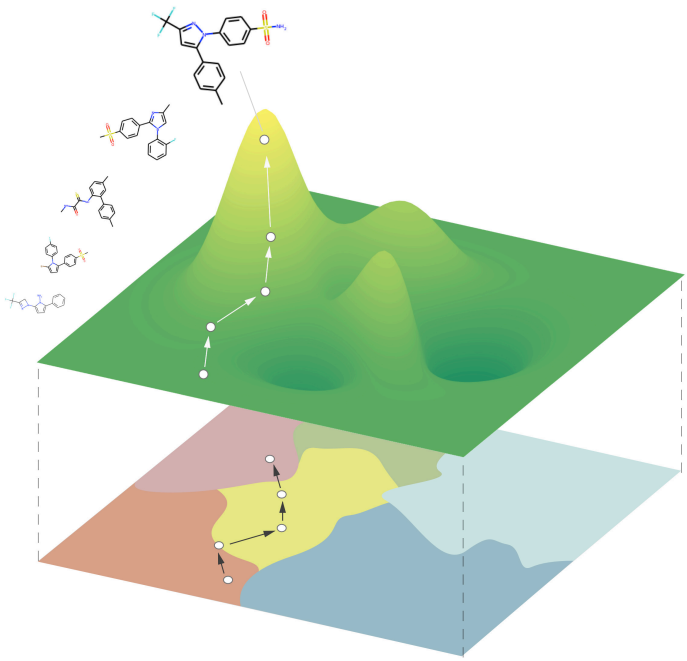


Comparison of State-of-the-art Algorithms for de novo Drug Design

Master's thesis in Computer Science and Engineering

Sebastian Nilsson
Jonathan Sundkvist

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Master's thesis 2020

Comparison of State-of-the-art Algorithms for de novo Drug Design

SEBASTIAN NILSSON
JONATHAN SUNDKVIST

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden 2020

Comparison of State-of-the-art Algorithms for de novo Drug Design

SEBASTIAN NILSSON
JONATHAN SUNDKVIST

Supervisor: Yinan Yu, Department of Computer Science and Engineering
Advisor: Atanas Patronov, AstraZeneca
Examiner: Graham Kemp, Department of Computer Science and Engineering

Master's Thesis 2020
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Visualization of optimization in chemical space

Typeset in LaTeX
Gothenburg, Sweden 2020

Comparison of State-of-the-art Algorithms for de novo Drug Design

SEBASTIAN NILSSON
JONATHAN SUNDKVIST
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

During the last decade, the application of deep learning in de novo drug design has increased. By employing generative models, in combination with suitable optimization algorithms, the chemical space can be explored to generate new and useful molecular compounds. Multiple models have been published for this purpose. While they all report promising results on various optimization tasks, there is a lack of continuity regarding what tasks on which the models are demonstrated. This makes a comparison of the models unfeasible. In this thesis, we provide a comprehensive evaluation and comparison of five state-of-the-art algorithms for de novo drug design. To this end, the selected models have been submitted to two experiments that evaluate their ability to generate and optimize molecules on a variety of different optimization tasks. The reported results show that a generative language model displays the best performance, both in terms of exploring and exploiting the chemical space. An application of particle swarm optimization on a continuous representation of the chemical space also shows promise, and we suggest further research on more elaborate usages of this method.

Keywords: Drug design, deep learning, generative model, optimization, variational autoencoder, language modelling, comparison.

Acknowledgements

We would like to extend a big thanks to our supervisor and advisors at AstraZeneca, Atanas Patronov, Kostas Papadopoulos and Ola Engkvist. Your guidance and helpfulness have been vital for this project. To the whole Molecular AI group: being around such talented and friendly people has been an inspiration and privilege. Special thanks to Panagiotis Kotsias for going the extra mile to ensure that our time with the group has been a great experience.

Many thanks to our examiner at Chalmers, Graham Kemp. And finally, to our academic supervisor, Yinan Yu, your sharp observations and ability to always answer any question has been of great help throughout the project, thank you.

Sebastian Nilsson, Jonathan Sundkvist, Gothenburg, June 2020

Contents

List of Figures

List of Tables

1
Introduction

In the field of drug discovery, one of the challenges biochemists face is the task of finding novel molecules with specific desired properties. Various approximations have been made regarding the number of potentially drug-like compounds, spanning from 10^{23} to 10^{160}. Out of these, *only* 10^8 have been successfully synthesised [1][2]. Given that the search space for this endeavour is enormous, multiple methods for screening the virtual molecular space have been developed in recent years to streamline the process [3][4]. The application of different Neural Network (NN) architectures and a plethora of Machine Learning (ML) training techniques have been employed in the race to find more efficient methods for generating and finding meaningful, novel, drug candidates.

In this introductory chapter we start by motivating the project by providing an overview of the drug development process. In Section 1.2, we give a brief review of deep learning in de novo drug design. Section 1.3 explains how molecular data can be represented as one-dimensional strings. In Sections 1.4 and 1.5 we define the objective and what the thesis aims to contribute to the field of drug design. The chapter is concluded with a description of the thesis outline.

1.1 Drug discovery and design

The development of new drugs is a complex, time consuming and expensive enterprise [5][6]. The process begins with identifying a target, commonly a protein, on which the new drug will act. Next, one must identify molecular compounds that bind to the selected target. Finding such a compound is referred to as discovering a *hit*. In addition to being able to interact with a target, a hit should also demonstrate other crucial properties such as synthesizability. Molecules that satisfy these requisites are named *leads* and are submitted to *lead optimization*, where one aims to maximise properties such as potency and affinity while minimizing toxicity. This can be achieved by various techniques where the lead is manipulated, for example, by removing certain substructures or imposing a scaffold known to interact with the designated target. This process is done iteratively and is often referred to as *de novo* drug design. These first steps in the drug development pipeline usually span over 4-5 years and assuming good results there are still clinical trials to be conducted before a drug is allowed to be distributed to the market.

A study by *Paul et al.*[6] approximates that *one* successfully commercialised drug

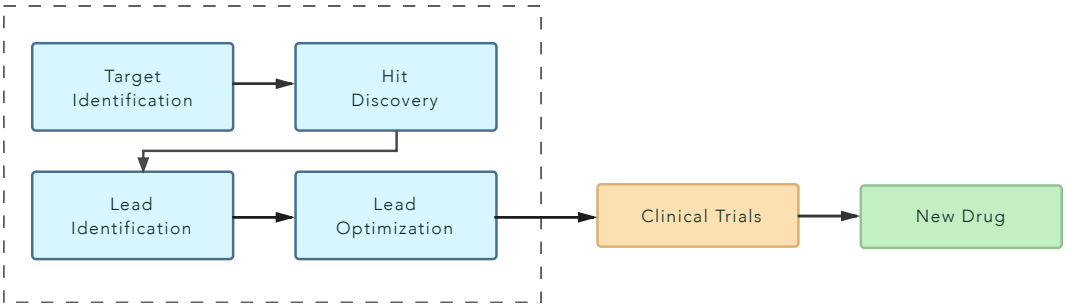


Figure 1.1: The different phases of drug development. The blue boxes correspond to the pre-clinical stage.

has been developed for 13.5 years, on average. They also report that only 4% of the compounds identified as hits in the first stage of the process get clinically approved in the end. For leads that are submitted to clinical trials, the corresponding success rate is 8%. When accounting for the whole pipeline from multiple research projects being conducted to a single new drug reaching the market, the total cost amounts to approximately $1.8 billion, whereas 33% is attributed to the steps before clinical trials. This poses a challenge, and opportunity, for the industry to find new, more efficient, methods for finding (and discarding) potential drug candidates early in the process.

1.2 Deep Learning in de novo Drug Design

During the last decade, deep learning (DL), a subclass of ML, has been successfully applied in disciplines such as image recognition, natural language processing, and self-driving cars [7]. Following the emergence of huge virtual libraries of molecular data, there has in recent years also been a surge of DL applications in the field of drug discovery. These libraries, containing huge numbers of known chemical compounds, make it possible to train DL architectures to learn some internal representation of the chemical space. This space, in turn, enables purpose-driven exploration and optimization where novel compounds of interest can be identified as hits and optimized given some predefined criteria. By finding efficient and sophisticated strategies for discovery and optimization using DL, one can potentially not only cut time and costs in the drug design process but also decrease the percentage of hits and leads that fail further down the pipeline.

Today there is a wide variation of DL architectures being employed in de novo drug design and discovery. The models often vary in multiple aspects, for example, what molecular representation format they operate on, optimization strategy and NN architecture. By surveying the literature, one finds that most methods report promising results on some selected task. However, comparing different models is non-trivial since these tasks are not necessarily the same or performed under the same premises. Assuming two separate models report some results given the same

optimization objective, one cannot merely rely on the results by themselves. One also has to take into account parameters such as what prior knowledge the models had (what data they had been trained on) or, if time efficiency is crucial, what hardware has been used. Therefore, in order to draw informed conclusions when comparing various models, one has to submit them to evaluations conducted under the same conditions. Such a survey could be valuable for the field by providing reliable and comparative data given a selection of promising models. This data can be used, for example, in decision-making processes when selecting a DL technique for a drug design task or as a stepping stone and baseline for future research.

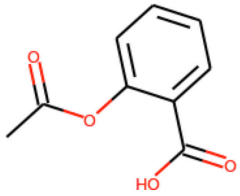
1.3 Molecular Data Representations

There are multiple ways to represent a molecule. In a non-data science context, the most common one is perhaps the *molecular formula*. A molecular formula is a text representation where every atom in the molecule is represented by its atomic symbol along with a subscript, denoting the count of that particular atom. An example is the molecular formula for water, H_2O, containing two hydrogen atoms and one oxygen atom. In order to provide structural information, one can also represent molecules with their *structural formula*, a graphical representation where the bonds are shown between the atoms.

To make molecular data more suitable for machine learning, one has to represent it in a way such that a program can easily read the data. Simplified molecular-input line-entry system (SMILES) [8] is a common notation for chemical compounds that represent 2-dimensional molecular graphs in one dimension. A SMILES is an ASCII string containing characters such as atom types (e.g. `'Br'`, `'C'`), double bonds (`'='`), ring openings (`'('`) and closings (`')'`), conformed to a predefined grammar. Benzene, as an example, has the chemical formula C_6H_6 and the corresponding SMILES is denoted `C1=CC=CC=C1` (hydrogen atoms can often be omitted [9]). A molecule may be represented using multiple SMILES, which in some cases might be suboptimal, for example, when one wants to guarantee uniqueness of entries in a dataset. On such occasions, one can use *canonical SMILES*, where every molecular compound is represented by a unique SMILES.

An extension of SMILES is the more flexible SMARTS notation[1]. While SMILES mainly uses symbols corresponding to atoms and bonds, SMARTS also provides symbols for specifying additional properties such as atom charge, ring size and chirality (a property related to molecule asymmetry). Furthermore, it allows for more general molecular representations in the sense that one may express a position in a molecule using, for example, `'[C,N]'`, indicating that either a `'C'` or an `'N'` can be used. This flexibility makes SMARTS strings suitable when searching for substructures in compounds.

[1] https://www.daylight.com/dayhtml/doc/theory/theory.smarts.html



CC(=O)OC1=CC=CC=C1C(=O)O

Figure 1.2: A molecular graph of Aspirin together with its SMILES string representation.

1.4 Objective

In this thesis work, we will evaluate and compare five state-of-the-art optimization algorithms for de novo drug design. The selected algorithms make use of different representations of the chemical space and utilize different strategies to optimize the generated molecules. This variation of methods indicates that there is yet no clear consensus as to which approach is the optimal one. By putting the algorithms side-by-side in a collection of benchmark tests, we aim to provide comprehensive data on their performances in relevant metrics such as their ability to generate molecules with high objective value, novelty, and uniqueness. The presented results may be used as guidance and a stepping stone for future research.

1.5 New contributions

This work aims to provide the field of *de novo* drug design with a rigorous evaluation and comparison of different strategies for molecular generation and optimization. Today there are numerous publications of tools and algorithms designed for this purpose. While these often include extensive benchmark and test results of relevant objectives, there is a noticeable lack of continuity and unanimity regarding the selection of benchmarks.

The main contribution of this work is the integration of five state-of-the art algorithms with a standardized benchmarking framework. By evaluating the algorithms on the same series of optimization task we enable a comparative analysis using a common denominator. We also hope this will encourage the field to use available frameworks for benchmarking in future publications, to facilitate easy and fair comparisons between algorithms.

The second contribution is a comparison of the algorithms on a test case that emulates an industry-like optimization task. This complements the results of the benchmarking evaluations with data of concrete, practical, value for the industry.

1.6 Thesis outline

This introductory chapter has provided an overview of the drug development process, and by highlighting the complexity of the enterprise, the advantages of employing deep learning to streamline the process have been established. We have also motivated the need for a comparative study on existing optimization methods.

In Chapter 2, we review deep learning architectures that are employed by the models covered in this work. Chapter 3 introduces three optimization methods that are utilized by the models to guide the generative process towards desirable areas of the chemical space. The five selected models are introduced in Chapter 4. In Chapter 5, we present a benchmarking framework and a multi-property function that is used in two experiments, in which the models are evaluated. In the subsequent Chapter 6 we provide details regarding these experiments followed by the results, in Chapter 7. Based on the reported results, the thesis concludes with a discussion and some final words in Chapter 8 and 9.

2

Generative Deep Learning

The state-of-the-art deep learning models evaluated in this report make use of generative models to find novel molecules. Generative modelling is a branch of deep learning, where models are trained to estimate a probability distribution [10]. This distribution can be sampled to find new data points, not previously seen in the training dataset. The models described in this report make use of three types of neural network architectures for data generation: recurrent neural networks, variational autoencoders and generative adversarial networks. Before we review these architectures we will introduce key concepts from deep learning in general.

2.1 Introduction to Artificial Neural Networks

Artificial Neural Networks (ANN) are computational systems that learn to perform tasks without being explicitly programmed with task-specific rules. In practice, they are used as universal function approximators, aiming to approximate an unknown function f. For example, they can be used to approximate a classifier $y = f(\mathbf{x})$ that maps input $\mathbf{x}$ to a category y.

The ANN, similar to the neural networks found in the brains of animals, is a collection of connected units called neurons. It can be represented as a graph, where the nodes are neurons, and the edges are the connections between them. Similar to biological neural networks, the artificial neurons transmit signals between each other. In the ANN, the signals are represented by scalar values.

The nodes in the graph are connected by directed edges, where each edge has a weight assigned to it. The signal sent from one neuron to another is multiplied with the weight of their shared edge. The receiving neuron processes the input signal by passing it through a nonlinear *activation function* and then sends it forward in the network. The neurons of a network are typically arranged in *layers*. The neurons in a layer are connected only to the neurons in the preceding and the immediately following layers. An ANN with this arrangement is called a Feedforward Neural Network (FFNN). We present a simple graph of an ANN in figure 2.1.

2.1.1 Training a Feedforward Neural Network

A *loss function*, $\mathcal{L}$, measures the performance of a neural network. For example, let $\mathcal{X} = \{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(m)}\}$ be a set of m inputs that a FFNN should map to one of

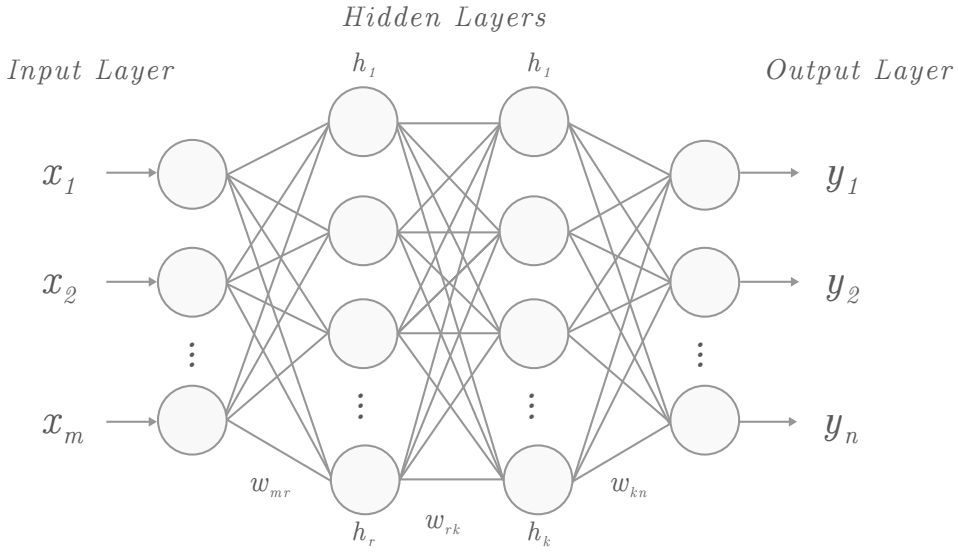


Figure 2.1: Computational graph of an artificial neural network. The circles represent neurons, and the edges between them represent a signal in the form of a scalar value. Each neuron has input to its left and produces an output to its right. The output is calculated by passing the sum of the input signals through an activation function. The edges are weighted, meaning that the signal from a neuron to another is multiplied with the weight of their connecting edge.

two classes $y \in \{0, 1\}$. Let y be the correct class of input $\mathbf{x}$ and let $\hat{y}$ be the class predicted by the network. For this task, the squared number misclassifications can be used as a loss function:

$$\mathcal{L} = \sum_{i=1}^{m} (\hat{y}_i - y_i)^2 \tag{2.1}$$

Training the neural networks means finding the weights of the edges in the graph such that the loss function is minimized. Using the chain rule from calculus, each weight in the network can be updated according to its contribution to the error. Let w_{ij} be the weight of the i^{th} edge in layer j, the change of a weight can be computed by:

$$\Delta w_{ij} = -\frac{\partial \mathcal{L}}{\partial w_{ij}} \tag{2.2}$$

The partial derivative is negative to ensure that the weights are always changed such that $\mathcal{L}$ decreases. The weight is then updated by:

$$w_{ij} = w_{ij} + \eta \Delta w_{ij} \tag{2.3}$$

where $\eta > 0$ is the *learning rate*, a scalar commonly in the range from $1e^{-5}$ to $1e^{-3}$. The process of minimizing the loss in this way is called *gradient descent*. By thinking of the loss as a function that depends on the weights of the network, we are moving

the value of the loss function in the direction of its negative gradient. The learning rate adjusts how big a step it takes.

In short, training the network means showing it samples from a *training dataset* and then updating its weights such that the loss function is minimized. In practice, the training data is divided into *batches* of size n. The loss is calculated and the weights updated for each batch. The reason for this is similar to the learning rate; the batch size adjusts the step size of our minimization. In addition to using training examples, it is common to use a *validation dataset* for which the *validation loss* is calculated to measure how well the network performs on unseen data.

The number of layers, the number of neurons in each layer, activation function, loss function, learning rate and batch size are all examples of *hyperparameters* of a neural network.

2.2 Recurrent Neural Networks

Recurrent Neural Networks [11] (RNNs) are a class of neural networks specialized for processing sequential data, such as text, audio, stock market data, or other data sources in which the ordering carries essential information. By learning to recognize patterns in sequences, RNNs have shown to be a proficient tool for tasks such as machine translation [12], speech recognition [13] and e-commerce product recommendation [14], amongst others.

What differentiates an RNN from other neural networks, is that an RNN assumes dependency between inputs. Keeping track of previous input is essential when applying machine learning to sequential data. An example can be found in machine translation, where words at the end of a sentence may have a different meaning depending on preceding words.

