 1.2.2). This step is fully simulated in the framework which provides a setting where the user can specify the probability that any molecules is successfully made. This value will be set to 1 for all runs. This will cause the make-step to have a 100% success rate, which is useful as we do not want to accidentally lose data that could have been useful for improving the selection algorithm and the generative model. However, a 100% molecule creation success rate is not realistic, and this could therefore be altered to better simulate real life where not every molecule may be possible to create.

[1] Introduced in Section 2.1.3

Further, restrictions based on molecular size could have been interesting to impose as large molecules might be difficult to create in a laboratory. The production process of chemical drugs is fairly well defined and streamlined, partly due to the larger number of competitors in the field, but also because of the often low number of ingredients required, which allows them to be created in large quantities [65]. Biologics (Section 1.2.2), however, are much more complicated to produce and production tends to yield small quantities. They are also very sensitive to physical conditions such as temperature and light [66].

3.1.4 Test

The selection algorithm will select molecules based on the scores assigned in the test-step. Two different ground-truths are used for scoring molecules to gain a better understanding of how the reinforcement learning models interpret the environment and if they behave differently under different scoring regiments. In one of the runs, the ground-truth used for scoring the selected molecules is based on both the principles of efficacy and novelty, while the other only uses efficacy. In both cases, binary scoring is used, with scores being either 1.0 or 0.0. First, the molecules are evaluated based on their novelty (in cases where novelty is used as a metric), with each molecule being scored based on its average Jaccard distance to all previously created molecules, as seen in Equation (3.1) and explained in Section 2.3.

$$\text{novelty}(A, B) = \frac{\frac{|A \cup B| - |A \cap B|}{|A \cup B|}}{\text{size}(B)} \tag{3.1}$$

$$5.5 \leq \frac{carbon}{oxygen} \leq 5.67 \tag{3.2}$$

$$7 \leq \frac{carbon}{nitrogen} \leq 7.39 \tag{3.3}$$

$$1.18 \leq \frac{oxygen}{nitrogen} \leq 1.34 \tag{3.4}$$

Any molecule with a novelty score > 0.3 is evaluated for its efficacy using Equations (3.2), (3.3) and (3.4) (cases not considering novelty skip straight to this step). These determine whether the molecular composition is deemed promising for the given task by looking at the ratios of carbon, nitrogen and oxygen molecules. If the compound has non-zero counts of at least two of these atoms, and the ratios of the non-zero counts are fulfilled by the equations previously mentioned, the compound is deemed active and given the score of 1.0. Otherwise, it is given the score of 0.0. After computing the scores, each molecule's score, along with their corresponding SMILES and fingerprint, are sent to the analyzer.

3.1.5 Analyze

The analyzer receives molecule scores with corresponding SMILES strings and fingerprints from the test-step. These are used to evaluate the selected molecules' performance and dictates how the framework should be updated for the next iteration. Molecule performance is based on a comparison between the assigned scores, as described in Section 3.1.4, and a target score set by the user. As the test-step uses binary scoring for molecules, the target score value will be set to 1 for all runs. It also provides a setting which controls whether the framework stops after the molecules reach the threshold value. This setting will be turned off for all runs as we wish to continue training until reaching the desired number of DMTA-cycles (500). Compounds reaching the threshold are then stored to be used for updating the design-step, giving REINVENT a better understanding of what constitutes a useful compound, hopefully allowing it to generate more effective ones in subsequent DMTA-cycles.

The next step is to update the QSAR model used by REINVENT which is based on scikit-learn, a machine learning library for Python, to include the newly tested compounds [67–69]. This is done using fingerprints and test scores of the molecules selected and made. All previously tested compounds' fingerprints are concatenated with the ones collected over the latest DMTA-cycle simulation, and the same procedure is performed for the test scores. The QSAR model is then re-fit using all concatenated data.

3.2 Reinforcement Learning

3.2.1 General

The reinforcement learning algorithm uses two neural network models with identical structure, both implemented with PyTorch [70]. Two network models, a primary_model, Q, and a secondary_model, Q′, are used to allow decoupling of the action selected and the target Q-value generation. The DQN algorithm uses only Q, while the DDQN algorithm uses both Q and Q′. For each model, the input layer has 2048 input features and 1024 output features. The first hidden layer has 1024 input features and 512 output features, and the second hidden layer has 512 input features and 256 output features. The output layer has 256 input features and 1 output feature. The input layer has 2048 input features as that is the amount of bits used to represent a molecule's fingerprint, and the output layer has 1 output feature which represents a molecule's q-value.

3.2.2 State Representation

The state for each DMTA-cycle iteration is represented by the number of molecules most recently generated by REINVENT, and a 2048 bit representation of their corresponding fingerprints. In an effort to evaluate how the RL selection algorithm's

performance depended on the number of input molecules, two different run setups were investigated. The design-step generated 1000 molecules in the first trial and 10000 in the second. The networks are fed with the entire state-space, consisting of either 1000 or 10000 molecules with 2048 features each. The RL network's input vector thus had the dimensions $1000 * 2048$ in the first version and $10000 * 2048$ in the second. The network then outputs the Q-values for each molecule, with the output vectors having the dimensions $1000 * 1$ or $10000 * 1$. The difference in selection performance between these settings is covered in Chapter 4. To ensure the RL networks received the same number of input molecules for each DMTA-cycle iteration, mock data comprised entirely of 0's was used as a complement in cases where REINVENT had created fewer molecules than desired for the current setup.

3.2.3 Explore/Exploit Function

As there are a considerable amount of molecules to investigate, it is important that our selection algorithms thoroughly explore the different options. To balance exploration and exploitation, the decaying ε-greedy algorithm (Section 2.1.1) was used with a decay rate of 0.99 and a minimum exploration rate of 0.02. The decay function is shown in Equation 3.5.

$$\text{exploration rate } \varepsilon_t = \begin{cases} \varepsilon_t * 0.99 & \text{if } \varepsilon_t \geq 0.02 \\ 0.02 & \text{otherwise} \end{cases} \tag{3.5}$$

A random selection is made with probability ε, and the algorithm will exploit by selecting the compound with the highest q-value with probability $1-\varepsilon$. This process is repeated until 100 compounds have been selected. The same compound cannot be selected multiple times.

RL settings	
Settings	Value
learning rate	0.00025
gamma	0.90
epsilon max	1.0
epsilon min	0.02
epsilon decay	0.99
batch size	16
optimizer	Adam
loss function	MSE

Table 3.2: Settings used for RL in the selection step.

3.2.4 Reward Setup

Two different approaches are used for rewards, as outlined in Section 3.1.4 [25]. In the first, the reward is based entirely on whether the selected molecule lies within the thresholds specified by Equations 3.2, 3.3 and 3.4. If the chemical composition of a compound satisfies these constraints, it is given the score 1.0. Otherwise it is given the score 0.0. In the second approach, molecules are also scored based on their novelty, with novelty computed as per Equation 3.1. If a molecule has an average Jaccard distance to all other molecules of ≥ 0.3, it is deemed novel and is subsequently subjected to the same test as in the first approach. If its average Jaccard distance is < 0.3, it is deemed not novel enough and given the score 0.0.

3.2.5 Molecule Selection

The network models are trained using the fingerprints of the molecules generated in the design step. These are presented to the model which outputs a Q-value for each fingerprint. These Q-values are either explored or exploited according to Equation 3.6, where R is a random threshold value between 0 and 1 and ε_t is the exploration rate at time t. If the Q-values are explored, one is chosen at random and the molecule corresponding to this value is stored.

$$\text{selected Q-value} = \begin{cases} \text{random} & \text{if } R < \varepsilon_t \\ \text{highest} & \text{otherwise} \end{cases} \tag{3.6}$$

If the Q-values are instead exploited, the highest available Q-value is selected (and subsequently marked as ineligible to prevent it being selected multiple times), and its corresponding molecule is stored. This process is repeated until 100 molecules have been selected, after which all selected molecules are stored in the experience replay buffer to be used for training. The parameter values used for molecule selection and in experience replay are shown in Table 3.2.

