



UNIVERSIDADE D
COIMBRA

Beatriz de Jesus Pereira Santos

**DRUG DISCOVERY WITH GENERATIVE
ADVERSARIAL NETWORKS**

**Dissertation in the context of the Master in Biomedical
Engineering, Specialization in Clinical Informatics and
Bioinformatics advised by Prof. Dr. Joel P. Arrais and Prof. Dr.
Bernardete Ribeiro and presented to the Faculty of Sciences and
Technology.**

August 2021



UNIVERSIDADE D
COIMBRA

Beatriz de Jesus Pereira Santos

Drug Discovery with Generative Adversarial Networks

Thesis submitted to the
University of Coimbra for the degree of
Master in Biomedical Engineering

Supervisors:
Prof. Dr. Joel P. Arrais
Prof. Dr. Bernardete Ribeiro

Coimbra, 2021



This work was developed in collaboration with:

Center for Informatics and Systems of the University of Coimbra



Esta cópia da tese é fornecida na condição de que quem a consulta reconhece que os direitos de autor são pertença do autor da tese e que nenhuma citação ou informação obtida a partir dela pode ser publicada sem a referência apropriada.

This copy of the thesis has been supplied on condition that anyone who consults it is understood to recognize that its copyright rests with its author and that no quotation from the thesis and no information derived from it may be published without proper acknowledgement.

Acknowledgments

This dissertation marks the end of a five-year journey at the University of Coimbra and while I am a firm believer that the journey matters more than the destination, I am very pleased with what I have achieved which would not have been possible without the help and support of countless people.

First, I would like to thank my supervisors Prof. Dr. Joel P. Arrais and Prof. Dr. Bernardete Ribeiro for giving me the opportunity to study what I am passionate about, the liberty to explore new ideas while also steering me in the right direction, and for introducing and advising me on the world of scientific research. I also want to express my deep gratitude to Dr. Maryam Abbasi, without whom this dissertation would unlikely see the light of day, for all the advice, discussion of ideas, formatting tips and overall, constant support and mentorship. I would also like to thank PhD student Tiago Pereira and the remaining colleagues at LARN for the invaluable suggestions and discussions.

Finally, and most importantly, a deep thank you to my family, in particular to my parents, for the unconditional love and support, for fostering my love for learning from an early age, believing in me when I didn't, keeping me motivated and, most of all, for always having a word of encouragement even in the hardest of times. A final note to my grandparents: this would not have been possible without your hard-work and resilience throughout life.

Thank you.

Acknowledgments

Financing

This research has been funded by the Portuguese Research Agency FCT, through D4 - Deep Drug Discovery and Deployment (CENTRO-01-0145-FEDER029266). This work is funded by national funds through the FCT - Foundation for Science and Technology, I.P., within the scope of the project CISUC - UID/CEC/00326/2020 and by European Social Fund, through the Regional Operational Program Centro 2020.

“O que impede de saber não são nem o tempo nem a inteligência, mas somente a falta de curiosidade.”

AGOSTINHO DA SILVA

Resumo

A descoberta de novos fármacos é um processo extremamente demorado, complexo, dispendioso e que apresenta taxas de sucesso muito baixas que podem ser atribuídas à elevada dimensionalidade do espaço químico. Estudar e avaliar o espaço químico de forma integral é simplesmente impraticável pelo que é importante encontrar novas formas de restringir o espaço de pesquisa. A utilização de algoritmos de *Deep Learning* tem surgido como uma possível solução para mitigar os problemas acima mencionados já que diminuem consideravelmente o tempo dispendido e, por conseguinte, as despesas associadas a todo o processo. As redes neuronais recorrentes (RNNs) e adversariais generativas (GANs) encontram-se entre os métodos mais promissores no que se refere à geração de novos potenciais fármacos.

O trabalho desenvolvido deu origem a duas contribuições independentes. Foi efetuado um estudo extensivo das arquiteturas e parâmetros associados às redes recorrentes do qual resultou um modelo otimizado capaz de gerar até 98.7% de moléculas válidas mantendo elevados níveis de diversidade. Este estudo permitiu ainda demonstrar que a informação estereoquímica, que é de extrema importância no desenvolvimento de fármacos mas frequentemente ignorada, pode ser incluída nestes modelos computacionais com elevado sucesso.

Para além disso, foi desenvolvida uma estratégia baseada em GANs que inclui uma componente de otimização. Este método é composto por duas técnicas de *Deep Learning*: um modelo Encoder-Decoder responsável por converter as moléculas em vetores do espaço latente, criando, desta forma, um novo tipo de representação molecular; e uma GAN com a capacidade de aprender e replicar a distribuição dos dados de treino para, posteriormente, gerar novos compostos. De modo a gerar moléculas otimizadas para uma determinada característica, a GAN treinada é conectada a um mecanismo de *feedback* que avalia as moléculas geradas a cada época e substitui os compostos do conjunto de treino que apresentam menor pontuação pelas novas moléculas com propriedades mais desejáveis. Desta forma, a distribuição dos compostos gerados vai-se aproximando sucessivamente do espaço químico de interesse, o que resulta na geração de um maior número de moléculas relevantes para o problema em estudo.

Palavras-Chave

Aprendizagem Profunda, Geração de Novos Fármacos, Redes Adversariais Genera-

tivas, Redes Neuronais Recurrentes, SMILES

Abstract

Drug discovery is a highly time-consuming, complex, and expensive process with low rates of success that can be mainly attributed to the high dimensionality of the chemical space. Evaluating the entire chemical space is prohibitively expensive, so it is of the utmost importance to find ways of narrowing down the search space. Deep Learning algorithms are emerging as a potential method to generate novel chemical structures since they can speed up the traditional process and decrease expenditure. Recurrent Neural networks (RNNs) and Generative Adversarial Networks (GANs) are two of the most promising methods for generating drug-like molecules from scratch.

The proposed work resulted in two independent contributions. A comprehensive study on RNNs' architectures and parameters that resulted in an optimized model capable of generating up to 98.7% of valid non-specific drug-like molecules while maintaining high levels of diversity. This work also proved that stereo-chemical information, often overlooked in most works, can be successfully incorporated and learned by these models.

Furthermore, a novel GAN-based framework that includes an optimization stage was developed. This approach incorporates two deep learning techniques: an Encoder-Decoder model that converts the string notations of molecules into latent space vectors, effectively creating a new type of molecular representation, and a GAN that is able to learn and replicate the training data distribution and, therefore, generate new compounds. In order to generate compounds with bespoke properties and once the GAN is replicating the chemical space, a feedback loop is incorporated that evaluates the generated molecules according to the desired property at every epoch of training and replaces the worst scoring entries in the training data by the best scoring generated molecules. This ensures a slow but steady shift of the generated distribution towards the space of the targeted property resulting in the generation of molecules that exhibit the desired characteristics.

Keywords

Deep Learning, Drug Design, Generative Adversarial Networks, Recurrent Neural Networks, SMILES

Contents

List of Tables	xix
List of Figures	xxiii
Acronyms	xxvii
1 Introduction	1
1.1 Context and Motivation	1
1.2 Objectives	2
1.3 Contributions	2
1.4 Scientific Outcomes	4
1.5 Dissertation Structure	5
2 Artificial Intelligence in Drug Discovery	7
2.1 Introduction	7
2.1.1 Artificial Intelligence	7
2.1.2 Drug Discovery and Development	9
2.1.3 Artificial Intelligence in Drug Discovery	10
2.2 Molecular Representation	10
2.2.1 Molecular Graphs	10
2.2.2 SMILES	11

2.2.3	Molecular Fingerprints	12
2.3	Review of Approaches to Property and Activity Prediction	13
2.4	Review of Approaches to Molecular Generation and Optimization	14
2.5	Evaluation Metrics	16
3	Deep Learning Models	21
3.1	Artificial Neural Networks	21
3.2	Fully Connected Neural Networks	23
3.3	Recurrent Neural Networks	24
3.3.1	Long Short-Term Memory	25
3.3.2	Gated Recurrent Units	28
3.3.3	Bidirectional RNNs	29
3.4	Autoencoder	29
3.4.1	Encoder-Decoder Sequence to Sequence	31
3.5	Generative Adversarial Networks	32
3.5.1	<i>Vanilla</i> GAN	32
3.5.2	Wasserstein GAN	35
3.5.3	Wasserstein GAN with Gradient Penalty	36
3.6	Regularization Techniques	37
4	RNN Generator	41
4.1	Introduction	41
4.2	Methods	42
4.2.1	Preprocessing Data	42
4.2.2	Training Models	43
4.2.3	Output Generation	46
4.2.4	Validation Strategy	46

4.3	Experimental Results and Discussion	47
4.3.1	Datasets	47
4.3.2	Performance Analysis and Results	47
4.4	Conclusions	52
5	GAN-based Framework	53
5.1	Introduction	53
5.2	Methods	54
5.2.1	Encoder-Decoder Model	54
5.2.2	Wasserstein GAN with Gradient Penalty	55
5.2.3	Case-study: Kappa Opioid Receptor	56
5.2.4	KOR Binding Affinity Predictor	57
5.2.5	Optimization through Transfer Learning	57
5.2.6	Optimization through FeedbackGAN	58
5.2.7	Validation Strategy	58
5.3	Experimental Results and Discussion	59
5.3.1	Datasets	59
5.3.2	Encoder-Decoder Model	60
5.3.3	WGAN-GP	68
5.3.4	Performance of the Predictor	70
5.3.5	Optimization through Transfer Learning	71
5.3.6	Optimization through FeedbackGAN	73
5.4	Conclusions	79
6	Conclusions and Future work	83
6.1	Conclusions	83
6.2	Future Work and Open Issues	85

Bibliography

87

List of Tables

4.1	The training time and percentage of valid and unique SMILES generated with 8 different models.	48
4.2	The training time and percentage of valid SMILES and diversity with 4 different optimizers for models 2 and 3.	49
4.3	The training time and percentage of valid and unique SMILES with different batch sizes for models 2 and 3.	50
4.4	The percentage of valid SMILES and diversity for Model 2 with different number of training samples.	50
4.5	The percentage of valid SMILES and diversity for different values of Sampling Temperatures	50
4.6	The percentage of valid SMILES and diversity for two different models and two types of input encoding.	51
4.7	Comparison of the results obtained for model 3 when applied to two different datasets: “Biogenic” and “ChEMBL” and using two types of encoding “Embedding” and “OHE” with Number of Samples equal to 100,000.	51
5.1	Summary of the datasets used throughout the experiment.	60
5.2	Search space for finding the optimal set of parameters of the proposed Encoder-Decoder model.	61
5.3	Results for different number of encoder BiLSTM layers and decoder LSTM layers (Encoder-Decoder with OHE structure)	62

5.4	Results for different number of BiLSTM/LSTM units (Encoder-Decoder with OHE structure)	62
5.5	Results for different number of batch size (Encoder-Decoder with OHE structure)	62
5.6	Results for different values Batch Normalization Momentum (BNM) (Encoder-Decoder with OHE structure)	63
5.7	Results for different latent dimensions (Encoder-Decoder with OHE structure)	63
5.8	Results for different values of Noise Standard Deviation (Encoder-Decoder with OHE structure)	64
5.9	Results for different number of SMILES in dataset (Encoder-Decoder with OHE structure)	64
5.10	Search space for finding the optimal set of parameters of the proposed Encoder-Decoder model (with Embedding structure).	65
5.11	Results for different number of encoder BiLSTM layers and decoder LSTM layers (Encoder-Decoder with Embedding structure)	65
5.12	Results for different number of BiLSTM/LSTM units (Encoder-Decoder with Embedding structure)	66
5.13	Results for different number of batch size (Encoder-Decoder with Embedding structure)	66
5.14	Results for different number of embedding dimension in encoder (Encoder-Decoder with Embedding structure)	67
5.15	Results for comparison between embedding dimension and latent dimension (Encoder-Decoder with Embedding structure)	67
5.16	Results for different number of SMILES in dataset (Encoder-Decoder with Embedding structure)	68
5.17	Performance of the Encoder-Decoder model for 100,000 and 500,000 training data.	69
5.18	Comparison of the Original Data with the Generated Data in terms of predicted <i>pIC</i> 50 and performance of the WGAN-GP model.	70

5.19	Comparison of the performance of the biased and unbiased models (maximization through TL).	72
5.20	Comparison of the performance of the biased and unbiased models (minimization through TL).	73
5.21	Comparison of $pIC50$ distribution measures throughout the optimization process (maximization of KOR affinity).	75
5.22	Comparison of $pIC50$ distribution measures throughout the optimization process (minimization of KOR affinity).	78

List of Figures

2.1	Schematic summary of AI concepts.	8
2.2	Drug Discovery and Development Process.	9
2.3	Example of the molecular graph and its corresponding matrices for the acetic acid.	11
2.4	Example of chemical compounds and corresponding skeletal forms and SMILES string.	12
2.5	Example of a Molecular Fingerprint.	13
2.6	Transfer Learning vs Reinforcement Learning.	15
3.1	Biological neuron and artificial neuron.	22
3.2	Schematic representation of a Fully Connected Neural Network.	24
3.3	Diagram of a recurrent layer and its unfolded representation.	25
3.4	LSTM Architecture.	26
3.5	LSTM Cell state.	26
3.6	Steps to update a single LSTM cell.	28
3.7	GRU Architecture.	29
3.8	Bidirectional RNN.	30
3.9	Schematic representation of an Autoencoder.	30
3.10	Schematic of an Encoder-Decoder network for English-German translation.	32

3.11	Schematic representation of a Generative Adversarial Network.	33
3.12	Schematic representation of a Wasserstein Generative Adversarial Network with Gradient Penalty	37
4.1	General workflow of the proposed Generator based on Recurrent Neural Network.	42
4.2	Data preprocessing for the molecule of Acetylsalicylic Acid using One-hot Encoding.	44
4.3	General structure of RNN Models for producing SMILES strings.	45
4.4	Workflow for generating the SMILES with a trained network.	46
4.5	Training epochs and percentage of valid and unique molecules for Models 2 and 3.	49
5.1	The general workflow of the proposed model.	55
5.2	The detailed structure of the Encoder (A) and Decoder (B) applied in the framework.	56
5.3	Implemented optimization mechanism via FeedbackGAN.	59
5.4	Comparison of the predicted $pIC50$ distributions for the original data and generated data.	70
5.5	Evaluation of properties (QED, SA score, $\log P$ and MW) for the original training data and generated data.	71
5.6	Predicted $pIC50$ versus true $pIC50$ and regression line for the test set.	71
5.7	Distribution of the predicted $pIC50$ values for the biased and unbiased model (maximization through TL).	72
5.8	Distribution of the predicted $pIC50$ values for the biased and unbiased model (minimization through TL).	73
5.9	Distribution of the predicted $pIC50$ values for the unbiased model and the biased model at every 100 epochs (maximization of KOR affinity).	75

5.10	Evaluation of properties (QED, SA score, $\log P$ and MW) for the biased model (500 epochs).	76
5.11	Best scoring examples in terms of predicted $pIC50$ generated by the proposed framework for every 100th epoch of optimization (maximization).	77
5.12	Distribution of the predicted $pIC50$ values for the unbiased model and the biased model at every 100 epochs (minimization of KOR affinity).	78
5.13	Evaluation of properties (QED, SA score, $\log P$ and MW) for the biased model (500 epochs).	79
5.14	Best scoring examples in terms of predicted $pIC50$ generated by the proposed framework for every 100th epoch of optimization (minimization).	80

Acronyms

K_d Dissociation Constant

K_i Inhibition Constant

$\log P$ Logarithm of the partition coefficient between n-octanol and water

pIC_{50} Negative Logarithm of the Half Maximal Inhibitory Concentration

Adam Adaptive Moment Estimation

ADMET Absorption, Distribution, Metabolism, Excretion and Toxicity

AI Artificial Intelligence

ANN Artificial Neural Network

BiLSTM Bidirectional Long Short-Term Memory

BN Batch Normalization

BNM Batch Normalization Momentum

DL Deep Learning

ECFP Extended-Connectivity Fingerprints

FCNN Fully Connected Neural Network

GAN Generative Adversarial Network

GPCR G-Protein-Coupled Receptors

GRU Gated Recurrent Unit

IC₅₀ Half Maximal Inhibitory Concentration

IUPAC International Union of Pure and Applied Chemistry

KNN K-Nearest Neighbors

KOR Kappa Opioid Receptor

LSTM Long Short-Term Memory

ML Machine Learning

MW Molecular Weight

NLP Natural Language Processing

NN Neural Network

OHE One-Hot Encoding

QED Quantitative Estimate of Drug-Likeness

QSAR Quantitative Structure-Activity Relationship

QSPR Quantitative Structure-Property Relationship

RF Random Forest

RL Reinforcement Learning

RNN Recurrent Neural Network

ROF Rule of Five

SA Synthetic Accessibility

SGD Stochastic Gradient Descent

SMILES Simplified Molecular-Input Line-Entry System

SVR Support Vector Regression

TL Transfer Learning

VAE Variational Autoencoder

WGAN Wasserstein Generative Adversarial Network

WGAN-GP Wasserstein GAN with Gradient Penalty

Introduction

1.1 Context and Motivation

The discovery and development of new drug candidates is a complex, lengthy and expensive process. For every drug that reaches the final step of this process and gets approved, an estimated \$2.8 billion have been spent and between 10 and 15 years of research were necessary [1, 2]. This is due to the fact that most drug candidates fail before reaching the last step of the process, with recent estimates pointing to a success rate of only 2% [3]. Such a low success rate implies that this is not just an expensive process but a high-risk one from a financial point of view, as most investments will fail.

One of the main challenges is the high dimension of the chemical space since it has been estimated that over 10^{60} drug-like molecules could be synthetically accessible [4] and only a small fraction of this chemical space has been explored [5]. Researchers have to select and analyze molecules from this vast space in order to find potential compounds that can be biologically active towards a defined target of interest and also exhibit a set of desired physicochemical and pharmacological properties. This is a prohibitively expensive process, and therefore it is critical to use computational tools to narrow down the search space. Search can be conducted using similarity-based metrics, which provide a quantifiable statistical indicator of closeness between molecules. Heuristics and modern virtual screening techniques can help this process narrow the space of possibilities, but the task remains daunting. In contrast, in *de novo* drug design, the practitioners try to directly design novel molecules that are active towards the desired biological target [6].

With the increase in computational power and available datasets, Deep Learning (DL) techniques have shown promising results in research fields such as radiology,

microscopy and genomics and are emerging as a promising solution for *de novo* drug design [7]. Generative Adversarial Network (GAN) [8] is a powerful state-of-the-art DL method that has revolutionized generative modeling and therefore is likely to improve *de novo* drug design. It should be noted that with the constant increase of the average life expectancy, prevalence of chronic illnesses and regulations, the drug discovery and development process is likely to become even more challenging. Therefore, an important strategy to cut costs and increase the rates of success can be summarized as “fail sooner” [7]. By identifying and filtering out compounds that will be rejected in the later stages of the process, it reduces the time and cost associated with clinical trials and improves the overall process.

While there has been impressive advances in the field of DL applied to drug design over the past years, there are still plenty of new techniques and avenues of research to be pursued in order for these methods to become reliable and be implemented in the pharmaceutical industry pipeline.

1.2 Objectives

The main goal of this thesis is to research, develop and evaluate a generative framework based on DL techniques with a focus on GANs that can integrate the stages of drug generation and optimization. The objectives can be summarized as follows:

- O1** - Overview DL techniques and their current applications to drug design.
- O2** - Explore different DL architectures and corresponding hyperparameters to train a molecule generator.
- O3** - Implement a discriminative model to predict the binding affinity for the Kappa Opioid Receptor (KOR).
- O4** - Implement a generative model based on GANs that can learn the distribution of the training data and generate new valid molecules.
- O5** - Propose and evaluate an optimization framework to bias the GAN towards generating molecules that exhibit the desired properties.

1.3 Contributions

In summary, the main contributions of this dissertation are as follows:

C1 - A RNN model with increased performance when compared to the current literature that is able to generate up to 98.7% of valid molecules while maintaining high levels of diversity among the generated compounds.

A comprehensive study on different RNNs frameworks applied to molecule generation with SMILES notation was performed. More specifically, the impact of different recurrent architectures and hyperparameters such as the batch size, data size, epochs, and the neural network optimizer were analyzed in terms of the rate of valid molecules, diversity of the generated compounds, and the time required to train the models. The effect of applying either embedding or one-hot encoding in the validity, diversity, and speed of generating the results was also studied. Further, the effect of using a different type of tokenization was explored, such as character by character (token by token) as proposed in the work of Olivecrona et al. [9] or by considering the stereo-chemistry notations with combined tokens grouped by a pair of square brackets [10]. Stereochemistry is vital in medicinal chemistry, but often overlooked by most works due to its increased complexity.

C2 - A novel Encoder-Decoder model inspired on LatentGAN [11] that is able to convert the SMILES notation of the compounds into latent vectors, and vice-versa, achieving up to 99.0% of correctly reconstructed molecules.

An exhaustive study of the proposed Encoder-Decoder model was performed and its architecture and hyperparameters independently studied. This resulted in a model with improved performance when compared to the literature as it can correctly reconstruct 99.0% of the compounds while also including stereochemical data, which is known for its higher complexity and not considered on LatentGAN [11].

C3 - A GAN model that is able to learn the distribution of the chemical space and generate new molecules.

A Wasserstein GAN with Gradient Penalty (WGAN-GP) [12] that uses the newfound representation given by the previously mentioned Encoder-Decoder model as real training data was implemented. Through adversarial training, the generator was able to approximate the distribution of the chemical space and therefore sample new molecules represented by their latent vectors from it.

These vectors must then be decoded by the Decoder of the Encoder-Decoder model in order to obtain the corresponding SMILES strings.

C4 - An optimization framework that can bias the generator of the aforementioned GAN model in order to generate compounds with desired properties.

An optimization framework inspired on FeedbackGAN [13] was implemented to bias the generation of the molecules towards the space of compounds with high binding affinity to the Kappa-Opioid Receptor (the chosen case-study). This framework works by iteratively sampling new molecules from the generator, evaluating them, and replacing the worst scoring molecules in the training data by the best sampled molecules, effectively biasing the distribution towards the desired property.

1.4 Scientific Outcomes

The contributions were submitted to international journals and presented/accepted at international conferences. They are listed, in chronological order for each type of venue, together with reference to the contribution.