A typical example of an RNN produces an output for each element in an input sequence $\mathcal{X} = \{\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(T)}\}$. We denote the input at time step t as $\mathbf{x}^{(t)}$. The RNN has an internal memory called the hidden state, $\mathbf{h}^{(t)}$, which is updated at each time step using the current input $\mathbf{x}^{(t)}$ and the hidden state from the previous time step, $h^{(t-1)}$. The hidden state is then used to calculate the output $o^{(t)}$. Two functions express the update of the hidden state and the output calculations:

$$\mathbf{h}^{(t)} = f(\mathbf{x}^{(t)}, \mathbf{h}^{(t-1)}) \tag{2.4}$$

$$o^{(t)} = g(\mathbf{h}^{(t)}) \tag{2.5}$$

Since the RNN is a neural network, both f and g are functions that are learned by the network during training. The functions are parameterized by weight matrices

and an activation function in the following way:

$$\mathbf{a}^{(t)} = \mathbf{b} + \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)} \tag{2.6}$$

$$\mathbf{h}^{(t)} = \sigma(\mathbf{a}^{(t)}) \tag{2.7}$$

$$o^{(t)} = \mathbf{c} + \mathbf{V}\mathbf{h}^{(t)} \tag{2.8}$$

where $\mathbf{U}$, $\mathbf{V}$ and $\mathbf{W}$ are weight matrices, $\mathbf{b}$ and $\mathbf{c}$ are biases and σ is a nonlinear activation function. A computational graph of the RNN is illustrated in figure 2.2.

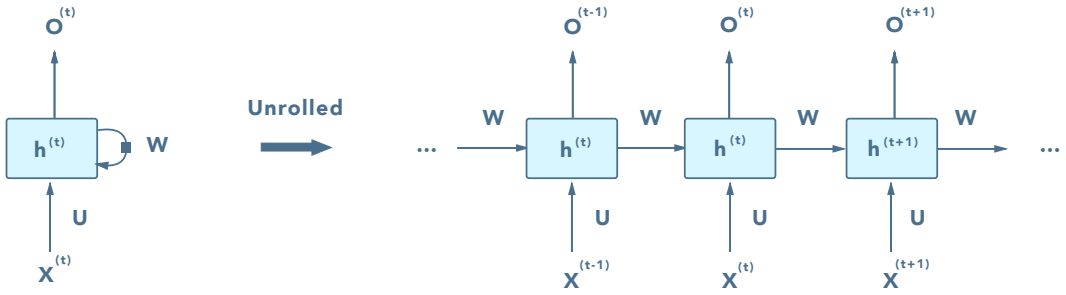


Figure 2.2: Computational Graph of the recurrent neural network, represented in a compact (left) and in a unrolled version (right). The black box in the left figure represents a delay of one timestep. The hidden state at time t, $\mathbf{h}^{(t)}$, is used to produce the output at time t, $\mathbf{o}^{(t)}$. The hidden state $\mathbf{a}^{(t)}$ is produced by a combination of the hidden state in the previous time step and the current input, $\mathbf{x}^t$.

In the context of generative models, RNNs can be trained for *sequence generation*[15] by processing real data sequences one step at a time and predicting what comes next. As a result, the network outputs a distribution that can be sampled in order to create sequences not previously seen in the training data. In addition, the sampled data can be fed back into the network as a base for its next prediction.

2.3 The Variational Autoencoder

An autoencoder [16] is an artificial neural network that attempts to copy its input to its output. Internally, an *encoder* learns to transform a dataset, $\mathcal{X} = \{\mathbf{x_1}, ..., \mathbf{x_n}\}$, into an encoded space $\mathcal{Z}$, also called latent space. From this encoding, which is typically a lower-dimensional representation of the original data, a *decoder* tries to reconstruct the input, and produces the network's output: $\hat{\mathbf{x}}$. The network is trained to compress and reconstruct the data such that the difference between input and output is as small as possible. The difference between input and output is called the reconstruction loss, $\mathcal{L}$, defined as:

$$\mathcal{L} = ||\mathbf{x} - \hat{\mathbf{x}}||^2 \tag{2.9}$$

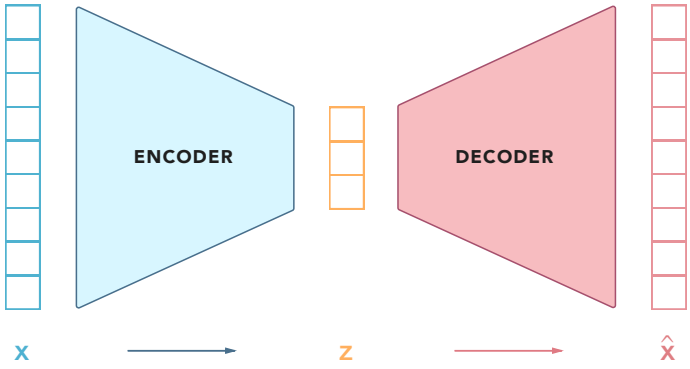


Figure 2.3: Overview of an autoencoder neural network architecture.

The generative aspect of the autoencoder is found in latent space $\mathcal{Z}$. The encoded vector $\mathbf{z}$ can be thought of as a coordinate in the latent space. In theory, by traversing the latent space and decoding the coordinates, new data points can be discovered, similar to the ones found in the training dataset. However, this puts several demands on the structure of the latent space. Using the autoencoder network described above does not guarantee that interpolation between latent points will produce meaningful output when decoded. In order to enforce a continuous structure, a probabilistic variant of the autoencoder can be used.

The variational autoencoder contains the same components found in a regular autoencoder and is also trained to minimize the reconstruction loss. However, instead of encoding an input $\mathbf{x}$ to a point $\mathbf{z}$ in the latent space, $\mathbf{x}$ is encoded as a distribution $p(\mathbf{z}|\mathbf{x})$. The distribution is sampled to attain a point in the latent space $\mathbf{z} \sim p(\mathbf{z}|\mathbf{x})$, which is passed to the decoder. In practice the input is encoded to a multivariate normal distribution, meaning that the decoder outputs a mean and covariance matrix for each $\mathbf{x}$.

The goal is for the latent space to represent the dataset continuously, making it possible to interpolate between data points. Furthermore, similar inputs should be found close together in the latent space. To achieve this, a regularization term is added to the loss to enforce that encoded distributions have a shape similar to standard normal distribution:

$$\mathcal{L} = ||\mathbf{x} - \hat{\mathbf{x}}||^2 + \mathbb{KL}[\mathcal{N}(\mu_{\mathbf{x}}, \sigma_{\mathbf{x}}) \,||\, \mathcal{N}(0, \mathbf{I})] \tag{2.10}$$

Where $\mathbb{KL}$ is the Kullback-Leibler divergence [17], measuring the difference between two distributions.

The variational autoencoder has been used as a tool for molecular design in several applications [18][19]. The latent space enables a continuous representation of molecules, which can be traversed to optimize both structure and chemical properties.

2.4 Generative Adversarial Networks

Generative Adversarial Networks [20] is an algorithm that uses two competing neural networks to generate new, previously unseen data. One neural network, the *generator*, creates new instances of data, while the *discriminator* neural network decides whether the data comes from the actual training dataset or not. The generator is trained to *fool* the discriminator by generating data that look increasingly similar to the training dataset. Simultaneously, the discriminator is trained to distinguish between the real data and the fake data from the generator.

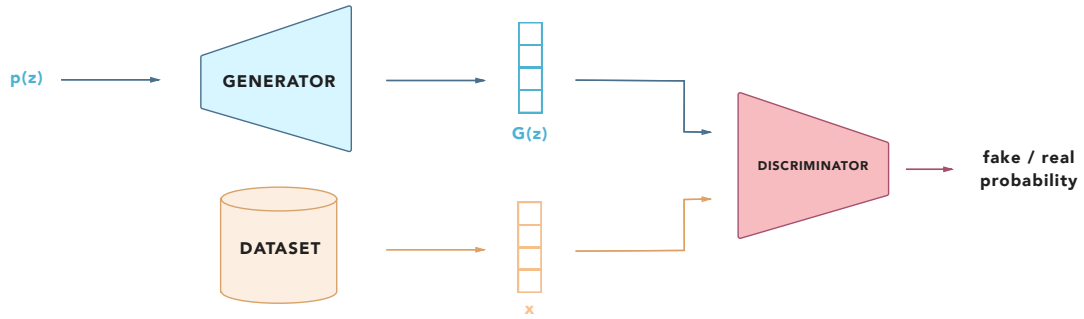


Figure 2.4: Overview of a generative adversarial network architecture

To create a new sample, the generator takes random noise as input. Let $\mathbf{z} \sim p_{\mathbf{z}}$ be a random noise vector and let $\mathcal{G}$ be the generator neural network which maps noise vectors to the training data space. Let $\mathcal{D}$ be the discriminator neural network, which takes in a sample $\mathbf{x}$ and outputs the probability (a scalar between 0 and 1) indicating if the sample is real. The following minimax game [21] describes the objective function of the network as a whole:

$$\min_{\mathcal{G}} \max_{\mathcal{D}} V(\mathcal{D}, \mathcal{G}) = \mathbb{E}_{\mathbf{x} \sim p_{data}}[\log \mathcal{D}(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_z}[\log(1 - \mathcal{D}(\mathcal{G}(\mathbf{z})))] \tag{2.11}$$

The discriminator is rewarded for making correct predictions, while the generator is rewarded for wrong predictions made by the discriminator.

In drug discovery, GANs have been used for generation of novel anti-cancer molecules [22], drug-target binding predictions [23] and for optimization of molecular properties [24].

3
Optimization

In a mathematical setting, optimization is the pursuit of an optimal element x^* such that a given *objective function* is minimised or maximised. In the case of maximisation, given a set $\mathcal{X}$, one can formulate this as the task of solving

$$x^* = \arg\max_{x \in \mathcal{X}} f(x) \tag{3.1}$$

There exist a multitude of various strategies to choose from when solving optimization problems, which one is preferable depends on the nature of the problem and the desired level of precision. However, a common problem for any method is to find a suitable trade-off between *exploration* and *exploitation* in the solution space. An optimization strategy that searches in an exploratory manner tends to query points in $\mathcal{X}$ with high uncertainty, while an exploitative strategy moves in the direction of areas in the search space that have already shown promise. A possible side-effect in the former case is that an optimum is not found within a reasonable time while the latter approach increases the risk of getting stuck in local optima.

In this chapter, we introduce three different optimization algorithms. In Section 3.1, we describe Bayesian Optimization where informed sampling approximates the underlying objective function in the search space. Section 3.2 describes the Particle Swarm Optimization algorithm that mimics the behaviour seen in insect swarms as a means to find an optimum. Lastly, in Section 3.3, we introduce Reinforcement Learning, a machine learning algorithm that aims to teach an agent a policy that will guide it in the search space such that a cumulative reward is maximized.

3.1 Bayesian Optimization

In cases where an unknown objective function f has multiple local optima and/or is computationally expensive to calculate, one might not be able to rely on local information, such as gradients, in an iterative manner and a large number of steps. In such cases a possible strategy is to perform less but more costly evaluations, and compensate by making use of the full optimization history to make as educated subsequent choices as possible as to where the global optimum is.

Bayesian Optimization (BO) is a technique based on Bayes' Theorem, where sampling is driven by accumulated observations throughout the optimization process. In a sequential manner, a prior probability distribution, that approximates the objective function, is updated as more observations are performed, enabling more

informed sampling decisions in the future steps. After some termination criteria are met, the output is the algorithm's final recommendation on where to find the optimum of f.

3.1.1 Bayes' Theorem

Bayes' theorem constitutes the foundation of the Bayesian optimization algorithm. Given events A and B along with the conditional events of A given B and vice versa, Bayes' theorem is summarised in the formula

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{3.2}$$

The theorem tells us that the *posterior* probability of some event A, given that we have observed event B, is conditioned under our *prior* knowledge of (possibly) related events. As an example, let A denote that a person has lung cancer and B that a person is a smoker. In this particular case, the prior probability, $P(A)$, of a person having lung cancer is known. The posterior probability, $P(A|B)$, is the probability of a person having lung cancer given that we *know* that the same person is a smoker. This can be calculated using our existing knowledge and observations.

By fixing B, one can highlight the proportionality of the posterior probability of an event A given event B to the probability of B given the prior likelihood of A multiplied with the probability of A.

$$P(A|B) \propto P(B|A)P(A) \tag{3.3}$$

This relation can be applied in the context of BO to describe how a prior distribution and an observation is used to create a posterior distribution over the space of objective functions.

3.1.2 Surrogate model

Any Bayesian scheme requires a prior distribution. This distribution represents the current belief regarding the space of potential objective functions and is constructed using a set of observations on f applied to a *surrogate model*. The purpose of a surrogate model is to approximate properties of the real objective function in order to guide the optimization process in either an exploratory or exploitative manner. There are numerous options when selecting a surrogate model. A common one is the *Gaussian processes* which also is the selected one in the BO algorithm evaluated in this thesis.

A Gaussian process (GP) is a modelling technique, and a generalization of a Gaussian distribution, that can be used in the context of BO to approximate the underlying objective function $f(\mathbf{x})$. While a Gaussian distribution is a distribution over a random variable, defined by its mean and variance, a GP is a distribution over

the space of possible functions, defined by its mean function $m(\mathbf{x})$ and covariance function $k(\mathbf{x}, \mathbf{x}')$, also known as the *kernel function* [25].

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')) \tag{3.4}$$

That is, when queried for $\mathbf{x}$, the GP will return the mean and variance of a normal distribution over the potential values at $\mathbf{x}$ rather than an actual scalar. The behaviour of the GP is determined by the choice of mean and covariance function. A common pair is the zero function and the squared exponential function as mean and covariance function respectively [26][27].

$$m(\mathbf{x}) = 0 \tag{3.5}$$

$$k(\mathbf{x}, \mathbf{x}') = \exp(-\frac{1}{2}||\mathbf{x} - \mathbf{x}'||^2) \tag{3.6}$$

Here we see that by investigating points close to each other, k will move towards 1, and towards 0 for points far away from each other. This indicates that neighbouring points influence each other to a greater extent than points far away from each other. Now, assuming $\mathcal{D}_{1:n}$ is a set of n observations on f such that

$$\mathcal{D}_{1:n} = \{f(\mathbf{x}_{1:n}), \mathbf{x}_{1:n}\} \tag{3.7}$$

we can define the *kernel matrix* $\mathbf{K}$ as

$$\mathbf{K} = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & k(\mathbf{x}_1, \mathbf{x}_2) & \dots & k(\mathbf{x}_1, \mathbf{x}_n) \\ k(\mathbf{x}_2, \mathbf{x}_1) & k(\mathbf{x}_2, \mathbf{x}_2) & \dots & k(\mathbf{x}_2, \mathbf{x}_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_1) & k(\mathbf{x}_n, \mathbf{x}_2) & \dots & k(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix} \tag{3.8}$$

In order to produce a posterior distribution, we assume that the next, arbitrary, point to sample is x_{n+1}. Thus, given that any linear combination of dimensions is Gaussian, we have that $f(x_{1:n})$ and $f(x_{n+1})$ are jointly Gaussian such that

$$\begin{bmatrix} f(\mathbf{x}_{1:n}) \\ f(\mathbf{x}_{n+1}) \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} \mathbf{K} & \mathbf{k}^T \\ \mathbf{k} & k(\mathbf{x}_{n+1}, \mathbf{x}_{n+1)} \end{bmatrix}\right) \tag{3.9}$$

with $\mathbf{k}$ defined as

$$\mathbf{k} = \begin{bmatrix} k(\mathbf{x}_{n+1}, \mathbf{x}_1) & \dots & k(\mathbf{x}_{n+1}, \mathbf{x}_n) \end{bmatrix} \tag{3.10}$$

The conditional probability $P(f(x_{n+1})|f(x_{1:n}))$ can now be expressed as the Gaussian distribution

$$P(f(\mathbf{x}_{n+1})|f(\mathbf{x}_{1:n})) \sim \mathcal{N}(\mu_n(\mathbf{x}_{n+1}), \sigma_n^2(\mathbf{x}_{n+1})) \tag{3.11}$$

where

$$\mu_n(\mathbf{x}_{n+1}) = \mathbf{k}^T \mathbf{K}^{-1} f(\mathbf{x}_{1:n}) \tag{3.12}$$

$$\sigma_n^2(\mathbf{x}_{n+1}) = k(\mathbf{x}_{n+1}, \mathbf{x}_{n+1}) - \mathbf{k}^T \mathbf{K}^{-1} \mathbf{k} \tag{3.13}$$

3.1.3 Acquisition function

In order to determine which point in the search space that should be sampled next, BO utilizes an acquisition function. Based on the current distribution of the surrogate model, the acquisition function is used to guide the optimization in either an exploratory or exploitative manner. The former strategy is carried out by sampling in areas of high uncertainty (high variance) while the latter by focusing on areas with high predictive values. To this end, the acquisition function has high objective values in such areas, and the selection of the next point to sample is decided by finding the point that maximizes the generic acquisition function $\alpha(.)$ as defined by equation 3.14.

$$\mathbf{x}_{n+1} = \underset{\mathbf{x}}{\operatorname{argmax}}\ \alpha(\mathbf{x}|f_{1:n}) \tag{3.14}$$

In figure 3.1, we provide a simple illustration of how an acquisition function is utilized in order to find the next point to sample, and how the underlying Gaussian distribution is updated.

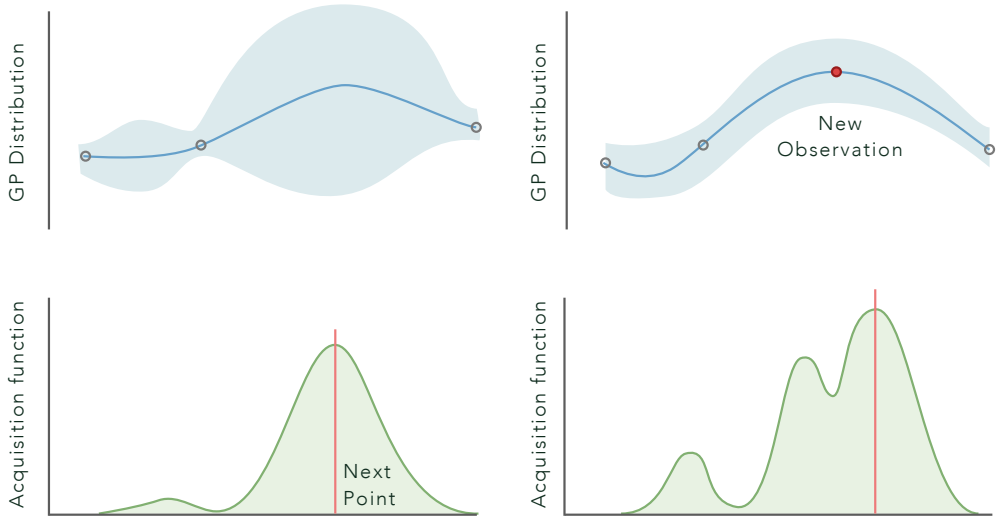


Figure 3.1: One-dimensional illustration of a single step in Bayesian Optimization. To the left is a GP distribution with three observations. Underneath the acquisition function is shown with a red line, indicating the location of optimum. This point is where the next observation is made. To the right, the updated GP distribution, and how the acquisition also has updated accordingly, is displayed.