3.2.6 Experience Replay

The analyzer calls on the selector to update when new test scores are available and the new information is added to the replay buffer. Next, the algorithm selects 16 samples from the buffer to be used for training the RL model with either DQN or DDQN. The DQN and DDQN update functions are shown in Equations 3.7 and 3.8, respectively. The target model Q′, used with DDQN, is updated every third cycle.

$$Q^*(s_t, a_t) \approx r_t + \gamma Q(s_{t+1}, argmax_{a'} Q(s_t, a_t)) \tag{3.7}$$

$$Q^*(s_t, a_t) \approx r_t + \gamma Q(s_{t+1}, argmax_{a'} Q'(s_t, a_t)) \tag{3.8}$$

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{3.9}$$

When the models have been updated, the loss is calculated using mean squared error (MSE). MSE is the mean value of the squared error of the observed values Y and the predicted values $\hat{Y}$, defined in Equation 3.9. Next, the gradients are computed, and the error backpropagated using the Adam optimizer, defined in Section 2.1.4. The final step is updating the exploration rate using Equation 3.5. This iteration of the DMTA-cycle is completed, and unless we just completed cycle 500, the process starts again from the design-step and incorporates the newly acquired knowledge.

4
Results

4.1 Settings

All of the results in the following sections were created with REINVENT setups as per Table 3.1 in Section 3.1.1. It covers the different variations of REINVENT settings for the runs we've produced, determining whether we run with novelty off, novelty on, and the amount of compounds we select during each selection process. With novelty on, the novelty calculation using Jaccard distance as described in Section 3.1.4 is utilized. With novelty off, there is no novelty calculation and compounds are not punished for being similar. As these settings vary across different runs, we can expect them to have significant impact on the results presented in the following sections. Table 3.1 also shows that we have the Diversity Filter enabled for all runs which should improve the average scores a decent amount for runs using novelty scoring. Specifics on how the Diversity filter works are found in Sections 2.5.2 and 3.1.1.

All the results from the DQN and DDQN selectors will also utilize the RL settings for the selection-step as presented in Table 3.2 in Section 3.2.3. Table 3.2 covers multiple variables used by Equations 3.7 and 3.8, namely learning rate (α), discount factor (γ), epsilon min (ε_{min}), epsilon max (ε_{max}) and epsilon decay (ε_{decay}). Batch size determines the number of samples that the network will use to train at each step. The optimizer refers to the optimization technique for gradient descent, and loss function defines which technique used for calculating loss. The amount these setting will affect the results vary, the variables should affect things a decent amount since they are used for multiple calculations each cycle. The batch size will likely not have a huge effect on the results as they mostly affect the speed and stability of the selector updates, as the batch size determines how much data to be used for training at the same time. The optimizer and loss functions could have a bigger effect, but since they are being kept the same across DQN and DDQN runs it seems unlikely these would cause large differences in selector performance. To summarize, we don't believe the RL setting in Table 3.2 will have a big effect on the results, largely due to all DQN and DDQN runs using the same settings.

4.2 Baseline

In order to properly evaluate the performance of our selection algorithms, six baseline results were established using greedy and random molecule selection. In the greedy selection, the molecules scored most highly by REINVENT's own scoring function were selected in each cycle, while the random selection randomly picked molecules created by REINVENT.

4.2.1 Baseline Results

Figure 4.1 shows the greedy and random selectors where they each select from 1000 compounds with novelty on. Figure 4.2 shows baseline selectors where they each select from 1000 compounds with novelty off. Finally, Figure 4.3 shows the two selectors where they each select from 10000 compounds with novelty on. All figures show the average results over 10 independent runs, each running for 500 cycles. All figures also use a moving average score over 10 cycles.

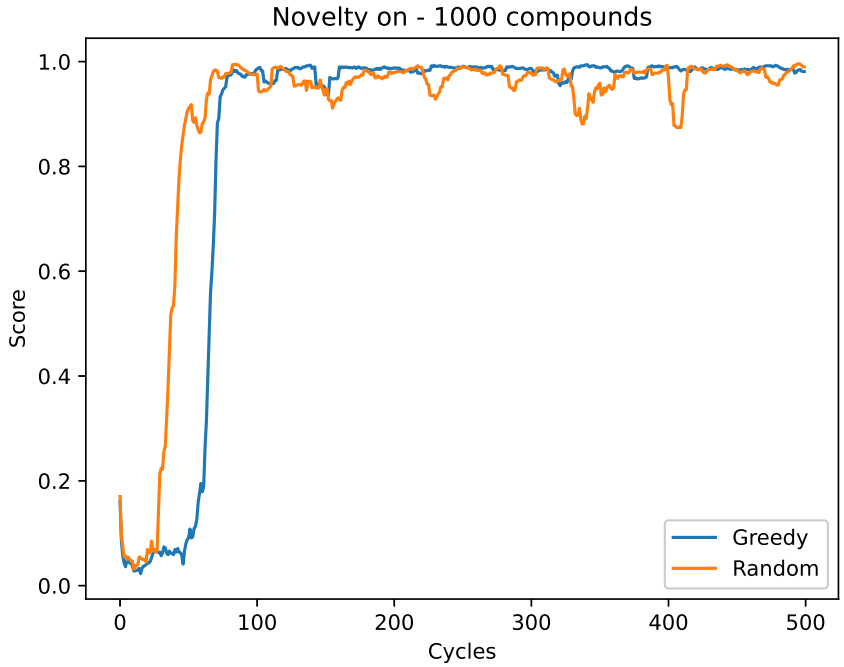

Figure 4.1: Plots of the greedy and the random selectors where they select from 1000 compounds with novelty on. The scores are computed using a moving average of 10 cycles.

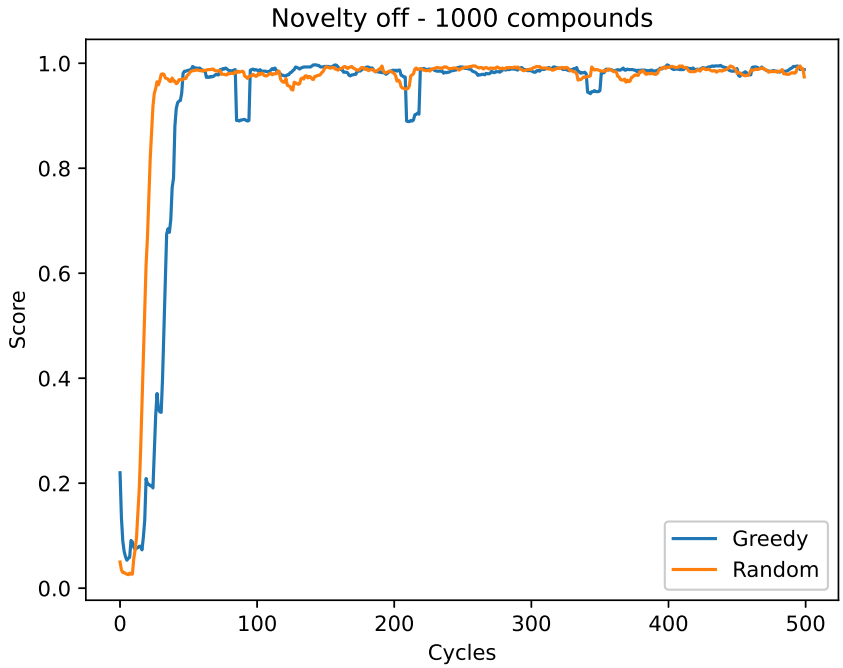

Figure 4.2: Plots of the greedy and the random selectors where they select from 1000 compounds with novelty off. The scores are computed using a moving average of 10 cycles.