- P1** - Beatriz P. Santos, Maryam Abbasi, Tiago Pereira, Bernardete Ribeiro and Joel P. Arrais. "Optimizing Recurrent Neural Network Architectures for De Novo Drug Design", IEEE CBMS, The 34th IEEE CBMS International Symposium on Computer-Based Medical Systems, 2021. (Published Paper) [14]
- P2** - Beatriz P. Santos, Maryam Abbasi, Tiago Pereira, Bernardete Ribeiro and Joel P. Arrais. "Improvement on Generative Adversarial Network for Targeted Drug Design", ESANN 2021, The 29th European Symposium on Artificial Neural Networks, 2021. (Accepted Paper)
- P3** - Maryam Abbasi, Tiago Pereira, Beatriz P. Santos, Bernardete Ribeiro and Joel P. Arrais. "Multiobjective Reinforcement Learning in Optimized Drug Design", ESANN 2021, The 29th European Symposium on Artificial Neural Networks, 2021. (Accepted Paper)
- P4** - Beatriz P. Santos, Maryam Abbasi, Carlos Simões, Bernardete Ribeiro and Joel P. Arrais. "Designing Optimized Drug Candidates with Generative Adversarial Network", Bioinformatics Journal. (In preparation)

During this work, there was also the opportunity of presenting and discussing the main topics of this dissertation:

- *Generative Adversarial Networks and their application to discrete data*, Department of Informatics Engineering, University of Coimbra, November 2020.
- *Generative Adversarial Networks*, Department of Informatics Engineering, University of Coimbra, March 2021.
- *Improvements on Generative Adversarial Networks and their application to drug design*, Department of Informatics Engineering, University of Coimbra, May, 2021.

All the implemented models presented and discussed in this dissertation are publicly available on:

- <https://github.com/larngroup/RNN-Drug-Generation>.
- <https://github.com/larngroup/GAN-Drug-Generator>.

1.5 Dissertation Structure

This thesis is divided into six chapters. The current and first chapter is concerned with a brief introduction that aims to contextualize and motivate the reader to the problem that is going to be addressed. Chapter 2, Artificial Intelligence in Drug Discovery, focuses on important introductory notions as well as a review of the current research approaches to improving the Drug Discovery process by resorting to computational methods. This review is further divided into two subcategories: Property Prediction and Molecular Generation. Chapter 3, Deep Learning Models, provides a theoretical overview of the relevant models. Chapters 4 and 5 cover the contributions of this dissertation: a generator based on Recurrent Neural Networks and an optimized generator based on Generative Adversarial Networks, respectively. The sixth and final chapter presents the conclusions of the dissertation and future potential directions of research.

Artificial Intelligence in Drug Discovery

This chapter clarifies the core principles of Artificial Intelligence (AI), Machine Learning (ML), Deep Learning (DL) and the process of Drug Discovery and Development. Moreover, it overviews the most common used computational molecular representations followed by a literature review of deep learning algorithms applied to property prediction and molecular generation and optimization. It closes with a section on evaluation metrics for molecules.

2.1 Introduction

2.1.1 Artificial Intelligence

Artificial Intelligence (AI) is defined by Rich as “*the study of how to make computers do things that people are better at*” [15] and can be thought of as intelligence demonstrated by machines in the sense that these machines will be able to learn and solve problems [16]. It is a broad field that encompasses a wide variety of computational methods and has been successfully applied to fields such as image recognition [17], language translation [18], music generation [19], gaming [20] and drug discovery [21].

Machine Learning (ML) is a sub-field of AI that refers to systems that have the ability of learning from experience, i.e., from given data and then use this knowledge to infer on new samples [7, 22]. ML algorithms can be further divided into three main groups: Supervised, Unsupervised and Reinforcement Learning (RL). Supervised learning requires the samples in the input data to be labeled so that the model can learn how to map each sample to its corresponding label. Once the model is trained, it will be able to predict the label of a new observation from the knowl-

edge that it has acquired during the training phase [22,23]. The label can either be quantitative or categorical making the problem a regression or classification, respectively. By contrast, Unsupervised Learning works with unlabeled data in order to find a structure in the input data, i.e. to identify patterns or relationships [22,24]. Clustering is an example of unsupervised learning where the goal is to organize the samples into a predefined number of groups based on a metric of similarity [24]. Reinforcement Learning is an approach that mimics the learning process of humans in a new environment. There is no data available but the agent needs to decide on which actions to take through a process of trial and error in order to maximize the long-term reward [7,25,26].

Deep Learning (DL) encompasses a group of neuro-inspired ML algorithms that use multilayer neural networks to create non-linear models. The more layers, the deeper the network, hence the term “*Deep Learning*” [27]. DL takes advantage of the recent increase in available data and computational power to find high-level representations of the input data [25,26]. Similarly to ML, it can be further divided into Supervised, Unsupervised and Reinforcement Learning. However, unlike most ML algorithms, DL has the advantage of not requiring structured data. Structured data refers to data that has a set of features for each observation, typically organized into a tabular format, while unstructured data, like images or text, is not naturally arranged into a predefined format [28]. Figure 2.1 summarizes the relationship between the aforementioned concepts.

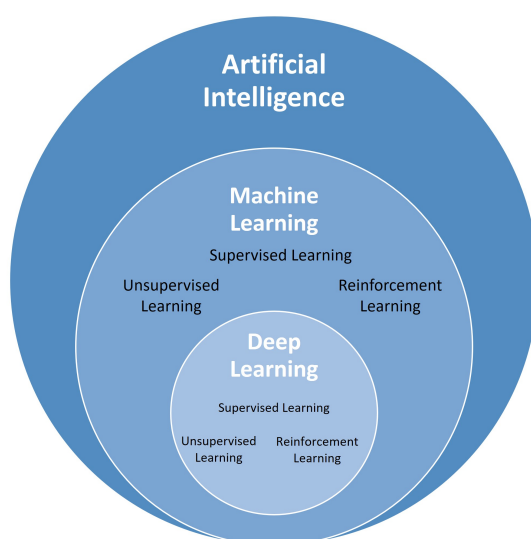


Figure 2.1: Schematic summary of AI concepts.

2.1.2 Drug Discovery and Development

Drug Discovery and Development is a lengthy, expensive and prone to failure process that aims to bring new drugs to the market and can be partitioned into several steps from finding a new candidate drug to its regulatory approval (Figure 2.2). The overall process is triggered when an unmet medical need, i.e., a medical condition for which there is no satisfactory treatment, is found [29]. Drug Discovery starts with Target Discovery and Validation which aims to identify the origin of the disease and demonstrate the functional role of the target in terms of therapeutic benefit and safety, respectively [29, 30]. Lead Discovery refers to the identification of a group of drug-like compounds that can bind to the chosen target. The final step of Drug Discovery is the Lead Optimization which is concerned with improving target specificity and selectivity so that off-target binding and toxicity can be mitigated [29, 30].

The Drug Development phase can start once a drug candidate or lead has been selected. The Pre-clinical Trials involve studying the safety, efficacy and toxicity of the compound in animals. A compound that passes the Pre-clinical Trials should also be stable, easily synthesizable, adhere to Lipinski's Rule of 5 (see Section 2.5 on Page 17) and possess acceptable solubility and bioavailability [29]. If the drug candidate passes all the trials and the regulatory bodies approve, the Clinical Trials can be conducted on humans. If successful, the process concludes with filing for regulatory approval [29, 30].

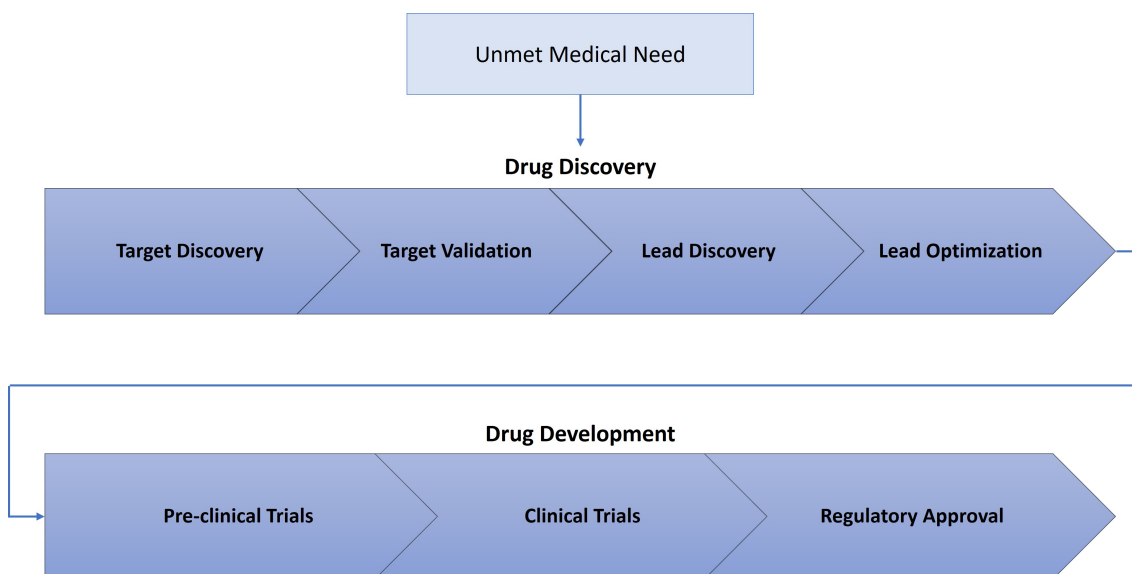


Figure 2.2: Drug Discovery and Development Process.

2.1.3 Artificial Intelligence in Drug Discovery

The increase in data digitalization and available computational power has prompted research on the integration of AI in the Drug Discovery process with the aim of decreasing costs, increasing the rates of success and exploring the extensive chemical space [5, 21, 31].

The current work is focused on the Drug Discovery phase, more specifically on *de novo* drug design. *De novo* drug design refers to the design of novel molecules with specific properties and that are active towards a predefined target [5,6]. Therefore, *de novo* drug design frameworks require the use of two types of models: generative and discriminative. Generative models are able to generate new data points by returning the probability of observing a certain observation while discriminative models learn a function that maps the input to a label [28]. In this context, generative models are used for molecular generation, i.e., generating a pool of new molecules, and discriminative models are used for property prediction. *De novo* drug design requires the generation and simultaneous optimization of compounds, therefore combining both of these types of models. Sections 2.3 on Page 13 and 2.4 on Page 14 provide a literature review on property prediction and molecular generation and optimization, respectively.

2.2 Molecular Representation

While chemists have several strategies for representing molecules, with the International Union of Pure and Applied Chemistry (IUPAC) nomenclature being the most widely used, they typically do not capture the structure of the molecule and cannot be interpreted by computational methods [32]. To solve this, notation systems such as Molecular Graphs, SMILES and Molecular Fingerprints have been developed in order to find a suitable way of computationally representing molecules.

2.2.1 Molecular Graphs

Molecules can be represented as mathematical graphs, with the atoms being the nodes and the bonds being the edges [7]. So that the computer can handle the information, the molecular graphs are mapped to matrices as represented in Figure 2.3. An adjacency matrix stores information regarding the manner by which the atoms are connected, a node features matrix identifies the type of atoms and an edge

features matrix the type of bonds. Since a graph can be traversed in many ways, this is a non-unique type of representation. Molecular graphs are very popular since they resemble how chemists draw molecules on paper, store some spacial information and all the sub graphs are interpretable [32]. However, they are not compact, require a lot of memory, cannot represent all types of molecules and cannot distinguish between right- and left-handed enantiomers, which is of the utmost importance in drug discovery [7, 32].

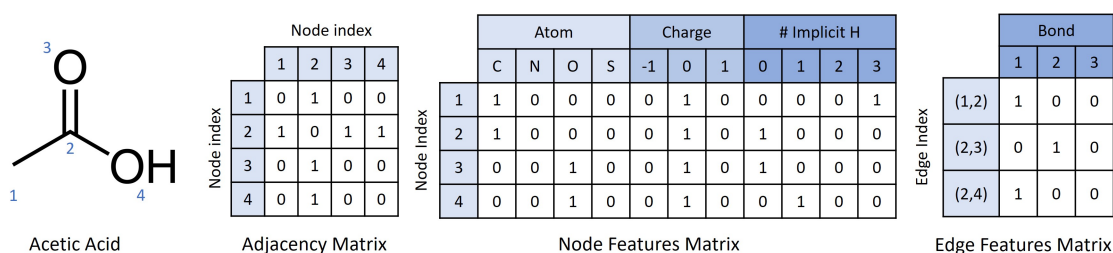


Figure 2.3: Example of the molecular graph and its corresponding matrices for the acetic acid.

2.2.2 SMILES

Simplified Molecular-Input Line-Entry System (SMILES) is a line notation that represents a chemical structure as a sequence of characters typically denoted by SMILES strings and that was first introduced by Weininger [33]. This is a non-unique type of representation, meaning that for each molecule there are several possible SMILES strings depending on the atom that was chosen at the beginning of the encoding process. In fact, there are as many possible SMILES strings as heavy atoms in the molecule [34]. In some situations, this might not be desirable, so tools like RDKit [35] include canonicalization algorithms that transform a molecule into a unique representation. While the SMILES encoding contains no 2D or 3D atom coordinates, it has the advantage of being extremely compact [34]. The encoding process adheres to the following set of basic rules [36]:

- Atoms are represented by their atomic symbols;
- Single, double and triple bonds are represented by '-', '=' and '#', respectively;
- Branches are represented by parentheses;
- Implicit Hydrogen atoms and single bonds can be omitted;

- Explicit Hydrogen atoms and charges are represented inside square brackets;
- Rings are represented by breaking one bond and numbering it, followed by the remaining atoms and a repetition of the same number to denote ring closure. Aromatic rings have their atoms written in lower case.

The SMILES syntax also supports stereochemistry and isotopism by including symbols such as "@" (anti-clockwise) and "@@" (clockwise) to indicate tetrahedral centers [36]. Figure 2.4 shows some examples of chemical compounds, their skeletal formula and the corresponding SMILES strings.




Compound	Skeletal Formula	SMILES String
Propane		CCC
Acetic acid		CC(=O)O
Acetylsalicylic Acid		O=C(C)Oc1ccccc1C(=O)O

Figure 2.4: Example of chemical compounds and corresponding skeletal forms and SMILES string.

2.2.3 Molecular Fingerprints

Molecular Fingerprints are fixed-length binary vectors that represent the presence or absence of structural and functional properties [7] as shown in Figure 2.5. Depending on the traversal algorithm, it is possible to distinguish between path-based and circular fingerprints. Extended-Connectivity Fingerprints (ECFPs) are the most commonly used circular fingerprints and represent the molecular structure by atom neighborhoods [37]. ECFPs are fast to compute and can be used for clustering, virtual screening, similarity searching and as input to Quantitative Structure-Activity Relationship (QSAR) models. However, they have the disadvantage of not encoding all the information regarding a molecule which can lead to two different molecules being represented by the same fingerprint, making this an ambiguous representation [32].

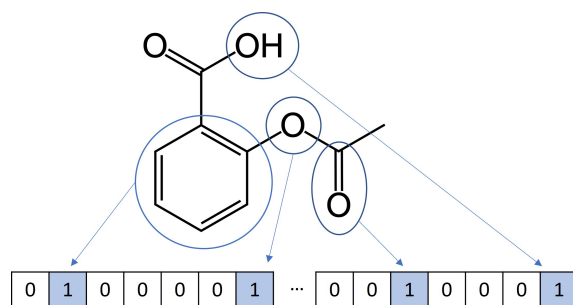


Figure 2.5: Example of a Molecular Fingerprint.

2.3 Approaches to Property and Activity Prediction

In drug discovery, for a molecule to be considered a drug it must satisfy several requirements that relate not only to its biological activity, but also to its intrinsic physicochemical properties that affect how the drug is processed in the organism, which makes drug discovery a multi-objective problem [16]. Therefore, finding a mathematical model that can correlate the chemical structure of a compound to a certain property is of the utmost importance [38]. It is possible to distinguish between two types of predictive models according to their goal: Quantitative Structure-Activity Relationship (QSAR) models aim to predict the biological activity of a given molecule to a predefined target while Quantitative Structure-Property Relationship (QSPR) models intend to predict a specific physicochemical property given a compound [39, 40]. These models are essential in computer-aided drug discovery since they allow the assessment of generator models and their optimization towards the space of desired properties so that valuable compounds can be found [16]. Moreover, another potential use for these types of models is to find, in the existing pool of commercially available drugs, compounds with interesting characteristics that can be applied to new therapeutic purposes. This is denoted by drug repurposing [41, 42] and is a remarkable worthwhile endeavour since, as these drugs have already been approved by the competent authorities, the cost and time associated with the evaluation of potential adverse effects is significantly reduced.

Several methodologies and molecular representations have been used for both QSAR and QSPR models. Initially, machine learning methods such as Support Vector Regression (SVR) [16, 43], Random Forest (RF) [16, 44], k-nearest neighbors (KNN) algorithm [45] or Bayesian Learning [46] were employed. However, with the increase in available datasets and computational power that has surged in the last decade, deep learning techniques, more specifically, predictive models based on deep neu-

ral networks have proved to be a more effective approach. Deep Neural networks present the following advantages when compared to traditional ML approaches: the possibility of tuning a variety of hyperparameters, changing the architecture and employing dropout layers, which considerably reduces over-fitting and thus, increases the robustness of the models [21].

Regarding the choice of the molecular representation, it is possible to use any of the representations presented in Section 2.2 on Page 10 or to learn a new one within the DNN predictive model. ECFPs have been successfully used in the works of Liu et al. [47] and Uesawa [48]. Lusci et al. took the first steps in the concept of learning a molecular representation directly from the data by translating molecular configurations into fixed length vectors that were then used to feed a fully connected neural network [16, 49]. Duvenaud et al. explored a convolutional neural network that receives graphs as inputs and creates a learnable neural fingerprint that can then be used as input to a Fully Connected Neural Network (FCNN) predictive model. This method showed improved performance and interpretability [16, 50]. The graph-based approach has the advantage that during the training of the full model, the molecular representation is also optimized to the current specific task [16].

Recently, Popova et al. implemented Recurrent Neural Networks (RNNs) with LSTM units to predict biological properties by directly resorting to SMILES strings [51]. This approach has the main advantage of directly using SMILES strings from a dataset without requiring any additional process to obtain the molecular descriptors.

Although different methodologies have been employed, the general trend clearly points towards the direct use of chemical structures and the avoidance of feature engineering techniques [42].

2.4 Approaches to Molecular Generation and Optimization

Deep Learning techniques, mainly generative models, have emerged as a promising solution for *de novo* drug design, whose aim is to generate novel molecules with specific predefined properties [6]. This process can typically be seen as two-fold: creating a general model that learns the distribution of the training data, and then optimizing this model to the desired property. The optimization is usually employed using Reinforcement Learning (RL), a reward-based approach, or Transfer Learning (TL) which requires the model to be trained on a smaller dataset that only includes samples with the desired properties (Figure 2.6).

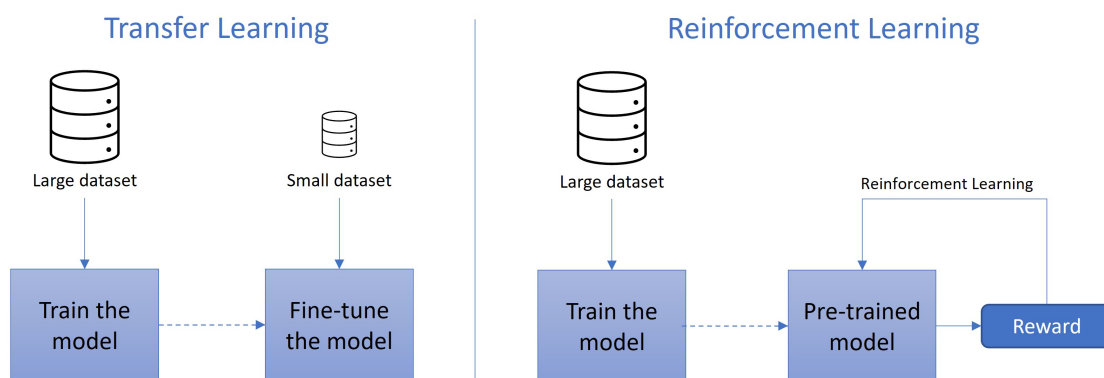


Figure 2.6: Transfer Learning vs Reinforcement Learning.

Earlier approaches to solve this problem resorted to the use of RNNs, which can learn the syntax of sequences of data, in a manner similar to what is done in the field of Natural Language Processing (NLP). They were used to generate new molecules represented as SMILES [36]. Olivecrona et al. and Liu et al. combined RNNs with the REINFORCE algorithm [52], a policy-based RL method, to bias the generation process towards the space of the desired properties [9] [47]. In the former, a Generator network is pre-trained and then tuned according to the defined reward function. The latter introduced the use of two Generator Networks, one for Exploration (the pre-trained model) and another for Exploitation (the optimized model), in order to ensure higher variability of the generated molecules by sampling a random number at each training step that defines which of the generators will be used. Popova et al. approached this problem by pre-training two independent networks: a stack-augmented RNN Generator and a Predictor, that are then used jointly through RL in order to optimize the generator. They showed that their model could be effectively biased towards physical properties like the melting temperature and partition coefficient, specific biological activity, and chemical complexity [51].

Zheng et al. trained an RNN model with GRU cells on a biogenic dataset that includes stereo-chemical information to learn the grammar of these SMILES strings with higher complexity, and then fine-tuned it employing TL; they used an embedding layer, and both normal tokens and combined tokens, which resulted in an increased vocabulary [10]. Gupta et al. and Segler et al. drew similar conclusions by implementing the same method but using LSTM cells [53] [54]. The former also showed that its method could be applied to low-data drug discovery and fragment-growing, while advocating that TL avoids the introduction of errors or unwanted bias when compared to RL.

The aforementioned approaches can suffer from exposure-bias [55,56] which prompted the appearance of other DL-based alternatives for targeted generation of compounds, mainly adversarial approaches.

Gómez-Bombarelli et al. proposed the use of a Variational Autoencoder (VAE) that transforms the discrete data (SMILES) into a latent real-valued continuous vector [57]. The VAE is trained together with a Predictor, that given a latent vector, predicts certain properties. They were able to generate new structures by perturbing latent vectors, to interpolate between molecules and also to guide the search for optimized compounds using gradient-based methods. Similarly, Blaschke et al. compared VAEs with Adversarial Autoencoders and showed that, in both cases, the latent space preserves the chemical similarity of the input molecules [58].

Sanchez et al. proposed the ORGANIC framework [59], which builds on SeqGAN [56] and ORGAN [55] and uses a Generative Adversarial Network (GAN) [60] to generate new molecules. A GAN comprises a Generator that is responsible for generating new samples that resemble the training data and a Discriminator that aims to distinguish between real and generated samples. Due to the inability to backpropagate through categorical samples (like SMILES Strings), the GAN is combined with RL, which is responsible for updating the generator’s weights during training. The already implemented RL can also be adapted to optimize the generated molecules. Putin et al. improved this architecture by using a differentiable neural computer as a more robust and powerful generator [61]. Their framework outperformed the ORGANIC.