There are multiple functions to choose from when selecting a suitable acquisition function. However, they should all be significantly computationally cheaper to evaluate than the actual objective function f. A common acquisition function is *expected improvement* (EI), which aims to find the point that is expected to increase the objective value of f as much as possible. The function is defined as

$$\mathbf{EI}(\mathbf{x}) = \mathbb{E}[\ \max\{0,\ f_{n+1}(\mathbf{x}) - f(\mathbf{x}^+)\}\ ] \tag{3.15}$$

where $f(\mathbf{x}^+)$ represents the best sampled value so far. Assuming a GP surrogate model one can evaluate the function analytically as [28][29]

$$\mathbf{EI}(\mathbf{x}) = \begin{cases} (\mu(\mathbf{x}) - f(\mathbf{x}^+))\Phi(Z) + \sigma(\mathbf{x})\phi(Z) & \text{if } \sigma(\mathbf{x}) > 0 \\ 0 & \text{if } \sigma(\mathbf{x}) = 0 \end{cases} \tag{3.16}$$

where Φ represent the cumulative distribution function, ϕ the probability density function and Z is defined as

$$Z = \frac{\mu(\mathbf{x}) - f(\mathbf{x}^+)}{\sigma(\mathbf{x})} \tag{3.17}$$

As alternatives to EI one can employ *upper confidence bound* [26], *probability of improvement* [30] or *entropy search* [31] as acquisition functions.

3.2 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a stochastic algorithm that emulates flock behaviour seen in birds and insects. It is an iterative method where the search for a global optimum is conducted using a population of solution candidates that traverse the search space in a swarm-like manner. The employed paradigm is inspired by the social sharing of information between members of flocks or swarms, that allow a seemingly choreographed behaviour regardless of the chaotic movement of each individual. This social sharing is one of the cornerstones of the algorithm where, in the same way birds flock to food, solution candidates converge to an optimum.

A PSO instance consists of a set of particles that together make up a swarm. A particle represents a possible solution to the optimization problem at hand, that is; a particle is a point in the search space. Given a D-dimensional search space a particle i is defined by three D-dimensional vectors; its position x_i, its velocity v_i and its best position so far x_i^{best}. Assuming an optimization has run for k iterations, the particle's best position *yet* is defined as

$$x_i^{best} = \text{argmax } f(x_i^k) \tag{3.18}$$

In addition to storing every separate particle's best position, a PSO instance keeps track of the best overall position of the swarm. There are two strategies one can employ for this cause; storing the best particle *historically* or the best particle at the *current* iteration. For the continuation of this thesis, the former will be assumed and the definition is expressed as

$$x_{swarm}^{best} = \text{argmax } f(x_i^{best}) \qquad i = 1, ..., n \tag{3.19}$$

The behaviour of a particle is dictated by both its individual history and the performance of the swarm as a whole. Furthermore, its movement is guided by a trade-off

between exploration and exploitation. These three components are summarised in the update step of the velocity of a particle i after k iterations

$$v_i^{k+1} \leftarrow wv_i^k + \underbrace{\varphi_1 r_1 (x_i^{best} - x_i^k)}_{cognitive\ component} + \underbrace{\varphi_2 r_2 (x_{swarm}^{best} - x_i^k)}_{social\ component} \tag{3.20}$$

The first term on the right-hand side is the component controlling the trade-off between exploration and exploitation. The constant w is called the *inertia weight* and is a scalar in between 0 and 1. It is multiplied with the current velocity of the particle and therefore low values will result in the particle "slowing down" while high values instruct the particle to maintain an exploratory behaviour.

The second term is often referred to as the *cognitive* component and can be interpreted as the particle's trust in its own performance. The difference between the particle's best position so far and the current position is multiplied with a uniformly distributed random variable $r_1 \in [0, 1]$ and an acceleration coefficient φ_1 that scales the magnitude of the cognitive component. This coefficient may vary or be set to a specific value throughout a run.

The third term is commonly called the *social* component and describes the particle's inclination to drift towards the best overall position of the swarm. Here the distance between the swarm's best position and the particle's current position is multiplied with uniformly distributed random variable $r_2 \in [0, 1]$ and coefficient φ_2 that has the same purpose, and may be manipulated under the same principle as φ_1.

After having updated the velocity of a particle, the final step is to update its position. This operation is performed by adding the velocity v_i^{k+1} to the current position x_i such that

$$x_i^{k+1} \leftarrow x_i^k + v_i^{k+1} \tag{3.21}$$

An optimization run is over when a predefined number of iterations has been reached or when some end condition has been satisfied. The algorithm should ideally be run multiple times due to the stochastic nature of it and the lack of certainty that the found optimum is the global optimum and not just a local one.

3.2.1 Hyperparameter settings

There are multiple possible hyperparameters to adjust when using PSO. The number of particles, the inertia weight and the two coefficients φ_1 and φ_2 all influence both the individual particles and the swarm as a unit. The optimal set of hyperparameter settings can vary depending on the objective function, the nature of the domain or the required precision. There is no consensus as to which single selection of hyperparameters is the best in a general sense, but rather there is a variation of proposed approaches to be found in the literature. In the following paragraphs we present a selection of examples on such approaches.

The inertia weight, dictating the momentum of a particle, is usually initiated around 0.9 to encourage exploration in the early stages of the optimization [32]. A typical utilization of the inertia weight is to let it decrease over time to promote exploitation of promising areas in the later stages of a run. This decaying process can be carried out using both linear and non-linear functions that decrease the inertia weight from 0.9 to 0.4 [33][34][35]. There has also been a study on scenarios where the application of a randomized inertia weight has been successfully applied [36].

The cognitive and social acceleration coefficients φ_1 and φ_2 may be adjusted to promote a certain "mindset" of the particles. With $\varphi_1 > \varphi_2$ the particles are more self-confident and weigh their personal history more than the swarm history. With $\varphi_1 < \varphi_2$ the opposite holds; the particles drift towards the best point found by the swarm rather than exploiting their own best observations. As with the inertia weight, there are a variation of different approaches to selecting these parameters. Setting the coefficients such that $\varphi_1+\varphi_2 \approx 4$ is one of the more common combination. In line with the gradual change towards an exploitative behaviour often implemented in the inertia weight, approaches with dynamic values have been proposed for the acceleration coefficients. In a method called PSO-TVAC (PSO with Time-Varying Acceleration Coefficients), the cognitive coefficient is decreased from 2.5 to 0.5 during optimization, while the opposite holds for the social coefficient [37].

3.3 Reinforcement learning

Reinforcement learning (RL) [38] is a learning paradigm describing the control of a system such that a long-term objective is optimized. On a high level, a reinforcement learning system is comprised of five main subelements: An *agent*, a *policy*, a *reward signal*, a *value function* and a model of the *environment*. The agent is the learner and decision-maker that interacts with the environment. The environment has a state which the agent can alter by performing an *action*. A policy defines what action an agent will take at a particular state. It represents the core of the algorithm since it decides the behaviour of the agent. A policy can be straightforward, mapping a state to an action from a lookup-table, or it can require extensive computation. In general, a policy is stochastic and returns a probability for each available action.

The reward signal is the immediate feedback an agent receives after selecting an action. It is a signal that may alter the policy to choose one action over another. The reward received from an action is unknown to the agent but can be learned over time by trial and error. The value function is the total accumulated reward, and it is the overall goal for the algorithm to maximize this value function. The challenge for the algorithm is to find a balance between choosing what is most beneficial now, to what will yield the highest reward in the long run.

3.3.1 Reinforcement learning as a finite Markov decision process

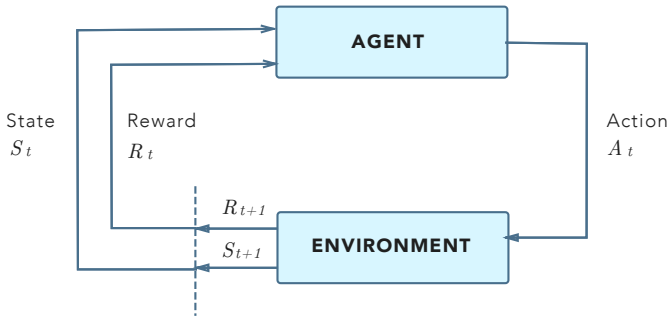


Figure 3.2: Interaction between agent and environment in a Markov decision process.

Formally, RL can be defined as a *finite Markov decision process* (MDP). MDP is a formalization of sequential decision making, in which an action influences current and future states. In the MDP framework, the system being controlled can be described by three sets: a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$ and a set of numerical rewards $\mathcal{R} \subset \mathbb{R}$. The finiteness of the MDP comes from the assumption that the sets $\mathcal{S}, \mathcal{A}$ and $\mathcal{R}$ have a finite number of elements.

The interactions between the agent and the environment, visualized in figure 3.2, are carried out in discrete time steps $t = 0, 1, 2, \ldots, T$. At each time step, the agent reads the current state, $S_t \in \mathcal{S}$, from the environment, which it uses to select its next action $A_t \in \mathcal{A}(s)$. In the next time step, the agent receives a new state S_{t+1} and a reward $R_{t+1} \in \mathcal{R}$ based on the previous action. Repeating this process for each time step creates a *trajectory*, τ, starting from an initial state S_0:

$$\tau = S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots \tag{3.22}$$

The length of the trajectory depends on what type of task is being optimized. If the task does not have a definite ending, it is called a *continuing* task. If the task has a subset of states which terminates the algorithm, it is called *episodic*. For instance, when an agent is playing a game that ends when a player wins or loses. A finite trajectory is called an *episode*.

The states, actions and rewards are all random variables from the discrete probability distributions $\mathcal{S}, \mathcal{A}$ and $\mathcal{R}$ respectively. The dynamics of the MDP can be described by a probabilistic transition function, p:

$$p(s|s', a) = Pr\{S_t = s'|S_{t-1} = s, A_{t-1} = a\} \tag{3.23}$$

for all $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}(s)$. The transition function p returns a probability of ending up in state s given that action a was carried out in the previous state s'. The reward received for carrying out action a in state s can be express as a function: $R(a, s) \in \mathcal{R}$

3.3.2 Optimal Policy Approximation

In the MDP framework, a policy, π, is defined as a function returning the probability of selecting action a given the state s:

$$\pi(A_t = a|S_t = s) \tag{3.24}$$

Since the policy is probabilistic, the trajectory τ of length T created by a policy is a random variable defined as:

$$\pi(\tau) = \pi(s_0, a_0, \ldots, s_T, a_T) = p(s_0) \prod_{t=1}^{T} \pi(a_t, s_t) p(s_{t+1}|s_t, a_t) \tag{3.25}$$

As mentioned previously, the overall goal is for the agent to accumulate as much reward as possible. To achieve this, the agent will need to learn a policy that on average generates the trajectories with the greatest reward. In the context of artificial neural networks, learning a function means parameterizing it using the connection weights of the network. Thus, the agent will learn a parameterized policy, π_θ, where θ represents the network weights.

Let $r(\tau)$ be the total accumulated reward given a trajectory τ. The cost function for a policy parameterized by θ is given by

$$\mathbf{J}(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[r(\tau)] = \int \pi_\theta(\tau) r(\tau) d\tau \tag{3.26}$$

The gradient of the cost function is thus given by

$$\begin{aligned} \nabla_\theta \mathbf{J}(\theta) &= \int \nabla_\theta \pi_\theta(\tau) r(\tau) d\tau \\ &= \int \pi_\theta(\tau) \nabla_\theta \log \pi_\theta(\tau) r(\tau) d\tau \\ &= \mathbb{E}_{\tau \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)] \end{aligned} \tag{3.27}$$

Since the gradient of the cost function is an expected value, it can be approximated using random sampling:

$$\nabla_\theta \mathbf{J}(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \Big(\sum_{t=1}^{T} \log \pi_\theta(s_{i,t}, a_{i,t}) \Big) \Big(\sum_{t=1}^{T} r(s_{i,t}, a_{i,t}) \Big) \tag{3.28}$$

With the weights of the network being updated with learning rate α:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \mathbf{J}(\theta) \tag{3.29}$$

This concludes the chapter about the optimization methods used by the evaluated models. Next, we introduce how these optimization methods can be combined with the neural network architectures from Chapter 2 to generate novel and useful molecules.

4

State-of-the-art Models for de novo Drug Discovery

In this section, we take a closer look at the five state-of-the-art generative models that we compare in this work. The models are based on combinations of the deep learning architectures and optimization algorithms described in Chapters 2 and 3. Together they represent a range of techniques that are considered to be of particular interest in the field of de novo drug design.

Although the models differ, they share the same core function: they generate molecules. By training their deep learning architectures on large amounts of molecules, the models learn an internal representation of the chemical space. This internal representation can in turn be traversed to find new, previously unseen molecules.

Although a model can find novel molecules, the challenge lies in generating something useful and realistic from a biochemical perspective. To this end, different optimization strategies can be used. A model must not only create molecules but also have the capability of changing the generation strategy based on outside feedback. In our project, this feedback comes from objective functions that score the molecules based on their structural and chemical properties. A model with a rich understanding of the chemical space will know how to alter the generation to maximize the objective function. It is this generative capability that the models will be evaluated on in our experiments.

4.1 Bayesian and Molecular Swarm Optimization

In their 2017 paper [18], *Bombarelli et al.* developed a variational autoencoder (VAE) trained to represent SMILES strings as distributions in a continuous latent space. This is made possible by training an encoder to encode the SMILES into multinomial distributions. Each SMILES string is encoded into two vectors, one where the elements represent a mean and another where the elements represent a standard deviation. The distributions are rewarded for having a shape similar to the standard normal distribution (with mean 0 and standard deviation 1). As a result, the encoded SMILES are forced together, creating overlaps between molecules in the latent space, illustrated in figure 4.1. This makes it possible to interpolate between molecules, which opens up the possibility of finding novel molecular structures.

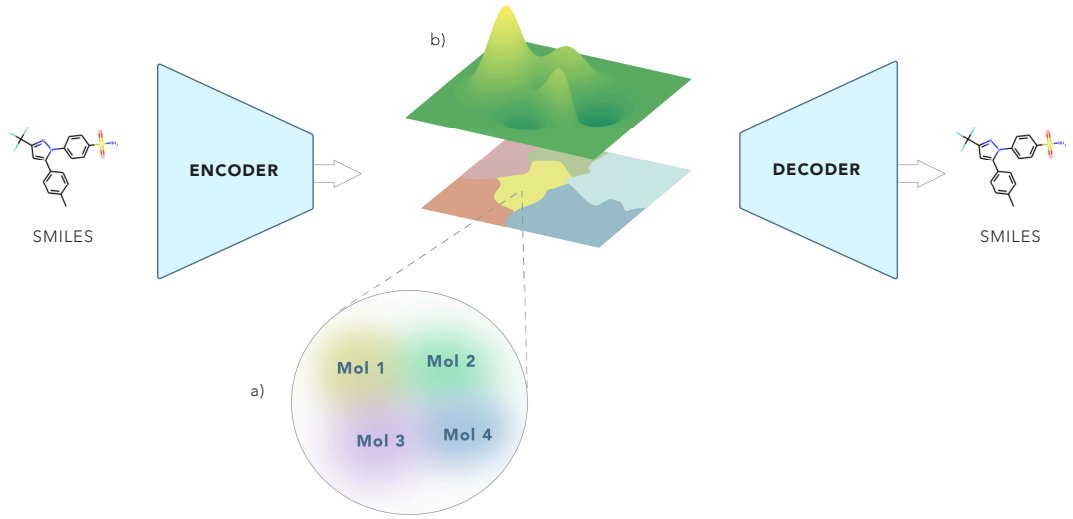


Figure 4.1: A variational autoencoder (VAE) trained to encode SMILES string as multinomial distributions. The encoded distributions are rewarded for having a mean of 0 with a standard deviation of 1. As a result, the distributions overlap as seen in a). Above the latent space is a landscape, b), representing the connection between the latent space and the value of a score function that scores molecules generated from this latent position. By traversing the latent space, the score is either increased or decreased. This is of interest when applying optimization algorithms to maximize the score of the generated molecules.

The variational autoencoder (Section 2.3) creates a *latent space*, possible to traverse in order to interpolate between and find novel molecules. By introducing an objective function that scores the generated molecules based on chemical properties, further meaning to the space can be added. In a sense, a score adds an additional dimension to the latent space, where moving in any direction results in an increase or decrease of the score. This opens up for the use of optimization algorithms to traverse the latent space in order to maximize the score. An illustration of the landscape that an objective function provides can be seen in figure 4.1. In the following subsections, we present two optimization algorithms that use different strategies to traverse the latent space to generate molecules that maximize a given score function. The algorithms are run on the latent space of a VAE architecture called *Continuous and Data-Driven Descriptors* (CDDD) [39], developed by a research group at Bayer pharmaceuticals.

4.1.1 Bayesian Optimization

With a latent representation of the chemical space, Bayesian Optimization (Section 3.1) can be applied to find new and useful molecules. Given a score function which returns how well a molecule fulfils a specific property, the latent space is sampled to find the highest scoring molecules. In this context, Bayesian Optimization is used to estimate the true objective function of the score over the latent space by repeated sampling. An acquisition function is used to guide the sampling to regions of the

latent space where it is most probable to find high scoring molecules.

In general, the latent space will have some regions that are considered *holes*, where the points do not decode into valid SMILES strings. To avoid sampling from these areas, a nudging function is used to guide the sampling into valid regions. Although this method requires no retraining of neural networks, it has been shown that the run-time increases exponentially as the dimensions of the latent space grow [40].

4.1.2 Molecule Swarm Optimization

Another optimization method on the latent space of CDDD created by *Winter et al.* makes use of particle swarm optimization (Section 3.2) to traverse the latent space in pursuit of high scoring molecules. The algorithm is illustrated and explained in figure 4.2.

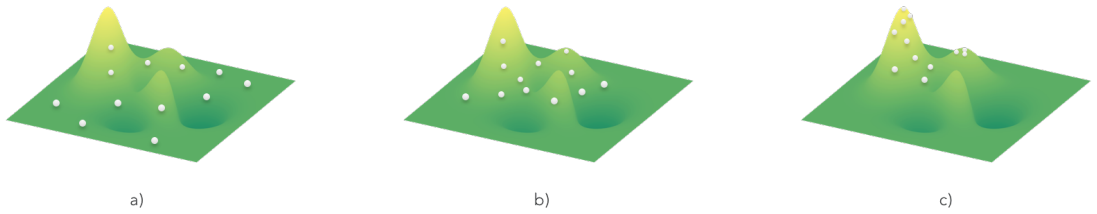


Figure 4.2: Three iterations of the Molecule swarm optimization algorithm on the latent space. The white circles, or particles, represent positions in the latent space that are decoded to SMILES strings. The landscape they are on represents the score of a decoded molecule from that latent position. In the first iteration, a), the molecules are in their starting position, randomly spread out throughout the latent space. In a subsequent iteration, b), the particles move in their own current direction (exploration) while being influenced to move toward the best molecule (exploitation) found by the swarm so far. By the end of the optimization, c), the particles have converged in the vicinity of an optimum.

In addition to the typical hyperparameters for PSO algorithms (inertia weight, φ_1 and φ_2), MSO make use of an additional acceleration coefficient, φ_3. While a particle's exploratory and exploitative inclinations are scaled by φ_1 and φ_2, this new scalar is used to manage a particles tendency to drift towards positions that have been the swarm's best at previous iterations, but not necessarily the current one. This behaviour is implemented by randomly selecting a point from the swarm's historically best particles and multiplying it with a uniformly random value $r_3 \in [0, 1]$ and φ_3. This term is thereafter included in the update step of a particle, as shown in equation 4.1.

$$v_i^{k+1} \leftarrow w v_i^k + \varphi_1 r_1 (x_i^{best} - x_i^k) + \varphi_2 r_2 (x_{swarm}^{best} - x_i^k) + \underbrace{\varphi_3 r_3 (x_{swarm}^{hist} - x_i^k)}_{MSO\ component} \quad (4.1)$$

Where x_{swarm}^{hist} denotes a randomly selected point from the swarm's previously best positions.