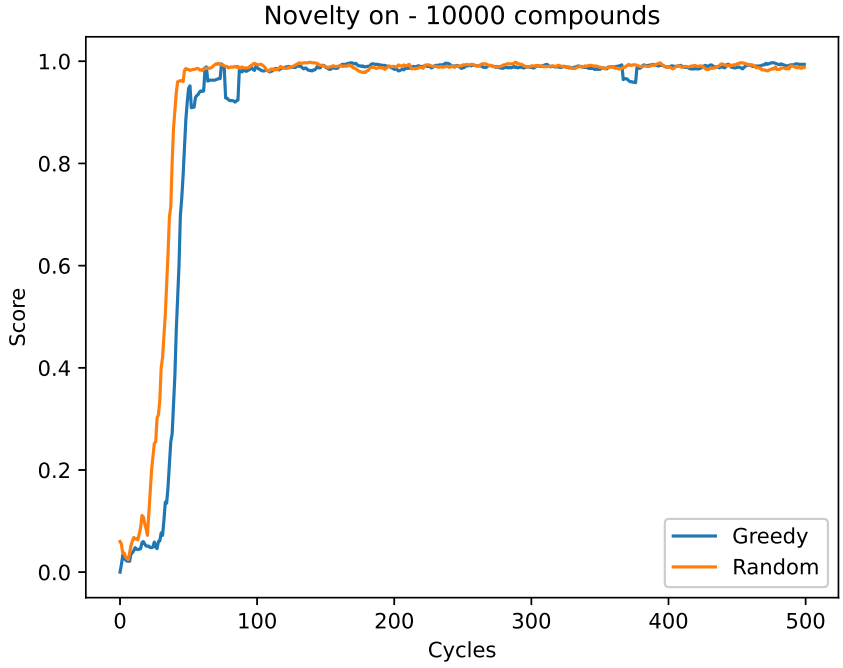

Figure 4.3: Plots of the greedy and the random selectors where they select from 10000 compounds and with novelty on. The scores are computed using a moving average of 10 cycles.

4.2.2 Baseline Observations

Comparing Figure 4.1 with Figure 4.2 we can see that the greedy selector appears to provide slightly better results, score wise, with novelty on and that the random selector appear to perform slightly better with novelty off. The random selector appears to converge to its maximum score faster than greedy across all three setups, with the greatest difference compared to the greedy selector being with novelty turned on and 1000 compounds used (Figure 4.1). Stability wise they both appear to be quite steady across all three configurations, with both selectors appearing to be the most stable when using 10000 compounds (Figure 4.3). Notably, the random selector barely oscillates off of 1.0 score at all. It is worth noting that the random selector is slightly unstable with novelty on and 1000 compounds used, it reaches 1.0 score at quite a few points but appears to oscillate around 0.90-0.95 with a fairly high frequency. None of the other five results across all three figures appear to fluctuate with such a high frequency. They all have the occasional dip in score, but nothing noteworthy.

In terms of speed, all the random selector configurations appear to reach a score of around 0.9 within less than 50 cycles with novelty off (Figure 4.2), and the 10000 compounds version (Figure 4.3) settling around 1.0 score in the same amount of cycles. The greedy selector is slightly slower, reaching a score of about 0.9 within less than 60 cycles with novelty on (Figure 4.1), and with novelty off (Figure 4.2) settling in around a score of 1.0 within the same cycle amount.

4.3 Deep Q-Learning Selectors

Two selection algorithms were implemented based on the theory described in Section 2.1.3. The first one is a Deep Q-Network (DQN), which is an RL algorithm that utilizes a neural network to approximate the Q-value function. The second is a Double Deep Q-Network (DDQN) selection algorithm which works similarly to the DQN selector, but also uses a second neural network to better approximate the Q-value function.

4.3.1 Deep Q-Learning Selectors Results

Similarly to Section 4.2.1, all these results use the setup as shown in Section 3.2.3. Figure 4.4 shows the results of the DQN and DDQN selectors where they each select from 1000 compounds with novelty on. Figure 4.5 shows results of the two implemented selectors where they select from 1000 compounds with novelty off. Lastly, Figure 4.6 shows the two selectors where they select from 10000 molecules with novelty on. Similarly to Section 4.2.1, the figures all show the average results over 10 independent runs, each running for 500 cycles. All figures also use a moving average of 10 cycles.

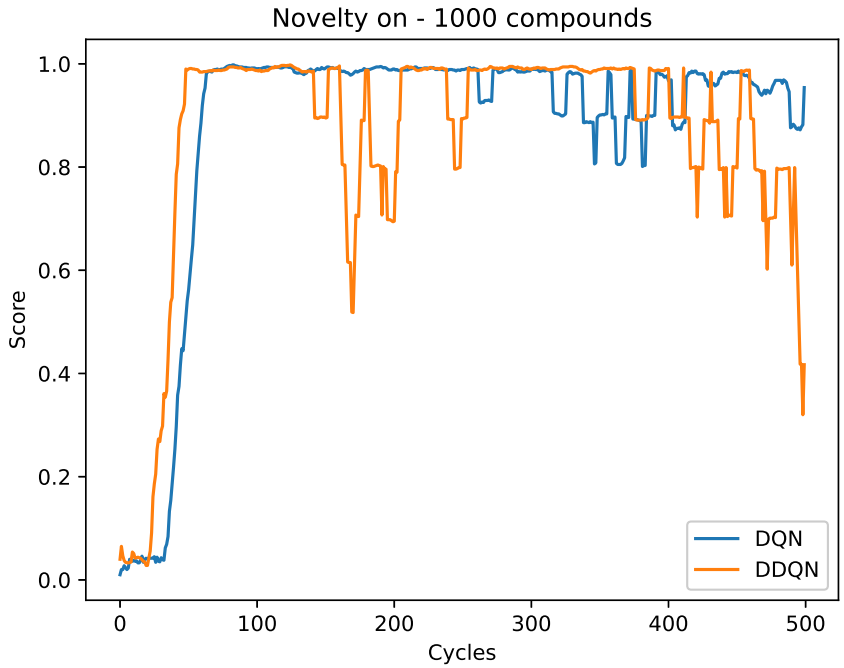

Figure 4.4: Plots of the DQN and the DDQN selectors where they select from 1000 compounds with novelty on. The scores are computed using a moving average of 10 cycles.

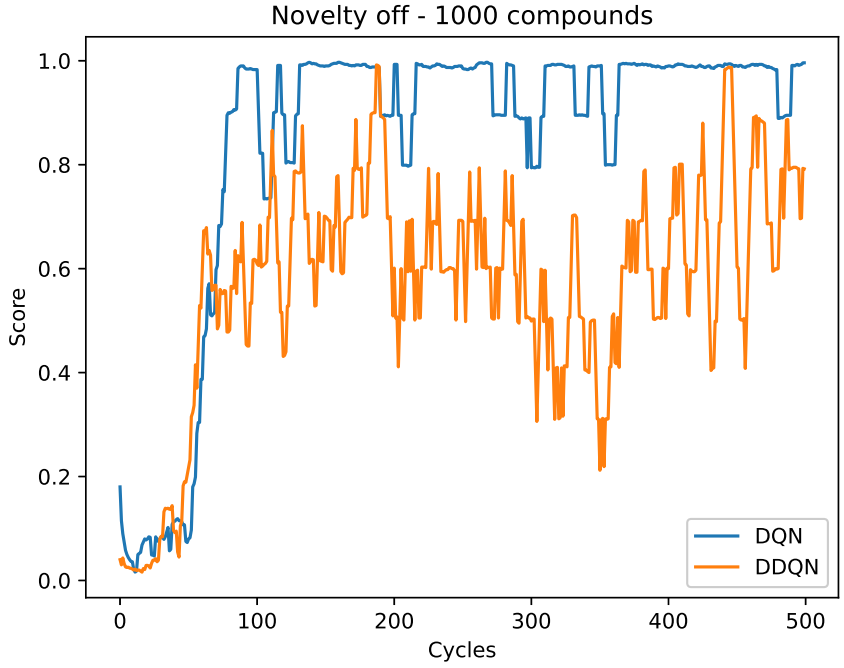

Figure 4.5: Plots of the DQN and the DDQN selectors where they select from 1000 compounds with novelty off. The scores are computed using a moving average of 10 cycles.