In the work of Prykhodko et al. [11], this differentiation problem was surpassed by first finding a numerical representation of the SMILES strings through an Autoencoder, as initially proposed by Bjerrum and Sattarov [62], and then using this newfound representation to train the GAN. Once the GAN is trained, it can generate multiple numerical vectors that must then be decoded into SMILES strings by the decoder part of the trained Autoencoder. In this way, the GAN can focus on optimizing the sampling process while the Autoencoder is responsible for learning the SMILES syntax. The generation process can be optimized by Transfer Learning.

2.5 Evaluation Metrics

When working with generative computational models applied to the field of drug discovery, it is of the utmost importance to have scoring metrics that can evaluate a

given property. It is not just important that the generated molecule has the atoms and their connections correctly and properly arranged according to the chemical rules (validity) but also that several other properties are present. In the following sections, some of the most commonly used scoring metrics are introduced.

Molecular Weight

An important property that can be easily computed through RDKit [35] is the Molecular Weight (MW) of a compound. The optimization of a molecule towards a specific target typically requires an increase in the molecular weight so that the desired features are present [63]. Nevertheless, a lower molecular weight has been associated with increased likelihood of absorption and blood brain barrier permeability, making this an important property to monitor and to keep as low as possible in most scenarios [64].

Lipophilicity

Lipophilicity is another important physicochemical property that has a major impact on absorption, distribution, metabolism, excretion, and toxicity (ADMET) of drugs [65]. It can be easily estimated by resorting to the logarithm of the partition coefficient between n-octanol and water ($\log P$). The $\log P$ value indicates whether the molecule will dissolve more in a lipid or water-based environment. Higher positive ($\log P$) values imply a lipophilic compound (increased lipid bilayer permeability and decreased solubility in water) while negative values are associated with a hydrophilic compound (decreased lipid bilayer permeability and increased water solubility) [65,66]. This metric is particularly useful as it approximates how a molecule will behave in the human body, with cytosol (the fluid part of the cytoplasm) instead of water and the lipid membranes taking the place of octanol [67].

According to Lipinski's Rule of Five, in order to have an acceptable absorption or permeation, a compound's $\log P$ should not exceed the value of 5 [64].

Drug-Likeness

Drug-Likeness is a concept that encompasses a series of properties that must be present in order for a molecule to be considered a drug candidate and survive Phase I clinical trials [68]. Such a molecule must not only show pharmacological activity, but also ADMET properties [69]. By considering the Drug-Likeness of the compounds

in the early stages of drug discovery, it becomes possible to reduce the high costs and attrition often associated with this process [65]. By analyzing the distribution of these critical properties, it has been found that most drug-like molecules fall into a very narrow range of values [70]. Lipinski’s “Rule of 5” (ROF) has been widely used as a guideline for drug-likeness since it was first introduced in 1997 due to its simplicity and straightforwardness [64]. It states that a molecule will likely exhibit poor absorption and permeation if two or more of the following properties are met:

- more than 5 hydrogen-bond donors;
- more than 10 hydrogen-bond acceptors;
- molecular weight greater than 500;
- $\log P$ greater than 5.

It should be noted that these criteria do not apply to natural products or substracts of biological transporters [64].

Despite the fact that the ROF was proposed as a guideline, it has often been used to filter datasets of compounds, which is similar to assigning a molecule to one of two groups: drug-like or not. To solve this issue, Bickerton et al. proposed the Quantitative Estimate of Drug-Likeness (QED) as a metric that is able to rank molecules according to their drug-likeness whether or not they have passed the ROF. QED combines several molecular properties including the ones in the ROF in order to produce a value in the range of 0 to 1, where a QED of 0 means that all properties are unfavorable and a QED of 1 stands for all properties being favorable. The fact that all compounds can be ranked according to their QED allows the researcher not to outright exclude a molecule because of exhibiting one unfavorable property when the remaining are near ideal [71].

Synthetic Accessibility Score

Synthetic Accessibility (SA) Score, proposed by Ertl and Schuffenhauer, is a method that estimates the synthesizability of a drug-like molecule as a value between 1 and 10, where the lower the score, the easier it is to make the compound. The synthetic accessibility, or ease of synthesis, is an extremely important property, particularly in *de novo* drug design, as it gives an indication of how feasible it will be to actually develop the molecule in the laboratory. If a molecule is not likely to be synthetic accessible, then it can be dropped in the early stages of the drug discovery process,

saving time and costs [72].

Binding Affinity

Binding affinity is crucial in Drug Discovery as it measures the strength of the interaction between a target and a ligand or drug. Therefore, the higher the binding affinity, the stronger the non-covalent interactions between the target and the drug and the more difficult it is to dissociate the created complex. By attributing a continuous binding affinity value, instead of a binary one, to a drug-target interaction, it becomes possible to rank a set of molecules according to how tightly they bind to the desired target. This property is typically measured by resorting to one of the following: inhibition constant (K_i), dissociation constant (K_d) or the half maximal inhibitory concentration (IC50). The lower these values, the stronger the interaction between the target and the ligand [73]. The IC50 is the concentration that is needed to inhibit 50% of a biochemical function and is the preferred measure during the early stages of drug discovery as it is the least time-consuming of the processes [74]. It is worth noting that IC50 values are frequently converted to a different scale by doing the negative logarithm of the IC50 ($pIC50$), which implies that the higher the $pIC50$, the higher the binding affinity.

3

Deep Learning Models

3.1 Artificial Neural Networks

Artificial Neural Networks (ANNs) are a set of varied computational models that are inspired by and attempt to simulate the biological neural networks found in many organisms. While the networks themselves are highly complex, their most straightforward unit, the neuron, can be easily explained [75].

A biological neuron is an idiosyncratic cell: it has a cell body with a nucleus and a set of cellular extensions called dendrites; from the cellbody protrudes the axon, a long structure covered in myelin that, among other functions, isolates it from its surroundings; at the opposite end, the axon branches to form a structure denoted by axon terminal. Due to being highly branched, the dendrites receive information from many other nerve cells, mainly relating to the presence of neurotransmitters in the environment. If their quantity can create a stimulus above a certain threshold, the neuron is activated. An action potential is generated and propagated through the axon until it reaches its terminals, causing the release of neurotransmitters, which might activate other neurons. It is worth noting that the neurons operate under the *all or none* law, which means that the action potential that results from an activated neuron is approximately the same independently of the number of neurotransmitters detected by the dendrites. If this quantity is below the threshold, then the neuron simply does not activate. This implies that the output of the neuron is not a linear function of its input [75, 76].

An artificial neuron, based on a single biological neuron, is the basic building block of an ANN and is depicted in Figure 3.1 alongside its biological counterpart. A single neuron requires [77]:

- inputs (x_i): independent values that are given to the neuron.

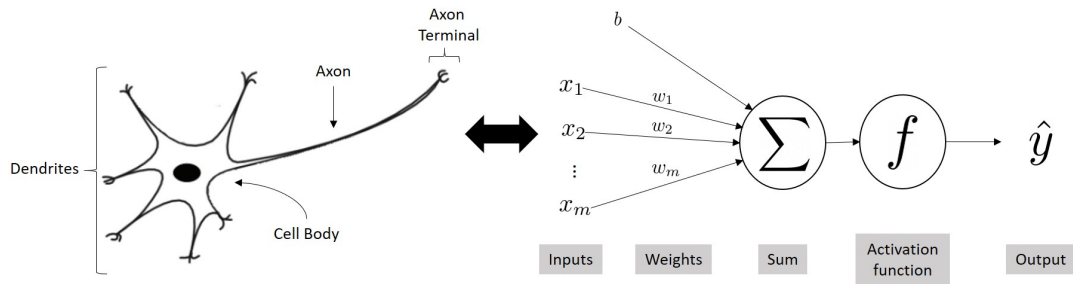


Figure 3.1: Biological neuron and artificial neuron¹.

- Bias (b): extra input that shifts the activation function.
- Weights (w_i): there is a weight associated with each input that defines its importance.
- Activation Function (f): analogous to the *all or none* law, it is responsible for activating the neuron and it is usually a non-linear function.
- output (\hat{y}): is the result of the weighted sums of the inputs and the bias passed through the activation function f . The output of a single neuron can be mathematically defined as:

$$\hat{y} = f\left(b + \sum_i w_i x_i\right). \quad (3.1)$$

Biological neurons don't occur in isolation. They are typically close to many other neurons, with the axon terminals of a neuron being close to the dendrites of others. This allows them to pass information to one another, forming a network [75]. In the same way, while a single artificial neuron may not be particularly useful, the combination and interconnection of several artificial neurons gives rise to ANNs, which are a powerful modeling tool able to solve extremely complex real-life problems. The manner by which the neurons are interlinked defines the architecture or topology of the ANN [77]. Nevertheless, one can typically identify the input layer, one or more hidden layers, and an output layer, where each layer contains a set of neurons. The input layer corresponds to the first layer of neurons that receives the independent values. The output of this layer is then fed to the following hidden layer and so on until the output layer is reached. The more hidden layers, the deeper the network, hence the term *Deep Learning* (DL).

Training the network is the process by which the best set of weights for each layer

¹Figure adapted from [77]

is found. For that, a loss function must be defined and computed at the end of each iteration. This loss compares the output of the network to the real value and, therefore, measures the cost incurred from incorrect predictions. The weights of the network are then modified in order to reduce the loss by backpropagation. The backpropagation algorithm corresponds to computing the gradient of the loss with respect to each parameter in the NN and then shifting the parameters in order to minimize that loss. The manner by which the parameters or weights are updated depends on the chosen optimizer. Still, theoretically, it will lead to a better model resulting in a decrease in the loss. The choice of the loss function and the optimizer depends heavily on the problem that needs to be solved [7, 78].

One of the advantages of DL methods like ANN is their ability to learn multiple levels of different representations of the raw input data. This means that there is no need for feature engineering, removing any previous assumptions required in traditional Machine Learning. The fact that activation functions are able to include non-linearities in the model allows it to learn extremely complex functions that learn to ignore irrelevant information and reveal hidden patterns in the data that can be represented in an abstract space and used, for example, for classification tasks. These outstanding characteristics have made DL state of the art for several applications, ranging from image recognition to language translation [78].

3.2 Fully Connected Neural Networks

Fully Connected Neural Networks (FCNN) or Dense Neural Networks NNs are very similar to the aforementioned ANN but comprise of several fully connected layers, i.e. every neuron in a layer is connected to every single neuron in the previous layer, as can be seen in Figure 3.2. This can be mathematically defined as:

$$z_{k,i} = b_i^{(k)} + \sum_{j=1}^{n_{k-1}} f(z_{k-1,j})w_{j,i}^{(k)} \quad (3.2)$$

where $z_{k,i}$ corresponds to the value at node i in hidden layer k .

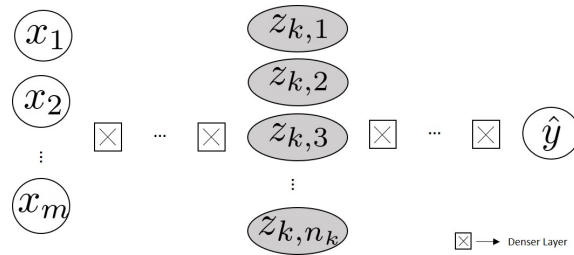


Figure 3.2: Schematic representation of a Fully Connected Neural Network ².

3.3 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a type of ANN devised to process sequences of data, where the order in which the data is presented is of the utmost importance, for example, when handling time series, text, music, or molecules [7]. They can be applied both to sequence generation and classification problems.

Their ability to process sequential data comes from the fact that RNNs have loops that allow the information to persist throughout time, keeping a sort of *memory*. This *memory* is kept in a vector called the hidden state, h_t . The output at each time step t , \hat{y}_t , depends not only on its current input x_t but also on the previous hidden state h_{t-1} ; the RNN cell will also produce a new updated hidden state, h_t , that will be taken into consideration in the next time step. This recurrence relation can be mathematically described for each time step by equations 3.3 and 3.4 [27].

$$h_t = \sigma_1(W_{hx}x_t + W_{hh}h_{t-1} + b_h) \quad (3.3)$$

$$\hat{y}_t = \sigma_2(W_{yh}h_t + b_y) \quad (3.4)$$

where σ_1 and σ_2 are activation functions (traditionally, *tanh* and *softmax*, respectively), W_{hx} , W_{hh} and W_{yh} are weight matrices (input-to-hidden, hidden-to-hidden and hidden-to-output, respectively) and b_h and b_y are bias vectors.

Depending on the goal of the problem, it is possible to output all the hidden states throughout time or just the final hidden state. Figure 3.3 represents an RNN with a single recurrent layer on the left, and its equivalent unfolded representation on the right. An unfolded RNN allows the visualization of how a single sequence is processed by the layer and how the *hidden state* is updated by the RNN cell at each time step. It is also worth noting that all the RNN cells in the diagram are, in fact, the same cell, and therefore share the same weights which are learned during

²Figure adapted from http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

training [28].

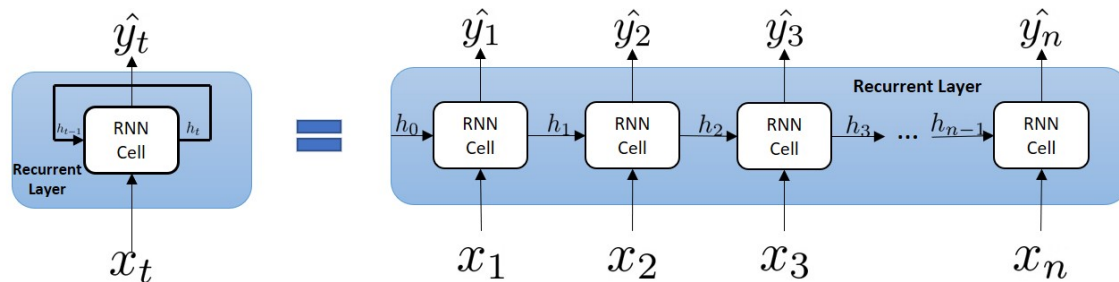


Figure 3.3: Diagram of a recurrent layer and its unfolded representation.

As explained in Section 3.1, to train a network, one must first compute the loss and then backpropagate it. In the case of RNNs, the total loss is considered as the sum of the losses at each individual time step, and backpropagation is done at each point in time (Backpropagation Through Time) as shown by equation 3.5 [79].

$$\frac{\partial L^{(T)}}{\partial W} = \sum_{t=1}^T \frac{\partial L^{(T)}}{\partial W_t} \quad (3.5)$$

where the left side stands for the derivative of the loss L w.r.t. the weight matrix W at time step T .

RNNs distinguish themselves from other types of ANN mainly due to being capable of dealing with variable length sequences by including the previously mentioned *memory* and the fact that an increase in the input size does not lead to an increase in model size [79]. The main drawback of these models is that they have difficulties in handling long-term dependencies, which can lead to exploding and vanishing gradients [80]. Vanishing gradients occur when the gradients become increasingly smaller and approach zero, making it impossible to train the network. Exploding gradients correspond to the opposite problem, in which the norm of the gradients constantly increases [81].

3.3.1 Long Short-Term Memory

Long Short-Term Memory (LSTM) networks [82] are a type of RNN designed to handle long-term dependencies by including a set of computational blocks with gates that control the flow of information. Gates resort to a sigmoid neural net layer and pointwise multiplication to decide how much information should be retained. With sigmoid activation function, the values are forced to be in the range $[0,1]$ and,

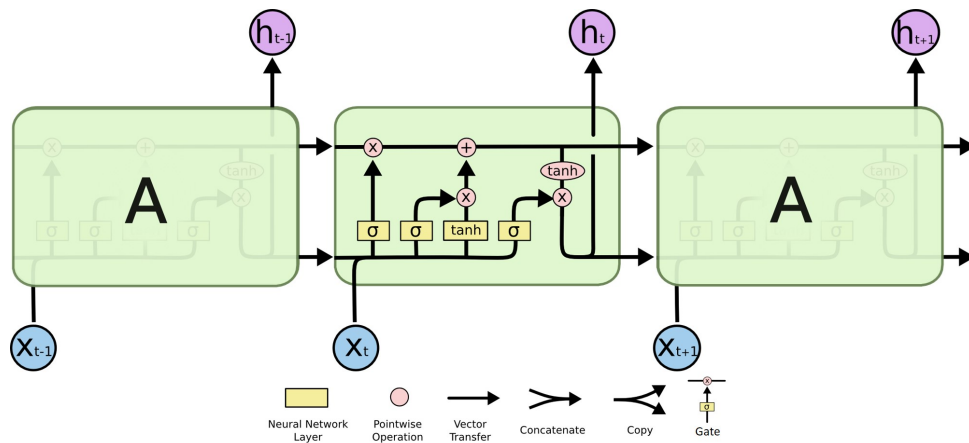


Figure 3.4: LSTM Architecture ³.

the higher the value, the more information is retained. Figure 3.4 represents the computational blocks that make up an LSTM and how they are interlinked. Note that a pointwise operation can be either a sum “+” or a multiplication “×”.

The key feature of an LSTM is its cell state C_t (Figure 3.5), which can be thought of as the current status of the sequence or the *memory*, and is regulated by the gates [83]. It depends on the previous cell state C_{t-1} , the previous hidden state h_{t-1} and the input at the current time step x_t . It’s the existence of this cell state that allows for an uninterrupted gradient flow during training that mitigates the vanishing gradients problem.

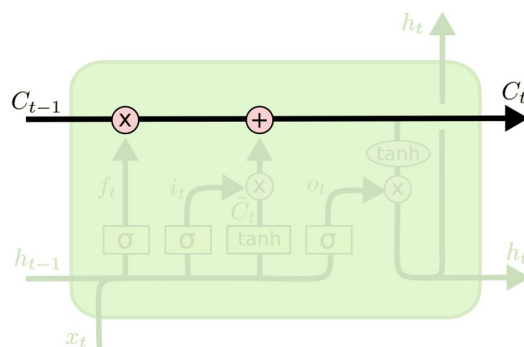


Figure 3.5: LSTM Cell state ⁴.

There are four steps in an LSTM cell, highlighted in Figure 3.6, that are ruled by six equations (equations 3.6 to 3.11) [28, 83]:

³Figure from [83].

⁴Figure from [83].

1. **Forget** (Figure 3.6a): the *forget gate* forgets irrelevant information from the previous hidden layer h_{t-1} and from the current input x_t , since some of it might not be important. The *forget gate* is a dense layer with a weights matrix W_f and bias b_f followed by a sigmoid activation function σ .

$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f) \quad (3.6)$$

2. **Store** (Figure 3.6b): in this step, that includes two phases, the cell decides what new information will be stored in the cell state. First, the vector that results from the concatenation of h_{t-1} and x_t is fed to an *input gate* which, similarly to the previously mentioned gate, comprises a weights matrix W_i , bias b_i and a sigmoid activation function. The output of this gate, i_t determines how much new information will be added to C_{t-1} . Second, the aforementioned concatenated vector goes through another dense layer (with weights matrix W_c and bias b_c) and a tanh activation function which results in vector \tilde{C}_t that contains the new information that the cell considers keeping.

$$i_t = \sigma (W_i \cdot [h_{t-1}, x_t] + b_i) \quad (3.7)$$

$$\tilde{C}_t = \tanh (W_C \cdot [h_{t-1}, x_t] + b_C) \quad (3.8)$$

3. **Update** (Figure 3.6c): the cell state C_t is updated by forgetting what has been decided in step 1. (first term of equation 3.9) and adding the new information scaled by how much it was decided to update each value in step 2 (second term of the same equation).

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \quad (3.9)$$

4. **Output** (Figure 3.6d): the concatenated vector also goes through the *output gate* (with weights matrix W_o , bias b_o and a sigmoid activation function) which results in the vector o_t that represents how much of C_t should be outputted. This vector is then multiplied pointwise with the result of passing C_t through a tanh activation function resulting in the current hidden state h_t that will be sent to the next cell along with the previously found C_t .

$$o_t = \sigma (W_o \cdot [h_{t-1}, x_t] + b_o) \quad (3.10)$$

$$h_t = o_t * \tanh(C_t) \quad (3.11)$$

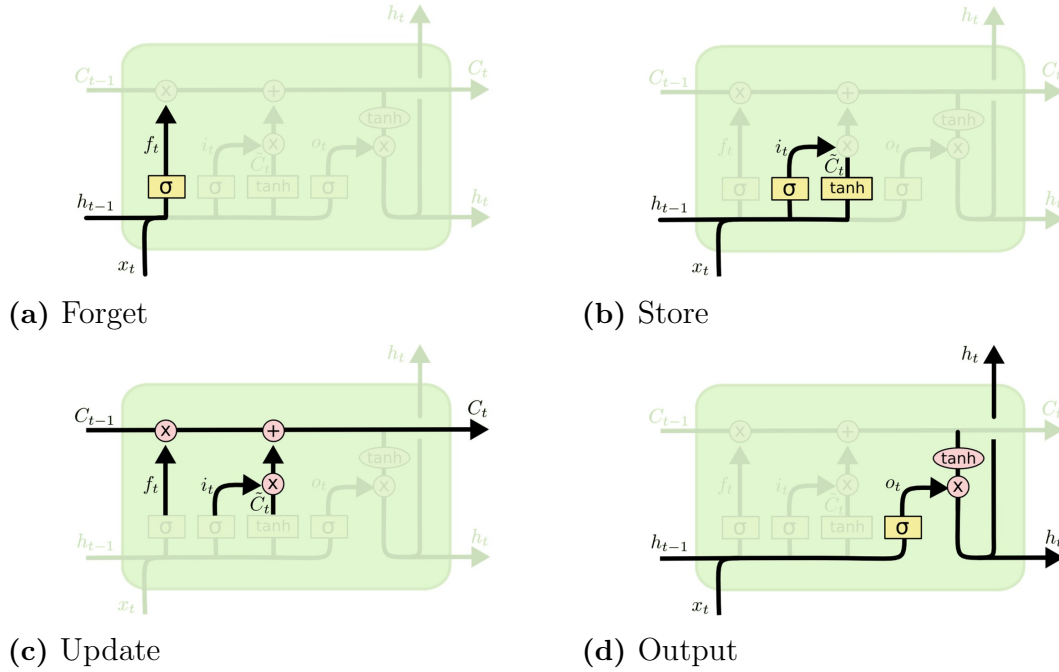


Figure 3.6: Steps to update a single LSTM cell ⁵.