4.2 REINVENT

REINVENT [41] is a generative model developed at AstraZeneca in Mölndal, Sweden, which consists of two separate RNN components with the same architecture but different policies in its generative process. The first component is referred to as the *prior* and is trained on SMILES strings to acquire general knowledge about the chemical space. The second component, the *agent*, is based on the prior but is trained using reinforcement learning (RL) to "focus" on some area of interest in the chemical space. REINVENT aims to generate not only high scoring compounds according to some objective function, but also to generate compounds with high diversity. To this end, policy-based RL is employed to guide the agent towards areas of high scoring molecules, exploit these areas, and then move on to other promising regions. The following sections will provide an overview of the prior and agent network architectures as well as present the underlying principles of the RL algorithm.

4.2.1 Prior Network

The prior network is an RNN where, during training, a probability distribution is created over the set of possible SMILES characters. The distribution depends on the previously sampled tokens, and the objective is to maximize the likelihood estimation linked to the next, *correct*, token as described in Section 2.2. A central component in the training of the prior is the randomization of the SMILES strings in the training dataset. Before each epoch, the order of the atoms in the SMILES strings is shuffled randomly. The order of which the atoms are written in the SMILES string makes no difference to the molecular graph it represents. This means that the network is trained on different string representations of the training molecules every epoch. Learning multiple representations of molecules is known to give a generative model deeper knowledge about molecular syntax [42].

When the network has completed its training, it can be sampled for new molecules. This procedure is carried out as a partially observable Markov decision process (POMDP), a generalisation of an MDP. The procedure is episodic, and we define $A = a_1 a_2 ... a_T$ as the sequence of actions resulting in the sampled SMILES. An action corresponds to a one-hot encoding being sampled from the vocabulary, and the probability of A being generated is defined as $P(A) = \prod_{t=1}^{T} \pi(a_t|s_t)$ where π is the policy. The sampling process is illustrated in figure 4.3.

4.2.2 Agent Network

Starting from the policy learned by the prior, the objective of the agent is to refine this policy such that it becomes biased towards some specific objective. We define this objective as an arbitrary function $Score(A) \in [0, 1]$ that takes a SMILES as input and returns a scalar. The *Score* function can be a single objective function or a composition of multiple objectives.

Figure 4.3: Generation of a new molecule. On the y-axis, we see the vocabulary and the conditional probability to sample each character at every generative step. On the x-axis, the sampled character at every step is shown. The graph of the generated molecules is displayed to the right.

The idea is to produce a probability distribution that makes use of the prior's general knowledge of the chemical space while also maximizing the expected output given the *Score* function. To this end, an augmented likelihood is defined as:

$$logP(S)_{Aug} = logP(S)_{Prior} + \sigma * Score(S) \tag{4.2}$$

where σ is a scalar coefficient making sure that the log probabilities and the score are of similar magnitude. Using policy based RL, the agent tries to minimise the loss defined as:

$$loss = [logP(S)_{Aug} - logP(S)_{Agent}]^2 \tag{4.3}$$

To encourage the agent to generate molecules of high diversity, a diversity filter (DF) is introduced. The DF keeps track of the scaffolds present in molecules with an objective score above a predefined threshold. When enough such molecules have been generated, the RL algorithm will start penalising further exploitation of the area and thus force the agent to find promising compounds elsewhere.

4.3 GENTRL

Generative Tensorial Reinforcement Learning (GENTRL) [19] is a machine learning strategy developed at Insilico Medicine, Hong Kong. GENTRL was presented in the context of a scenario in which the full *de novo* drug design pipeline is taken into account with the ultimate goal of finding molecules that are proved to be active towards a specific target. This process included extensive preprocessing of data, generating novel molecules using GENTRL, filtering of the output and actual synthesizing of a small number of selected candidate compounds. The published experiment highlighted how the usage of generative models could be optimized by taking the whole drug design process into account by prioritizing synthesizability and activeness towards a predefined target. In this paper, GENTRL will be analyzed as a generative model in a more limited scope, where not as much emphasis is put on the full pipeline. The following sections will provide an overview of the

key aspects of the generative tool, consisting of an autoencoder and reinforcement learning.

4.3.1 Chemical Space Representation

GENTRL use a variational autoencoder (VAE) (Section 2.3) to model the chemical space. In the experiment reported by *Zhavoronkov et al.* the VAE was trained on three separate datasets. The first dataset was used to provide general knowledge of the chemical space while the two other sets aimed to bias the distribution given the specific objective. The VAE accepts input in the form of SMILES strings but also allow the user to provide a selection of values corresponding to some chemical properties. As a result, the VAE can learn a mapping that takes both chemical structures and associated properties into account. To encode the relationship between a molecular conformation and its properties a technique called *tensor decomposition* [43][44] was applied.

4.3.2 Generation Strategy

In order to further exploit an already objective focused chemical space, reinforcement learning is applied. The decoder component of the VAE is trained to update its distribution according to a user-defined objective function. When the training using the reinforce algorithm is completed, the decoder can be sampled for high scoring compounds.

4.4 LatentGAN

LatentGAN [45] was developed at AstraZeneca and utilizes a heteroencoder alongside a GAN, consisting of a generator and discriminator component. The idea is to teach the discriminator a distribution corresponding to an area of the chemical space that contains desirable compounds, according to some objective function. The generator tries to learn the same distribution such that its generated compounds are considered "true" by the discriminator. When the generator is able to produce data satisfactory to the discriminator, it can be sampled for novel compounds. The following sections aim to provide an overview of the relevant components and principles of LatentGAN. We give a brief description of the heteroencoder, followed by an explanation of the workflow and dynamics of the GAN.

4.4.1 Heteroencoder

A heteroencoder is a generalized autoencoder in the sense that it allows translation between different representations of the same entity. In contrast, an autoencoder promotes the recreation of the original representation of an object. In the case of LatentGAN [45], the heteroencoder is trained on pairs of SMILES strings representing the same molecular compound. The encoder translates the SMILES into a latent vector representation whereas the decoder tries to reconstruct it as a non-canonical

SMILES of the same molecule. The implementation is a modified version of an architecture reported by *Bjerrum* and *Sattorov* [42].

4.4.2 GAN workflow

Using the encoder component of the heteroencoder, data from a training set is translated to latent vectors $\mathbf{h}$ which in turn are fed to the discriminator as the true data. The generator is provided with uniformly randomized latent vectors as input and attempts to output vectors $\mathbf{h}'$ similar to $\mathbf{h}$. When the generator has learned the distribution defined by the discriminator, it can be sampled for new, latent, vectors. Using the decoder component of the heteroencoder, these vectors are in turn translated into SMILES.

In order to generate optimal molecules according to some specific objective, the distribution of the discriminator can be focused on any area of the chemical space. By feeding it data from a specialized dataset, containing compounds with some desired properties, its scope can be narrowed down to the sub-space in question. As a result, the generator is forced to learn this more objective-oriented distribution, and upon completion may be sampled for high scoring molecules.

5
Evaluation Methods

To evaluate the generative capability of the models and their ability to optimize, we carry out two experiments. The first experiment, described in Section 5.1, involves evaluating the models using a benchmarking tool called GuacaMol [46]. GuacaMol provides a set of optimization tasks where generated molecules are scored using different multi-objective functions for chemical properties. The second experiment evaluates the models' ability to generate molecules that are likely to bind to a specific biological target while avoiding undesired substructure patterns. More details are provided in Section 5.2. All code used for the experiments is available on GitHub[1].

5.1 GuacaMol Benchmarking Framework

GuacaMol is an evaluation framework for generative de novo design models. It consists of a set of single- and multi-property benchmarks that maps a molecular structure to a real-valued score between 0 and 1. The score reflects how well a generated molecule fits a chemical property profile, which is a set of desired molecular features such as structural features, physiochemical properties, similarity to a target molecule or presence of substructures, functional groups or atoms types. A benchmark consists of one or more scoring functions that define such features, and the objective of a model is to maximize the score of the generated molecules. Out of the 27 benchmarks included in the experiment, seven are considered trivial, while the others are considered non-trivial. The trivial benchmarks were deemed too easy by the authors of GuacaMol since the training dataset already contained molecules that solved the tasks almost perfectly. We still included them in this experiment as a means to expose poorly performing models.

Using GuacaMol as a tool to evaluate the models has multiple advantages. By using a standardized and publicly available framework, we allow future research to use our results as baselines. Furthermore, the large number of varying benchmarks enables a fair and comprehensive evaluation of the models, and since the framework was released along with a peer-reviewed paper, we can also be confident that the results we present will be relevant from a drug design perspective.

In the following sections, we provide details regarding the framework. In Section 5.1.1, we describe the various scoring functions that are used in the benchmark objectives. The benchmarks are introduced in Section 5.1.2 and how they are inte-

[1] https://github.com/sebastiandro/de-novo-evaluation

grated in the experiments can be read in Section 5.1.3. In Section 5.1.4 we briefly introduce the concept of measuring the quality of generated molecules.

5.1.1 Scoring Functions

Here we present the scoring functions that are used in combinations to define the GuacaMol benchmarks. Where applicable we also include a brief explanation to why particular functions might be of interest in the context of drug design.

Similarity: Compound similarity is one of the core concepts of chemoinformatics because of its usage in virtual screening [47]; the process of finding structures that bind to a particular target. Given a molecule that is known to interact with a target, odds are that if we find a similar molecule, this will also bind. The function `sim` accepts a target and a query molecule and returns a value between 0 (query not similar to target) and 1 (query similar to target).

Isomers: Isomers are molecules that share the same chemical formula, but that differ in structure (structural isomers) or spatial orientation (stereoisomers). The scoring function `isomer` accepts a SMILES string corresponding to a chemical formula and a query molecule. The function returns a score of 1 if the query has the same chemical formula as the target, but penalizes deviations.

TPSA: TPSA, or topological polar surface area, is a measurement of a compound's ability to penetrate the membrane of a cell [48]. It is, therefore, a suitable metric when predicting a drug's transportability, that is; how well it disperses in the body [49]. The function `TPSA` returns a positive real-numbered value for any query molecule.

logP: logP is a measurement that indicates a compounds dissolvability in water versus lipids (fats, oils, etc.) [50]. It is therefore often used when predicting the transportability of molecules. The function `logP` accepts a query molecule and returns a real-numbered value. A negative value indicates that the compound is more likely to dissolve in water while a positive value indicates higher dissolvability in lipids.

Number of fluorine atoms: The function `number of fluorine atoms` returns a positive integer indicating the number of fluorine (`'F'`) atoms present in a query molecule. Adding fluorine to a drug candidate can potentially enhance crucial properties and decrease the probability of failure in the design process [51].

Number of aromatic rings: The number of aromatic rings in a molecule can be of particular interest in drug design. Aromatic rings are well-documented and often easy to synthesize and therefore, a potential means to reduce attrition rates in drug development. However, too many rings can have the opposite effect and instead decrease the developability of a drug [52]. The function `number aromatic rings` returns a positive integer corresponding to the number of aromatic rings in a query

molecule.

Number of rings: The number of rings, in general, is also an important aspect of drug design. Approximately 74% of all drugs include one or two rings fragments. The function `number rings` returns an integer according to the number of rings present in a query molecule.

SMARTS: The `SMARTS` function is used to locate a specific substructure pattern in a query molecule. It can be used to reward such structures or to penalize them. In the former case, the presence of a target substructure is rewarded the score of 1 while its absence returns a 0. The opposite holds for the case where we want to avoid the specific pattern. The function accepts a SMARTS string (see Section 1.3) that denotes the substructure, a boolean to signal if the substructure is desired or not and a query molecule.

Bertz: Bertz is a topological index that quantifies the complexity of molecules [53]. The function `Bertz` accepts a query molecule and returns a real numbered value where high values indicate complex molecules.

Number of hydrogen bond donors: When evaluating a compounds ability to interact with a target, hydrogen bonds are of particular interest [54]. A hydrogen bond consists of a donor, an acceptor and a hydrogen atom. The function `num hydrogen bond donors` returns the number of hydrogen bond donors given a query.

Molecular weight: Molecular weight can have an impact on a drug's oral bioactivity and is preferably kept below 500 [55]. The function `molecular weight` returns the weight of a molecule.

QED: Quantitative estimation of drug-likeness, or QED, is a common metric used in drug design. The QED value of a molecule is a mean of various desirability functions, based on molecular properties such as logP, molecular weight and number of aromatic rings [56]. The `QED` function accepts a query molecule as input and returns a value in the range 0 to 1, depending on the molecule's drug-likeness

Number rotatable bonds: The number of rotatable of bonds in a molecular compound has been shown to influence the oral bioactivity of drugs [57]. The more rotatable bonds, the less bioactivity. The function `number of rotatable bonds` accepts a query molecule and returns the number of rotatable bonds in it.

5.1.2 Benchmarks

Here we introduce the benchmarks that are used to evaluate the models. In table 5.1 we include the 20 non-trivial benchmarks while the 7 trivial benchmarks are found in table 5.2.

A benchmark consists of a selection of scoring functions. These scoring functions

may be subject to a scoring modifier (see *Modifier* column) that, for example, normalises the score to a value in the range 0 to 1. There are four different types of modifiers which we introduce in appendix A.2. Furthermore, for the multi-function benchmarks, the score is calculated as either a geometric or an arithmetic mean, as indicated in the *Mean* column. The final output for a benchmark is a weighted average of the top scoring molecules in certain ranges. For example, most benchmarks are calculated using the top-1, top-10 and top-100 molecules. The average score for every range is summed and the final output is the mean of that sum, as defined in equation 5.1.

$$S = \frac{1}{3}\left(s_1 + \frac{1}{10}\sum_{i=1}^{10} s_i + \frac{1}{100}\sum_{i=1}^{100} s_i\right) \tag{5.1}$$

Here, the score s_i denotes the generated molecule scores sorted in decreasing order, $s_i \geq s_j$ for $i < j$.

5.1.3 Integration

The benchmarks are employed in two phases; optimization and evaluation. In the optimization phase, a benchmark is used as a scoring function that guides the generation process of a model. A benchmarking function accepts a single SMILES string as input and returns a value between 0 and 1 that is fed back to the model, which in turn tries to find a better molecule in the next iteration. Here we stress that we are only concerned with scoring one molecule in isolation, and not calculating a weighted score based on multiple molecules.

In the evaluation phase the models are evaluated based on their highest scoring molecules from the optimization. Which of the best molecules the final score on a benchmark is based on is indicated in the *Scoring* column of tables 5.1 and 5.2. The score on a benchmark is a value in the range 0 to 1, thus a model can as best receive a score of 27 in total for this experiment.

5.1.4 Quality Measures

Although a model scores high on a benchmark function, the generated molecules may be unstable, hard to synthesize or simply unpleasant for a medicinal chemist to look at. If a model is used in a drug discovery pipeline, it is vital that the molecules that it generates hold a certain level of quality. To assess the quality of the molecules generated for the 27 benchmarks, we employ a number of quality filters [58], included in the GuacaMol benchmarking suite. The filters look for structural patterns in the molecules that are deemed as undesired from a medical chemistry standpoint. The top 100 scoring molecules from each benchmark are run through the filters, and the proportion of molecules that pass the filters is compared between models.

Benchmark name	Scoring	Mean	Scoring functions	Modifier
Celecoxib rediscovery	top-1		sim(celecoxib)	
Troglitazone rediscovery	top-1		sim(troglitazone)	
Thiothixene rediscovery	top-1		sim(thiothixene)	
Aripiprazole similarity	top-1/10/100		sim(aripiprazole)	Thresh(0.75)
Albuterol similarity	top-1/10/100		sim(albuterol)	Thresh(0.75)
Mestranol similarity	top-1/10/100		sim(mestranol)	Thresh(0.75)
$C_{11}H_{24}$	top-159		isomer($C_{11}H_{24}$)	
$C_9H_{10}N_2O_2PF_2Cl$	top-250		isomer($C_9H_{10}N_2O_2PF_2Cl$)	
Median molecules 1	top-1/10/100	geom	sim(camphor) sim(menthol)	
Median molecules 2	top-1/10/100	geom	sim(tadalafi) sim(sildenafi)	
Osimertinib MPO	top-1/10/100	geom	sim(osimertinib) sim(osimertinib) TPSA logP	Thresh(0.8) MinGauss(0.85, 2) MaxGauss(100, 2) MinGauss(1, 2)
Fexofenadine MPO	top-1/10/100	geom	sim(fexofenadine) TPSA logP	Thresh(0.8) MaxGauss(90, 2) MinGauss(4, 2)
Ranolazine MPO	top-1/10/100	geom	sim(ranolazine) logP TPSA number fluorine atoms	Thresh(0.7) MaxGauss(7, 1) MaxGauss(95, 20) Gauss(1, 1)
Perindopril MPO	top-1/10/100	geom	sim(perindopril) number aromatic rings	 Gauss(2, 0.5)
Amlodipine MPO	top-1/10/100	geom	sim(amlodipine) number rings	 Gauss(3, 0.5)
Sitagliptin MPO	top-1/10/100	geom	sim(sitagliptin) logP TPSA isomer($C_{16}H_{15}F_6N_5O$)	Gauss(0, 0.1) Gauss(2.0165, 0.2) Gauss(77.04, 5)
Zaleplon MPO	top-1/10/100	geom	sim(zaleplon) isomer($C_{19}H_{17}N_3O_2$)	
Valsartan SMARTS	top-1/10/100	geom	SMARTS(s_2, true) logP TPSA Bertz	 Gauss(2.0165, 0.2) Gauss(77.04, 5) Gauss(896.38, 30)
Scaffold Hop	top-1/10/100	arithm	SMARTS(s_2, true) SMARTS(s_3, false) SMARTS(s_4, false) sim(s_5)	 Thresh(0.85)
Deco Hop	top-1/10/100	arithm	SMARTS(s_2, false) SMARTS(s_6, true) sim(s_5)	 Thresh(0.75)

Table 5.1: GuacaMol non-trivial benchmarks specification. SMARTS and SMILES strings abbreviated as s_x are specified in appendix A.1. This table was adopted from the GuacaMol paper [46].

Benchmark name	Scoring	Mean	Scoring functions	Modifier
logP(target: -1.0)	top-1/10/100		logP	Gauss(-1, 2)
logP(target: 8.0)	top-1/10/100		logP	Gauss(8, 2)
TPSA(target: 150.0)	top-1/10/100		TPSA	Gauss(150, 2)
CNS MPO	top-1/10/100	arithm	TPSA TPSA num hydrogen bond donors logP molecular weight	MinGauss(90, 2) MaxGaus(40, 2) MinGauss(0, 2) MinGauss(5, 2) MinGauss(360, 2)
QED	top-1/10/100		QED	
$C_7H_8N_2O_2$	top-159		isomer($C_{11}H_{24}$)	
Pioglitazone MPO	top-100	geom	sim(pioglitazone) molecular weight number rotatable bonds	Gauss(0, 0.1) Gauss(356.447, 10) Gauss(2, 0.5)

Table 5.2: GuacaMol trivial benchmarks specification. This table was adopted from the GuacaMol paper [46].