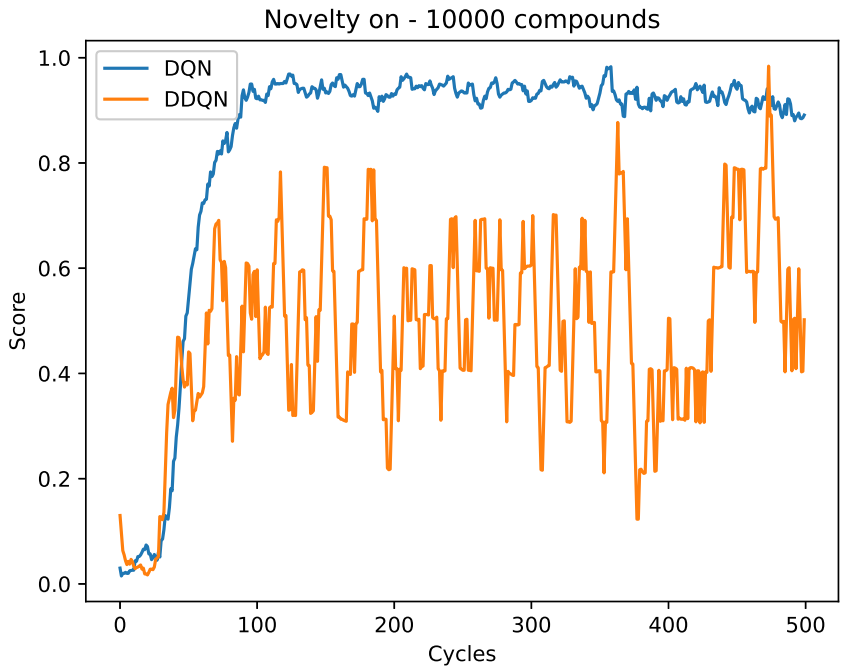

Figure 4.6: Plots of the DQN and the DDQN selectors where they select from 10000 compounds and with novelty on. The scores are computed using a moving average of 10 cycles.

4.3.2 Deep Q-Learning Selectors Discussions

Both selectors start of well and rise quickly with the initial cycles, although this is most likely the diversity filter's work, since it will filter all SMILES below a user-set threshold [16]. The DQN selector with novelty on (Figure 4.4) reaches a score of roughly 0.95 after about 60 cycles. After this, it rises slightly to around 1.0 and then hovers around this score. The DDQN selector with novelty on (Figure 4.4) rises faster and reaches 1.0 after about 50 cycles. Both selectors fluctuate a decent amount afterwards, particularly later on for the DQN selector. The fluctuations for the DDQN selector appear both earlier and appear to have larger swings than the DQN selector. The average results in Figure 4.4 indicate the DQN selector performance appear to be slightly more stable than the DDQN selector when presented with 1000 molecules each cycle.

Looking at Figure 4.5 in Section 4.3.1 we can see how the DQN and DDQN selectors compare to each other with novelty turned off. The DQN selector is slower to reach a score near 1.0 than with novelty on (Figure 4.4), taking around 90 cycles in this case. It also fluctuates a lot more and starts fluctuating earlier, even before the initial big growth of score. The DDQN selector with novelty off appears to be doing significantly worse than with novelty on (Figure 4.4), reaching around 1.0 score at 2 peaks around 200 cycles and 400 cycles in. The DDQN selector is constantly fluctuating with this setup and does not appear to have any significant intervals where it appears to be stable. Comparing the two selectors with novelty

off, the DQN selector appears to be a clear winner. It's not as stable as with novelty off but it does appear to be stable the majority of the time. The DDQN selector appears to be quite unstable with these settings. It appears that the novelty calculations in the test-step affect the average results more for the DDQN selector than the DQN selector based on these two figures (Figure 4.4 and Figure 4.5).

The final figure in Section 4.3.1 is Figure 4.6 which shows the results for the DQN and DDQN selectors with novelty on, but this time selecting from 10000 compounds each cycle, as opposed to the 1000 compounds in the previous two figures in the same section. The DQN selector appears to reach its max score at around the same speed as the previous figures. The max score is however slightly lower, maxing out around 0.95. It also appears to be more unstable than the 1000 compound figures, constantly fluctuating with a high frequency and low amplitude. The DDQN selector appears to perform significantly worse when it uses 10000 compounds, compared to with novelty off and using 1000 compounds. It only reaches its max score once (close to 1.0) after around 470 cycles. Before and after that it appears to oscillate violently around 0.5, reaching 0.8 and then dipping down to around 0.2-0.3. The DQN selector oscillates around a higher score (0.9) and with a higher frequency than the DDQN selector. Both selectors appear to perform worse when using 10000 compounds, with the performance decrease appearing larger with the DDQN selector.

We can also look at and draw comparisons from a few selected runs from each selector, selecting from 1000 molecules with novelty on. The runs were handpicked to more clearly illustrate the difference in performance between DQN and DDQN. In Figure 4.7 we see three DQN selector runs from the selection of runs used to compose the DQN plot in Figure 4.4. Three DDQN selector runs used by the DDQN plot in Figure 4.4 can be found in Figure 4.8.

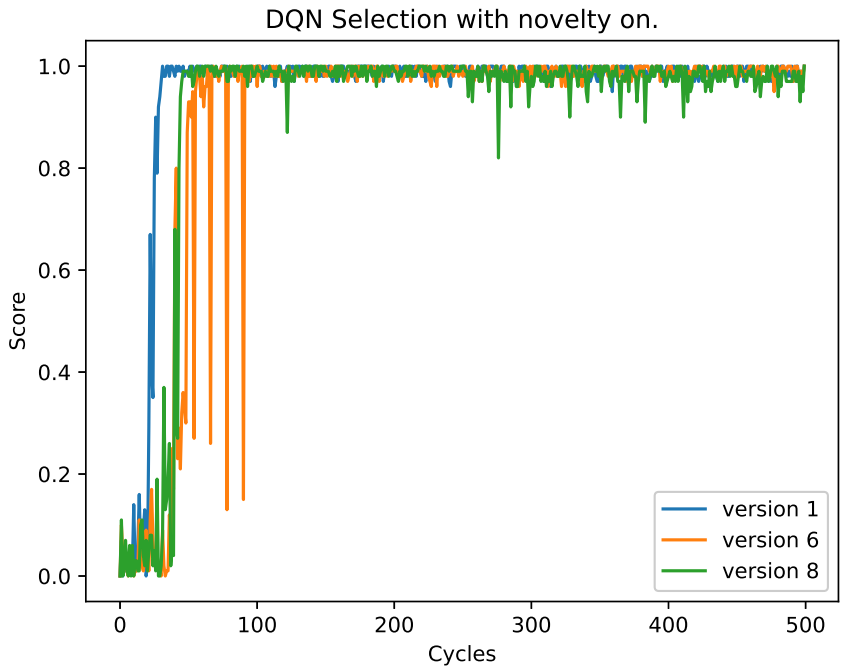

Figure 4.7: Three selected individual runs (version 1, 6, 8) using the DQN selector. The algorithm selected from 1000 molecules and used novelty scoring.