3.3.2 Gated Recurrent Units

Over the years, several variants of LSTMs have been proposed. The most successful one is the Gated Recurrent Units (GRUs) which are simpler to compute and implement [84]. They differ from the LSTM units by replacing the *forget* and *input* gates with *reset* and *update* gates, and removing the cell state and the *output gate* [28]. Therefore, the only output of the cell is its hidden state. This hidden state h_t is obtained using the following four expressions (equations 3.12 to 3.15):

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r) \quad (3.12)$$

$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t] + b) \quad (3.13)$$

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z) \quad (3.14)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t \quad (3.15)$$

The *reset gate* is responsible for producing the r_t vector which, due to the sigmoid activation function, contains values between 0 and 1 that represent how much of the previous hidden state h_{t-1} should be considered for the new state of the cell. From r_t , and using equation 3.13, the vector \tilde{h}_t is obtained and it stores the new state of the cell (values between -1 and 1 due to the tanh activation function). The

⁵Figure from [83].

update gate, in a manner similar to the previously mentioned gates, outputs a vector z_t whose values determine how much of the new state \tilde{h}_t should be blended with h_{t-1} . Finally, the output of the cell, i.e. its updated hidden state, h_t is computed by blending in \tilde{h}_t with h_{t-1} in the proportion determined by the update gate z_t [28, 84]. Figure 3.7 shows a diagram of these computations.

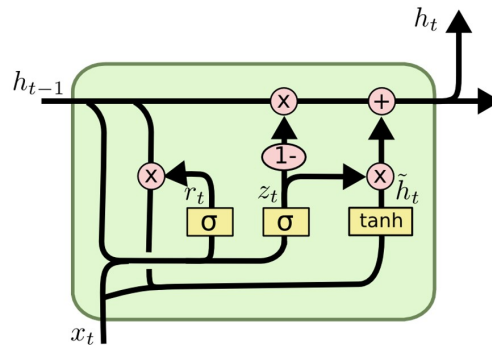


Figure 3.7: GRU Architecture ⁶.

3.3.3 Bidirectional RNNs

The aforementioned types of RNNs process the sequence only in a single direction, the forward direction, which means that the state at a given time t , and therefore the prediction y_t , depend only on past values, x_1, \dots, x_{t-1} . Bidirectional RNNs solve this by combining two RNNs, one that evaluates the sequence in the forward direction and another that evaluates it in the backward direction making y_t dependent on both the past and future values of the sequence through the two sets of hidden states that are stored. A bidirectional RNN can be made up of traditional RNNs, LSTMs or GRUs [27, 28, 85]. A schematic representation of a bidirectional RNN is shown in Figure 3.8 with the forward direction represented by the blue line and the backward one by the green line.

3.4 Autoencoder

An Autoencoder is a neural network that consists of two distinct and often symmetrical networks: the *encoder* and the *decoder*. The *encoder* is responsible for mapping the input data into a lower-dimensional representation vector (also denoted bottleneck or latent vector), and the *decoder* has to reconstruct the original data from

⁶Figure from [83].

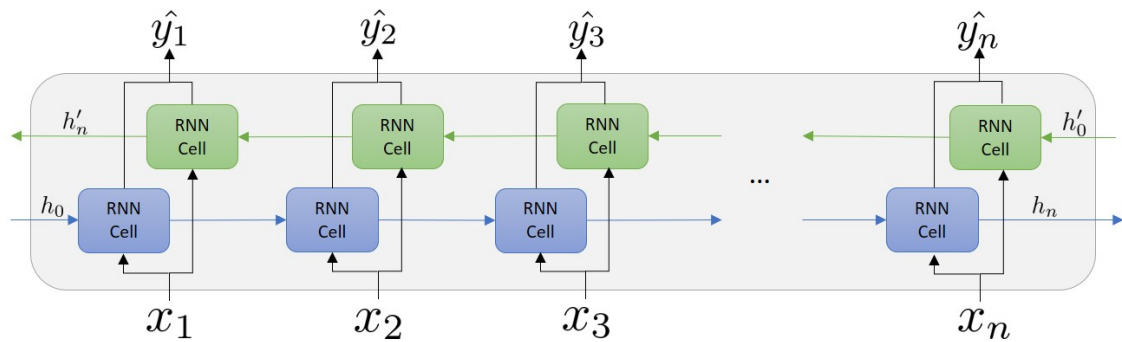


Figure 3.8: Bidirectional RNN ⁷.

this representation vector. The autoencoder is trained using backpropagation to find the weights for both networks that minimize the loss between the input x and its reconstruction \hat{x} (figure 3.9). Since it compares the input to its reconstruction, there is no need for previously defined labels, and therefore this is an unsupervised technique. Root mean squared error and binary cross-entropy are the most often used loss functions [28].

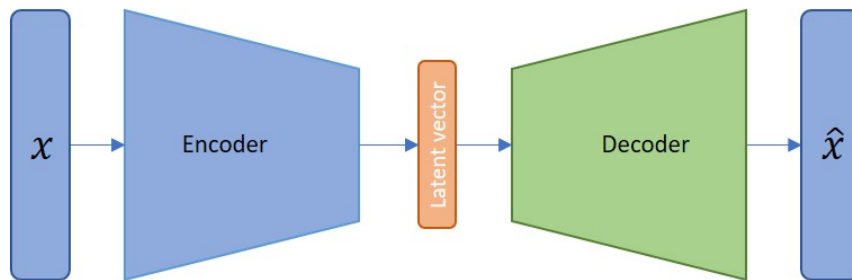


Figure 3.9: Schematic representation of an Autoencoder.

The representation vector is a compression of the original data into a lower dimensional space, called the latent space, and so, it can be seen as an encoding of the data that retains its most important features. The lower the dimensionality of this latent space, the lower the quality of the reconstruction. It is important to note that the model is not required to work for any possible input, it works only for inputs that belong to the same distribution as the training data, so, in a way, it learns to compress data from this distribution into the latent vector and then reconstruct it [7].

The idea is that by randomly picking a point in the latent space, the decoder is able to convert it into a viable data instance. Since there are no stochastic nodes in this

⁷Figure adapted from [83].

network, once the autoencoder is trained it works as a deterministic encoding of the data.

3.4.1 Encoder-Decoder Sequence to Sequence

Encoder-Decoder Sequence to Sequence models, commonly known as *seq2seq*, are typically used when the goal is to predict a new sequence of words that is in some way related to the given input sequence and where both sequences can have different lengths. This type of network is used for tasks such as Language Translation [86], Question Generation [87] and Text Summarization [88]. The original model was independently proposed by Cho et al. and Sutskever et al. and the process can be summarized in three main steps [28, 84, 86]:

1. An encoder RNN maps the input sequence into a fixed-length vector: the context or latent vector. This context vector is usually a function of the final hidden state of the RNN and so it can be seen as a summary of the input sequence.
2. The context vector is then used to initialize the hidden state of the decoder RNN.
3. Given a <start>token as input, the decoder RNN, with its previously initialized hidden state, produces a new hidden state which is followed by a dense layer that outputs a probability distribution over the vocabulary, from which the next token can be retrieved. At each timestep, the decoder RNN takes the previous hidden state and predicted token in order to produce the next. In doing so, it is able to generate a novel sequence of tokens that terminates either by sampling a <stop>token or achieving a pre-defined maximum allowed length of the sequence.

During training, teacher forcing is employed so that the output of the decoder at each timestep is compared to the real token and the loss can be calculated [28].

Due to the fact that *seq2seq* works with variable length sequences it could suffer from problems associated with long-term dependencies. To minimize this, Cho et al. proposed the use of GRU cells instead of standard RNN cells [84] and Sutskever et al. proposed the use of LSTM cells and also found that deep LSTMs outperformed shallow ones and that better results could be obtained when reversing the order of tokens in the input sequence [86].

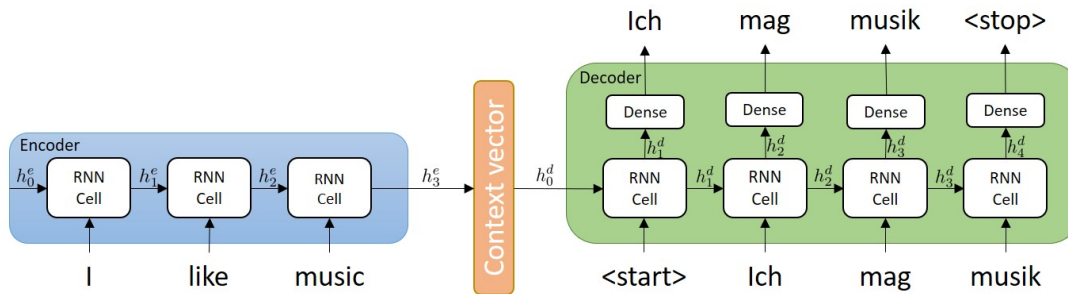


Figure 3.10: Schematic of an Encoder-Decoder network for English-German translation.

When using LSTM cells, since they output both an hidden state and a cell state, the context vector can be obtained from combining both of these vectors.

An example of a simple *seq2seq* model employed for Language Translation is depicted in Figure 3.10.

3.5 Generative Adversarial Networks

Generative Adversarial Networks (GANs) [60] are a type of unsupervised deep generative model that aims to implicitly define the probability density function of the training data, making it possible to sample from its high dimensional complex distribution. They achieve this by pairing two competing neural networks with opposing objectives. In this section, three types of GANs are explored: *vanilla* GAN, Wasserstein GAN, and Wasserstein GAN with Gradient Penalty, with each model building up on the previous one.

3.5.1 *Vanilla* GAN

The first GAN, commonly referred to as *vanilla* GAN, was proposed by Goodfellow et al. [60] and consists of two competing neural networks: a *generator* and a *discriminator*. The *generator* is responsible for converting a random noise vector \mathbf{z} into synthetic samples that resemble the training data, while the *discriminator*, given a sample, attempts to identify it as real or synthetic/fake. The output of the *discriminator* is, therefore, the probability of the input sample belonging to the training/real data.

These two networks have different and opposing objectives since while the *discriminator* D aims to correctly label the data, the *generator* G wants to fool D into

believing that its data is real. This translates into a minimax game with the following objective [60]:

$$\min_G \max_D \mathbb{E}_{\mathbf{x} \sim p_{data}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} [\log (1 - D(G(\mathbf{z})))] . \quad (3.16)$$

where p_{data} is the data distribution and $p_{\mathbf{z}}$ is a simple noise distribution from which we sample \mathbf{z} (e.g., the uniform distribution or the normal distribution). In this way, the *generator* G maps the noise distribution $p_{\mathbf{z}}$ to the model distribution p_g . \mathbf{z} is the source of randomness in the model, allowing the *generator* to output a different variety of vectors.

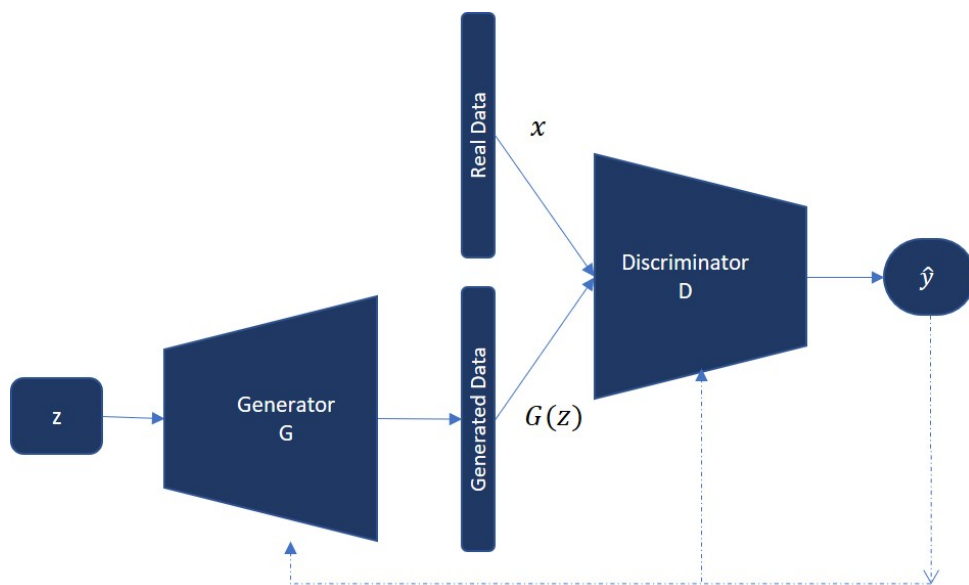


Figure 3.11: Schematic representation of a Generative Adversarial Network.

Figure 3.11 represents a GAN, where \hat{y} is the output of the *discriminator* and therefore, $\hat{y} = D(\mathbf{x})$ and $\hat{y} = D(G(\mathbf{z}))$ for real and synthetic samples, respectively.

In short, the *discriminator* wants to maximize the objective in equation 3.16 so that $D(\mathbf{x})$ is close to 1 and $D(G(\mathbf{z}))$ is close to 0. The *generator*, on the other hand, has no influence on the first term of the objective and wants to minimize the second term so that $D(G(\mathbf{z}))$ is close to 1, implying that the generated data has been perceived as real by the *discriminator*.

The process of training GANs requires a delicate balance between training the *discriminator* and training the *generator*. Ideally we would like the *discriminator* to be trained to optimality before switching to training the *generator* but this is computationally prohibitive and could lead to overfitting. In practice, we resort to training

the *discriminator* for k steps, and the generator for a single step [60] since it has also been shown that if the *discriminator* gets too strong, then it can lead to vanishing gradients on the *generator* [89]. The GAN is trained until it reaches a point at which neither the *generator* nor the *discriminator* can improve because $p_g = p_{data}$, and thus, the *discriminator* cannot distinguish between the two distributions and it constantly outputs a value of 0.5.

It should be noted that while training the generator, the weights of the discriminator must be frozen, and vice-versa. Otherwise, the generated data would be predicted as real because of having a weak discriminator and not a strong generator, which is the goal [28].

In practice, and as stated by Goodfellow et al., during the beginnings of training, the *generator* is weak and therefore, the *discriminator* can easily distinguish between generated and real samples which can lead to the saturation of $\log(1 - D(G(\mathbf{z})))$ [60]. To solve this, instead of minimizing $\log(1 - D(G(\mathbf{z})))$ when training the *generator*, Goodfellow et al. propose maximizing $\log D(G(Z))$ which is able to provide stronger gradients in the early stages of training but was later shown to cause unstable updates [89].

Overall, GANs attempt to minimize the divergence between the real data distribution and the generated distribution, so that once the model is trained, the *generator* is able to generate new data instances that resemble the training data.

While GANs can be seen as a major breakthrough in the field of generative modeling and promptly became a popular method, they are particularly difficult to train and present several specific challenges [28]:

- **Mode Collapse:** can occur when the *generator* collapses and finds an output (i.e. mode) or a limited set of outputs that constantly fool the *discriminator* and is therefore, unable to diversify its output. This is often associated with the *discriminator* being stuck in a local minimum and leads to a poor generalization of the model.
- **Vanishing Gradients:** this can occur when the *discriminator* becomes so good at distinguishing samples that the *generator's* gradients vanish and it is unable to learn.
- **Uninformative Loss:** since the *generator* is evaluated against the current best *discriminator* and this *discriminator* is constantly being updated, the loss

function cannot be compared at different points during the training process. This results in a lack of correlation between the *generator* loss and the quality of the generated samples making it impossible to monitor the quality of the training by its loss. In fact, sometimes the loss function of the generator can increase throughout the epochs and be accompanied by an increase in the quality of the generated samples.

- **Failure to Converge:** there are several known and unknown reasons for a GAN to fail to converge, some of them associated with the above mentioned problems, but it typically happens when the loss of both the *discriminator* and *generator* starts to oscillate and spiral out of control instead of achieving long-term stability. It is worth noting that, unlike other DL models, stability in GANs can also be a gradually increase or decrease of the loss, due to its uninformative loss. Moreover, GANs have a great number of hyper-parameters that need to be tuned and are highly sensitive to the slightest change to any of them which means that achieving stability is a process of trial and error.

3.5.2 Wasserstein GAN

The Wasserstein-GAN (WGAN), proposed by Arjovsky et al., is an alternative algorithm to the traditional GAN that reduces mode collapse, provides a meaningful loss metric and shows improved stability [90]. They propose the use of the *Earth-Mover* distance or Wasserstein-1 as the distance metric between both probability distributions, p_{data} and p_g , $W(p_{data}, p_g)$, which provides a specific value for the distance, no matter how far apart the distributions are⁸. This distance can be interpreted as the minimum cost of transporting mass required to transform p_{data} into p_g [12].

However, the Wasserstein distance is intractable but by resorting to the Kantorovich-Rubinstein duality [91], it is possible to obtain the following WGAN value function [12, 90]:

$$\min_G \max_{D \in \mathcal{D}} \mathbb{E}_{\mathbf{x} \sim p_{data}} [D(\mathbf{x})] - \mathbb{E}_{\mathbf{z} \sim p_z} [D(G(\mathbf{z}))] \quad (3.17)$$

where \mathcal{D} is the set of 1-Lipschitz functions. A function is considered to be 1-Lipschitz

⁸The traditional GAN resorts to the Jensen-Shannon Divergence which, for two non-overlapping distributions (frequently the case at the beginning of training), is always equal to $\log 2$. This results in the fact that, for an ideal *discriminator*, the gradient will be zero, resulting in a *generator* that is unable to learn due to vanishing gradients [89].

if it satisfies inequality 3.18:

$$\frac{|D(\mathbf{x}_1) - D(\mathbf{x}_2)|}{|\mathbf{x}_1 - \mathbf{x}_2|} \leq 1 \quad (3.18)$$

where \mathbf{x}_1 and \mathbf{x}_2 are any two input samples. This means that there is a limit enforced on the rate at which the discriminator’s predictions can change between two samples, i.e. the absolute value of the gradient must never surpass 1 [28]. This is of the utmost importance since when using the Wasserstein loss, real samples are compared to the label 1 and fake samples to the label -1. This means that the output of the *discriminator*, denoted by *critic* in this context, is no longer bound to the range $[0,1]$ and so its predictions can fall anywhere in the range $]-\infty, +\infty[$. If the Lipschitz constraint was not enforced, this could lead to extremely large loss values, which must be avoided in NN [28]. Arjovsky et al. propose the use of weight clipping after each gradient update so that the weights of the critic will lie in the range $[-0.01, 0.01]$. They also state that this is a “terrible way” of enforcing the Lipschitz constraint because it diminishes the critic’s capacity to learn [90].

Another important thing to notice is that with WGANs the *critic* can be trained to optimality giving rise to more reliable gradients for the *generator* which, not only removes the difficulty in balancing the training of the *discriminator* and *generator* in GANs, but also reduces mode collapse [90]. Typically, the *critic* is updated five times (to ensure it is close to convergence) and the *generator* only one.

Overall, the greatest contributions of the WGAN are its meaningful loss (which correlates with sample quality) and its improved stability.

3.5.3 Wasserstein GAN with Gradient Penalty

The Wasserstein GAN with Gradient Penalty (WGAN-GP) (Figure 3.12), proposed by Gulrajani et al., solves the following two problems of WGAN that are inherited from the use of weight clipping as a way to enforce the 1-Lipschitz constraint: the *Discriminator* has reduced capacity and is unable to model complex functions, and without careful tuning of the clipping threshold, the interactions between weight clipping and the cost function can lead to either exploding or vanishing gradients [12].

The use of weight clipping is avoided by directly including a gradient penalty in the overall loss function that encourages the model to satisfy the Lipschitz constraint by penalizing it whenever the gradient norm of the *critic* deviates from 1 [12, 28].

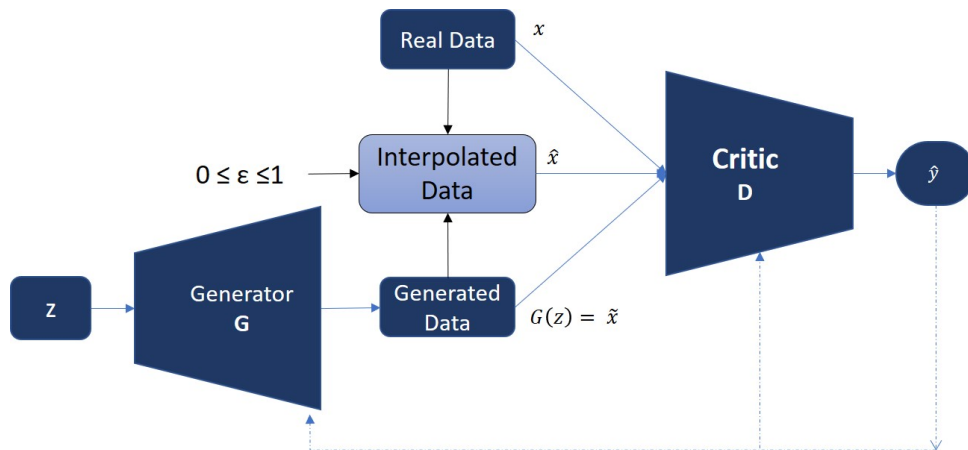


Figure 3.12: Schematic representation of a Wasserstein Generative Adversarial Network with Gradient Penalty

This loss function is expressed by:

$$L = \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}} [D(G(\mathbf{z}))] - \mathbb{E}_{\mathbf{x} \sim p_{data}} [D(\mathbf{x})] + \lambda \mathbb{E}_{\hat{\mathbf{x}} \sim p_{\hat{\mathbf{x}}}} [(\|\nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}})\|_2 - 1)^2] \quad (3.19)$$

where the first two terms correspond to the original critic loss from the WGAN and the third to the gradient penalty with λ as its weight factor (typically $\lambda = 10$). If we consider $\tilde{\mathbf{x}} = G(\mathbf{z})$ then $\hat{\mathbf{x}}$ is obtained by:

$$\hat{\mathbf{x}} = \epsilon * \tilde{\mathbf{x}} + (1 - \epsilon) * \mathbf{x} \text{ with } 0 \leq \epsilon \leq 1 \quad (3.20)$$

where ϵ is uniformly sampled between 0 and 1. In practice, this means that we use a set of interpolated samples, that result from randomly choosing points that lie on the lines that connect the batch of real samples to the batch of fake samples, and evaluate its gradients [28].

As a final note, batch normalization should not be employed in the *critic* since it creates correlation between samples in the same batch and therefore decreases the effectiveness of the gradient penalty loss. Nevertheless, WGAN-GP has shown improved performance and stability during the training process when compared to the previously presented methods [12, 28].