5.2 Activity Towards Dopamine Receptor D2

While GuacaMol evaluates the models' generative capabilities on a range of chemical optimization tasks, the framework does not provide any benchmark that includes generating compounds predicted to interact with a biological target. In the development of a new drug, a generative model is used to find novel molecular structures that bind to a specific target in the body. To assess how well the models perform in a more industry-like scenario, they are evaluated on the task of generating molecules which bind to the Dopamine Receptor D2 (DRD2).

Receptors are proteins, often found in the cell membrane, that transfers signals between a cell's outside environment, and its core. The signals are, usually, chemical messengers in the form of ligands that bind to the receptor, which in turn activates a response from the cell. As suggested by the name, the DRD2 dopamine receptor binds to the neurotransmitter *dopamine*. The receptor is involved in several neurological processes, including motivation, pleasure, cognition and memory, amongst others [59]. Problems with the dopamine receptor signalling can cause a number of disorders, which makes it a common target for neurological drugs.

In the following sections, we give an overview of the experiment involving DRD2. In Section 5.2.1, we define a multi-property function that will be used to guide the models in their generation. The experiment is arranged into three sub-experiments where the models try to include certain scaffolds in the generated compounds. These scaffolds are introduced in Section 5.2.2. The chapter is concluded with Section 5.2.3, where we describe the parameters that will be analyzed when evaluating and comparing the models.

5.2.1 Scoring Function

Finding a molecule that binds to a target is usually not enough when developing a new drug. The compound needs to have certain properties which make it suitable as a drug. In addition to not being toxic or too reactive, it should also be possible to synthesize it. For this purpose, a set of constraints on the molecular structure is added. The molecule needs to have drug-like properties, while also containing substructures that are interesting for a medicinal chemist. As a result, the task can be described as the maximization of a multi-objective scoring function:

$$f(m, s, \mathcal{U}) = CA(m, \mathcal{U}) \times MS(m, s) \left(\frac{QED(m) + Binding(m)}{2} \right) \tag{5.2}$$

comprised by four objectives:

- $CA(m, \mathcal{U}) \in \{0, 1\}$. *Custom Alerts*, returns 0 if the molecule contains one of the undesired substructures in $\mathcal{U}$ and 1 otherwise.
- $MS(m, s) \in \{0.5, 1\}$. *Matching Substructure*, returns 1 if the molecule contains the substructure s and 0.5 otherwise.

- $QED(m) \in (0, 1)$. Measure of drug-likeness [56], returns a scalar between 0 and 1.
- $Binding(m) \in \{0, 1\}$. A scikit-learn Random Forest classifier [60] trained on data from molecules synthesized in a lab that were measured for DRD2 bioactivity. [61]. Returns the probability, a number between 0 and 1, that this molecule bind to DRD2.

The function rewards molecules that bind toward DRD2 while having a drug-like structure. Custom alerts can be thought of as a hard penalization since any occurrence of unwanted substructures returns in a score of 0. This could steer the optimization algorithms away from the regions of undesirable structures. Matching substructures is a softer penalty by allowing the optimization to sample molecules which do not contain the target substructure. This opens up for exploration and could guide the algorithms to intermediate steps before finding both substructure, drug-likeness and binding.

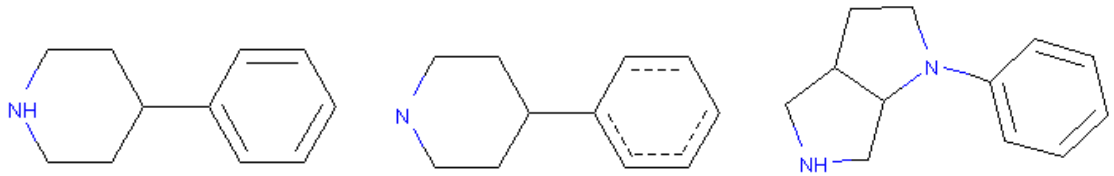
5.2.2 Scaffolds

The DRD2 experiment is performed in three sub-experiments where three different scaffolds will act as targets in the $MS(m, s)$ function. A scaffold is a core structure of a small molecule. In the context of drug design, scaffolds can be of particular interest if they possess some documented, desirable, properties [62]. The scaffolds for these experiments were selected by a senior chemist at AstraZeneca and have been selected such that they all have varying frequencies in the training dataset. That way, we can study the models' generative capability under different levels of difficulty by providing them with varying amount of exposure to the requested scaffolds. In table 5.3 we give the SMARTS representations of the scaffolds along with their frequency in the training dataset. In figure 5.1 we include a visualization of the scaffolds.

	SMARTS	Frequency	Graph
Scaffold 1	`c1ccc(cc1)C2CCNCC2`	$0.00768e^{-3}$	see Figure 5.1a
Scaffold 2	`[c,n]1[c,n][c,n]c([c,n][c,n]1)[C,N]2CCNCC2`	$0.03532e^{-2}$	see Figure 5.1b
Scaffold 3	`c1ccc(cc1)N2CCC3C2CNC3`	$1.64252e^{-5}$	see Figure 5.1c

Table 5.3: DRD2 experiment scaffolds

As can be seen in the *Frequency* column the second scaffold is a more general and flexible target structure than the other two, while the third scaffold should be the most difficult for the models to find, due to their minimal exposure to it during training.



(a) scaffold 1 **(b)** scaffold 2 **(c)** scaffold 3

Figure 5.1: DRD2 experiment scaffolds

5.2.3 Evaluation

To evaluate the models, we analyze and compare the distribution of the generated molecules for each sub-experiment. By looking at the distributions, we get insight into how well the models manage to find and exploit areas of high scoring compounds. Additionally, we take a closer look at the top-scoring molecules for each model. This can be of particular interest in cases where a model has a distribution centred around a relatively low value but still manages to find a small set of high scoring compounds. To further enrich the analysis, we also provide a breakdown of the generated molecules' scores in terms of the multi-objective function's separate components. This way, an in-depth analysis can be made for the models to draw conclusions regarding their performance. For example, if a model is performing poorly compared to the others, a breakdown of the scores can be useful to see if the problem is a general one or if the model is struggling with any of the scoring functions in particular.

6
Experimental Setup

In this chapter, we describe how the experiments are carried out in practice to ensure a qualitative and fair evaluation of the models. The chapter is split into four sections that each describes a part of the procedure. Section 6.1 describes the different datasets that were used in this work. In Section 6.2 we give a brief description of our approach to hyperparameter tuning. The execution of the experiments is presented, model by model, in Section 6.3. The chapter is concluded with Section 6.4 where we give describe how the stochasticity of the models is handled to ensure verifiable results.

6.1 Datasets

In order to make a fair comparison between the five models, they need to have access to the same prior knowledge. That is; they should be trained on the same data. This is to ensure that no model gets an unfair advantage, for example, by training it on data containing the molecules to be found in the different similarity tasks. With this in mind, we retrained the models on two separate datasets, one in preparation for the GuacaMol benchmarks, and one for the DRD2 experiments. In the two following sections, we describe in detail the nature of the datasets and how they have been preprocessed to fit the design of the experiments.

6.1.1 GuacaMol Training Dataset

The authors of GuacaMol provide users with a prepared dataset [63] to use for retraining of any model that will be benchmarked. The dataset is a filtered version of the ChEMBL dataset [64], a dataset containing 2 million bioactive compounds that have already been synthesized, making it suitable in this particular context.
The final dataset provided by the authors of GuacaMol is derived from the 24:th release of ChEMBL and has been post-processed in the following five steps:

- Removal of salts: *Whether or not a molecule will form a salt in a solution can be inferred from its SMILES string. Hence it is not something necessary for the model to learn to generate.*
- Charge neutralization: *Some molecules may be charged positively or negatively in the dataset. As with the salts, charge can be added later on and is not interesting to generate. The charged molecules are run through software that neutralizes charges.*

- Removal of molecules containing atoms other than H, B, C, N, O, F, Si, P, S, Cl, Se, Br and I: *This is done to limit the vocabulary of the models to atoms relevant to the benchmarking suite.*
- Removal of molecules that are too similar to the similarity and rediscovery tests (Section 5.1.1): *This makes sure that the models must create molecules it has not seen before.*
- Removal of molecules with SMILES string length over 100 characters: *For the functions in the benchmarking suite we are not interested in molecules with more than 100 atoms.*

6.1.2 GuacaMol Top-1000 Datasets

Using the scoring functions from the benchmarking tasks, the 1000 best molecules for all objectives were found in the aforementioned dataset. These compounds were extracted and put into 27 separate subsets, one for every benchmark. These subsets were later used as the true data input for the discriminator in LatentGAN. We also ran experiments with BO and MSO where we investigated the impact of a high scoring prior and initiating swarms at high scoring molecules. The top-1000 datasets were used in these cases.

6.1.3 DRD2 Inactives dataset

To facilitate fair comparisons of the models when assessed on the DRD2 experiments, we created a filtered subset of ChEMBL where all molecules with a probability of being active of 0.5 or higher were removed. This was done to force the models to create DRD2 active molecules through optimization rather than replicating molecules seen in the training dataset. This dataset was used to retrain the models as well as providing MSO and BO with randomly selected molecules to initiate the optimization runs.

6.2 Hyperparameter Search

Another important aspect of making a fair comparison is to assure that each model performs as well as possible for each test. The results produced by a machine learning model depend heavily on what hyperparameters were used during training [65]. However, finding the near-optimal hyperparameters for each model and benchmark requires a large number of runs. As this project is limited by time, a constrained approach to hyperparameter optimization is used.

The constrained approach used is to start by running the benchmarks using the default hyperparameters provided with the code for each model. The results are then compared to the dataset max scores, which are used as a baseline. If the model cannot generate molecules that score better than what its prior was trained on, the hyperparameters are adjusted around the vicinity of the default values. If a manual hyperparameter optimization also failed, the authors of the model were contacted to get guidelines on how to improve the results. If the default hyperparameters lead to

results that on average score higher than the dataset max, a hyperparameter search is carried out with the aim to further improve the performance of the model by varying the hyperparameters around the default values.

6.3 Experiment Execution

The following sections aim to give the reader an overview of the execution of the experiments. Model by model, we provide a chronological account of how the models were submitted to the various optimization tasks, evaluated and adjusted, in terms of hyperparameters. Here we also provide the configurations and hyperparameters that were used to generate the results reported in Chapter 7.

6.3.1 REINVENT

The neural network is trained using an adaptive learning rate, meaning that the learning rate change according to the changes in the validation loss. Initially, the learning rate will start relatively high and decrease as the validation loss plateaus. When the learning rate is at a minimum, it will be kept there as long as the validation loss decreases. If no change in the loss has been seen for a set number of training batches, called *patience*, the learning rate starts to increase again until a maximum learning rate is reached.

During reinforcement learning, the generated molecules that score above a certain threshold will be stored in memory. The agent is penalized for generating molecules that look too similar to the ones in memory. This ensures diversity amongst the generated molecules. At the end of the training, the 10,000 highest scoring molecules in memory are returned as the model's output.

To train the prior, we are using the default hyperparameters (table 6.1) found in the code repository. Due to time restrictions, we will assume that this architecture is sufficient for the experiments.

hyperparameter	value
num_epochs	5
batch_size	128
randomize	true

Table 6.1: Default configurations for training REINVENT

For the hyperparameters used during reinforcement learning, we also chose the default settings available in the code. Batch sizes of 128, 256 and 512 were evaluated without any significant change in performance. The number of steps was evaluated at 200, 500, 1000, 1500 and 2000 steps. The increase of performance reaches a max around 1000 steps, with small or no improvement for a larger number of steps. The final hyperparameters used during reinforcement learning for all steps are listed in table 6.2.

hyperparameter	value
num_steps	1000
sigma	128
learning_rate	0.0001
batch_size	256
memory_size	10000

Table 6.2: Hyperparameters for reinforcement learning in REINVENT

6.3.2 LatentGAN

The LatentGAN model uses an autoencoder architecture from the Deep Drug Decoder framework [66] as a base for its generator and discriminator. The autoencoder is trained using a decaying learning rate, meaning that the learning rate is reduced by a factor of 2 after no decrease in the loss has been seen for a set number of batches. The reduction is repeated until a minimum learning rate is reached.

In the deep drug decoder code repository[1], there is an autoencoder trained on another version of the ChEMBL dataset. We used the same architecture when training the two priors used for our experiments.

The LatentGAN differs from the other models in that it has no element of learning from feedback. Instead, it uses what it has learned about the chemical space to generate molecules that have a certain structure. To guide the generation, the LatentGAN is shown top scoring molecules for each GuacaMol score function and also the known actives of DRD2. The generator and discriminator networks in the LatentGAN architecture are then trained using this dataset of top-scoring molecules. The idea is for the GAN to learn to generate molecules that look similar to the top scoring ones. The hyperparameters that have shown to produce the best results are listed in table (6.3).

hyperparameter	value
num_epochs	10000
learning_rate	0.001
n_critics	8

Table 6.3: Final configurations for training the LatentGAN

6.3.3 GENTRL

Unlike the other models, the GENTRL neural network architecture is trained on both the SMILES string representation and a set of chemical properties for the molecules in the training dataset. We used two approaches to augment the training dataset with additional chemical properties. The first approach is to add general chemical knowledge by adding eight physicochemical properties to each molecule (see appendix B.2). The second approach is to add the score from an objective

[1] https://github.com/pcko1/Deep-Drug-Coder

function as the only additional property. For instance, if the model were to optimize the *celecoxib similarity* benchmark function from GuacaMol, that similarity score would be added to each of the molecules in the dataset. As a result, the network is retrained on SMILES and their corresponding score, before running reinforcement learning to maximize the score functions from GuacaMol or DRD2.

As mentioned previously, the strategy for hyperparameter tuning included running the experiments with the default hyperparameters as a starting point. GENTRL showed poor performance in the GuacaMol benchmark when optimizing using the default hyperparameters. In this context, performance is assessed by the model's ability to generate molecules that surpass the score of the molecules in the training dataset.

To improve performance, the hyperparameters used during training were varied, and the network architecture modified. The latent size was increased from 50 to 100 and 128. The number of epochs was increased, and the batch size used was varied between 128, 256 and 512. The number of neurons per layer in the RNN used in the encoder was changed from 256 to 512, and the number of layers increased from 2 to 4. We implemented a decaying learning rate in addition to using fixed learning rates of 1e-3, 1e-4 and 1e-5. The changes to the architecture showed no improvement of the final scores; thus, the default architecture was used. In addition, the authors of GENTRL were contacted in order to receive guidance on how to improve performance. The authors recommended an increase in the number of feature descriptors. The change had little to no effect on the final scores. The final network architecture and training settings used for the variational autoencoder and the learnable prior is presented in table 6.4.

hyperparameter	value
latent_size	50
num_epochs	20
learning_rate	1e-4
batch_size	128
layer size RNN	256
latent descriptors	50
feature descriptors	8

Table 6.4: Hyperparameters for the variational autoencoder and learnable prior used in GENTRL.

For reinforcement learning, the default number of steps was set to 100,000. This would require days running for each objective function. The 100,000 steps were evaluated on the *celecoxib similarity* benchmark in GuacaMol, which showed a small improvement on the final score compared to running it for 2000 steps. However, it was not large enough to validate using a number of steps of that magnitude for all benchmarks. Batch sizes were varied between 128, 256 and 512 with no significant difference in the final scores. Learning rate of the learnable prior achieved better scores when it was set to 1e-4 and a decrease in performance when the values was set

below that. The final hyperparameters used for reinforcement learning are shown in table 6.5.

hyperparameter	range
num_steps	2000
batch_size	200
learning rate	1e-4
learnable prior	
learning rate	1e-6
decoder	

Table 6.5: Hyperparameter settings for optimizing GENTRL with reinforcement learning

6.3.4 CDDD

The CDDD package, containing the VAE used to represent the chemical space used in MSO and BO, provides the user with an interface for retraining the model. A file containing all necessary configurations, such as hyperparameters and paths to required files, is provided along with default values. When retraining, we used the default settings with some exceptions. The selected non-default settings, listed in table 6.6, were also used by *Winter et al.* when training the model for MSO.

argument	value
input_sequence_key	canonical_smiles
model	NoisyGRUSeq2Seq
input_pipeline	InputPipeline
infer_input	canonical
emb_noise	0.05
cell_size	[512, 1024, 2048]
input_dropout	0.15
emb_size	512

Table 6.6: Non-default configurations used for retraining CDDD VAE

6.3.5 Molecular Swarm Optimization

All experiments using MSO were executed using 40 swarms of 200 particles each, as per the default settings. For the GuacaMol benchmarks, each swarm was initiated at randomly selected molecules from the datasets described in Section 6.1.1. In the DRD2 experiments, the swarms were initiated at randomly selected compounds from the DRD2 inactives dataset presented in Section 6.1.3. During every optimization run the 10,000 best molecules so far, aggregated from all 40 swarms, was tracked by the optimizer. At the end of a run, these 10,000 molecules returned as the output of the optimization.

As described in Section 3.2.1, there is a large variety of hyperparameter settings to be found in the literature, when it comes to different applications of PSO algorithms.

Not only are there variations in what fixed values to use for the various parameters, but there are also versions of PSO that employs hyperparameters that vary over time. For MSO, we decided to explore different usages of the latter approach. In addition to the default, fixed, hyperparameters provided by the authors of MSO, we implemented the possibility to let the inertia weight, w, φ_1 and φ_2 vary over time. Thus, w could either be fixed or decrease linearly over time from 0.9 to 0.4 in order to reduce momentum and promote exploitation in the later stages of a run. The acceleration coefficients φ_1 and φ_2 could either be set to a fixed value or vary over time. If the latter was selected, φ_1 was linearly decreased from 2.0 to a minimum of 0.5 throughout the optimization run while φ_2 increased linearly from 0.5 to a maximum of 2.0. φ_3 were fixed in all scenarios. The hyperparameters were tested in various combinations of settings, as presented in table 6.7, where the first setting contains the default values. The number of particles was set to 200 for all runs.

MSO Hyperparameters

	w	φ_1	φ_2	φ_3
Setting 1	0.9	2.0	2.0	2.0
Setting 2	0.9	2.5	2.5	2.5
Setting 3	$lin(0.9, 0.4)$	2.0	2.0	2.0
Setting 4	0.9	$lin(2.0, 0.5)$	$lin(0.5, 2.0)$	2.0
Setting 5	$lin(0.9, 0.4)$	$lin(2.0, 0.5)$	$lin(0.5, 2.0)$	2.0

Table 6.7: Hyperparameter settings used with MSO

For the GuacaMol experiment, our initial goal was to reproduce the results as reported by *Winter et al.*. To this end, the default parameters (see *Setting 1* in table 6.7) were used. However, the resulting scores turned out to be significantly subpar to the ones previously reported. To improve the scores, we submitted the model to multiple optimization runs using the various hyperparameter settings. Even though the utilization of varying hyperparameters showed some promise the model still could not generate satisfying scores.

As a next step, the authors of MSO was contacted and a new set of hyperparameter values were suggested by them (see *Setting 2*). Furthermore, it was also indicated that the top molecules given a benchmark had been used as starting points, in order to reach higher scores. The results showed significant improvements at this stage, and by starting the optimization from the 40 best molecules for any given benchmark, we managed to match the results reported by *Winter et al.* with some minor deviations in both directions. For the DRD2 experiments, there were no previously reported baselines. Here our approach was simply to use all hyperparameter settings and use the results from the top-performing one in the evaluation. We give the hyperparameters that performed best for each experiment in table 6.8. For the DRD2 experiments the best results were produced when using a linearly decreasing inertia weight. However we stress that these results were not better by a significant margin compared to using the same hyperparameter values as in the GuacaMol experiment.

hyperparameter	value (GuacaMol)	value (DRD2)
w	0.9	lin(0.9, 0.4)
φ_1	2.5	2.0
φ_2	2.5	2.0
φ_3	2.5	2.0

Table 6.8: Final hyperparameter settings for MSO

6.3.6 Bayesian Optimization

The Bayesian Optimization (BO) algorithm used in this project is not based on published work, but was put together for this thesis. To this end, we used surrogate models, acquisition functions and optimizers provided in the python libraries GPFlow [67] and GPFlowOpt [68]. We used a Gaussian process as a surrogate model. The prior distribution was created using 1000 randomly selected molecules from the training dataset. A modified version (see appendix B.1 for details) of a `Matern52`[2] kernel was used as kernel function and expected improvement (EI) was selected as acquisition function.