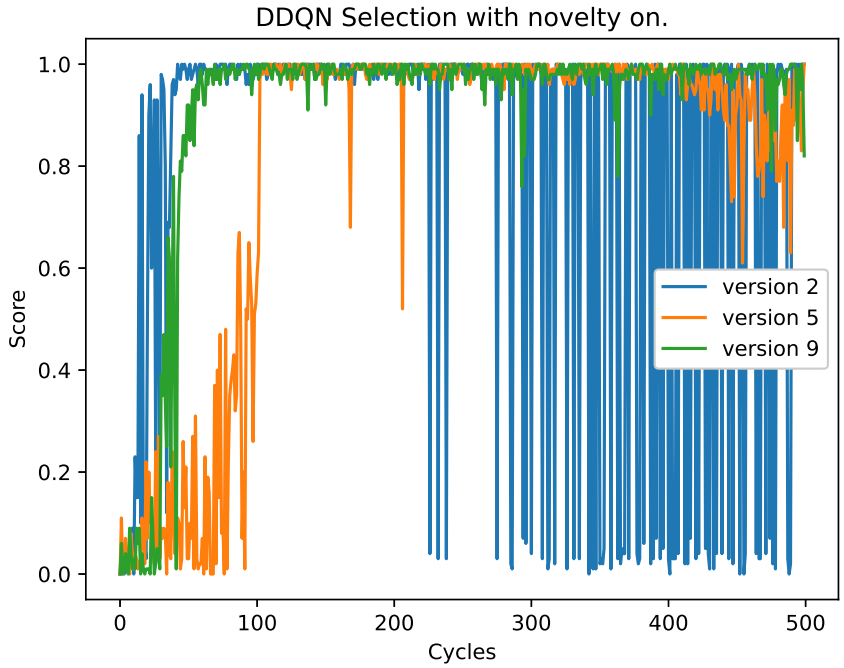

Figure 4.8: Three selected individual runs (version 2, 5, 9) using the DDQN selector. The algorithm selected from 1000 molecules and used novelty scoring.

The DQN selector runs appear to be pretty stable, with the version 6 plot from Figure 4.7 being a bit more unstable until it settles around cycle 100. The DDQN selectors are quite volatile, with some runs experiencing extreme scenarios as can be seen with the version 2 plot from Figure 4.8 where the DDQN selector quickly reaches a score of near 1.0, but after 225 cycles starts experiencing runs scoring near 0. This is likely the reason the composite DDQN figure (Figure 4.4) converges to roughly 0.8, while most other selection algorithms trend toward 0.9. This issue could potentially be explained by what is known as catastrophic forgetting, where a neural network forgets previously learned information upon acquisition of new information [71]. As the chemical space is enormous, it is likely that some molecules may not reoccur for a long time, and in these scenarios, network information retention is important while still allowing for acquisition of new knowledge. It might seem as though the experience replay buffer should prevent this. However, when the model has achieved a large number of successes, these experiences will make up the majority of the buffer and the algorithm will start forgetting what failure looks like. It will therefore struggle to predict values for these states. A way of maintaining proficiency on tasks not recently experienced was proposed by Kirkpatrick et al., where old tasks are remembered by slowing down the learning on weights important to those tasks [72]. Resolving this issue in a machine learning framework for drug discovery is something that could yield great benefits, as the DDQN shows great promise with runs potentially quickly converging to scores near 1.0, as can be seen in the runs with dedicated plots in Figure 4.8. A possible explanation as for why DQN does not seem to experience the catastrophic forgetting is due to it using the same network for calculating both the current and target values, whereas DDQN uses two different networks. With two networks, each susceptible to catastrophic forgetting, DDQN may suffer from an increased likelihood of encountering this issue. One of the networks is used for calculating the current values, while the other calculates the estimated target values. The difference between these two values is used to calculate the RL model's loss. If one of these networks were to "forget" what failure looks like, the loss would become large and the model would be unable to properly estimate how good a specific state is.

To summarize, the DQN selector appears to score well and is close to 1.0 in most cases, as can be seen by both the figures in Section 4.3.1 and the individual runs in Figure 4.7. The DDQN selector also scores well and is close to 1.0, although it does appear to have some instability issues. It is most notable in Figure 4.5 and Figure 4.6, as well as the version 9 run in Figure 4.8.

4.4 Comparisons and Discussions

We are now going to present composite figures of the figures from Section 4.2.1 and Section 4.3.1. This is done to make it easier to draw comparisons between the two baseline selectors (greedy and normal) and the two RL selectors (DQN and DDQN). Since these figures are just composites, they all use the same settings as in Section 4.2.1 and Section 4.3.1. Figure 4.9 shows the results of all the selectors where they each select from 1000 compounds with novelty on. Figure 4.10 shows the

results of all the selectors where they each select from 1000 compounds with novelty off. Lastly, Figure 4.11 shows all the selectors where they each select from 10000 compounds with novelty on. Like the individual figures in Section 4.2.1 and 4.3.1, all the figures show the average results over 10 independent runs, each running for 500 cycles. All figures also use a moving average of 10 cycles.

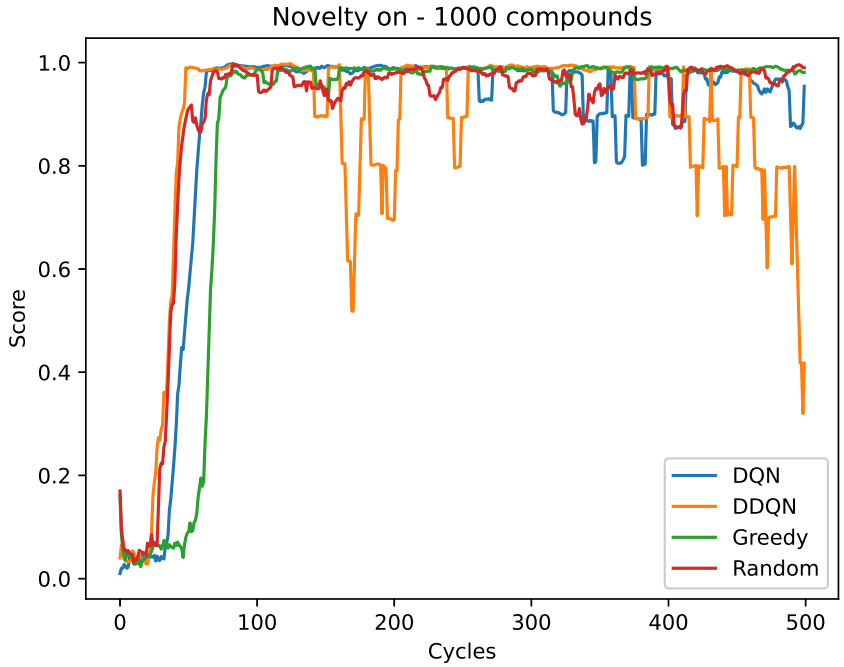

Figure 4.9: Plots of all four selectors (greedy, random, DQN, DDQN) where they select from 1000 compounds with novelty on. The scores are computed using a moving average of 10 cycles.

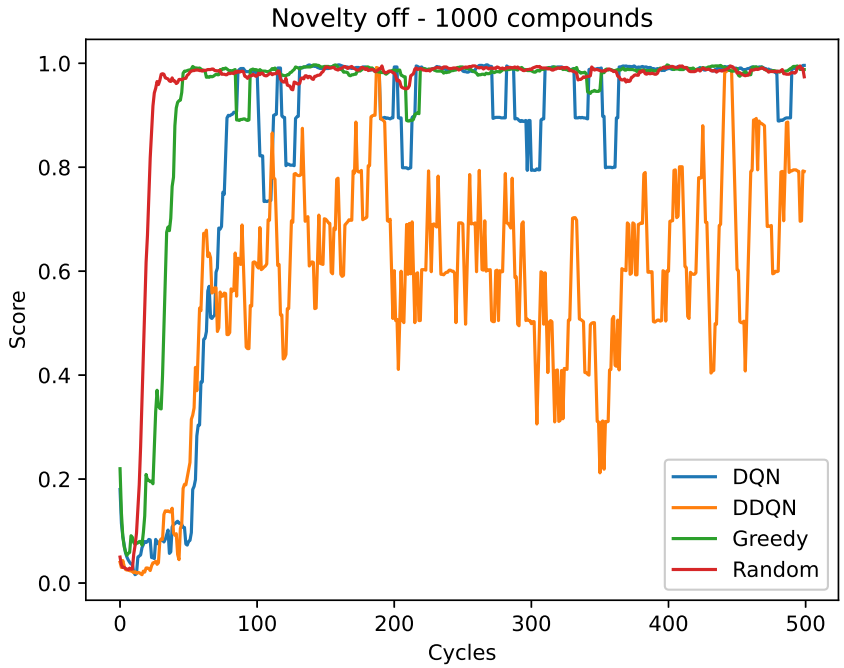

Figure 4.10: Plots of all four selectors (greedy, random, DQN, DDQN) where they select from 1000 compounds with novelty off. The scores are computed using a moving average of 10 cycles.