3.6 Regularization Techniques

Regularization techniques are a set of strategies that aim to improve the generalization of a model, i.e. its capacity to perform well on new unobserved data, and

in doing so, reduce overfitting (the situation in which a model performs well on the training set but poorly on the test set) [27]. The following are some of the more commonly used regularization techniques:

- **Dropout** is a very simple and effective regularization technique. It consists of setting a user-defined percentage of units in a network layer to 0, which reduces overfitting by making sure that the network doesn't rely solely on a certain group of units or memorizes the input data. During training, the dropped units are picked randomly and change for every batch. While testing, all the units are active and the weights are scaled-down according to the probability of the unit being retained during training [27, 28, 92, 93].
- **Batch Normalization (BN)** [94] is a technique that aims to reduce a phenomenon known as *covariate shift*, which occurs when, during training, as the previous layers change, the distribution of the input to a layer also changes. This impairs training by requiring careful parameter initialization and lower learning rates to avoid a possible collapse of the network. A batch normalization layer normalizes its inputs by subtracting the mean and dividing by the standard deviation of each training mini-batch. To preserve the model's capacity, β and γ are introduced as two trainable parameters, which shift and normalize the values, respectively. The BN algorithm, as presented by Ioffe and Szegedy [94], is as follows:

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad \text{mini-batch mean} \quad (3.21)$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad \text{mini-batch variance} \quad (3.22)$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\mu_{\mathcal{B}}^2 + \epsilon}} \quad \text{normalize} \quad (3.23)$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \mathbf{BN}_{\gamma, \beta}(x_i) \quad \text{scale and shift} \quad (3.24)$$

where $\mathcal{B} = \{x_{i \dots m}\}$ is the mini-batch, and γ and β are parameters to be learned. It is worth noting that this technique frequently eliminates the need for dropout, with recent works resorting only to batch normalization [28, 93, 94].

- **Early Stopping** consists on monitoring a specified metric and stopping the training procedure when it hasn't improved in a user-defined number of epochs.

The weights of the best epochs are saved, so that once the training is halted, it is possible to retrieve the best recorded set of parameters. This technique has the advantage of requiring no change to the architecture and overall training procedure of the model. It should be noted, however, that as the performance of the model is typically evaluated on a validation set using the defined metric, the number of available training samples decreases [27].

RNN Generator

4.1 Introduction

Recurrent architectures are amongst the most promising methods for generating new molecules [9, 10, 47, 53, 54]. However, one current challenge consists in finding the optimal architecture and parameters for the recurrent network that assures the generation of valid molecules that span the chemical space. Also, with the exception of the work of Zheng et. al [10], previous approaches have not explicitly considered stereo-chemistry which is of the utmost importance in drug design. Moreover, to optimize the probability of finding interesting hits for a given target, drug candidates should be produced chemically diverse while containing similar chemical properties to already known ligands [47]. Although many researchers have studied DL’s application to produce molecules as drug candidates, most of the existing generative models do not consider diversity (Tanimoto similarity) as one of the objectives in the generated libraries [95].

This chapter presents a comprehensive study on different RNN frameworks applied to molecule generation with SMILES notation. More specifically, the impact of different recurrent architectures and its inherent parameters are analyzed in terms of the rate of valid molecules, diversity in the generated compounds, and the time required to train the models. The effect of applying embedding layer or one-hot encoding in the validity, diversity of the generated compounds, and speed of generating the results is also studied. Further, the effect of using a different type of tokenization for the SMILES strings is explored, such as character by character (token by token), as proposed in the work of Olivecrona et al. [9], or by considering the stereo-chemistry notations with combined tokens grouped by a pair of square brackets [10]. This molecular information is vital in medicinal chemistry, but most works often overlook it due to its complexity in terms of increased vocabulary for

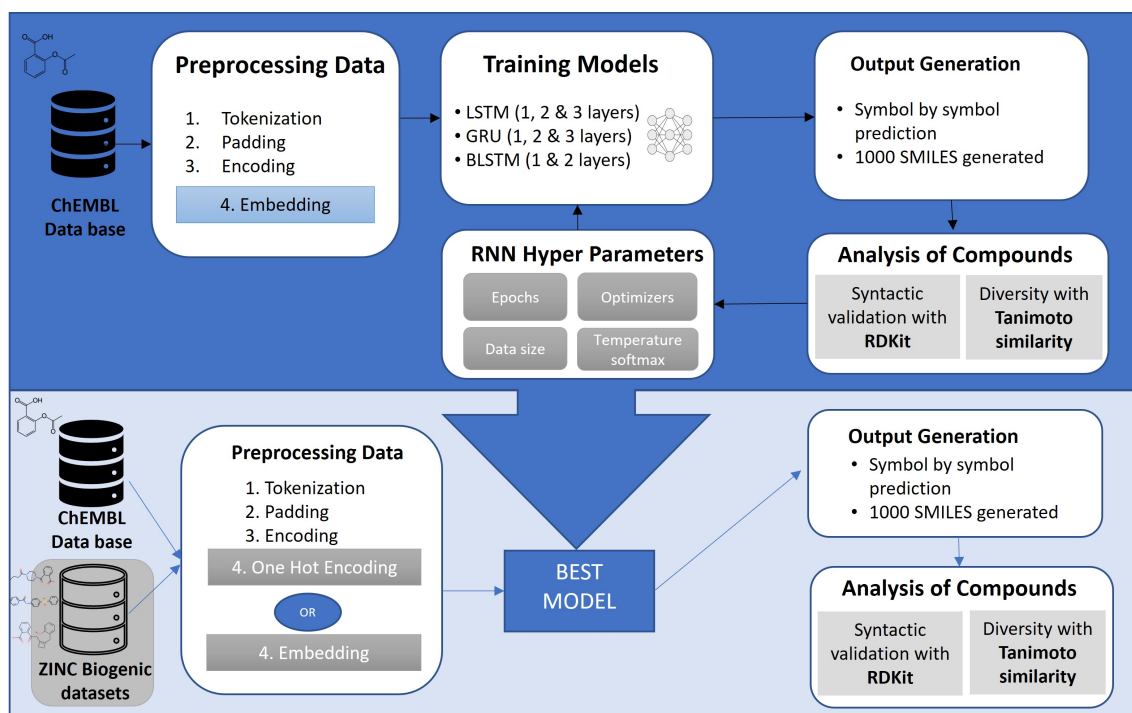


Figure 4.1: General workflow of the proposed Generator based on Recurrent Neural Network.

computational models.

4.2 Methods

This study is divided into three main parts. Firstly, three different types of Recurrent Neural Networks are explored: Long Short-Term Memory (LSTM), Gated Recurrent Units (GRU), and Bidirectional LSTM (BiLSTM) (see Section 3.3 on Page 24 for more details). Secondly, the effect of the various hyperparameters on the efficiency of the model is evaluated. Finally, the use of two different types of encoding in the structure of the DL model: embedding and one-hot encoding and the effect of using datasets with and without stereo-chemical information is analyzed. Figure 4.1 describes the general schema of the proposed model in order to generate valid plausible drug compounds.

4.2.1 Preprocessing Data

There are two main ways of encoding categorical data: embedding and one-hot encoding. The first preprocessing steps are common to both techniques and correspond

to steps 1 through 3 in Figure 4.2. First, the SMILES strings must be tokenized. This tokenization is performed character by character, with the atoms ‘Cl’ and ‘Br’ being previously substituted by the characters ‘L’ and ‘R’, respectively. In this way, each chemical symbol corresponds to a single token. As a second step, two extra tokens are added; the character ‘G’ is added at the beginning of each SMILES, and the character ‘A’ is added at the end and is also used for padding to the length of the longest allowed SMILES string. The set of these characters comprises the vocabulary. Thirdly, a dictionary with as many entries as the length of the vocabulary, voc , is created, which associates each token to an integer so that the SMILES string can be encoded as a list of integers. To perform one-hot encoding, each encoded token is transformed into a binary vector filled with zeros, except for the index that corresponds to the encoded token. This results in a matrix where each column is a one-hot vector. The size of this matrix is $voc \times \ell$, where ℓ is the maximum length of the sequence (defined by the user). As an example, Figure 4.2 shows the one-hot encoding of a molecule of Acetylsalicylic Acid. An embedding layer can be seen as a lookup table that converts each encoded token into a dense vector of a user-defined length and is an alternative to one-hot encoding. Therefore it results in a matrix of size $|voc \times e|$, where e is the embedding dimension. The values of the embedding are trainable parameters which allows the model to find its own representation for each token. After passing the sequence through the embedding layer, one obtains a matrix of size $|\ell \times e|$, which is then passed on to the following layers. It should be noted that for the biogenic dataset, the tokenization process is slightly different since it includes combined tokens, meaning that we consider sections of the SMILES string that are enclosed in brackets (for example, ‘[C@@H]’ and ‘[N+]’) as a single token, which results in an extended vocabulary [10].

4.2.2 Training Models

RNN models [96] can be used to generate sequences one token at a time, as these models can output a probability distribution over all possible tokens at each time step, therefore, they can predict the next token based on all the tokens seen so far. Eight different model structures are analyzed: GRU [84], LSTM [82], and BiLSTM; for the first two, 1, 2, and 3 layers are considered, for the last one, only the influence of 1 and 2 layers is studied. Figure 4.3 shows the main structure of an RNN model. In this example, the RNN model processes a sequence of data $X = x_1x_2 \dots x_\ell$ by taking as input each item x_i in the sequence, passing it through a series of gates and

4. RNN Generator

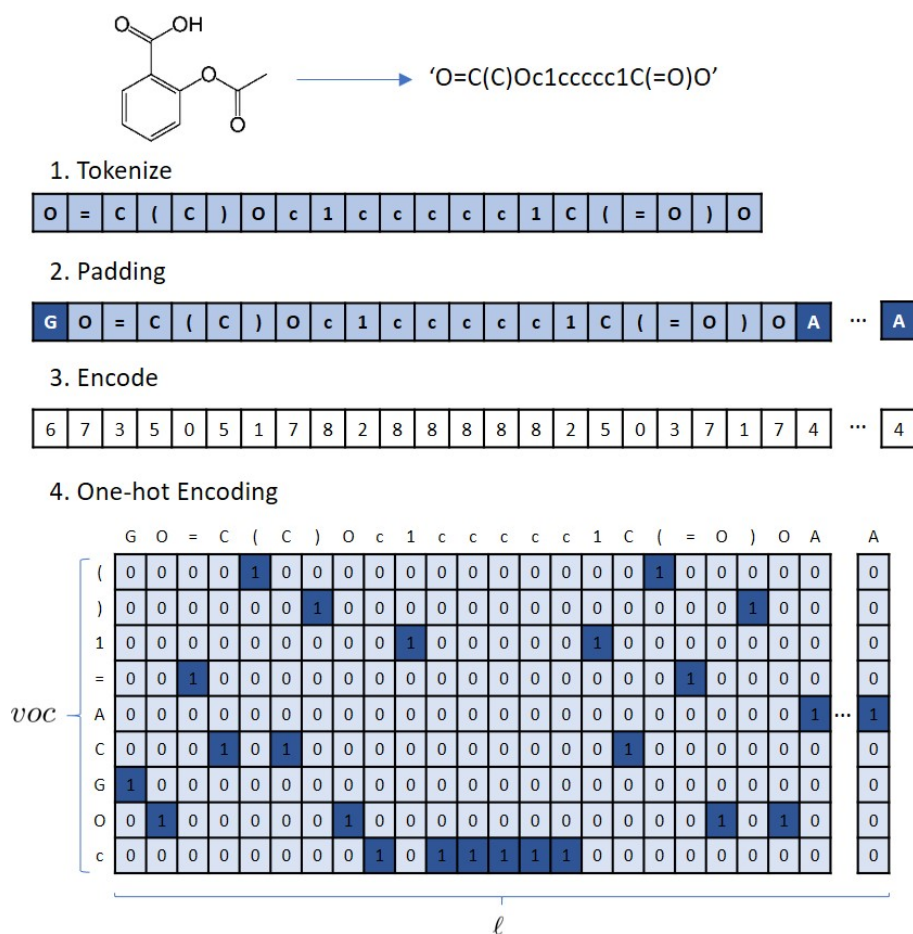


Figure 4.2: Data preprocessing for the molecule of Acetylsalicylic Acid using One-hot Encoding.

creating the output vector $Y = y_1 y_2 \dots y_\ell$.

The RNN layers are followed by a dense layer and a neuron unit with a *softmax* activation function with temperature. A dense layer is a linear operation in which every input is connected to every output by a set of weights. After each RNN layer, dropout is also added as a regularization technique [92]. During the training step, this type of layer randomly sets a user-defined percentage of the input units to 0, which reduces overfitting by making sure that the network doesn't rely solely on a certain group of units. The loss L is calculated at each position as the categorical cross-entropy between the predicted and actual next token and the network's weights are updated based on the gradient of this loss function by the optimizer. As a starting point, the 'Adam' optimizer is used and then the performance of three other optimizers are also evaluated: "Adam_clip" (the Adam optimizer with the gradients clipped to a maximum value of 3), "SGD" and "RMSProp". The remaining models

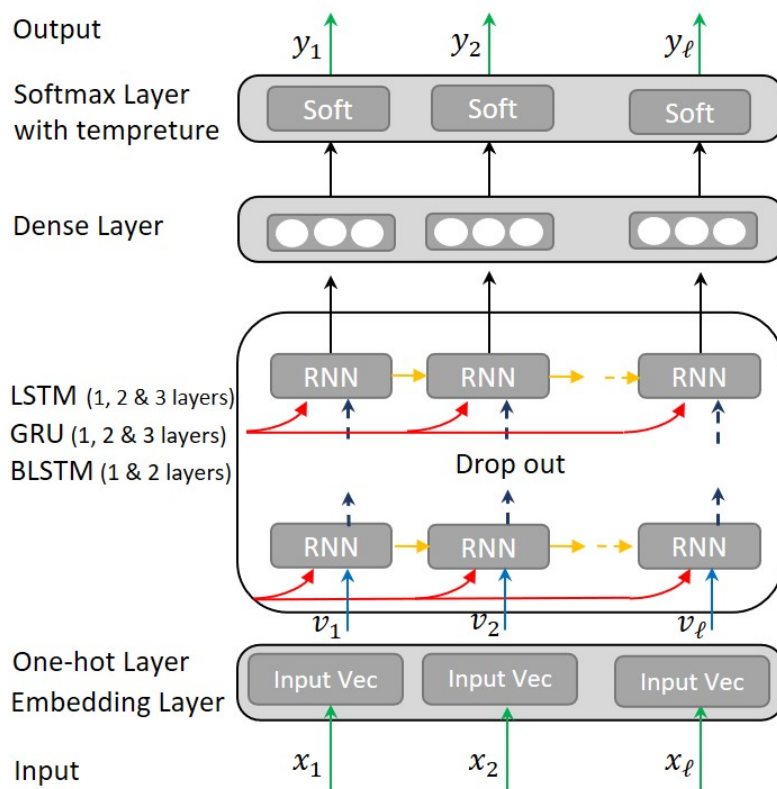


Figure 4.3: General structure of RNN Models for producing SMILES strings.

are implemented in the same way, differing only in the type of neural network and its layers.

The “Teacher Forcing” algorithm [27] is used during the training of the RNN and corresponds to always inserting the correct token at the following time step, independently of the predicted token at the current step. By doing so, we prevent the accumulation of errors due to wrong predictions, which in turn allows the model to converge faster. Early stopping is a regularization technique that can be employed during the training of a model, and it is used in some of the experiments. It essentially monitors the performance at the end of each epoch and stops training when it hasn’t improved in the last n epochs, where n is user-defined. In doing so, it avoids overtraining the model and prevents overfitting. Overfitting occurs when the model resorts to “memorizing” the training data and therefore it is unable to generalize well to new data samples [97].

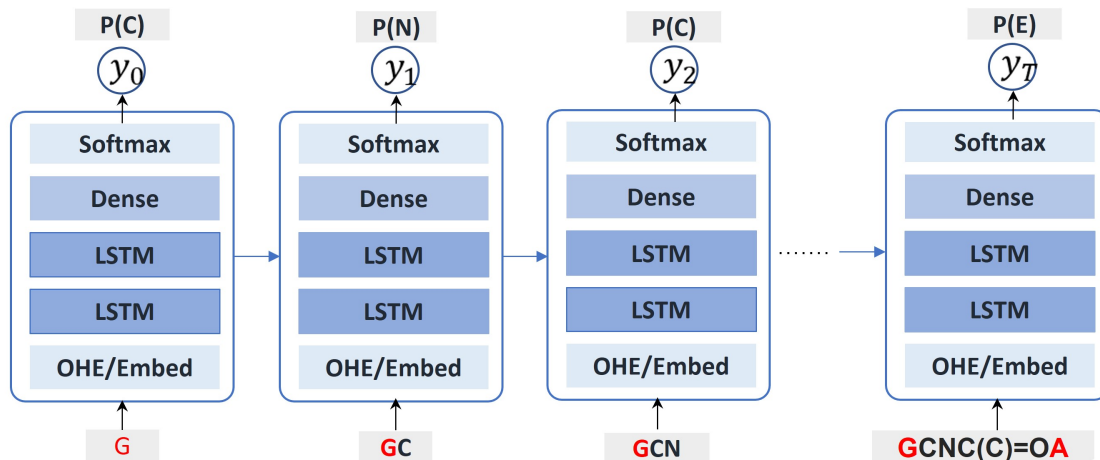


Figure 4.4: Workflow for generating the SMILES with a trained network. At every time step t , the model samples the next token of the SMILES y_t from the probability distribution and introduces it as the next input x_{t+1} .

4.2.3 Output Generation

RNN models generate sequences one token at a time, as these models can output a probability distribution over all possible tokens at each time step. Given a certain input, the RNN predicts the next token; it is worth noting that this input can be one or more tokens in length. Figure 4.4 shows a general workflow for this process for an example of SMILES string “GCNC(C)=OA”. We start by feeding the network with $x_0 = \text{'G'}$ and then it predicts the next token $\tilde{x}_1 = \text{'C'}$. The network will then predict $\tilde{x}_2 = \text{'N'}$ given $\{x_0, \tilde{x}_1\} = \text{'GC'}$, and so forth until the ‘A’ token is predicted or the maximum length of the sequence ℓ is reached, at which point the generation of the sequence ends.

4.2.4 Validation Strategy

The SMILES generated by the proposed RNN models are syntactically and bio-chemically validated by RDkit [35]. It is also important for a model to generate diverse SMILES strings both in terms of *Internal Diversity* (the diversity inside the generated compounds set) and *External Diversity* (the diversity between the generated and the training dataset). To evaluate the diversity, we resort to the Tanimoto Similarity T_s , which computes the similarity between two molecules in terms of their circular fingerprints. The Tanimoto distance can be defined as $1 - T_s$. From it, we can define the diversity between two sets of generated molecules A and B as the

average of the Tanimoto distance between every single pair of molecules:

$$Div(A,B) = \frac{1}{|A| \cdot |B|} \sum_{a \in A} \sum_{b \in B} (1 - T_s) \quad (4.1)$$

Therefore, to obtain the internal diversity we simply compute $Div(A,A)$ and for the external diversity we compute $Div(A,B)$, where A is the set of generated SMILES and B is the set of the training data.

4.3 Experimental Results and Discussion

This section describes the exhaustive analysis that has been done to find the best set of hyperparameters for the proposed RNN model. The goal is to find a model that can generate new compounds with a high percentage of valid molecular structure while simultaneously being as diverse as possible and different from the training dataset.

4.3.1 Datasets

To do the experiments, two different datasets are considered. The first and the one that is used for most of the experiments, is analogous to the one used by Olivecrona et al. [9] which was obtained from the ChEMBL database [98] and contains 1,179,477 SMILES Strings previously filtered and canonicalized by RDKit. The molecules in this dataset contain only the elements $\{H, B, C, N, O, F, Si, P, S, Cl, Br, I\}$ and have between 10 and 50 heavy atoms. The second dataset is from the ZINC biogenic library [99] and is the same used in the work of Zheng et al. [10]. It comprises 153,733 biogenic structures in canonicalized SMILES format, therefore, unlike the previous dataset, this one includes stereo-chemical properties. This dataset does not contain metal elements, or molecules with less than 10 non-hydrogen atoms or more than 100.

4.3.2 Performance Analysis and Results

The primary purpose of the first set of tests is to analyse and evaluate the set of network hyper-parameters that generates the highest ratio of valid SMILES for the ChEMBL dataset. Table 4.1 shows the obtained results for the 8 proposed models. It contains the percentage of valid and unique SMILES and the training time. A

SMILES is considered "valid" if it passes the validation process (see Section 4.2.4) and "unique" if it is not repeated in the generated set. The models are trained on 100,000 SMILES strings from the ChEMBL dataset by using embedding as the type of data encoding. The results are obtained from evaluating 1,000 generated SMILES strings. The models are trained for 25 epochs with a batch size of 16, a dropout rate of 0.2, and using the Adam optimizer. The network units are set to 512, the embedding dimension to 256, and the maximum length of a sequence to $\ell = 100$. For the sampling process, the temperature is fixed at 0.75. The highest validity values are obtained using the LSTM cells, reaching a maximum of 96.20% for Model 3 and 100% of unique SMILES strings. Regarding the GRU cells (models 4 to 6), the validity decreased from 92.70% when using 1 layer to 0.80% for 3 layers. The BiLSTM based models performed the poorest with 0.70% and 0.0% of validity for 1 and 2 layers, respectively, often generating empty SMILES strings or constantly generating the same non-valid SMILES. In total, they were not able to capture the complexities of the SMILES syntax. From the results presented in Table 4.1, we discarded the use of GRU and BiLSTM cells and chose the two best models: model 2 and 3 with a validity of 95.70% and 96.20%, respectively, to use in the following experiments.

Table 4.1: The training time and percentage of valid and unique SMILES generated with 8 different models.

Model	Description	Valid %	Unique %	Time (hh:mm:ss)
Model 1	LSTM - 1 Layer	94.6%	100.0%	0:39:56
Model 2	LSTM - 2 Layer	95.7%	100.0%	1:08:38
Model 3	LSTM - 3 Layer	96.2%	100.0%	1:38:09
Model 4	GRU - 1 Layer	92.7%	99.9%	0:38:16
Model 5	GRU - 2 Layer	11.9%	99.9%	1:00:13
Model 6	GRU - 3 Layer	0.8%	100.0%	1:21:26
Model 7	BiLSTM -1 Layer	0.7%	1.6%	0:58:38
Model 8	BiLSTM -2 Layer	0.0%	9.0%	1:39:31

The effect of validity and uniqueness through the epochs was analyzed. In particular, epochs 4, 8, 12, 16, 20, 24 and 28 were examined for the two best models from the previous experiment. Fig 4.5 shows these results. It can be observed that the percentage of unique generated molecules stays close to 100% for the evaluated epochs and for both models, while the percentage of valid molecules is more erratic, reaching 97.2% at epoch 16 for the first model and 96.8% at epoch 12 for the second model.