In order to optimize the acquisition function, a Monte Carlo optimizer[3] and scipy's [69] `minimize`[4] functions were used in succession. The Monte Carlo optimizer samples and evaluates 3000 randomly selected points and the optimization is thereafter focused by applying `minimize` on the most promising point.

The experiments were conducted in 2000 steps or until 1000 novel molecules had been sampled from the latent space. The output from the GuacaMol evaluations showed that BO struggled to find compounds that surpassed the dataset max and to improve its performance, we tried tuning the likelihood variance used in the surrogate model. We used values in the range of 0.02 to 0.1 but with little to no obvious effect. Experiments were also run using lower confidence bound and the probability of feasibility as acquisition function in place of EI but again, without any significant effect. Furthermore, we investigated the effect of using a linear and a polynomial kernel function instead of `Matern52`, but the results did not improve noticeably. In table 6.9 we give the final hyperparameters.

hyperparameter	value
acquisition function	expected improvement
variance	0.02
kernel	Matern52

Table 6.9: Hyperparameter settings for Bayesian Optimization

To study the impact of the prior distribution, we decided to run the GuacaMol experiments using the top-1000 molecule datasets to initiate the optimization. The

[2]https://gpflow.readthedocs.io/en/master/gpflow/kernels/#gpflow-kernels-matern52
[3]https://github.com/GPflow/GPflowOpt/blob/master/gpflowopt/optim.py#L131
[4]https://github.com/GPflow/GPflowOpt/blob/master/gpflowopt/optim.py#L201

idea was that by providing the model with a set of high scoring molecules, the optimization would become more focused towards those areas. While there was a noticeable improvement, the model still had difficulties with finding molecules that surpassed the dataset max.

6.4 Stochasticity of models

Since all models include some level of stochasticity, one has to ensure that any sequence of random choices can be repeated on command. To this end, the models are initiated using a seed, specified by the user, upon starting an optimization run. By setting a seed for the modules that perform stochastic actions for each model, the same results can be guaranteed if running a task using the same seed and hyperparameters multiple times. When starting an optimization run, all relevant configurations are saved to enable reproducibility and verification of the results.

7
Results

Here we report the results from the experiments described in Chapter 5. Section 7.1 contains the score output for all models when benchmarked using GuacaMol. Here we also report the aggregated quality of the top-100 generated molecules for each model, and the models' ability to generate novel and unique compounds. Section 7.2 contains the outcome of the experiments aiming to optimize molecules toward DRD2.

7.1 GuacaMol Benchmarks

As displayed in Table 7.1, REINVENT outperforms the other models by acquiring the top scores for almost all benchmarks. MSO outputs the highest scores on five of the benchmarks and the second-best on all others. GENTRL, LatentGAN and BO all fail to beat the dataset max on the majority of the benchmarks, including the trivial ones as shown in table 7.2.

GuacaMol non-trivial benchmarks								
Benchmark name	REINVENT	GENTRL	LatentGAN	MSO	MSO*	BO	BO*	Dataset
Celecoxib rediscovery	**1.000**	0.439	0.318	0.753	1.000	0.369	0.543	0.506
Troglitazone rediscovery	**1.000**	0.261	0.304	0.451	0.827	0.349	0.317	0.419
Thiothixene rediscovery	**1.000**	0.387	0.292	0.590	1.000	0.398	0.352	0.456
Aripiprazole similarity	**1.000**	0.380	0.294	0.734	0.999	0.380	0.523	0.609
Albuterol similarity	**1.000**	0.908	0.564	0.985	1.000	0.637	0.645	0.765
Mestranol similarity	**1.000**	0.542	0.370	0.846	0.996	0.412	0.574	0.660
C11H24	0.960	0.874	0.005	**0.982**	0.999	0.020	0.156	1.000
C9H10N2O2PF2Cl	**0.939**	0.835	0.421	0.895	1.000	0.372	0.390	0.869
Median molecules 1	**0.453**	0.346	0.194	0.349	0.394	0.279	0.264	0.400
Median molecules 2	**0.395**	0.166	0.170	0.276	0.373	0.209	0.159	0.362
Osimertinib MPO	**0.954**	0.797	0.771	0.881	0.917	0.823	0.843	0.856
Fexofenadine MPO	**0.998**	0.732	0.639	0.882	0.965	0.787	0.813	0.856
Ranolazine MPO	**0.907**	0.760	0.512	0.863	0.900	0.290	0.793	0.794
Perindopril MPO	**0.832**	0.452	0.402	0.633	0.742	0.348	0.443	0.625
Amlodipine MPO	**0.905**	0.573	0.516	0.668	0.874	0.531	0.557	0.716
Sitagliptin MPO	0.553	0.429	0.258	**0.618**	0.719	0.115	0.146	0.598
Zaleplon MPO	**0.677**	0.540	0.470	0.576	0.708	0.367	0.503	0.585
Valsartan SMARTS	0.869	0.000	0.000	**0.915**	0.994	0.000	0.000	0.603
Scaffold Hop	**1.000**	0.578	0.564	0.903	1.000	0.802	0.932	0.965
Deco Hop	**1.000**	0.452	0.443	0.607	0.980	0.522	0.400	0.808
Total score	**17.442**	10.451	7.506	14.407	17.387	8.011	9.354	12.144

Table 7.1: Scores for the five models assessed on non-trivial benchmarks from GuacaMol. MSO and BO are initiated using randomly selected molecules from the training dataset. The highest score for every benchmark is indicated with bold digits. The MSO* and BO* columns display results when the models are initiated using top-scoring molecules as described in Section 7.1.1. If the highest score of a benchmark is matched or improved this is indicated with an underline.

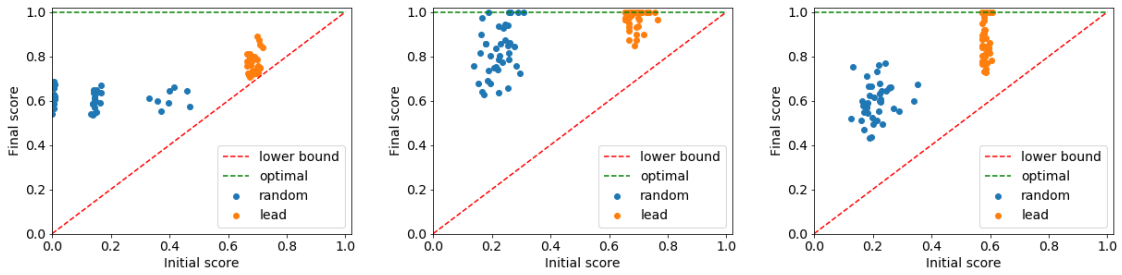
GuacaMol trivial benchmarks								
Benchmark name	REINVENT	GENTRL	LatentGAN	MSO	MSO*	BO	BO*	Dataset
logP (target: -1.0)	**1.000**	**1.000**	0.972	**1.000**	1.000	0.899	0.999	1.000
logP (target: 8.0)	**1.000**	**1.000**	0.886	**1.000**	1.000	0.455	0.999	1.000
TPSA (target: 150.0)	**1.000**	**1.000**	0.938	**1.000**	1.000	0.757	0.980	1.000
CNS MPO	**1.000**	**1.000**	**1.000**	**1.000**	1.000	**1.000**	1.000	1.000
QED	**0.948**	0.939	0.932	**0.948**	0.948	0.946	0.947	0.948
C7H8N2O2	0.996	**1.000**	0.607	**1.000**	1.000	0.649	0.601	1.000
Pioglitazone MPO	0.986	0.887	0.732	**0.995**	1.000	0.344	0.668	0.986
Total score	6.930	6.826	6.067	**6.943**	6.948	5.049	6.194	6.902

Table 7.2: Scores for the five models assessed on trivial benchmarks from GuacaMol. MSO and BO are initiated using randomly selected molecules from the training dataset. The highest score for every benchmark is indicated with bold digits. The MSO* and BO* columns display results when the models are initiated using top-scoring molecules as described in Section 7.1.1. If the highest score of a benchmark is matched or improved this is indicated with an underline.

7.1.1 Lead Optimization

To investigate the impact of initiating the swarms of MSO and the prior distribution of BO at high scoring molecules, we ran the benchmarks as a lead-optimization task for both models. The swarms of MSO were initiated at the 40 best molecules for each benchmark, and the prior distribution of BO was created using the top-1000 molecules. We present the results in table 7.1 and 7.2 in the columns titled *MSO** and *BO**.

While BO only manages to match or surpass the dataset max for two benchmarks, MSO shows a distinct improvement. By starting from a collection of already relatively high scoring molecules, the swarms managed to exploit areas of the chemical space not reached before. In figure 7.1, we compare the swarms' performance on three benchmarks with respect to their starting conditions.



(a) Amlodipine MPO **(b)** Albuterol similarity **(c)** Aripiprazole similarity

Figure 7.1: Scatter plot of the initial best score of swarms in relation to the final best score. A swarms position at the x-axis represents its best score at initialization, and its position on the y-axis represents the score of the best particle found during the optimization. The blue dots represent swarms initiated at randomly selected points. The orange dots represent swarms initiated at the 40 best molecules in the training dataset for the respective benchmark. The further up to the left a swarm is positioned, the better it performed since this indicates a low starting score but a high final score.

LatentGAN, GENTRL and REINVENT were not submitted to lead-optimization

experiments. REINVENT were left out due to it already outperforming the other models with a margin. GENTRL, while utilizing reinforcement learning, lack a mechanism that enables lead optimization in the same explicit manner as MSO and BO. LatentGAN has already been exposed to the top-scoring molecules and has no way of modifying its own distribution further.

7.1.2 Quality of Generated Molecules

After evaluating each model on the 27 benchmarks, the top 100 compounds from each benchmark were extracted and assessed using the quality measurements described in Section 5.1.4. That is, each model is evaluated based on its 2700 best molecules. The results are shown in table 7.2 where REINVENT reports an 85% ratio of molecules passing the quality filters. LatentGAN reports 70% while the other models all report ratios around 60%. While REINVENT and MSO produced the best results using the 27 benchmarks, this indicates that REINVENT generated higher-quality compounds.

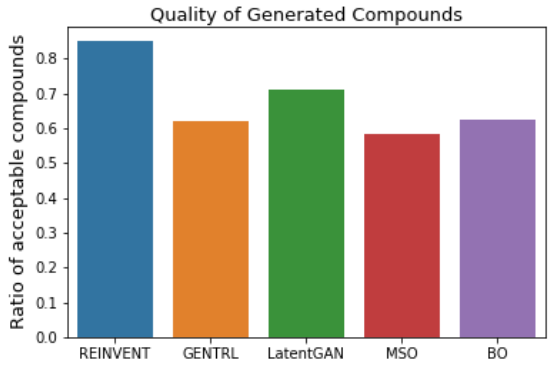


Figure 7.2: Quality measurements of the top-100 molecules generated for each benchmark

We also conducted a separate analysis of the validity, uniqueness and novelty of the generated molecules. This analysis was based on the top 10,000 best molecules for every model, with the exception of BO that generated 1000 molecules (or less if the optimization reached 2000 steps). As seen in table 7.3, all models report a high percentage of valid molecules with low variance. LatentGAN and MSO excel in terms of generating unique molecules. The low ration of unique molecules generated by REINVENT might seem contradictory to its performance on the benchmarks. However, it should once again be noted that these figures are based on the best 10,000 molecules for every model, while the benchmark scores take no more than 250 molecules into account. We see that the novelty ratios match the uniqueness ratios, both in terms of mean value and variance.

Properties of the generated molecules			
Model	Validity $\mu \pm \sigma$	Uniqueness $\mu \pm \sigma$	Novelty $\mu \pm \sigma$
BO	0.996 ± 0.002	0.876 ± 0.164	0.857 ± 0.163
BO (Lead optimization)	0.997 ± 0.002	0.860 ± 0.199	0.840 ± 0.201
GENTRL	0.967 ± 0.170	0.578 ± 0.262	0.577 ± 0.263
LatentGAN	1.000 ± 0.000	0.941 ± 0.074	0.933 ± 0.077
MSO	0.999 ± 0.001	0.999 ± 0.001	0.998 ± 0.002
MSO (Lead optimization)	1.000 ± 0.000	1.000 ± 0.000	0.997 ± 0.003
REINVENT	0.998 ± 0.009	0.446 ± 0.340	0.444 ± 0.337

Table 7.3: The validity, uniqueness and novelty of the 10,000 SMILES generated for each of the objective functions (with the exception for Bayesian Optimization). LatentGAN has functionality in place which prevents it from sampling invalid molecules, hence the validity of 1.

7.2 DRD2 Activity Experiments

Here we present the results from the experiments using the multi-objective function described in Section 5.2. The experiments were conducted using three different scaffolds as substructure target. We present the results in three subsections, one for each scaffold. The results are based on the 10,000 best molecules obtained when optimizing using REINVENT and MSO. BO is assessed using all samples of an optimization run conducted in 2000 steps or until 1000 unique molecules have been sampled.

7.2.1 Scaffold 1

The first experiment aimed to find active compounds with the scaffold seen in figure 7.3 present. As the distribution plot in figure 7.4 shows, REINVENT manages to generate high-quality compounds with small deviations. MSO also manages to generate several high scoring molecules but fail to exploit the high scoring areas to the same extent as REINVENT. GENTRL, LatentGAN and BO produce mainly low scoring molecules. While LatentGAN does this exclusively, GENTRL and BO do generate a small number of relatively high scoring compounds.

By looking at the score breakdown in table 7.4 we see that GENTRL, on average, generates molecules with higher probability of binding to DRD2 and that have a better QED score than MSO. However, since it struggles to generate compounds containing the requested scaffold, the mean score in total is heavily penalized.

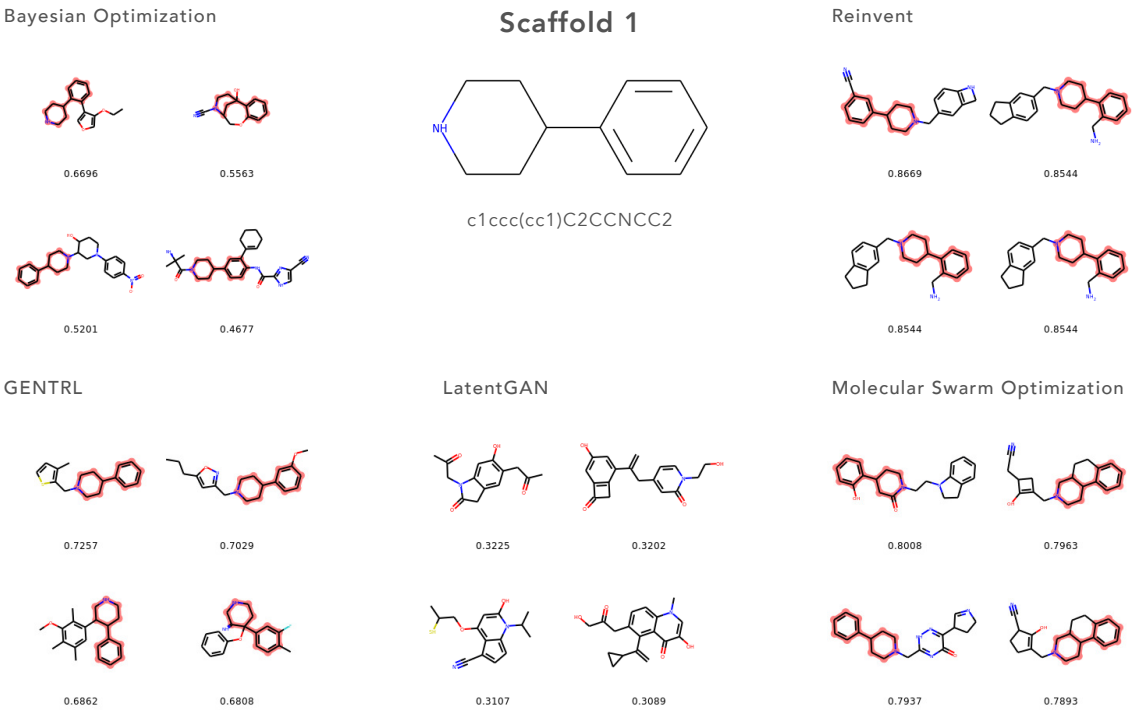


Figure 7.3: The top four molecules from each model optimizing the DRD2 benchmark using scaffold 1. The number under each molecule represents its score. If the substructure is present, it is highlighted with red in the 2D molecular graph.

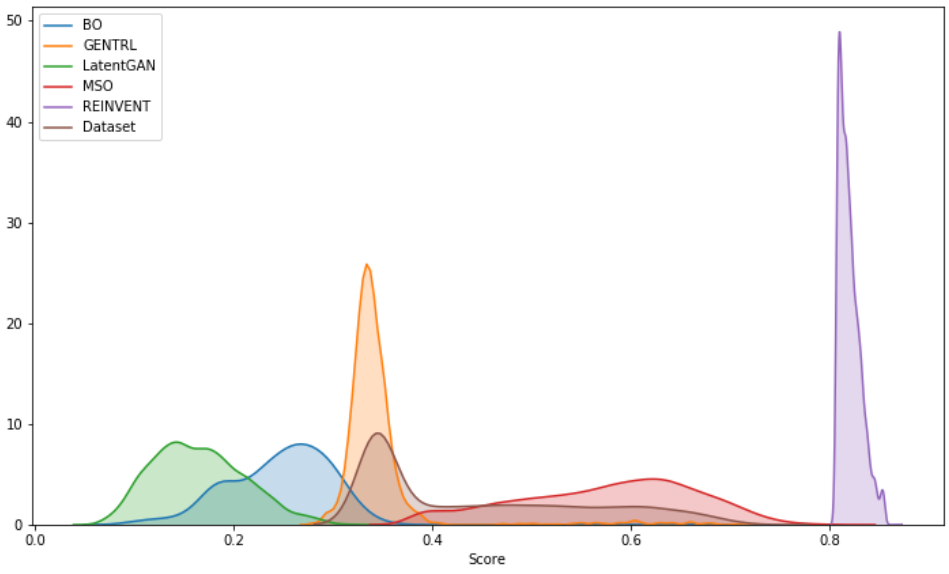


Figure 7.4: Score distributions of the models when optimizing DRD2 with scaffold 1 as target substructure

Score Breakdown for DRD2 MPO for Scaffold 1						
Model	Max Score	Score $\mu \pm \sigma$	CA	MS	DRD2 $\mu \pm \sigma$	QED $\mu \pm \sigma$
BO	0.670	0.166 ± 0.124	0.671	0.501	0.349 ± 0.076	0.616 ± 0.183
GENTRL	0.726	0.345 ± 0.049	1.000	0.515	0.467 ± 0.074	0.878 ± 0.052
LatentGAN	0.322	0.106 ± 0.087	0.641	0.500	0.279 ± 0.041	0.370 ± 0.164
MSO	0.801	0.575 ± 0.089	1.000	0.991	0.428 ± 0.091	0.736 ± 0.140
REINVENT	0.867	0.820 ± 0.011	1.000	1.000	0.722 ± 0.027	0.918 ± 0.019
ChEMBL	0.721	0.444 ± 0.111	1.000	0.805	0.428 ± 0.062	0.728 ± 0.203

Table 7.4: Score breakdown of generated molecules when optimizing DRD2 with scaffold 1 as target substructure. The score is a combination of custom alerts (CA), matching substructure (MS), DRD2 binding and drug-likeness (QED). If CA equals 1 none of the generated molecules contained unwanted substructures.