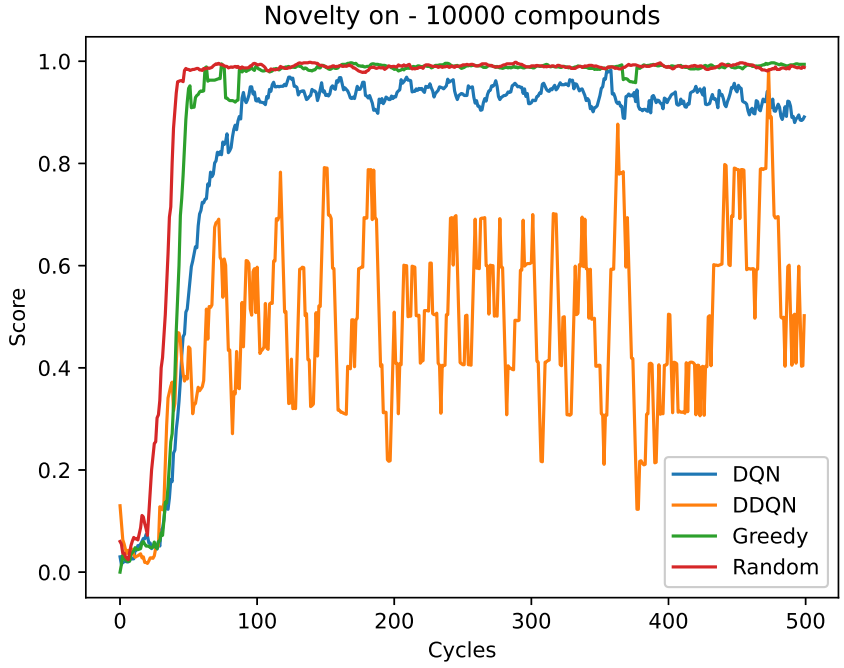

Figure 4.11: Plots of all four selectors (greedy, random, DQN, DDQN) where they select from 10000 compounds and with novelty on. The scores are computed using a moving average of 10 cycles.

We can now draw some comparisons between the baseline runs in Section 4.2.1 and the two Deep Q-Network Selectors in Section 4.3.1 using these new figures. Starting with Figure 4.9, 1000 used compounds with novelty on, the first observation is that the DQN selector appears to perform equally well as the greedy and random selectors from the same figure. It even appears to be getting faster results than the greedy selector. This is not entirely surprising, as the DQN selector utilizes a neural network that learns which molecules score well rather than just picking the ones REINVENT scores highly, as the greedy selector does [73]. What is surprising is how well the random selector performs. One could expect that just randomly picking the molecules created by REINVENT would result in a more random, and in return lower, score than what the greedy and DQN selector can provide as a random selection would at times choose very poorly performing molecules without learning from the negative experience, but this does not appear to be the case. The amount of runs used for our results, i.e., 10 runs, could be the culprit. We are a bit vary of pointing this out specifically though, as more runs of a few select series are available, the DQN selector in Figure 4.9 being amongst them, and we struggle to see a noticeable difference when more runs are included.

Looking at the DDQN selector's result from Figure 4.9 and comparing it to the baseline selectors in the same figure, it leaves more to be desired. We believe that the apparent instability of the DDQN selector is the culprit here. If we look at the individual runs of the DDQN selector in Figure 4.8, we see that the DDQN selector is quick to converge with a score approaching 1.0. However, we can also see the instability over the moving average in Figure 4.8, particularly in the version 2 plot, but also somewhat in the version 5 plot. We believe that this instability is what causes the plot of the average DDQN selector scores in Figure 4.9 to perform significantly worse than the baseline and the DQN selectors in the same figure.

Next, we will look at Figure 4.10 with 1000 used compounds and novelty off. The situation for the DQN selector is largely similar to the previous case. It is a bit slower to reach max score, taking around 100 cycles to do so. It is largely stable with a few drops to around 0.8 score, lasting around 20 cycles each until it once again settles around 1.0. The greedy selector exhibits behavior similar to the DQN selector, with roughly the same amount of dips but only dropping to about 0.9. The random selector is once again the best performing selector, being both the fastest to reach the max score and being the most stable.

Switching over to the DDQN results from Figure 4.10, we note that it appears to be even more unstable than in Figure 4.9. It only reaches a test score of 1.0 at two points, around cycle 200 and 430. The rest of the time it appears to be oscillating around a test score of roughly 0.6, never really having any stable periods. Our analysis was unable to explain the reason both the DQN and the DDQN selectors perform worse, slightly in the DQN selector case and significantly in the DDQN selectors case. Looking at the two baselines, the random selector performs even better with novelty off (Figure 4.10) with novelty on (Figure 4.9), while the greedy selector performs near the same level for both settings, with perhaps a slight

reduction in performance when novelty is on. One thing we are considering is that the more stringent novelty calculations, i.e., Figure 4.9 with novelty on, cause the selectors to be forced to make better choices. This favors the three selectors that do not pick completely at random, i.e., all but the random selector.

Looking at Figure 4.11, where selectors picked from 10000 compounds with novelty on, the first observation is that the two baseline selectors' performances do not differ from the previous two figures (Figure 4.9 and Figure 4.10) to any significant degree. Looking at the DQN selector, it performs significantly worse than the results where it selects from 1000 compounds, regardless of the novelty setting used. It never manages to reach a 1.0 in test score and instead oscillates around 0.9, never truly converging. The DDQN selector is also performing worse when selecting from 10000 compounds compared to 1000 compounds, oscillating around 0.5 score with no stable periods, reaching a max score of 1.0 once at around cycle 460. It behaves largely the same as Figure 4.10 (selecting from 1000 compounds and novelty off), which is interesting because Figure 4.11 does have novelty on. If the opposite had been true, i.e., novelty off when using 10000 compounds, it could have been easy to look at that when considering why we are seeing performance problems with the DQN and the DDQN selector and not with the baselines. If anything, based on our results, the baselines are performing slightly better when selecting more compounds than less.

5
Conclusions

Diversity filter makes REINVENT very good at finding novel compounds, with the most notable effect being the how quick all plots in the three figures in Section 4.4 rise toward their test score maximum [16]. The novelty setting appears to fill a similar role when it comes to the speed on the non-random selectors. Comparing the amount of cycles it takes for the DQN selector to reach its test score maximum with novelty off in Figure 4.10 with the same selector in Figure 4.9 with novelty on shows this effect. It does not appear to have any significant effect on the baseline selectors, if anything it appears to make the random selector perform slightly worse, as can be seen in Figure 4.9. One could argue whether the implemented novelty setting provides any significant improvements over just utilizing diversity filter. We would argue that it does, with the noted improvement of the DQN selector using the novelty setting suggesting this. The DDQN selector also performs significantly better with novelty turned on, which becomes apparent when comparing Figure 4.9 (novelty on) with Figure 4.10 (novelty off). As both the implemented DQN and DDQN selectors utilize one additional layer of RL in their selection method in comparison to the baselines, it appears that the novelty setting has a notable effect on the speed if the selection method is advanced enough. So, the need for the novelty setting appears to be tied to the complexity of the selection method. If it is sufficiently complex, the diversity filter alone does seem not offer enough of an improvement to the speed of the selector.

As we performed many runs where only 1000 of the molecules designed by REINVENT were used, random selection might be performing unreasonably well compared to other selection algorithms as a smaller pool of available molecules increases the likelihood of it selecting good ones by chance. Therefore, it could be interesting to perform runs using all available molecules to gain a better understanding of how the different selection algorithms compare to one another in large state spaces. The runs using 10000 molecules, in Figure 4.11, provide a glimpse into what this might look like. Still, for 10000 molecules, random selection performs just about identically to greedy.