The number of epochs was fixed at 16 and the impact of using the optimizers mentioned in 4.2.2 was evaluated. As shown in Table 4.2, for both models, "Adam",

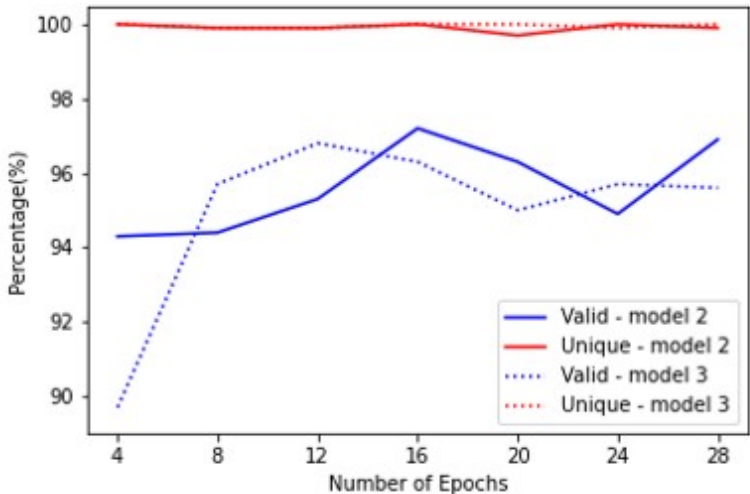


Figure 4.5: Training epochs and percentage of valid and unique molecules for Models 2 and 3.

“Adam_clip”, and “RMSprop” performed well, keeping the uniqueness close to 100% and the validity higher than 94% but the use of “RMSprop” yielded a higher number of valid molecules: 97.5% and 96.8% for model 2 and 3, respectively. Therefore, from here onwards, the “RMSprop” optimizer was used.

Table 4.2: The training time and percentage of valid SMILES and diversity with 4 different optimizers for models 2 and 3.

Model	Optimizer	Valid %	Internal Diversity	External Diversity
Model 2	Adam	95.9%	0.864	0.875
Model 2	Adam_clip	94.5%	0.863	0.874
Model 2	SGD	4.6%	0.876	0.899
Model 2	RMSProp	97.5%	0.862	0.875
Model 3	Adam	94.6%	0.859	0.872
Model 3	Adam_clip	95.2%	0.861	0.873
Model 3	SGD	4.2%	0.874	0.899
Model 3	RMSProp	96.8%	0.866	0.876

The effect of using different batch sizes: 16, 32, 64, 128 and 256 for models 2 and 3 was then tested. These models were set to run for 24 epochs using early stopping with a patience of 5, though it should be noted that early stopping was never applied because all models ran until the final epoch. From the observed results in Table 4.3, we can conclude that the highest percentage of validity was obtained when using a batch size of 16, yielding 97.5% and 96.9% for models 2 and 3, respectively. It is also worth noting that, by comparing Table 4.2 with Table 4.3, an increase in training epochs from 16 to 24 was not accompanied by an increase in validity for model 2, and for model 3 there was only an increase of 0.01%.

Table 4.3: The training time and percentage of valid and unique SMILES with different batch sizes for models 2 and 3.

Model	Batch Size	Valid %	Unique %	Time (hh:mm:ss)
Model 2	16	97.5%	99.9%	1:08:31
Model 2	32	94.7%	99.9%	0:38:51
Model 2	64	94.7%	99.8%	0:29:29
Model 2	128	94.6%	100.0%	1:23:49
Model 2	256	93.7%	99.9%	0:21:21
Model 3	16	96.9%	100.0%	1:38:07
Model 3	32	96.2%	100.0%	0:56:55
Model 3	64	95.3%	99.9%	0:44:03
Model 3	128	94.9%	99.9%	0:35:27
Model 3	256	95.8%	100.0%	0:31:43

To study the impact of the number of training samples on the quality of the generated data, Model 2 was trained for 16 epochs and with a batch size of 16 on successively increasing datasets. The results are presented in Table 4.4 where it is clear that an increase in the training samples results in a higher percentage of valid molecules without negatively affecting the external diversity, with the downside that it takes longer to train.

Table 4.4: The percentage of valid SMILES and diversity for Model 2 with different number of training samples.

Samples Number	Valid %	Int Div	Ext Div	Time (hh:mm:ss)
10,000	85.0%	0.858	0.873	0:04:49
50,000	92.8%	0.861	0.873	0:23:27
100,000	95.4%	0.868	0.876	0:45:19
200,000	96.7%	0.864	0.874	1:30:15
500,000	98.7%	0.869	0.877	3:41:24

Table 4.5 presents the results from evaluating the effect of different sampling temperatures during the generation phase when using Model 2 trained on 100,000 samples. We can observe that there is a trade-off between diversity and validity, the lower the sampling temperature, the higher the percentage of valid molecules and the lower the diversity. We kept on using a sampling temperature of 0.75 as a compromise between diversity and validity.

Table 4.5: The percentage of valid SMILES and diversity for different values of Sampling Temperatures

Model	Sampling Temperature	Valid %	Internal Diversity	External Diversity
Model 2	0.50	98.1%	0.844	0.873
Model 2	0.75	94.9%	0.862	0.874
Model 2	1.00	88.6%	0.877	0.879
Model 2	1.20	75.3%	0.885	0.884

To compare between the use of one-hot encoding and embedding, all the parameters were kept fixed as in the previous experiment but with the number of epochs set to 24 without early stopping. As can be seen in Table 4.6, the results were quite similar both in terms of validity of the generated data and diversity. They were better when using model 3, and the percentage of valid molecules slightly higher for the embedding, at 96.5% when compared to the OHE, with 96.1%. It should also be noted that the percentage of unique SMILES was constantly close to 100.0%.

Table 4.6: The percentage of valid SMILES and diversity for two different models and two types of input encoding.

Model	Encoding	Valid %	Internal Diversity	External Diversity
Model 2	OHE	94.2%	0.863	0.874
Model 2	Embedding	95.9%	0.867	0.876
Model 3	OHE	96.1%	0.864	0.875
Model 3	Embedding	96.5%	0.865	0.875

Table 4.7: Comparison of the results obtained for model 3 when applied to two different datasets: “Biogenic” and “ChEMBL” and using two types of encoding “Embedding” and “OHE” with Number of Samples equal to 100,000.

Dataset	Encode	Ep	Valid%	Uni%	IntDiv	ExtDiv	Time
ChEMBL	Embed	43	97.2%	100.0%	0.868	0.877	3:02:17
Biogenic	Embed	18	93.6%	99.1%	0.859	0.899	1:19:31
ChEMBL	OHE	62	94.8%	100.0%	0.859	0.873	3:58:40
Biogenic	OHE	61	94.7%	99.1%	0.866	0.902	4:00:30

The best model found (model 3 with “RMSProp” as the optimizer and a batch size of 16) was applied to both of the previously mentioned datasets and using two different types of encoding: Embedding and One-hot Encoding. The models were trained on 100,000 SMILES and until convergence. The results are represented in Table 4.7 where we can observe that, when using ‘Embedding’ as the encoding process, 97.2% and 93.6% of the generated SMILES were valid for the “ChEMBL” and “Biogenic” dataset, respectively. It is interesting to note that, even though the hyper-parameters for our model had been tuned for the “ChEMBL” dataset, the model was able to return a really high validity for only 18 epochs of training for the “Biogenic” dataset, which contains more complex information relating to the stereo-chemical properties of the molecules. For an encoding process of OHE, both datasets reported similar results in terms of percentage of valid molecules, with 94.8% and 94.7% for the “ChEMBL” and “Biogenic” datasets, respectively. For OHE both datasets converged at around the same epoch taking, therefore, a similar time to train, while for ‘Embedding’, these values were clearly distinct.

4.4 Conclusions

From the comprehensive study of RNN architectures for molecular generation presented in this chapter, it was possible to conclude that LSTM cells outperform other types of RNN when it comes to generating molecules as SMILES strings. Also, as expected, lower batch sizes and an increased number of training data resulted in a better performance. Regarding the optimizer, “RMSProp” yielded the best results while “SGD” performed extremely poorly producing only about 4% of valid molecules. The chosen strategy produced an optimized model that is able to generate 98.7% of valid SMILES strings with high internal and external diversity which shows improved performance when compared to the current literature [9]. When considering the “Biogenic” dataset, known for its higher level of complexity, a validity of 94.7% and a diversity of 0.90 were obtained. Regarding the use of either embedding or OHE as the input layer, the differences in validity and diversity were not particularly significant but it is worth mentioning that the use of an embedding layer required significantly less epochs and therefore, less training time.

Overall, the results presented in this chapter indicate not only that RNN generators are powerful tools for learning and generating new molecules but also that it is possible to obtain good results when considering stereo-chemical information. As this type of information is of the utmost importance in drug design, we believe that further frameworks that resort to RNNs for molecular generation should take in consideration the stereo-chemistry of the molecules.

Technical notes: The results shown in this chapter were coded in Python 3.8.3. The models were coded using Tensorflow 2.3. The chemistry library used throughout is RDKit 2020.09.2 [35]. The GPU hardware used to train and sample the models was Nvidia RTX 2070 8GB of GDDR6 VRAM using CUDA 10.1.

GAN-based Framework

5.1 Introduction

GANs have been successfully employed in a variety of research fields, but their application to drug design is only at the beginning. This is due to the fact that GANs cannot be straightforwardly applied to discrete data like SMILES strings. The reason being that the process of generating discrete samples introduces a non-differentiable layer, which implies that the backpropagation algorithm cannot be applied [100]. In order to do so, and as explored in Section 2.4 on Page 14, researchers have resorted to RL or finding an alternative continuous representation for the discrete data [11]. The former has a tendency to focus on local minima and therefore return very similar and sometimes duplicate molecules while the latter, even though it is a novel approach, does not consider stereo-chemistry nor the diversity of the generated compounds.

In this Chapter, the aforementioned issues are addressed. Firstly, an exhaustive grid search of an Encoder-Decoder model is performed so that a continuous representation of the SMILES strings can be found. SMILES strings that contain stereo-chemical information are included in the training data, which is vital when working in drug design but often overlooked due to its higher complexity. Secondly, a GAN is trained on the latent space vectors created by the Encoder-Decoder model, and its capability to create diverse molecules is evaluated. Lastly, two approaches are studied to optimize the framework: TL and a FeedbackGAN-inspired method [13] with

the goal of producing molecules that exhibit a high affinity to the KOR receptor.

5.2 Methods

The general proposed framework is illustrated in Figure 5.1 and is composed of an autoencoder, more specifically, an encoder-decoder architecture based on RNNs [84,86], and a Wasserstein GAN with gradient penalty [12]. The encoder-decoder architecture allows the model to learn a context vector (Figure 5.1-C) that summarizes the SMILES strings in such a way that they can be reconstructed only from their context vectors. By passing a dataset that consists of SMILES through the encoder (Figure 5.1-A), an equivalent dataset made up of context vectors is obtained. This new dataset is used as real data to train the GAN so that, once trained, its generator is able to generate new samples from the same distribution as the context vector’s dataset. These samples are then passed through the decoder (Figure 5.1-B) in order to obtain the corresponding SMILES strings. By combining these two models, it becomes possible to train the GAN, surpassing the differentiation problem associated with categorical data, such as SMILES strings, that would otherwise arise [8].

5.2.1 Encoder-Decoder Model

The encoder-decoder model is an autoencoder that works with sequences of data by resorting to recurrent networks. The encoder-decoder architecture allows the learning of a context vector. The model summarizes the input SMILES in such a way that it can reconstruct them only from the context vector. The encoder, Figure 5.2-A, contains an embedding layer and two bidirectional LSTM layers with batch normalization in between. From these bidirectional layers, the final cell and hidden states, from both directions, are retrieved and concatenated. Then the result is passed through a dense layer with a size equal to the desired length of the context vector. Next, there is a dense layer followed by batch normalization and a Gaussian noise layer (during training) to make the model more flexible and the context vector more robust. Regarding the decoder, Figure 5.2-B, the context vector serves as input to four independent dense layers whose goal is to reconstruct the hidden and cell states that will be given as initial states to two LSTM layers. These stacked LSTM layers, with batch normalization in between, are followed by a dense layer with *softmax* activation function so that it outputs the probabilities associated with the next token. It should be noted that, during training, the “Teacher Forcing”

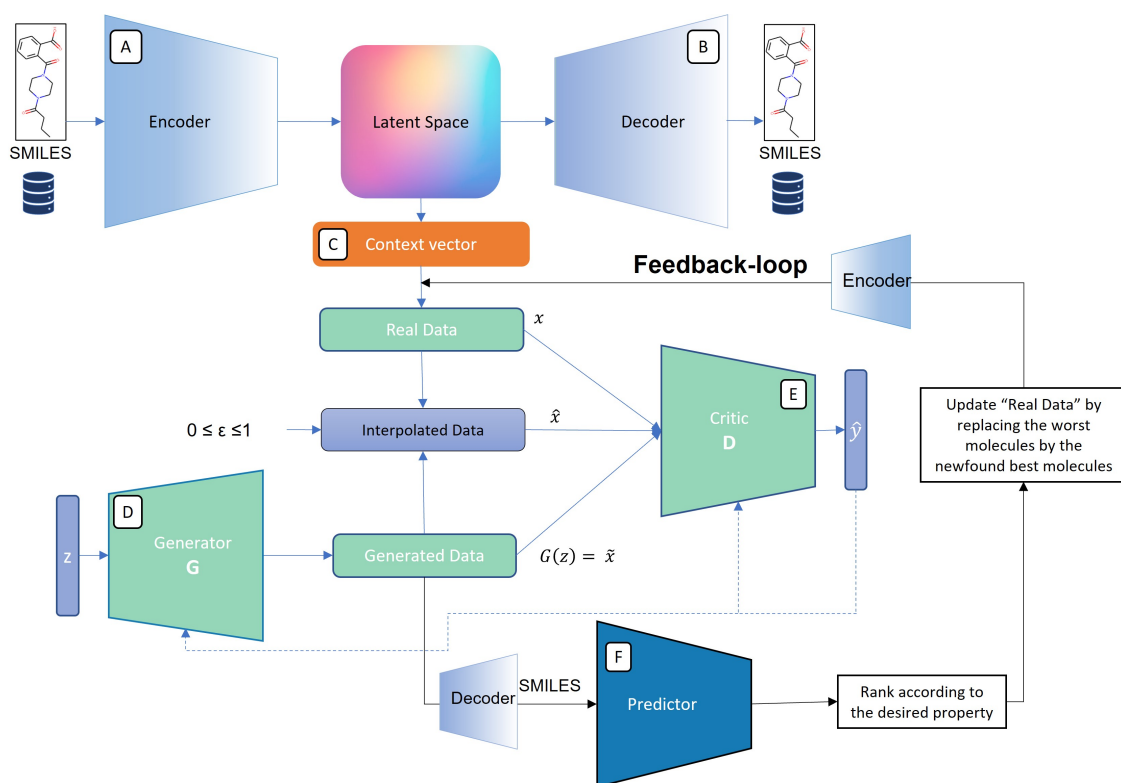


Figure 5.1: The general workflow of the proposed model that is composed of an Encoder-Decoder that converts SMILES into latent space vectors that are then used as real data in the training of a WGAN-GP network that comprises a Generator and Critic.

algorithm is employed, and so the target output of the decoder is the same as its input but shifted by one time step. The complete model is trained using the categorical cross-entropy between the input and the predicted output as the loss function. The network’s weights are then updated by the Adam optimizer, considering the gradient of the loss.

5.2.2 Wasserstein GAN with Gradient Penalty

The second part of the framework is based on a Wasserstein GAN with gradient-penalty (WGAN-GP) (see Section 3.5 on Page 32) since this type of model has better performance and stability during the training process [12] when compared to the traditional GAN as proposed by Goodfellow et al. [8]. GANs cannot be directly applied to categorical data, like SMILES strings, due to the fact that the sampling process at the end of the generator does not allow for the backpropagation of the errors through that layer. However, since the previously mentioned encoder-decoder

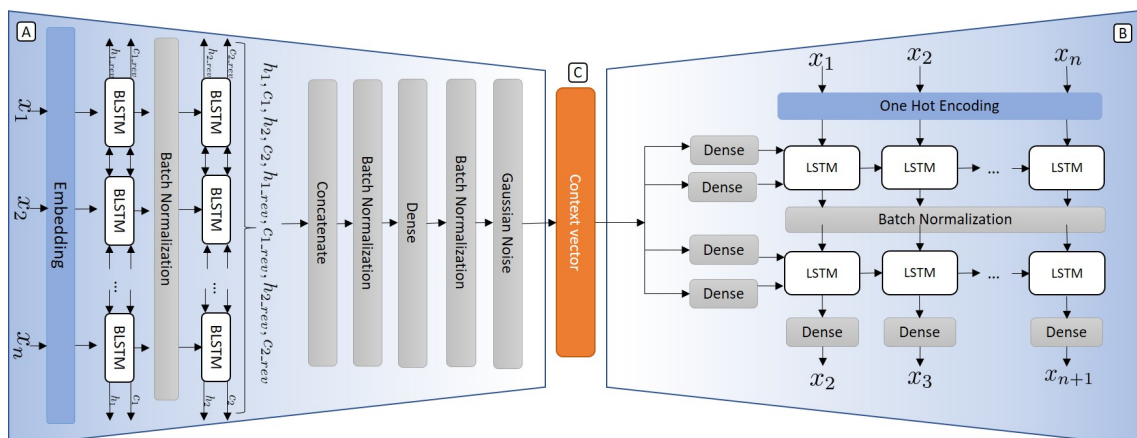


Figure 5.2: The detailed structure of the Encoder (A) and Decoder (B) applied in the framework. This model is used to convert the SMILES strings into vectors in the latent space (context vector (C)).

model can find an alternative continuous representation for said SMILES strings, this new latent space representation can be used as real data to train the WGAN-GP without requiring intrinsic changes to the model. As the training data now comprises of vectors, both the Critic and the Generator can be simple Feed Forward Neural Networks.

5.2.3 Case-study: Kappa Opioid Receptor

The Kappa Opioid Receptor (KOR) is one of four opioid receptors that belong to the G-protein-coupled receptors (GPCR) superfamily. Research on opioid receptors, particularly on KOR, has been gaining momentum as it mediates affective disorders such as depression and anxiety, neurological diseases like epilepsy, but also pain and drug addiction making it a promising pharmacological target [101–104]. It is possible to identify two main groups of ligands that bind to the KOR: antagonists and agonists. An agonist is a drug that binds to the receptor and activates it triggering a biological response. An antagonist is a drug that also binds to the receptor but blocks its biological response.

The goal of the current experiment is to find ligands that bind antagonistically to the KOR as evidence suggests that KOR antagonists may serve as treatment for depression [102], anxiety [105], and psychostimulant disorders by attenuating, for example, cocaine consumption [106].

In order to attain it, the generator of the WGAN-GP model must be optimized to

generate compounds with a high affinity to bind antagonistically to the KOR. To measure this, the $pIC50$ is used, which is the negative logarithm of the half maximal inhibitory concentration. Therefore, the higher the $pIC50$, the more potent the inhibition will be.

5.2.4 KOR Binding Affinity Predictor

So that the following optimization processes can be evaluated, a QSAR model, henceforth denominated by Predictor, based on the work of Pereira et al. was implemented [107]. This work also showed that a RNN Predictor has improved performance when compared to standard QSAR approaches [107]. This particular Predictor aims to predict the binding affinity of a given molecule for the KOR as measured by the $pIC50$. The Predictor is a LSTM-based model that receives tokenized and padded SMILES strings (see Section 4.2.1 on Page 42) as input which are then passed through an embedding layer followed by two LSTM layers and two dense layers. Since this is a regression problem the last layer has a single unit and a linear activation function. The use of RNN-based QSAR models is of particular interest for two reasons: first, this type of model works with inputs with different lengths and second, it also works with SMILES strings which means that there is no need to find other types of molecular representations that might add human bias [108].

Before training the Predictor, the labels were normalized using percentile normalization which reduces the impact of outliers in the overall performance of the model. The dataset was randomly split into training/validation (85%) and testing (15%). A 5-fold cross-validation strategy was implemented. This results in 5 trained models that are combined and evaluated using the Mean Squared Error on the hold-out test set. In this way, when given a new molecule, the final prediction of the overall model is the average of the predictions from the 5 independent models.

To avoid overfitting, early stopping was employed (see Section 3.6 on Page 37) with a patience of 15 epochs which stops the training of the model if there has been no improvement in the last 15 epochs.

5.2.5 Optimization through Transfer Learning

After training the implemented WGAN-GP model, the generator is able to produce context vectors that are then decoded into SMILES strings that span the chemical

space. In order to optimize the framework, Transfer Learning was employed to bias the model towards specific properties by retraining the WGAN-GP model with a smaller dataset containing only molecules with the desired property.

In order to do this, the KOR dataset (dataset that contains SMILES strings and their corresponding pIC_{50} values) (ChEMBL identifier 237) was split in two subsets: one with only pIC_{50} values higher than 7 and another with values lower than 7. By doing so, two different scenarios can be evaluated: maximization and minimization of the KOR affinity, respectively.

Once TL has been employed, 1,000 SMILES strings are sampled and evaluated by resorting to the Predictor.

5.2.6 Optimization through FeedbackGAN

As a second approach to fine-tune the WGAN-GP model, a strategy based on FeedbackGAN [13] was implemented. FeedbackGAN is an optimization framework proposed by Gupta and Zou [13] that resorts to a feedback-loop and a function analyzer to optimize a GAN towards the space of desirable properties.

In the context of the current problem, the Predictor (see Section 5.2.4 on Page 57) takes the place of the function analyzer. After training the WGAN-GP model, the Predictor is linked to the GAN through a feedback mechanism. The GAN then enters a retraining phase in which, at the end of each epoch, the Generator is used to sample a set of new molecules that are fed to the Predictor to be evaluated. Taking in consideration the predictor’s output, the n best SMILES strings replace the worst n SMILES strings from the training data that is used in the following epoch. By doing so, the training data is constantly being updated with new and better molecules according to the goal that has been set (either maximization or minimization). This results in a fine-tuned Generator that gradually approaches the space of the desired property. Figure 5.3 represents this optimization mechanism.

The framework is evaluated by sampling 1,000 SMILES strings and comparing them to 1,000 SMILES sampled from the WGAN-GP before optimization.