7.2.2 Scaffold 2

In this experiment the substructure target was the more generic scaffold displayed in figure 7.5, easing the constraints for the models, such that high scoring molecules can be found in a larger subspace of the chemical space. However, the resulting distributions, shown in figure 7.6, do not change notably and REINVENT still outperforms the other models with a margin. MSO does find a higher percentage of high scoring molecules compared to when queried for the first scaffold, but lack in its ability to generate compounds that score high on the DRD2 and QED components, as seen in table 7.5. GENTRL is once again penalized for not generating the target scaffold to a large enough extent, which also seem to be the most challenging task for BO and LatentGAN.

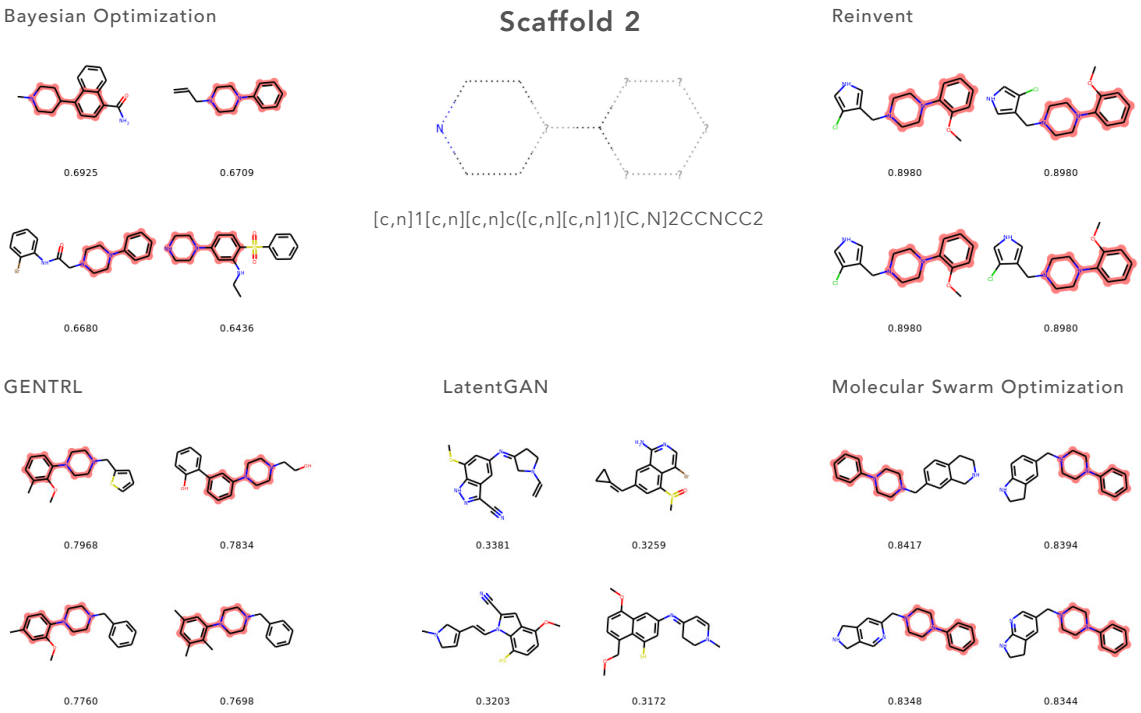


Figure 7.5: The top four molecules from each model optimizing the DRD2 benchmark using scaffold 2. The number under each molecule represents its score. If the substructure is present, it is highlighted with red in the 2D molecular graph.

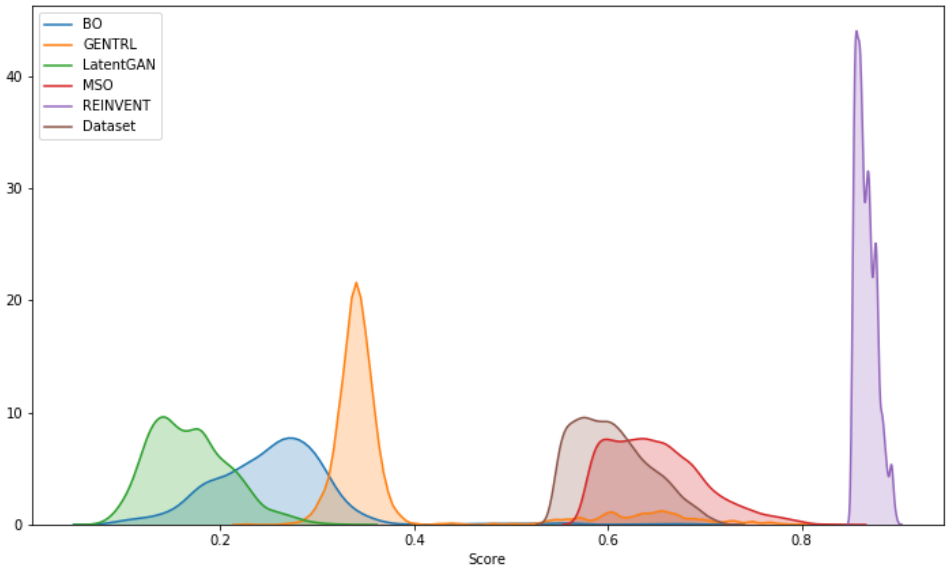


Figure 7.6: Score distributions of the models when optimizing DRD2 with scaffold 2 as target substructure

Score Breakdown for DRD2 MPO for Scaffold 2						
Model	Max Score	Score $\mu \pm \sigma$	CA	MS	DRD2 $\mu \pm \sigma$	QED $\mu \pm \sigma$
BO	0.692	0.168 ± 0.131	0.663	0.508	0.348 ± 0.079	0.625 ± 0.185
GENTRL	0.797	0.382 ± 0.107	1.000	0.573	0.471 ± 0.075	0.874 ± 0.064
LatentGAN	0.338	0.097 ± 0.089	0.579	0.500	0.286 ± 0.039	0.365 ± 0.153
MSO	0.842	0.649 ± 0.048	1.000	1.000	0.484 ± 0.091	0.815 ± 0.078
REINVENT	0.898	0.866 ± 0.010	1.000	1.000	0.814 ± 0.027	0.919 ± 0.023
ChEMBL	0.721	0.605 ± 0.038	1.000	1.000	0.413 ± 0.058	0.796 ± 0.076

Table 7.5: Score breakdown of generated molecules when optimizing DRD2 with scaffold 2 as target substructure. The score is a combination of custom alerts (CA), matching substructure (MS), DRD2 binding and drug-likeness (QED). If CA equals 1 none of the generated molecules contained unwanted substructures.

7.2.3 Scaffold 3

In the final experiment, the models attempt to generate compounds with the scaffold shown in figure 7.7 present. This task proved to be more challenging for REINVENT, generating a distribution centred at 0.432, as seen in figure 7.8. MSO generates top scoring molecules that match REINVENT, but with a distribution centred at a slightly lower value, 0.366. REINVENT and MSO are the only models that manages to generate a small number of molecules containing the target scaffold. The full score breakdown is presented in table 7.6.

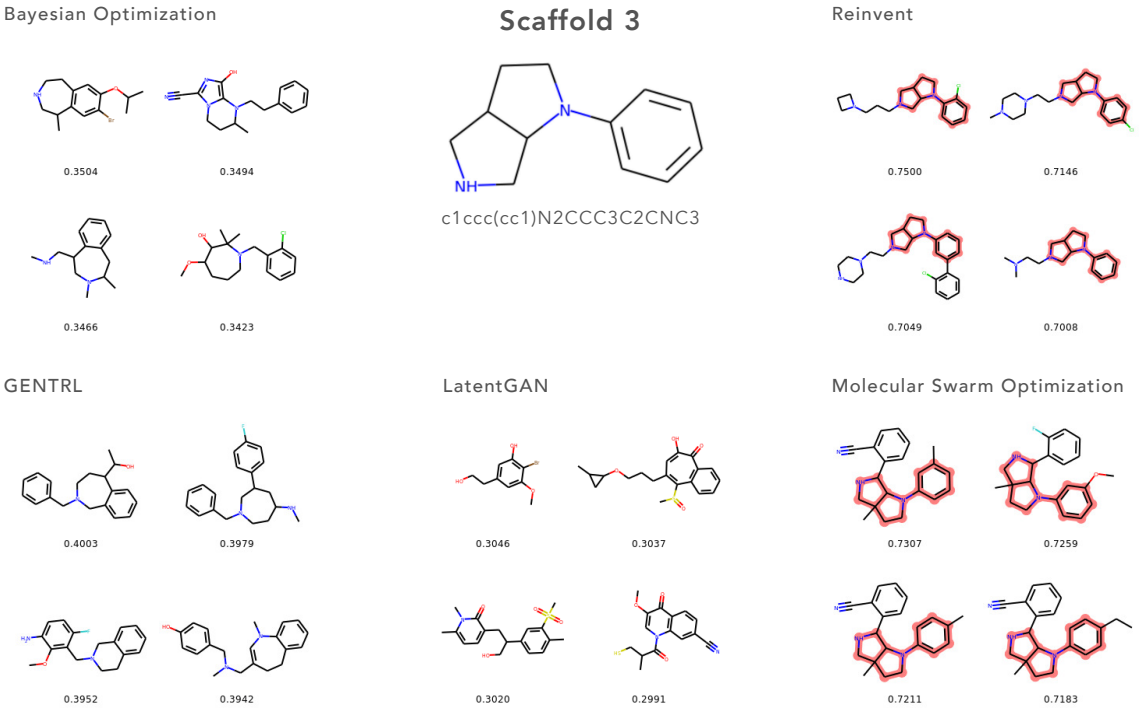


Figure 7.7: The top four molecules from each model optimizing the DRD2 benchmark using scaffold 3. The number under each molecule represents its score. If the substructure is present, it is highlighted with red in the 2D molecular graph.

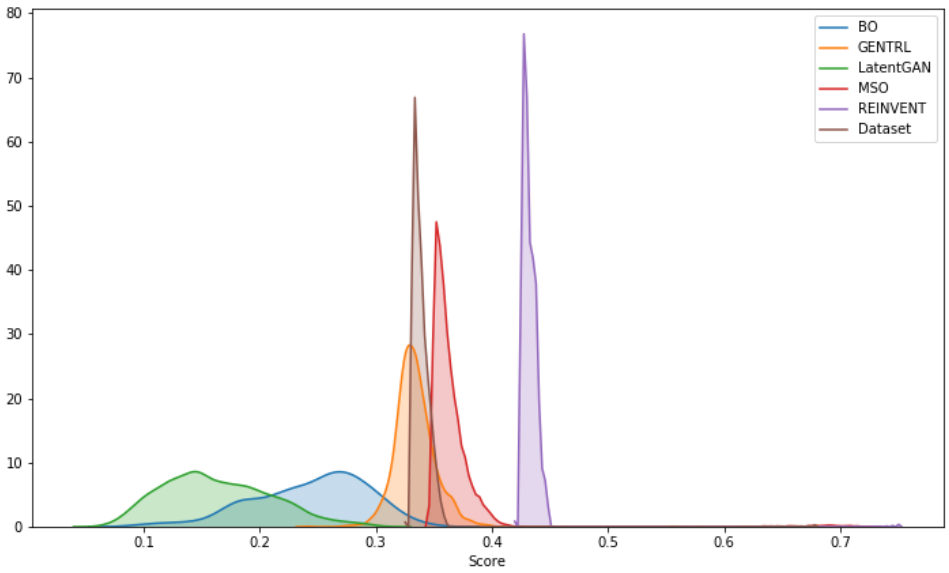


Figure 7.8: Score distributions of the models when optimizing DRD2 with scaffold 3 as target substructure

Score Breakdown for DRD2 MPO for Scaffold 3						
Model	Max Score	Score $\mu \pm \sigma$	CA	MS	DRD2 $\mu \pm \sigma$	QED $\mu \pm \sigma$
BO	0.350	0.165 ± 0.121	0.677	0.500	0.346 ± 0.074	0.616 ± 0.182
GENTRL	0.400	0.334 ± 0.017	1.000	0.500	0.453 ± 0.071	0.882 ± 0.047
LatentGAN	0.305	0.102 ± 0.087	0.625	0.500	0.279 ± 0.040	0.364 ± 0.164
MSO	0.731	0.366 ± 0.035	1.000	0.506	0.557 ± 0.058	0.892 ± 0.042
REINVENT	0.750	0.432 ± 0.009	1.000	0.500	0.814 ± 0.029	0.911 ± 0.028
ChEMBL	0.677	0.338 ± 0.014	1.000	0.501	0.455 ± 0.031	0.894 ± 0.032

Table 7.6: Score breakdown of generated molecules when optimizing DRD2 with scaffold 3 as target substructure. The score is a combination of custom alerts (CA), matching substructure (MS), DRD2 binding and drug-likeness (QED). If CA equals 1 none of the generated molecules contained unwanted substructures.

8

Discussion

8.1 Performance of models

The combined results from the GuacaMol and DRD2 experiments show that REINVENT outperformed the other models with distinction. We think the model's primary strength is its ability to adapt its generative strategy. While MSO shows promise in its ability to traverse a continuous representation of the chemical space to find desirable compounds, it does not match the performance of REINVENT when it comes to exploiting these high scoring areas efficiently. This restriction can be seen in the DRD2 experiments where MSO is able to find relatively high scoring molecules but fails to converge its particles in those areas. However, we think that there is reason to believe that further research into more sophisticated applications of MSO would be fruitful. We also hypothesize that MSO, BO and GENTRL are constrained by their underlying generative model, the variational autoencoder. While REINVENT manipulates the probability distribution during optimization, the distribution of the autoencoder is fixed throughout the whole process. So while the algorithms move in the chemical space in a direction of improved scores, the highest scores might be unreachable due to the decoder having a distribution that has a very small chance of sampling the very best compounds.

The poor performance of Bayesian Optimization was evident throughout the whole project. Regardless of thorough experimentation with different kernels, acquisition functions and likelihood variance, the results showed no significant improvements. The model showed some promise on the trivial GuacaMol benchmarks, especially when used in a lead optimization context. This is in line with previously reported results where Bayesian Optimization has been successfully applied to sample molecules using simple, single-property, functions. However, to the best of our knowledge, there are no reported results using Bayesian Optimization on more complicated, multi-objective, optimization tasks. This, in combination with our restricted approach to hyperparameter tuning, makes the reported results on Bayesian Optimization harder to interpret with confidence. We suggest that the results of Bayesian Optimization are used as a baseline for future research on the method, rather than as an absolute truth of its capabilities.

GENTRL was another model that performed below expectation, given the results the authors got in the original paper. It is important to note that the datasets and reward functions mentioned in the paper are not available in the code published on

GitHub. By not being able to reproduce their results, we could not confirm that our implementation was done correctly and that the code did not contain bugs. In addition, it is possible that the more advanced reward function described in the paper could have yielded better results on the benchmarks.

Finally, LatentGAN could not generate molecules that scored higher than the maximum scores found in the datasets. This was expected since it is the only model that lacks an optimization mechanism. In the original paper, LatentGAN found several molecules that were predicted to bind to DRD2, while in our evaluation none were found. This could be explained by the fact that we removed all molecules that had a probability of binding greater than 0.5, while in the original paper, only a set of known actives were removed. This suggests that a generative model without an optimization algorithm must be trained on data containing the desired properties. Though its results lacked high scores, it showed an ability to generate diverse sets of valid molecules.

8.2 Future research

MSO showed promise as a lightweight, easy to use, lead optimization strategy. In this work, we experimented with time-varying hyperparameters, as explained in Section 6.3.5, which slightly increased the performance of the model on the DRD2 experiments. This indicates that further research into more sophisticated usages of the hyperparameters could improve the performance of the model. The DRD2 experiments showed that MSO indeed found high scoring molecules but failed to properly exploit the areas containing them. We hypothesise that one could, for example, implement a mechanism in the algorithm such that the particles of a swarm more efficiently converge at the most promising areas towards the end of an optimization run.

One of the features of REINVENT is its usage of data augmentation by randomizing SMILES when training the prior state of the generative model. We believe it would be interesting to evaluate how the same approach would impact the performance of the other models. The results of such experiments could be used to shed light on our hypothesis of MSO and BO being constrained by the underlying generative model. Furthermore, the DRD2 experiments showed that there could be benefits of using two or more of the models in tandem, to properly explore the chemical space. The models generated potentially active compounds that were unique for each generative model, and were not produced by the other models. This indicates that the models can be used in a complementary manner to cover a more substantial portion of the space.

9
Conclusion

In this thesis, a comparative study of five different optimization algorithms for de novo drug design is presented. The investigated models, REINVENT, LatentGAN, Molecule Swarm Optimization, GENTRL and Bayesian Optimization, have been evaluated using a standardized benchmark framework, GuacaMol, and a multi-property function that encourages the models to generate molecules likely to bind to the dopamine receptor D2.

The results clearly highlight the merits of a model called REINVENT, which employs a recurrent neural network and reinforcement learning to sample high-scoring molecular data from a probability distribution. The approach excels both in terms of its ability to generate high-scoring molecules but also efficiently exploring promising regions of the chemical space.

An application of particle swarm optimization, on a latent representation of the chemical space, Molecule Swarm Optimization, also report good results. While it produces high-scoring molecules, it is not with the same diversity nor in the same numbers as REINVENT. The model had an increased performance when initiated at high-scoring molecules, indicating applicability in lead-optimization tasks. Based on this, suggestions on possible directions of future research are provided.

The three other models in this study report subpar results, when compared to the two aforementioned models. In the case of LatentGAN, a model employing a generative adversarial network in latent chemical space, this might be due to the strict training setup that was imposed on the underlying autoencoder, which excluded all high-scoring compounds from the training dataset. Bayesian Optimization applied to a latent chemical space displays results on par with results reported in the literature on simpler, single-property, tasks but underperforms on more complex objectives.

The last model, GENTRL, uses a variational autoencoder combined with reinforcement learning. While the model's authors report promising data in their paper, it has not been possible to reproduce the reported results since not all of the relevant data is publicly available. As a result, the correctness of the used implementation has not been verified. We hypothesize that Bayesian Optimization and GENTRL have the potential to perform better on the tasks in this work than what is reported.

All code can be found in the GitHub repository: `https://github.com/sebastiandro/`

`de-novo-evaluation`.

Bibliography

[1] Gisbert Schneider and Uli Fechner. "Computer-based de novo design of drug-like molecules". In: *Nature Reviews Drug Discovery* 4.8 (2005), pp. 649–663. ISSN: 14741776. DOI: 10.1038/nrd1799.

[2] P. G. Polishchuk, T. I. Madzhidov, and A. Varnek. "Estimation of the size of drug-like chemical space based on GDB-17 data". In: *Journal of Computer-Aided Molecular Design* 27.8 (2013), pp. 675–679. ISSN: 0920654X. DOI: 10.1007/s10822-013-9672-4.