One potential reason for this is that our ground truth, described in Section 3.1.4, is not complex enough. This means that the molecules produced by REINVENT are considered promising too easily, thus favoring a random selection more. A potential counterargument can be attained by looking at Figure 4.11 again, but this time focusing on the other two plots, the DQN and DDQN selector plots. Both the DQN and the DDQN perform notably worse when selecting 10000 molecules, compared

to when selecting 1000 molecules in Figure 4.9, with all other settings being the identical across the two versions. Both of the implemented selectors perform better when selecting 1000 molecules rather than 10000 molecules, with the DQN selector performing approximately as well as the greedy selector. A more complex ground truth could potentially mean that the scores REINVENT assigns to molecules may be a bit less accurate and could therefore consequently make the non-random selectors less accurate. A larger selection of molecules could potentially show this effect. The results in Figure 4.11 compared to Figure 4.9 points towards this effect for the implemented RL selectors, but not for the greedy selector. Therefore, the hypothesis that our ground truth is too simple is left unanswered.

Both DQN and DDQN fail to achieve the results of greedy and random, with DDQN especially lacking. DQN is near identical in performance to the baselines in runs with 1000 molecules and novelty on (Figure 4.9), showing some signs of promise. DDQN however reaches the same maximum scores as the others, but is prone to massive drops in performance. It may be that REINVENT is so good at suggesting molecules that the learning performed by our RL selection algorithm interferes with REINVENT's own learning. As there are additional steps in DDQN compared to DQN and the baselines, REINVENT's own understanding of what constitutes a good molecule might get even more clouded. Figure 4.10 seems to support this idea.

Neither DQN nor DDQN perform as well without novelty scoring as with it, with DDQN again being significantly worse than the rest. Without novelty scoring, our RL selection algorithm does not reward compounds for being novel. This might cause confusion for REINVENT which uses the diversity filter but is not explicitly rewarded for novelty. This could potentially be cause for uncertainty as to what REINVENT should generate, leading to larger drops in performance than when novelty was rewarded. Both DQN and DDQN are particularly far from the baseline performances in runs with 10000 compounds (Figure 4.11). Here, even more molecules are ran through the RL selection algorithm, further decreasing the purity of REINVENT's knowledge by polluting it with data passing through a different RL system. Random and greedy both converge to scores near 1 faster without novelty scoring, showing that REINVENT is fully capable of generating useful molecules quickly when the data is not adulterated by another RL system.

To summarize, neither of the implemented RL selectors (DQN and DDQN) performed well enough to compare with the baseline selectors in any of our test series. The DDQN version performed particularly poorly and we suspect it could be due to data pollution caused by there being to many agents involved, clouding REINVENT's understanding of what constitutes a good molecule. The DQN selector on the other hand shows promise and could with some polish potentially be a viable alternative to the greedy and random selectors.

Bibliography

[1] Petra Schneider, W Patrick Walters, Alleyn T Plowright, Norman Sieroka, Jennifer Listgarten, Robert A Goodnow, Jasmin Fisher, Johanna M Jansen, José S Duca, Thomas S Rush, et al. Rethinking drug design in the artificial intelligence era. *Nature Reviews Drug Discovery*, 19(5):353–364, 2020.

[2] Amol Deore, Jayprabha Dhumane, Rushikesh Wagh, and Rushikesh Sonawane. The stages of drug discovery and development process. *Asian Journal of Pharmaceutical Research and Development*, 7(6):62–67, Dec. 2019.

[3] Petra Schneider, W. Patrick Walters, and Alleyn T. et al. Plowright. Rethinking drug design in the artificial intelligence era. *Nature Reviews Drug Discovery*, 19(6):353–364, May 2020.

[4] Serge Mignani, Scot Huber, Helena Tomás, João Rodrigues, and Jean-Pierre Majoral. Why and how have drug discovery strategies in pharma changed? what are the new mindsets? *Drug Discovery Today*, 21(2):239–249, 2016.

[5] Kenneth I Kaitin. Deconstructing the drug development process: the new face of innovation. *Clinical Pharmacology & Therapeutics*, 87(3):356–361, 2010.

[6] Christopher P Adams and Van V Brantner. Estimating the cost of new drug development: is it really $802 million? *Health affairs*, 25(2):420–428, 2006.

[7] Joshua Meyers, Benedek Fabian, and Nathan Brown. De novo molecular design and generative models. *Drug Discovery Today*, 26(11):2707–2715, 2021.

[8] Jean-Louis Reymond. The chemical space project. *Accounts of Chemical Research*, 48(3):722–730, 2015.

[9] Melodie Christensen, Lars PE Yunker, Folarin Adedeji, Florian Häse, Loïc M Roch, Tobias Gensch, Gabriel dos Passos Gomes, Tara Zepel, Matthew S Sigman, Alán Aspuru-Guzik, et al. Data-science driven autonomous process optimization. *Communications Chemistry*, 4(1):1–12, 2021.

[10] Allan M. Jordan. Artificial intelligence in drug design—the storm before the calm? *ACS Medicinal Chemistry Letters*, 9(12):1150–1152, 2018.

[11] Alleyn T Plowright, Craig Johnstone, Jan Kihlberg, Jonas Pettersson, Graeme Robb, and Richard A Thompson. Hypothesis driven drug design: improving quality and effectiveness of the design-make-test-analyse cycle. *Drug discovery today*, 17(1-2):56–62, 2012.

[12] Daniel Merk, Lukas Friedrich, Francesca Grisoni, and Gisbert Schneider. De novo design of bioactive small molecules by artificial intelligence. *Molecular informatics*, 37(1-2):1700153, 2018.

[13] W Patrick Walters and Mark Murcko. Assessing the impact of generative ai on medicinal chemistry. *Nature biotechnology*, 38(2):143–145, 2020.

[14] A. Ng and Michael I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *NIPS*, 2001.

[15] Wenhao Gao, Tianfan Fu, Jimeng Sun, and Connor W Coley. Sample efficiency matters: A benchmark for practical molecular optimization. *arXiv preprint arXiv:2206.12411*, 2022.

[16] Thomas Blaschke, Josep Arús-Pous, Hongming Chen, Christian Margreitter, Christian Tyrchan, Ola Engkvist, Kostas Papadopoulos, and Atanas Patronov. Reinvent 2.0: an ai tool for de novo drug design. *Journal of Chemical Information and Modeling*, 60(12):5918–5922, 2020.

[17] Steven M. Mennen, Carolina Alhambra, C. Liana Allen, Mario Barberis, Simon Berritt, Thomas A. Brandt, Andrew D. Campbell, Jesús Castañón, Alan H. Cherney, Melodie Christensen, David B. Damon, J. Eugenio de Diego, Susana García-Cerrada, Pablo García-Losada, Rubén Haro, Jacob Janey, David C. Leitch, Ling Li, Fangfang Liu, Paul C. Lobben, David W. C. MacMillan, Javier Magano, Emma McInturff, Sebastien Monfette, Ronald J. Post, Danielle Schultz, Barbara J. Sitter, Jason M. Stevens, Iulia I. Strambeanu, Jack Twilton, Ke Wang, and Matthew A. Zajac. The evolution of high-throughput experimentation in pharmaceutical development and perspectives on the future. *Organic Process Research & Development*, 23(6):1213–1242, 2019.

[18] Favour Danladi Makurvet. Biologics vs. small molecules: Drug costs and patient access. *Medicine in Drug Discovery*, 9:100075, 2021.

[19] Jungseog Kang, Chien-Hsiang Hsu, Qi Wu, Shanshan Liu, Adam D Coster, Bruce A Posner, Steven J Altschuler, and Lani F Wu. Improving drug discovery with high-content phenotypic screens by systematic selection of reporter cell lines. *Nature biotechnology*, 34(1):70–77, 2016.

[20] JP Hughes, S Rees, SB Kalindjian, and KL Philpott. Principles of early drug discovery. *British Journal of Pharmacology*, 162(6):1239–1249, 2011.

[21] Independent High-Level Expert Group on Artificial Intelligence. Ethics guidelines for trustworthy ai. *European Commission*, 2019.