5.2.7 Validation Strategy

The encoder-decoder is a deterministic model; therefore, it is evaluated based on the percentage of molecules that it can correctly reconstruct. It is worth noting that a

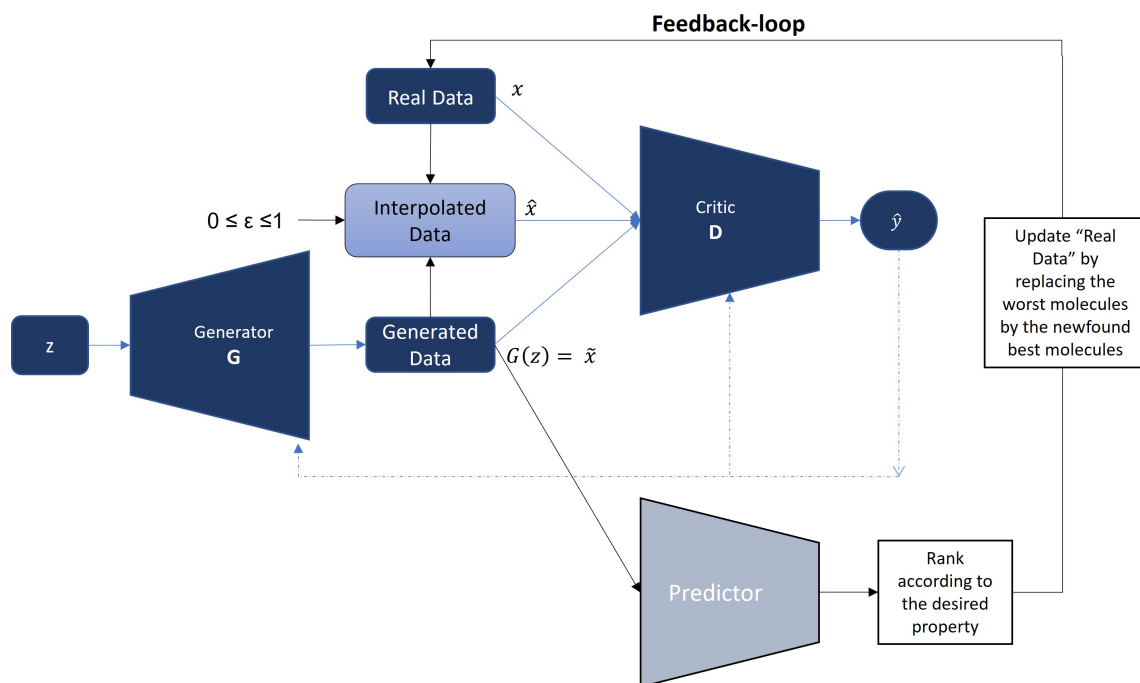


Figure 5.3: Implemented optimization mechanism via FeedbackGAN. During the optimization phase, at the end of each epoch, a set of new SMILES strings are sampled, evaluated and ranked according to the output of the Predictor. The best scoring samples are then incorporated into the “Real Data” by replacing the worst molecules.

correctly reconstructed SMILES string is automatically valid.

The generated SMILES are syntactically and biochemically validated by RDkit [35]. From the group of valid molecules, its uniqueness is also computed as the percentage of different generated valid molecules. Since it is not just important that the generated molecules are unique but also that they are diverse, the *Internal Diversity* (Int Div) and *External Diversity* (Ext Div) are also computed using the Tanimoto Similarity, similar to the previous experiment (see Section 4.2.4 on Page 46).

5.3 Experimental Results and Discussion

5.3.1 Datasets

The dataset used to do the experimental analysis on the Encoder-Decoder model in order to find the best architecture and set of parameters was the ChEMBL dataset [98].

Once the best architecture and set of hyperparameters had been defined, the model was trained on two other more complex datasets: `composed_dataset_1` and `composed_dataset_2` which contain 100,000 and 500,000 drug-like molecules, respectively, that were retrieved from the remaining datasets mentioned in Table 5.1. This resulted in datasets that include a wider variety of compounds and molecules with and without stereochemistry.

All the SMILES strings were canonicalized, and there were no duplicates. The SMILES strings are preprocessed by being tokenized character by character, adding ‘G’ as the first token of each SMILES and ‘A’ at the end and for padding. The SMILES are then either One-hot Encoded (OHE), where each token becomes a binary vector or passed through an embedding layer that converts each token into a dense vector that is learned by the model (Decoder).

Table 5.1: Summary of the datasets used throughout the experiment.

Dataset	# Compounds	Labeled	Observations
ChEMBL [98]	1,178,946	No	-
Zinc Biogenic [10]	108,283	No	-
ad2a	4,729	Yes	ChEMBL ID 251
KOR	5,262	Yes	ChEMBL ID 237
jak2	1,697	Yes	ChEMBL ID 2971
bbbp [109]	1,340	Yes	-
<code>composed_dataset_1</code>	100,000	No	-
<code>composed_dataset_2</code>	500,000	No	-

5.3.2 Encoder-Decoder Model

This section explains the experimental analysis and exhaustive grid search strategy that was employed to find the best structure and set of hyper parameters for the Encoder-Decoder model. The aim of this model is to convert the molecular compounds into continuous latent space vectors and reconstruct them correctly. It should be noted that two different structures for the Encoder input layer were considered: one with One Hot Encoding (OHE) and the other with Embedding (see Section 5.2.1 on Page 54 for more details).

Encoder-Decoder with OHE Structure

This section presents the results for the model with the structure that contains OHE layer as input to the Encoder. Table 5.2 shows the set of parameters that were

studied in this part. The models are evaluated on their ability to correctly reconstruct 1,000 SMILES strings from the training set and 1,000 SMILES strings from the hold-out test set. This approach was chosen in order to analyse the generalization capability of the model. The closer the percentage of correctly reconstructed molecules in the test set to the one in the train set, the better the generalization of the model.

Table 5.2: Search space for finding the optimal set of parameters of the proposed Encoder-Decoder model.

Parameters	Search Space
Number of Layers	[1,2,3]
Number of LSTM/BiLSTM Units	[256,512,1024]
Batch Size	[16,32,64,128,256]
Batch Normalization Momentum	[0.7,0.8,0.9,0.95]
Latent Dimension	[64,128,512,1024]
Noise Standard Deviation	[0.1,0.15,0.2,0.25]
Training Data	[10,000, 100,000, 200,000, 500,000]

In order to independently evaluate the effect of each hyperparameter, several experiments were conducted and evaluated on an hold-out test set that contained 1,000 SMILES strings.

Table 5.3 shows the results obtained when keeping all the hyperparameters fixed with the exception of the number of encoder BiLSTM layers which is always the same as the number of decoder LSTM layers. The number of training data was fixed at 100,000 SMILES strings, 512 units were used for the BiLSTM layers (256 for each direction) and for the LSTM layers. The batch normalization momentum was set to 0.9 and the Noise standard deviation to 0.1. The model was trained using the Adam Optimizer with a batch size of 128. As it can be seen, regarding the percentage of correctly reconstructed SMILES in the train set, all models performed well with 99.7%, 100.0% and 100.0% for 1, 2, and 3 layers, respectively. However, regarding the test set, the model with two layers clearly outperformed the remaining models by reaching 94.1% of correctly reconstructed compounds, showing its improved generalization capability. The use of two encoder BiLSTM layers was chosen for the following experiments and a structure of this model is represented in Figure 5.2 on page 56.

Regarding the number of BiLSTM/LSTM units, the results of the experiments are shown in table 5.4 from which it can be concluded that 512 units produces the model

Table 5.3: Results for different number of encoder BiLSTM layers and decoder LSTM layers (Encoder-Decoder with OHE structure). Dataset# =100,000 test#=1000 BiLSTM/LSTM units=512 Latent dimension=512 Batch size=128 Batch normalization momentum=0.9 Optimizer=Adam Noise std=0.1

Encoder BiLSTM layers#	Decoder LSTM Layers#	Last Ep.	%Correctly Reconstruct (Train)	%Correctly Reconstruct (Test)	%Valid (Train)	%Valid (Test)	Train Time (hh:mm:ss)	Total Run Time (hh:mm:ss)
1	1	48	99.7	90.5	100.0	96.2	00:35:10	02:05:03
2	2	33	100.0	94.1	100.0	97.7	00:55:55	02:30:45
3	3	27	100.0	91.2	100.0	97.7	01:12:11	02:48:49

with better generalization capabilities.

Table 5.4: Results for different number of BiLSTM/LSTM units (Encoder-Decoder with OHE structure). Dataset#=100,000 test#=1000 BiLSTM/LSTM layers=2 Latent dimension=512 Batch size=128 Batch normalization momentum=0.9 Optimizer=Adam Noise std=0.1

BiLSTM /LSTM units	Last Ep.	Correctly Reconstruct (Train)%	Correctly Reconstruct (Test)%	Valid (Train) %	Valid (Test) %	Train Time (hh:mm:ss)	Total Run Time (hh:mm:ss)
256	38	100.0	92.9	100.0	97.5	00:40:17	02:13:37
512	33	100.0	94.1	100.0	97.7	00:55:55	02:30:45
1024	45	99.6	89.6	99.8	94.7	03:03:34	04:34:52

After setting the number of units to 512, the effect of using different batch sizes: 16, 32, 64, 128 and 256 was evaluated. According to table 5.5, a batch size of 64 returned the best results with 93.7% of correctly reconstructed molecules in the test set. It is interesting to note that the worst result, both in terms of train and test set, was obtained for a batch size of 16.

Table 5.5: Results for different number of batch size (Encoder-Decoder with OHE structure) Dataset# = 00,000 test#=1000 BiLSTM/LSTM layers=2 BiLSTM/LSTM units=256 Latent dimension=512 Batch normalization momentum=0.9 Optimizer=Adam Noise std=0.1

Batch Size	Last Ep.	Correctly Reconstruct (Train)%	Correctly Reconstruct (Test)%	Valid (Train) %	Valid (Test) %	Train Time (hh:mm:ss)	Total Run Time (hh:mm:ss)
16	20	94.3	86.0	98.3	96.4	01:38:46	03:12:46
32	28	99.0	92.0	99.9	97.5	01:48:40	03:20:16
64	44	99.9	93.7	100.0	98.4	01:26:39	03:01:33
128	38	100.0	92.9	100.0	97.5	00:40:17	02:13:37
256	45	99.9	92.7	100.0	96.6	00:28:44	02:00:16

The proposed structure for the Encoder-Decoder model (see Figure 5.2 on Page 56) includes a Batch Normalization Layer between every other type of layer. This type

of layer requires the user to define the value for the Batch Normalization Momentum (BNM) which is the momentum of the moving average used during inference. Table 5.6 shows the results for four different BNM values: 0.95, 0.9, 0.8 and 0.7. The four models performed fairly similar, indicating that this parameter does not have a critical effect on the overall performance. Nevertheless, this parameter was set to 0.9 from here onwards.

Table 5.6: Results for different values Batch Normalization Momentum (BNM) (Encoder-Decoder with OHE structure). Dataset#=100,000 test#=1000 BiLSTM/LSTM layers=2 BiLSTM/LSTM units=256 Latent dimension=512 Batch size=128 Optimizer=Adam Noise std=0.1

Batch Normalization Momentum	Last Ep.	Correctly Reconstruct (Train)%	Correctly Reconstruct (Test)%	Valid (Train) %	Valid (Test) %	Train Time (hh:mm:ss)	Total Run Time (hh:mm:ss)
0.95	38	99.9	92.6	99.9	97.5	00:40:57	02:15:51
0.9	30	100.0	92.7	100.0	96.3	00:32:51	02:06:20
0.8	27	99.8	92.4	100.0	97.4	00:28:57	02:00:46
0.7	29	99.2	92.2	99.6	96.9	00:29:39	01:48:29

The Latent Dimension defines the dimension of the context vectors. The lower this value, the more the information will have to be compressed and potentially lost. Table 5.7 summarizes the performance of models with different latent dimensions: 64, 128, 256, 512 and 1024. Even though the best model in terms of the percentage of correctly reconstructed molecules was the one with a latent dimension of 1024 (with 93.9%), this parameter was fixed at 256 for two reasons: the performance was similar (93.2%) and a latent dimension of 256, as opposed to 1024, would require less running time when implementing the full framework.

Table 5.7: Results for different latent dimensions (Encoder-Decoder with OHE structure). Dataset#=100,000 test#=1000 BiLSTM/LSTM layers=2 BiLSTM/LSTM units=512 Batch size=128 Batch normalization momentum=0.9 Optimizer=Adam Noise std=0.1

Latent Dimension	Last Ep.	Correctly Reconstruct (Train)%	Correctly Reconstruct (Test)%	Valid (Train) %	Valid (Test) %	Train Time (hh:mm:ss)	Total Run Time (hh:mm:ss)
64	41	100.0	91.5	100.0	96.6	01:08:07	02:41:29
128	35	99.9	91.9	99.9	96.0	00:58:19	02:31:33
256	35	100.0	93.2	100.0	97.4	00:58:40	02:33:03
512	30	100.0	92.7	100.0	96.3	00:32:51	02:06:20
1024	34	100.0	93.9	100.0	97.6	00:57:08	02:30:53

The last layer of the Encoder part of the model is responsible for adding Gaussian Noise to the vector so that the model can become more robust at translating between

SMILES strings and context vectors. Therefore, the effect of the standard deviation of the distribution was studied and the results are shown in Table 5.8. The model with a standard deviation of 0.2 returned the highest performance with 92.7% of correctly reconstructed molecules in the test set.

Table 5.8: Results for different values of Noise Standard Deviation (Encoder-Decoder with OHE structure). Dataset#=100,000 test#=1000 BiLSTM/LSTM layers=2 BiLSTM/LSTM units=512 Latent dimension=256 Batch size=128 Batch normalization momentum=0.9 Optimizer=Adam

Noise Std	Last Ep.	Correctly Reconstruct (Train)%	Correctly Reconstruct (Test)%	Valid (Train) %	Valid (Test) %	Train Time (hh:mm:ss)	Total Run Time (hh:mm:ss)
0.1	36	100.0	91.3	100.0	97.3	01:00:35	02:31:44
0.15	27	99.9	91.2	100.0	96.6	00:46:17	02:17:11
0.2	31	100.0	92.7	100.0	98.1	00:52:02	02:26:19
0.25	30	100.0	90.8	100.0	97.3	00:50:21	02:25:24

As a final experiment, the impact of the number of training data was also evaluated and the results are summarized in Table 5.9. As expected, the higher the number of training data, the higher the performance of the model. With 500,000 SMILES strings it was possible to achieve 100.0% and 99.1% of correctly reconstructed molecules in the train and test sets, respectively.

Table 5.9: Results for different number of SMILES in dataset (Encoder-Decoder with OHE structure). Test#=1000 BiLSTM/LSTM layers=2 BiLSTM/LSTM units=512 Latent dimension=256 Batch Size=128 Batch normalization momentum=0.9 Optimizer=Adam Noise std=0.1

SMILES Num	Last Ep.	Correctly Reconstruct (Train)%	Correctly Reconstruct (Test)%	Valid (Train) %	Valid (Test) %	Train Time (hh:mm:ss)	Total Run Time (hh:mm:ss)
10000	49	25.5	8.5	63.0	53.7	00:08:29	01:44:47
100000	34	99.8	83.4	100.0	95.2	00:56:39	02:29:42
200000	44	100.0	96.5	100.0	98.7	02:25:00	03:59:04
500000	33	100.0	99.1	100.0	99.8	07:01:27	09:10:33

Encoder-Decoder with Embedding Structure

As OHE is known to be a sparse and high-dimensional type of encoding [110], the use of an embedding layer as the input layer to the Encoder is studied in this section. An Embedding layer is expected to be a more computationally efficient approach and to retain information about the relations between atoms that would not be present when employing OHE.

Therefore, this section summarizes the results for the model that contains an embedding layer as input following a strategy similar to the previous section. Table 5.10 shows the set of parameters that are studied in this part.

Table 5.10: Search space for finding the optimal set of parameters of the proposed Encoder-Decoder model (with Embedding structure).

Parameters	Search Space
Number of Layers	[1,2,3]
Number of BiLSTM Units	[256,512,1024]
Batch Size	[16,32,64,128,256]
Embedding Dimension	[64,128,256,512]
Latent Dimension	[64,128,256,512,1024]
Training Data	[50,000, 100,000, 500,000, 1,000,000]

Table 5.11 shows the results obtained for a different number of encoder BiLSTM layers (which is equal to the number of decoder LSTM layers): 1, 2 and 3. From it we can conclude that, as with the previous section, a model with two layers returns the highest percentage of correctly reconstructed molecules with 100.0% and 94.3% for the train and test sets, respectively. For this experiment the number of training data was set to 100,000, the number of BiLSTM/LSTM units and latent dimension were both set to 512 and the embedding dimension to 256. The models were trained with a batch size of 128 and using the Adam Optimizer. The BNM was set to 0.9 and the noise standard deviation to 0.1.

Table 5.11: Results for different number of encoder BiLSTM layers and decoder LSTM layers (Encoder-Decoder with Embedding structure). Dataset#=100,000 test#=1000 BiLSTM/LSTM units=512 Embedding dimension=256 Latent dimension=512 Batch size=128 Batch normalization momentum=0.9 Optimizer=Adam Noise std=0.1

Encoder BiLSTM Layers #	Decoder LSTM Layers #	Last Ep.	Correctly Reconstruct (Train)%	Correctly Reconstruct (Test)%	Valid (Train) %	Valid (Test) %	Train Time (hh:mm:ss)	Total Run Time (hh:mm:ss)
1	1	23	99.9	90.5	100.0	96.9	00:21:04	01:51:14
2	2	33	100.0	94.3	100.0	98.3	01:00:56	02:33:26
3	3	29	100.0	91.7	100.0	98.2	01:22:05	02:58:28

The next step was to evaluate the number of BiLSTM/LSTM units. A summary of the results is shown in Table 5.12 where 256, 512 and 1024 units were considered. The model with 512 BiLSTM/LSTM units clearly outperformed the competing ones by reaching 100.0% and 94.3% of correctly reconstructed molecules from the train and test sets, respectively.

Table 5.12: Results for different number of BiLSTM/LSTM units (Encoder-Decoder with Embedding structure). Dataset#=100,000 test#=1000 BiLSTM/LSTM layers=2 Embedding dimension=256 Latent dimension=512 Batch size=128 Batch normalization momentum=0.9 Optimizer=Adam Noise std=0.1

BiLSTM /LSTM units	Last Ep.	Correctly Reconstruct (Train)%	Correctly Reconstruct (Test)%	Valid (Train) %	Valid (Test) %	Train Time (hh:mm:ss)	Total Run Time (hh:mm:ss)
256	27	99.9	92.8	100.0	97.2	00:30:15	02:04:32
512	33	100.0	94.3	100.0	98.3	01:00:56	02:33:26
1024	32	99.7	83.0	99.8	95.3	02:28:34	04:05:02

Table 5.13 shows the results obtained for different batch sizes: 16, 32, 64, 128 and 256. The percentages of correctly reconstructed train and test molecules were similar for batch sizes of 64 and 128. A batch size of 64 was used from here onwards.

Table 5.13: Results for different number of batch size (Encoder-Decoder with Embedding structure). Dataset#=100,000 test#=1000 BiLSTM/LSTM layers=2 BiLSTM/LSTM units=512 Embedding dimension=256 Latent dimension=512 Batch normalization momentum=0.9 Optimizer=Adam Noise std=0.1

Batch Size	Last Ep.	Correctly Reconstruct (Train)%	Correctly Reconstruct (Test)%	Valid (Train) %	Valid (Test) %	Train Time (hh:mm:ss)	Total Run Time (hh:mm:ss)
16	36	35.3	70.0	98.5	93.5	04:05:51	05:37:50
32	12	18.7	17.0	81.5	78.9	00:52:41	02:24:09
64	35	100.0	92.2	100.0	98.5	01:26:57	03:01:01
128	31	99.7	92.2	100.0	97.6	00:57:14	02:30:01
256	29	100.0	91.0	100.0	96.3	00:47:11	02:19:49

Regarding the choice of the embedding dimension, the results are presented in table 5.14. There was no outstanding model which prompted the experiment showed in Table 5.15 where an exhaustive comparison between the values chosen for the embedding and latent dimensions is performed. From Table 5.15, the model with an embedding dimension of 256 and a latent dimension of 256 was chosen due to its higher performance and better generalization capability by correctly reconstructing 95.2% of the molecules in the test set.

Lastly, an evaluation regarding the size of the training data was performed. Once again, and as shown in Table 5.16, using a larger dataset results in an higher performance of the model.

Table 5.14: Results for different number of embedding dimension in encoder (Encoder-Decoder with Embedding structure). Dataset#=100,000 test#=1000 BiLSTM/LSTM layers=2 BiLSTM/LSTM units=512 Latent dimension=512 Batch Size=128 Batch normalization momentum=0.9 Optimizer=Adam Noise std=0.1

Embedding Dimension	Last Ep.	Correctly Reconstruct (Train)%	Correctly Reconstruct (Test)%	Valid (Train) %	Valid (Test) %	Train Time (hh:mm:ss)	Total Run Time (hh:mm:ss)
64	39	100.0	94.5	100.0	97.9	01:11:27	02:44:08
128	31	100.0	94.2	100.0	98.1	00:57:22	02:22:57
256	31	99.7	92.2	100.0	97.6	00:57:14	02:30:01
512	30	99.9	94.6	99.9	98.2	00:55:26	02:28:16

Table 5.15: Results for comparison between embedding dimension and latent dimension (Encoder-Decoder with Embedding structure). Dataset#=100,000 test#=1000 BiLSTM/LSTM layers=2 BiLSTM/LSTM units=512 Batch size=128 Batch normalization momentum=0.9 Optimizer=Adam Noise std=0.1

Embedding Dimension	Latent Dimension	Last Ep.	Correctly Reconstruct (Train)%	Correctly Reconstruct (Test)%	Valid (Train) %	Valid (Test) %	Train Time (hh:mm:ss)	Total Run Time (hh:mm:ss)
64	64	34	100.0	92.1	100.0	97.1	00:27:48	01:02:32
64	128	32	100.0	92.3	100.0	97.3	00:26:15	01:01:07
64	256	33	100.0	93.7	100.0	98.0	00:27:04	01:01:59
64	512	25	99.9	91.7	100.0	97.1	00:20:44	00:55:37
64	1024	27	99.7	92.9	99.9	97.3	00:22:23	00:57:13
128	64	50	100.0	93.2	100.0	97.4	00:40:32	01:15:26
128	128	46	100.0	92.0	100.0	96.7	00:37:25	01:12:13
128	256	25	99.9	90.7	100.0	98.0	00:20:44	00:55:38
128	512	33	100.0	92.6	100.0	96.5	00:27:08	01:02:04
128	1024	33	100.0	92.1	100.0	97.4	00:27:06	01:01:55
256	64	35	100.0	91.2	100.0	97.7	00:28:39	01:03:32
256	128	35	100.0	93.2	100.0	97.6	00:28:31	01:03:21
256	256	39	99.9	95.2	100.0	97.8	00:31:54	01:06:47
256	512	37	100.0	93.3	100.0	98.3	00:30:26	01:05:14
256	1024	26	99.9	94.1	100.0	97.9	00:21:24	00:56:20
512	64	38	99.9	91.3	99.9	96.6	00:31:01	01:05:57
512	128	44	100.0	92.7	100.0	97.5	00:35:48	01:10:40
512	256	36	100.0	90.7	100.0	96.6	00:29:28	01:04:25
512	512	27	100.0	92.2	100.0	96.7	00:22:21	00:57:28
512	1024	31	100.0	93.5	100.0	97.6	00:25:33	01:00:35
1024	64	34	100.0	92.6	100.0	96.8	00:27:56	01:03:00
1024	128	29	99.9	93.8	100.0	98.0	00:24:00	00:59:00
1024	256	30	99.8	92.1	99.8	97.3	00:24:41	00:59:40
1024	512	31	100.0	93.4	100.0	98.2	00:25:30	01:00:34
1024	1024	32	100.0	93.6	100.0	97.2	00:26:16	01:01:21

Table 5.16: Results for different number of SMILES in dataset (Encoder-Decoder with Embedding structure). Test#=1000 BiLSTM/LSTM layers=2 BiLSTM/LSTM units=512 Embedding dimension=256 Latent dimension=512 Batch Size=128 Batch normalization momentum=0.9 Optimizer=Adam Noise std=0.1

# SMILES	Last Ep.	Correctly Reconstruct (Train)%	Correctly Reconstruct (Test)%	Valid (Train) %	Valid (Test) %	Train Time (hh:mm:ss)	Total Run Time (hh:mm:ss)
50000	38	100.0	84.5	100.0	93.8	00:15:22	00:49:55
100000	37	99.9	94.5	99.9	98.0	00:29:44	01:04:28
500000	21	99.9	98.8	100.0	99.8	01:26:54	02:01:49

The Proposed Encoder-Decoder Model

Taking in consideration the previous results, the chosen encoder-decoder model contains two bidirectional LSTM layers with 512 units each, 256 for each direction. Both the embedding dimension and latent dimension were set to 256. All the batch normalization layers had a batch normalization momentum of 0.9. The Gaussian noise layer added noise with a standard deviation of 0.1. The model was trained using the Adam optimizer with a learning rate of 0.01, a batch size of 128, and the total number of epochs was set to 100, but only the best models regarding the validation loss were kept (10% of the training data was set as validation data).