[3] Lucas G. Viviani et al. "Virtual Screening Approach for the Identification of Hydroxamic Acids as Novel Human Ecto-5-Nucleotidase Inhibitors". In: *Journal of Chemical Information and Modeling* 60.2 (2020), pp. 621–630. ISSN: 15205142. DOI: 10.1021/acs.jcim.9b00884.

[4] Thomas Scior et al. "Recognizing pitfalls in virtual screening: A critical review". In: *Journal of Chemical Information and Modeling* 52.4 (2012), pp. 867–881. ISSN: 15499596. DOI: 10.1021/ci200528d.

[5] J. P. Hughes et al. "Principles of early drug discovery". In: *British Journal of Pharmacology* 162.6 (2011), pp. 1239–1249. ISSN: 00071188. DOI: 10.1111/j.1476-5381.2010.01127.x.

[6] Steven M. Paul et al. "How to improve RD productivity: The pharmaceutical industry's grand challenge". In: *Nature Reviews Drug Discovery* 9.3 (2010), pp. 203–214. ISSN: 14741776. DOI: 10.1038/nrd3078.

[7] Hongming Chen et al. *The rise of deep learning in drug discovery*. 2018. DOI: 10.1016/j.drudis.2018.01.039.

[8] David Weininger. "SMILES, a Chemical Language and Information System: 1: Introduction to Methodology and Encoding Rules". In: *Journal of Chemical Information and Computer Sciences* 28.1 (1988), pp. 31–36. ISSN: 00952338. DOI: 10.1021/ci00057a005.

[9] *OpenSMILES specification*. URL: http://opensmiles.org/opensmiles.html.

[10] Tony Jebara. "Discriminative , Generative and Imitative Learning". In: *Media* (2002), pp. 1–212. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.8.2731&rep=rep1&type=pdf.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016, pp. 367–415.

[12] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks". In: *Advances in Neural Information Processing Systems* 4.January (2014), pp. 3104–3112. ISSN: 10495258.

[13] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. “Speech Recognition With Deep Recurrent Neural Networks”. In: 3 ().

[14] B Hidasi et al. “Session-based Recommendations With Recurrent Neural Networks”. In: (2016), pp. 1–10.

[15] Alex Graves. *Generating Sequences With Recurrent Neural Networks*. Tech. rep.

[16] Mark A. Kramer. “Nonlinear principal component analysis using autoassociative neural networks”. In: *AIChE Journal* 37.2 (1991), pp. 233–243. ISSN: 15475905. DOI: 10.1002/aic.690370209.

[17] James M Joyce. “Kullback-Leibler Divergence”. In: *International Encyclopedia of Statistical Science*. Ed. by Miodrag Lovric. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 720–722. ISBN: 978-3-642-04898-2. DOI: 10.1007/978-3-642-04898-2{_}327. URL: https://doi.org/10.1007/978-3-642-04898-2_327.

[18] Rafael Gómez-Bombarelli et al. “Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules”. In: *ACS Central Science* 4.2 (Feb. 2018), pp. 268–276. ISSN: 23747951. DOI: 10.1021/acscentsci.7b00572.

[19] Alex Zhavoronkov et al. “Deep learning enables rapid identification of potent DDR1 kinase inhibitors”. In: *Nature Biotechnology* 37.9 (Sept. 2019), pp. 1038–1040. ISSN: 15461696. DOI: 10.1038/s41587-019-0224-x.

[20] Ian J. Goodfellow et al. “Generative adversarial nets”. In: *Advances in Neural Information Processing Systems* 3.January (2014), pp. 2672–2680. ISSN: 10495258. DOI: 10.3156/jsoft.29.5{_}177{_}2.

[21] Morton D. Davis and Steven J. Brams. *Game theory*. 2016. URL: https://www.britannica.com/science/game-theory/Two-person-constant-sum-games#ref22617.

[22] Artur Kadurin et al. “druGAN: An Advanced Generative Adversarial Autoencoder Model for de Novo Generation of New Molecules with Desired Molecular Properties in Silico”. In: *Molecular Pharmaceutics* 14.9 (Sept. 2017), pp. 3098–3104. ISSN: 1543-8384. DOI: 10.1021/acs.molpharmaceut.7b00346. URL: https://doi.org/10.1021/acs.molpharmaceut.7b00346.

[23] Lingling Zhao et al. “GANsDTA: Predicting Drug-Target Binding Affinity Using GANs”. In: *Frontiers in Genetics* 10.January (2020), pp. 1–8. ISSN: 16648021. DOI: 10.3389/fgene.2019.01243.

[24] Łukasz Maziarka et al. “Mol-CycleGAN: A generative model for molecular optimization”. In: *Journal of Cheminformatics* 12.1 (2020), pp. 1–18. ISSN: 17582946. DOI: 10.1186/s13321-019-0404-1. URL: https://doi.org/10.1186/s13321-019-0404-1.

[25] Carl Edward Rasmussen. “Gaussian Processes in Machine Learning”. In: *Advanced Lectures on machine Learning*. Vol. 3176. 2004, pp. 69–71. ISBN: 978-3-540-28650-9. DOI: 10.1007/978-3-540-28650-9{_}8.

[26] Eric Brochu, Vlad M. Cora, and Nando de Freitas. “A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning”. In: (2010). URL: http://arxiv.org/abs/1012.2599.

[27] Mark Ebden. “Gaussian Processes: A Quick Introduction”. In: August (2015). URL: http://arxiv.org/abs/1505.02965.

[28] A. Candelieri, R. Perego, and F. Archetti. “Bayesian optimization of pump operations in water distribution systems”. In: *Journal of Global Optimization* 71.1 (2018), pp. 213–235. ISSN: 15732916. DOI: 10.1007/s10898-018-0641-2. URL: https://doi.org/10.1007/s10898-018-0641-2.

[29] Donald R Jones, Matthias Schonlau, and W. J. Welch. “Efficient Global Optimization of Expensive Black-Box Functions," , vol. 13, no. 4, pp. 455-492, 1998.” In: *Journal of Global Optimization* 13 (1998), pp. 455–492. URL: https://link.springer.com/content/pdf/10.1023%2FA%3A1008306431147.pdf.

[30] H. J. Kushner. “A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise”. In: *Journal of Fluids Engineering, Transactions of the ASME* 86.1 (1964), pp. 97–106. ISSN: 1528901X. DOI: 10.1115/1.3653121.

[31] Philipp Hennig and Christian J. Schuler. “Entropy search for information-efficient global optimization”. In: *Journal of Machine Learning Research* 13 (2012), pp. 1809–1837. ISSN: 15324435.

[32] Riccardo Poli, James Kennedy, and Tim Blackwell. “Particle swarm optimization: An overview”. In: *Swarm Intelligence* 1.1 (2007), pp. 33–57. ISSN: 1935-3812. DOI: 10.1007/s11721-007-0002-0.

[33] Shigenori Naka et al. “Practical distribution state estimation using hybrid particle swarm optimization”. In: *Proceedings of the IEEE Power Engineering Society Transmission and Distribution Conference* 2.WINTER MEETING (2001), pp. 815–820. DOI: 10.1109/pesw.2001.916969.

[34] Y. Shi Eberhart and R. C. “Empirical study of particle swarm optimization, “in Proc”. In: *Evolutionary Comput* (1995), pp. 1942–1948.

[35] Saptarshi Sengupta, Sanchita Basak, and Richard Peters. “Particle Swarm Optimization: A Survey of Historical and Recent Developments with Hybridization Perspectives”. In: *Machine Learning and Knowledge Extraction* 1.1 (2018), pp. 157–191. DOI: 10.3390/make1010010.

[36] R. C. Eberhart and Y. Shi. “Tracking and optimizing dynamic systems with particle swarms”. In: *Proceedings of the IEEE Conference on Evolutionary Computation, ICEC* 1 (2001), pp. 94–100. DOI: 10.1109/cec.2001.934376.

[37] Asanga Ratnaweera, Saman K. Halgamuge, and Harry C. Watson. “Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients”. In: *IEEE Transactions on Evolutionary Computation* 8.3 (2004), pp. 240–255. ISSN: 1089778X. DOI: 10.1109/TEVC.2004.826071.

[38] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. 2nd ed. Westchester Publishing Services, 2018. ISBN: 9780262039246.

[39] Robin Winter et al. “Learning continuous and data-driven molecular descriptors by translating equivalent chemical representations”. In: *Chemical Science* 10.6 (2019), pp. 1692–1701. ISSN: 20416539. DOI: 10.1039/c8sc04175j.

[40] Ziyu Wang et al. “Bayesian Optimization in High Dimensions via Random Embeddings”. In: *Twenty-Third International Joint Conference on Artificial Intelligence Bayesian* (2012), pp. 1778–1784. URL: https://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/download/6971/6964.

[41] Marcus Olivecrona et al. “Molecular de-novo design through deep reinforcement learning”. In: *Journal of Cheminformatics* 9.1 (Sept. 2017). ISSN: 17582946. DOI: 10.1186/s13321-017-0235-x.

[42] Esben Jannik Bjerrum and Boris Sattarov. “Improving chemical autoencoder latent space and molecular de novo generation diversity with heteroencoders”. In: *Biomolecules* 8.4 (2018), pp. 1–17. ISSN: 2218273X. DOI: 10.3390/biom8040131.

[43] I. V. Oseledets. “Tensor-train decomposition”. In: *SIAM Journal on Scientific Computing* 33.5 (2011), pp. 2295–2317. ISSN: 10648275. DOI: 10.1137/090752286.

[44] Stephan Rabanser, Oleksandr Shchur, and Stephan Günnemann. “Introduction to Tensor Decompositions and their Applications in Machine Learning”. In: (2017), pp. 1–13. URL: http://arxiv.org/abs/1711.10781.

[45] Oleksii Prykhodko et al. “A de novo molecular generation method using latent vector based generative adversarial network”. In: *Journal of Cheminformatics* 11.1 (Dec. 2019). ISSN: 17582946. DOI: 10.1186/s13321-019-0397-9.

[46] Nathan Brown et al. “GuacaMol: Benchmarking Models for de Novo Molecular Design”. In: *Journal of Chemical Information and Modeling* 59.3 (Mar. 2019), pp. 1096–1108. ISSN: 15205142. DOI: 10.1021/acs.jcim.8b00839.

[47] Camille Wermuth et al. *Practice of Medicinal Chemistry (4th Edition)*. Second edi. Elsevier, 2015, p. 94. URL: https://app.knovel.com/hotlink/pdf/id:kt00UCXH02/practice-medicinal-chemistry/virtual-screening.

[48] P. Ertl, B. Rohde, and P. Selzer. “Fast calculation of molecular polar surface area as a sum of fragment-based contributions and its application to the prediction of drug transport properties”. In: *Journal of Medicinal Chemistry* 43.20 (2000), pp. 3714–3717. ISSN: 00222623. DOI: 10.1021/jm000942e.

[49] V. J. Gillet and A. R. Leach. “Chemoinformatics”. In: *Comprehensive Medicinal Chemistry II* 3 (2006), pp. 235–264. ISSN: 0040-1706. DOI: 10.1081/e-elis3-120043664.

[50] Sanjivanjit Bhal. “Log P — Making Sense of the Value”. In: *Advanced Chemistry Development* (2007), pp. 1–4. URL: https://www.acdlabs.com/download/app/physchem/making_sense.pdf.

[51] Eric P. Gillis et al. “Applications of Fluorine in Medicinal Chemistry”. In: *Journal of Medicinal Chemistry* 58.21 (2015), pp. 8315–8359. ISSN: 15204804. DOI: 10.1021/acs.jmedchem.5b00258.

[52] Simon E Ward et al. “Expert Opinion on Drug Discovery What does the aromatic ring number mean for drug design ? What does the aromatic ring number mean for drug design ?” In: 0441 (2014). DOI: 10.1517/17460441.2014.932346.

[53] Steven H. Bertz. “The First General Index of Molecular Complexity”. In: *Journal of the American Chemical Society* 103.12 (1981), pp. 3599–3601. ISSN: 15205126. DOI: 10.1021/ja00402a071.

[54] Suqing Zheng et al. “Proposed Hydrogen-Bonding Index of Donor or Acceptor Reflecting Its Intrinsic Contribution to Hydrogen-Bonding Strength”. In: *Journal of Chemical Information and Modeling* 57.7 (2017), pp. 1535–1547. ISSN: 15205142. DOI: 10.1021/acs.jcim.7b00022.

[55] Adam Todd, Roz Anderson, and Paul W. Groundwater. "Rational drug design - Designing a molecule that binds to a target". In: *Pharmaceutical Journal* 283.7563 (2009), pp. 131–132. ISSN: 00316873.

[56] G Richard Bickerton et al. "Quantifying the chemical beauty of drugs". In: *Nat Chem* 4.2 (2012), pp. 90–98. DOI: 10.1038/nchem.1243. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3524573/pdf/emss-50746.pdf.

[57] Daniel F. Veber et al. "Molecular properties that influence the oral bioavailability of drug candidates". In: *Journal of Medicinal Chemistry* 45.12 (2002), pp. 2615–2623. ISSN: 00222623. DOI: 10.1021/jm020017n.

[58] Pat Walters. *rd_filters*. 2019. URL: https://github.com/PatWalters/rd_filters.

[59] Edoardo R. de Natale and Marios Politis. "Imaging in Movement Disorders: Imaging Methodology and Applications in Parkinson's Disease". In: *International Review of Neurobiology,* (2018).

[60] Fabian Pedregosa et al. "Scikit-learn: Machine learning in Python". In: *Journal of Machine Learning Research* 12.January (2011), pp. 2825–2830. ISSN: 15324435.

[61] Jiangming Sun et al. "ExCAPE-DB: An integrated large scale dataset facilitating Big Data analysis in chemogenomics". In: *Journal of Cheminformatics* 9.1 (2017), pp. 1–9. ISSN: 17582946. DOI: 10.1186/s13321-017-0203-5.

[62] Ansgar Schuffenhauer et al. "The scaffold tree - Visualization of the scaffold universe by hierarchical scaffold classification". In: *Journal of Chemical Information and Modeling* 47.1 (2007), pp. 47–58. ISSN: 15499596. DOI: 10.1021/ci600338x.

[63] *GitHub - BenevolentAI/guacamol: Benchmarks for generative chemistry*. URL: https://github.com/BenevolentAI/guacamol.

[64] David Mendez et al. "ChEMBL: Towards direct deposition of bioassay data". In: *Nucleic Acids Research* 47.D1 (2019), pp. D930–D940. ISSN: 13624962. DOI: 10.1093/nar/gky1075.

[65] Philipp Probst, Anne Laure Boulesteix, and Bernd Bischl. "Tunability: Importance of hyperparameters of machine learning algorithms". In: *Journal of Machine Learning Research* 20 (2019), pp. 1–32. ISSN: 15337928.

[66] Panagiotis-Christos Kotsias et al. "Direct Steering of de novo Molecular Generation using Descriptor Conditional Recurrent Neural Networks (cRNNs)". In: 1 (2019). DOI: 10.26434/CHEMRXIV.9860906.V1.

[67] Alexander G De et al. "GPflow: A Gaussian Process Library using TensorFlow Mark van der Wilk". In: *Journal of Machine Learning Research* 18 (2017), pp. 1–6. URL: http://jmlr.org/papers/v18/16-537.html..

[68] Nicolas Knudde et al. "GPflowOpt: A Bayesian Optimization Library using TensorFlow". In: (2017), pp. 0–1. URL: http://arxiv.org/abs/1711.03845.

[69] Pauli Virtanen et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python". In: *Nature Methods* 17.3 (2020), pp. 261–272. ISSN: 15487105. DOI: 10.1038/s41592-019-0686-2.

[70] Stacey S. Cherny. "Cholesky Decomposition". In: *Wiley StatsRef: Statistics Reference Online* (2014). DOI: 10.1002/9781118445112.stat06454.

[71] Ruth Brenk et al. “Lessons learnt from assembling screening libraries for drug discovery for neglected diseases”. In: *ChemMedChem* 3.3 (2008), pp. 435–444. ISSN: 18607179. DOI: 10.1002/cmdc.200700139.

A

Appendix - GuacaMol

A.1 SMILES and SMARTS abbreviations

- s_1 = `CN(C=O)Cc1ccc(c2ccccc2)cc1`
- s_2 = `[#7]-c1n[c;h1]nc2[c;h1]c(-[#8])[c;h0][c;h1]-c12`
- s_3 = `[#7]-c1ccc2ncsc2c1`
- s_4 = `CS([#6])(=O)=O`
- s_5 = `CCCOc1cc2ncnc(Nc3ccc4ncsc4c3)c2cc1S(=O)(=O)C-(C)(C)C`
- s_6 = `[#6]-[#6]-[#6]-[#8]-[#6][#6][#6][#6][#6]-[#7]-c1ccc2ncsc2c1`

A.2 GuacaMol score modifiers

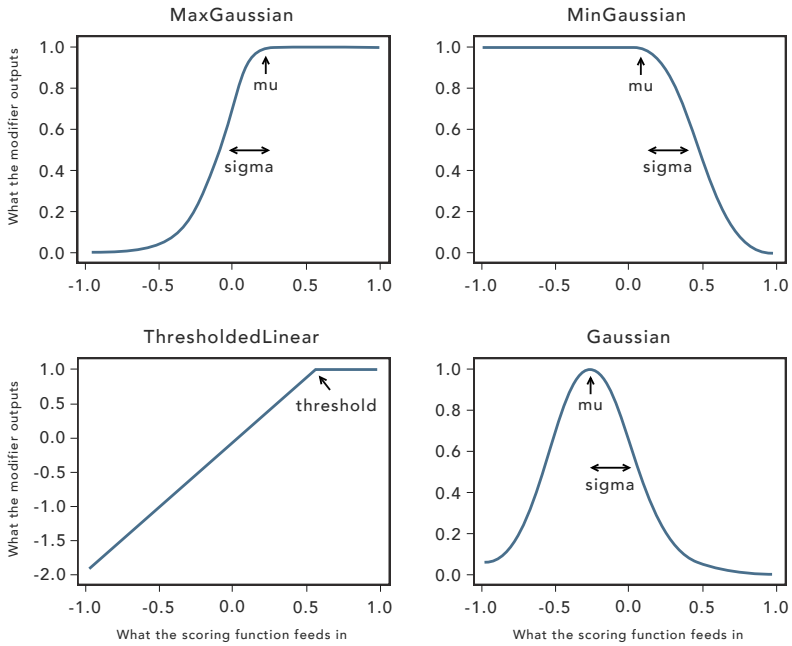


Figure A.1: Score modifiers used in GuacaMol framework. Figures are adopted from GuacaMol paper [46].

B

Appendix - Experimental Setup

B.1 Modification of `Matern52` kernel

Initially, when running Bayesian Optimization for some of the optimization tasks, the runs failed due to numerical exceptions being raised in TensorFlow. These errors were reported as failures when trying to perform Cholesky decomposition [70] on the kernel matrix. The issue was patched by overriding the `euclid_dist`-method defined in the `Matern52` kernel. By adding a small amount of jitter (10^{-6}) to the euclidean distance, the stability of the matrix operations was improved.

B.2 Physicochemical properties used in GENTRL pretraining

- Molecular Weight (see Section 5.1.1 for more details)
- logP (see Section 5.1.1 for more details)
- Hydrogen Bond Donors and Acceptors (see Section 5.1.1 for more details)
- The number of aromatic rings (see Section 5.1.1 for more details)
- Number of unwanted substructures in the context of drug development [71]
- Number of rotatable bonds (see Section 5.1.1 for more details)
- The polar surface area (TPSA) of molecule (see Section 5.1.1 for more details)