[22] Xiaochu Tong, Xiaohong Liu, Xiaoqin Tan, Xutong Li, Jiaxin Jiang, Zhaoping Xiong, Tingyang Xu, Hualiang Jiang, Nan Qiao, and Mingyue Zheng. Generative models for de novo drug design. *Journal of Medicinal Chemistry*, 64(19):14011–14027, 2021.

[23] Woosung Jeon and Dongsup Kim. Autonomous molecule generation using reinforcement learning and docking to develop potential novel inhibitors. *Scientific Reports*, 10(1):22104, December 2020.

[24] Jessica Vamathevan, Dominic Clark, Paul Czodrowski, Ian Dunham, Edgardo Ferran, George Lee, Bin Li, Anant Madabhushi, Parantu Shah, Michaela Spitzer, et al. Applications of machine learning in drug discovery and development. *Nature reviews Drug discovery*, 18(6):463–477, 2019.

[25] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[26] Shweta Bhatt. Reinforcement learning 101, Learn the essentials of Reinforcement Learning!, 2018. `https://towardsdatascience.com/reinforcement-learning-101-e24b50e1d292`, Last accessed on 2022-02-21.

[27] Fabio Pardo, Arash Tavakoli, Vitaly Levdik, and Petar Kormushev. Time limits in reinforcement learning. In *International Conference on Machine Learning*, pages 4045–4054. PMLR, 2018.

[28] Quyet Nguyen, Noel Teku, and Tamal Bose. Epsilon greedy strategy for hyper parameters tuning of a neural network equalizer. In *2021 12th International Symposium on Image and Signal Processing and Analysis (ISPA)*, pages 209–212. IEEE, 2021.

[29] Aakash Maroti. Rbed: Reward based epsilon decay. *arXiv preprint arXiv:1910.13701*, 2019.

[30] Ian Osband, Benjamin Van Roy, Daniel J Russo, Zheng Wen, et al. Deep exploration via randomized value functions. *J. Mach. Learn. Res.*, 20(124):1–62, 2019.

[31] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado Van Hasselt, and David Silver. Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933*, 2018.

[32] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[33] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

[34] Yuxi Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

[35] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G Bellemare, Joelle Pineau, et al. An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*, 11(3-4):219–354, 2018.

[36] Ankit Choudhary. A hands-on introduction to deep q-learning using openai gym in python. `https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/`. Accessed: 2022-04-07.

[37] Shangtong Zhang and Richard S Sutton. A deeper look at experience replay. *arXiv preprint arXiv:1712.01275*, 2017.

[38] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[39] Hado Hasselt. Double q-learning. *Advances in neural information processing systems*, 23, 2010.

[40] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[41] Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl ^{2}: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

[42] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[43] Léon Bottou. Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*, pages 421–436. Springer, 2012.

[44] Zijun Zhang. Improved adam optimizer for deep neural networks. In *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, pages 1–2. Ieee, 2018.

[45] David Weininger. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of chemical information and computer sciences*, 28(1):31–36, 1988.

[46] William J Wiswesser. 107 years of line-formula notations (1861-1968). *Journal of Chemical Documentation*, 8(3):146–150, 1968.

[47] John J Vollmer. Wiswesser line notation: an introduction. *Journal of Chemical Education*, 60(3):192, 1983.

[48] Esben Jannik Bjerrum. Smiles enumeration as data augmentation for neural network modeling of molecules. *arXiv preprint arXiv:1703.07076*, 2017.

[49] Noel M O'Boyle. Towards a universal smiles representation-a standard method to generate canonical smiles based on the inchi. *Journal of cheminformatics*, 4(1):1–14, 2012.

[50] James L Melville, Edmund K Burke, and Jonathan D Hirst. Machine learning in virtual screening. *Combinatorial chemistry & high throughput screening*, 12(4):332–343, 2009.

[51] Adrià Cereto-Massagué, María José Ojeda, Cristina Valls, Miquel Mulero, Santiago Garcia-Vallvé, and Gerard Pujadas. Molecular fingerprint similarity search in virtual screening. *Methods*, 71:58–63, 2015.

[52] Ulrich Rester. From virtuality to reality-virtual screening in lead discovery and

lead optimization: a medicinal chemistry perspective. *Current opinion in drug discovery & development*, 11(4):559–568, 2008.

[53] Jérôme Hert, Peter Willett, David J Wilton, Pierre Acklin, Kamal Azzaoui, Edgar Jacoby, and Ansgar Schuffenhauer. New methods for ligand-based virtual screening: use of data fusion and machine learning to enhance the effectiveness of similarity searching. *Journal of chemical information and modeling*, 46(2):462–470, 2006.

[54] David Rogers and Mathew Hahn. Extended-connectivity fingerprints. *Journal of chemical information and modeling*, 50(5):742–754, 2010.

[55] Robert C Glen, Andreas Bender, Catrin H Arnby, Lars Carlsson, Scott Boyer, and James Smith. Circular fingerprints: flexible molecular descriptors with applications from physical chemistry to adme. *IDrugs*, 9(3):199, 2006.

[56] Harry L Morgan. The generation of a unique machine description for chemical structures-a technique developed at chemical abstracts service. *Journal of chemical documentation*, 5(2):107–113, 1965.

[57] Thomas J Jech, Thomas Jech, Thomas J Jech, Great Britain Mathematician, Thomas J Jech, and Grande-Bretagne Mathématicien. *Set theory*, volume 14. Springer, 2003.

[58] Suphakit Niwattanakul, Jatsada Singthongchai, Ekkachai Naenudorn, and Supachanun Wanapu. Using of jaccard coefficient for keywords similarity. In *Proceedings of the international multiconference of engineers and computer scientists*, volume 1, pages 380–384, 2013.

[59] Sunyoung Kwon, Ho Bae, Jeonghee Jo, and Sungroh Yoon. Comprehensive ensemble in qsar prediction for drug discovery. *BMC bioinformatics*, 20(1):1–12, 2019.

[60] Kunal Roy, Supratik Kar, and Rudra Narayan Das. *A primer on QSAR/QSPR modeling: fundamental concepts*. Springer, 2015.

[61] Jie Zhang, Rocío Mercado, Ola Engkvist, and Hongming Chen. Comparative study of deep generative models on chemical space coverage. *Journal of Chemical Information and Modeling*, 61(6):2572–2581, 2021.

[62] Anna Gaulton, Anne Hersey, Michał Nowotka, A Patricia Bento, Jon Chambers, David Mendez, Prudence Mutowo, Francis Atkinson, Louisa J Bellis, Elena Cibrián-Uhalte, et al. The chembl database in 2017. *Nucleic acids research*, 45(D1):D945–D954, 2017.

[63] Nathan Brown, Marco Fiscato, Marwin HS Segler, and Alain C Vaucher. Guacamol: benchmarking models for de novo molecular design. *Journal of chemical information and modeling*, 59(3):1096–1108, 2019.

[64] Thomas Blaschke, Ola Engkvist, Jürgen Bajorath, and Hongming Chen. Memory-assisted reinforcement learning for diverse molecular de novo design. *Journal of cheminformatics*, 12(1):1–17, 2020.

[65] Favour Danladi Makurvet. Biologics vs. small molecules: Drug costs and patient access. *Medicine in Drug Discovery*, 9:100075, 2021.

[66] Thomas Morrow and Linda Hull Felcone. Defining the difference: what makes biologics unique. *Biotechnology healthcare*, 1(4):24, 2004.

[67] Alexander Tropsha. Best practices for qsar model development, validation, and exploitation. *Molecular informatics*, 29(6-7):476–488, 2010.

[68] Oliver Kramer. Scikit-learn. In *Machine learning for evolution strategies*, pages 45–53. Springer, 2016.

[69] Michel F Sanner et al. Python: a programming language for software integration and development. *J Mol Graph Model*, 17(1):57–61, 1999.

[70] Nikhil Ketkar and Jojo Moolayil. Introduction to pytorch. In *Deep learning with python*, pages 27–91. Springer, 2021.

[71] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.

[72] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[73] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. *Advances in neural information processing systems*, 29, 2016.