This model was trained using the “composed_dataset_1” and “composed_dataset_2” (see Section 5.3.1 on Page 59 for more details). The results from training with these datasets which are more complex due to including a wider range of molecules and also stereo-chemical information are summarized in Table 5.17.

In both cases, the model reaches high percentages of correctly reconstructed molecules for the train and test sets. As expected, the model trained with 500,000 molecules learned to generalize better (with 99.2% and 99.0% for the train and test set, respectively) when compared to the one trained on only 100,000. It should be noted that molecules that are correctly reconstructed are automatically valid. Validity (evaluated by RDkit [35]) is constantly higher than the percentage of correct reconstruction, which means that some molecules are reconstructed into valid molecules but not the intended ones.

5.3.3 WGAN-GP

The composed_dataset_2 with 500,000 SMILES strings was passed through the Encoder-Decoder model presented in the previous section in order to separate the

Dataset	#Training Data	%Correct R. (train set)	%Correct R. (test set)	%Validity (train set)	%Validity (test set)
composed_dataset_1	100,000	98.6	96.8	99.7	99.1
composed_dataset_2	500,000	99.2	99.0	99.9	99.8

Table 5.17: Performance of the Encoder-Decoder model for 100,000 and 500,000 training data.

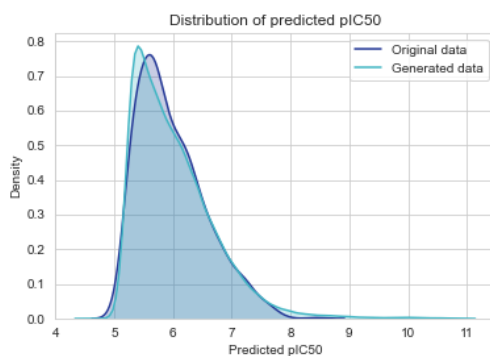
correctly reconstructed molecules from the ones that the model could not properly reconstruct. From the set of correctly reconstructed molecules, 100,000 were used as training data for the WGAN-GP. This is an important step so that the WGAN-GP would not train on vectors that the Encoder-Decoder would not be able to reconstruct.

The implemented WGAN-GP comprises a critic made up of three dense layers with 256 units and with Leaky-Relu as the activation function (with $\alpha = 0.3$), except for the final layer, which does not include an activation function. A vector of 64 dimensions is drawn from the uniform distribution and then passed through the Generator that contains five dense layers with 256 dimensions each, except for the first layer, which contains 128 dimensions; between these layers, Leaky-Relu activation function ($\alpha = 0.3$) and batch normalization (momentum of 0.9) layers are applied. Both the Generator and the Critic are trained using the Adam optimizer with a learning rate of 0.0001 and include dropout layers with a value of 0.2. The WGAN-GP was trained on 100,000 molecules and for 10,000 epochs. The results of sampling 1,000 valid molecules using the full framework are presented in table 5.18 and Figure 5.4 from which it is possible to conclude that the GAN effectively learned the training data distribution as it generates data that follows that same distribution regarding the values of predicted $pIC50$. It is worth noting that the validity of the generated data was only of 30.2%. This low value is attributed to the fact that the WGAN-GP is learning to mimic a distribution of continuous vectors that are then converted into a series of discrete tokens that need to abide to certain rules to be considered valid SMILES strings, hence the difficulty in generating valid molecules.

To further compare the generated data to the original training data in terms of general drug-like properties, Figure 5.5 shows the relationship between the QED and SA score (Figure 5.5a) and the relationship between $\log P$ and MW (Figure 5.5b). By interpreting them, it is possible to conclude that the original data and generated data are clearly overlapping which means that the GAN successfully learned the

Table 5.18: Comparison of the Original Data with the Generated Data in terms of predicted $pIC50$ and performance of the WGAN-GP model.

	Original Data	Generated Data
$pIC50$ Maximum	9.974	8.454
$pIC50$ Mean	6.003	5.984
$pIC50$ Minimum	4.548	5.083
$pIC50$ Standard Deviation	0.684	0.577
External Diversity	-	0.890
Internal Diversity	-	0.887
% Unique	-	100.0
% Valid	-	30.2

**Figure 5.4:** Comparison of the predicted $pIC50$ distributions for the original data and generated data.

training data distribution in terms of several of its properties. It should be noted that, to make the plots more perceptive, instead of resorting to the 100,000 molecules in the training data, only 5,000 were used but care was taken to make sure that the distribution of this smaller set in terms of predicted $pIC50$ was kept the same.

5.3.4 Performance of the Predictor

The optimal architecture for the KOR predictor consisted of an embedding layer with 128 units, followed by two LSTM layers and a dense layer (128 units each) and a final layer with a single unit. The model was trained using the Adam optimizer (with $\beta_1 = 0.9$ and $\beta_2 = 0.999$) with a learning rate of 0.0001 and a batch size of 16. The maximum number of epochs was set to 100 but, as explained in Section 5.2.4 on Page 57, early stopping was employed and all the models stopped training before reaching the 100th epoch. Figure 5.6 shows the scatter plot that results from applying the predictor to the hold-out test set where it can be observed that the model performs constantly well for the complete range of values.

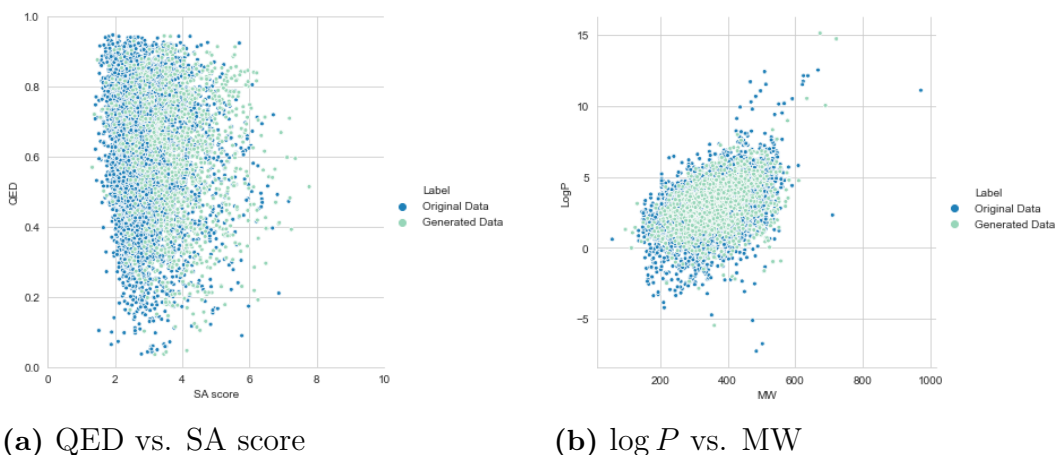


Figure 5.5: Evaluation of properties (QED, SA score, $\log P$ and MW) for the original training data and generated data.

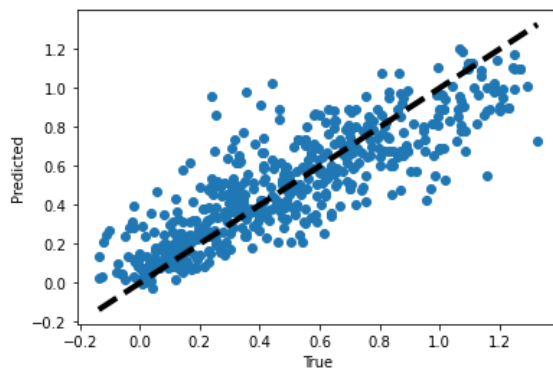


Figure 5.6: Predicted pIC_{50} versus true pIC_{50} and regression line for the test set.

5.3.5 Optimization through Transfer Learning

As a first approach to optimize the model towards the space of the desired properties, Transfer Learning was employed.

Two experiments were devised: one that aims to maximize the affinity for KOR and another that aims to minimize it. The affinity of a molecule for KOR is measured in terms of pIC_{50} which is obtained by $-\log(IC_{50})$ where IC_{50} is the half-maximal inhibitory concentration. Therefore, the higher the pIC_{50} value, the higher the affinity for the receptor and the more powerful the inhibition will be.

Maximizing KOR affinity

From the KOR dataset, a subset with only molecules with a pIC_{50} higher than 7 was created. This dataset was used to bias the WGAN-GP model with Transfer Learning

towards the space of higher KOR affinities by retraining the trained model for 4,000 additional epochs. After employing TL, 1,000 valid SMILES strings were sampled. Figure 5.7 shows the comparison of the KOR pIC_{50} distribution produced by the unbiased and biased models where it is clear that the employed strategy resulted in a shift towards the desired region of the chemical space which implies that the model is generating molecules with an higher probability of inhibiting the KOR receptor. Table 5.19 presents a comparison between these two models regarding distribution metrics, diversity, uniqueness and validity. It should be highlighted that the mean of the pIC_{50} distribution increased over 1.3 points for the biased model and the maximum also increased from 8.896 to 10.444. It is also important to note that the uniqueness and the internal and external diversity remained high for the biased model.

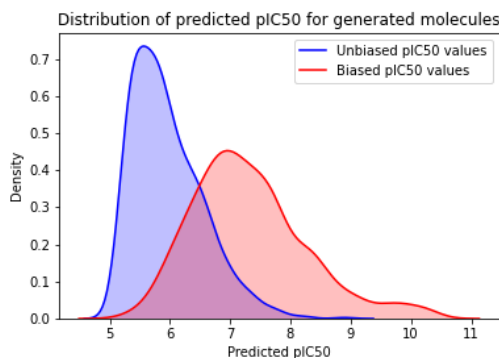


Figure 5.7: Distribution of the predicted pIC_{50} values for the biased and unbiased model (maximization through TL).

Table 5.19: Comparison of the performance of the biased and unbiased models (maximization through TL).

	Unbiased	Biased
pIC_{50} Maximum	8.896	10.444
pIC_{50} Mean	5.961	7.317
pIC_{50} Minimum	5.055	5.178
External Diversity	0.891	0.896
Internal Diversity	0.891	0.868
% Unique	100.0	98.2
% Valid	34.5	44.5

Minimizing KOR affinity

To study the versatility of the framework, an experiment with the goal of minimizing KOR affinity was also devised. From the KOR dataset, a subset with only molecules with a pIC_{50} lower than 7 was created and used to train the model for an additional

4,000 epochs; 1,000 valid molecules were sampled, and their $pIC50$ were predicted. As observed in Table 5.20, the biased model retains high levels of uniqueness and diversity. Figure 5.8 shows the $pIC50$ distribution plots for both models. From it we can conclude that 4,000 epochs were not enough to provide an effective distribution shift towards lower values of $pIC50$ since the molecules from the unbiased model already had fairly low $pIC50$ values when compared to the subset used for the TL procedure.

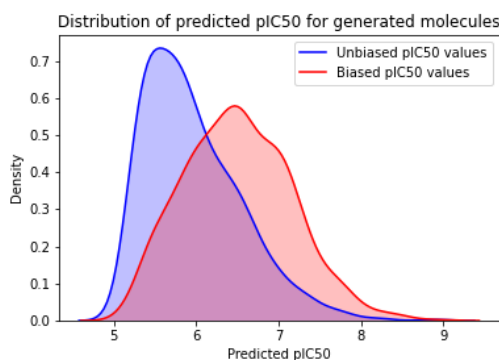


Figure 5.8: Distribution of the predicted $pIC50$ values for the biased and unbiased model (minimization through TL).

Table 5.20: Comparison of the performance of the biased and unbiased models (minimization through TL).

	Unbiased	Biased
$pIC50$ Maximum	8.896	8.912
$pIC50$ Mean	5.961	6.506
$pIC50$ Minimum	5.055	5.122
External Diversity	0.891	0.890
Internal Diversity	0.891	0.878
% Unique	100.0	99.3
% Valid	34.5	40.0

5.3.6 Optimization through FeedbackGAN

The previous approach biases the model towards a space where the desired property belongs to the range of values presented in the dataset used to retrain the model. This does not result in an effective minimization or maximization of the property of interest but in a moderate shift of the overall distribution. In order to have a finer control of the maximization/minimization of the desired property, a feedbackGAN-based framework was employed. Since this strategy constantly updates the training data by replacing the worst scoring molecules by the best, it gradually shifts the

distribution towards the desired property. At the end of each epoch, 200 valid molecules are sampled and the 20 best replace the worst molecules in the “real data” training set. So that the changes in the dataset may be significant to effectively bias the distribution that is being learned, only 5,000 molecules were used as “real data” for the optimization process. These 5,000 molecules were sampled from the dataset used to train the WGAN-GP taking in consideration the distribution of the original dataset in terms of $pIC50$.

Maximizing KOR affinity

The unbiased WGAN-GP model (see Section 5.3.3 on Page 68) was optimised for 500 epochs considering that the best scoring molecules were the ones with the highest $pIC50$. 1,000 valid molecules were sampled in intervals of 50 epochs and evaluated in terms of distribution of predicted $pIC50$, validity, uniqueness and diversity. These results are presented in table 5.21. Figure 5.9 shows the distribution of predicted $pIC50$ values in terms of a probability density for intervals of 100 epochs of optimization where it can be observed that the implemented strategy results in a successive shift of the overall distribution towards higher values of predicted $pIC50$. The same conclusion can be drawn from Table 5.21 where it is clear that the mean of the distributions is constantly increasing from 5.984 for the unbiased model to 7.283 for the model optimised for 500 epochs. It is important to note that the oscillations in the minimum and maximum values of the distributions are expected as sampling is a stochastic process.

Regarding the internal diversity of the generated molecules, it remains fairly constant, at around 0.88. Also, the percentage of unique molecules in the sampled set is almost always 100.0%. This implies that the model is not getting stuck on a local minimum and collapsing into a restricted set of outputs.

Interestingly, the external diversity increases from 0.890 to 0.938. This means that the model is not only generating diverse molecules but also constantly finding new valid molecules as it moves towards a new area of the chemical space.

The increase in the percentage of valid molecules as the optimization process progresses is explained by the fact that, as this process starts, the training data is reduced from 100,000 to 5,000 molecules, which means that the model can better focus on learning between what is “real” and “fake” data. While the low validity is clearly a downside of this method, it is counterbalanced by the high diversity and

uniqueness.

Table 5.21: Comparison of $pIC50$ distribution measures throughout the optimization process (maximization of KOR affinity).

	Unbiased	50 epoch	100 epoch	150 epoch	200 epoch	250 epoch	300 epoch	350 epoch	400 epoch	450 epoch	500 epoch
Max $pIC50$	8.454	8.341	8.852	9.340	9.354	8.911	9.149	9.000	8.939	9.241	9.179
Mean $pIC50$	5.984	6.501	6.719	6.791	6.851	7.015	7.074	7.168	7.204	7.273	7.383
Min $pIC50$	5.083	5.198	5.205	5.380	5.181	5.496	5.271	5.340	5.522	5.380	5.380
Int Div	0.887	0.890	0.890	0.887	0.891	0.886	0.885	0.885	0.883	0.878	0.877
Ext Div	0.890	0.900	0.909	0.908	0.915	0.923	0.922	0.926	0.930	0.937	0.938
% Unique	100.0	100.0	100.0	100.0	100.0	100.0	99.9	100.0	100.0	99.8	100.0
% Valid	30.2	15.1	25.7	32.0	38.7	44.0	46.9	51.6	52.1	60.4	62.3

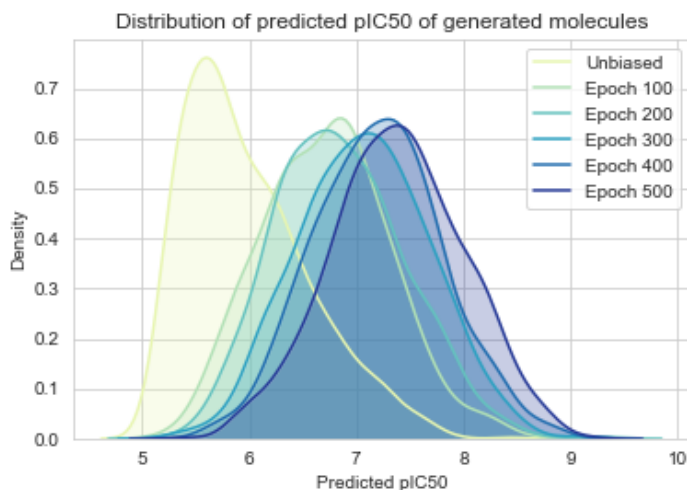


Figure 5.9: Distribution of the predicted $pIC50$ values for the unbiased model and the biased model at every 100 epochs (maximization of KOR affinity).

Although, as it has been shown, the proposed model was successful in maximizing the overall KOR affinity of the generated molecules, for them to be considered as potential drugs it is also important to evaluate metrics such as QED, $\log P$, MW and SA score (see Section 2.5 on Page 16 for more details). Figure 5.10 shows the scatter plots and distributions of these properties of 1,000 molecules generated after 500 epochs of optimization; the red square represents the region of drug-like properties. From Figure 5.10a it can be concluded that most of the molecules are outside the desired region (high QED and low SA score), with most of them exhibiting a SA score higher than 5 which implies that they are difficult to synthesize. From Figure 5.10b it can be observed that a significant amount of the generated molecules have good values for both $\log P$ and MW, even though there is a general tendency towards higher molecular weights.

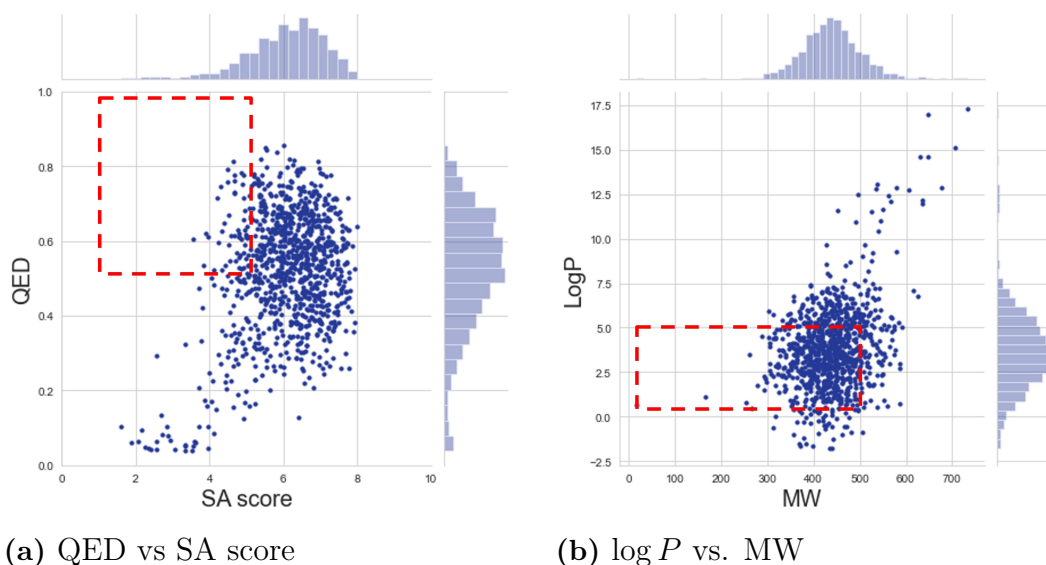


Figure 5.10: Evaluation of properties (QED, SA score, $\log P$ and MW) for the biased model (500 epochs). The red square represents the region of desired properties.

Figure 5.11 shows the highest scoring molecules together with their properties for the sets generated with the unbiased model (at the top) and the optimized models at epochs 100, 200, 300, 400 and 500. It can be observed that as the $pIC50$ increases, there seems to be a tendency to generate more complex molecules which is in agreement with the generally high SA score values in Figure 5.10a.

Minimizing KOR affinity

Once again, to evaluate the versatility of the framework and also to study possible off-target effects, the unbiased WGAN-GP model was optimised for 500 epochs with the goal of shifting the probability distribution towards the minimization of the predicted $pIC50$. Figure 5.12 shows the distribution of predicted $pIC50$ values in terms of a probability density for intervals of 100 epochs while Table 5.22 presents more detailed information regarding the distributions of predicted $pIC50$, diversity, uniqueness and validity of the sampled molecules for every 50 epochs. From this information it is possible to conclude that there was a first shift towards higher predicted $pIC50$ values at the beginning of the optimization process before it started to move towards the goal of minimization. This can be attributed to the fact that, as previously explained, before starting the optimization process we reduced the training dataset from 100,000 to 5,000 compounds effectively breaking the flow of training. We also believe that this is the reason for the decrease in validity from

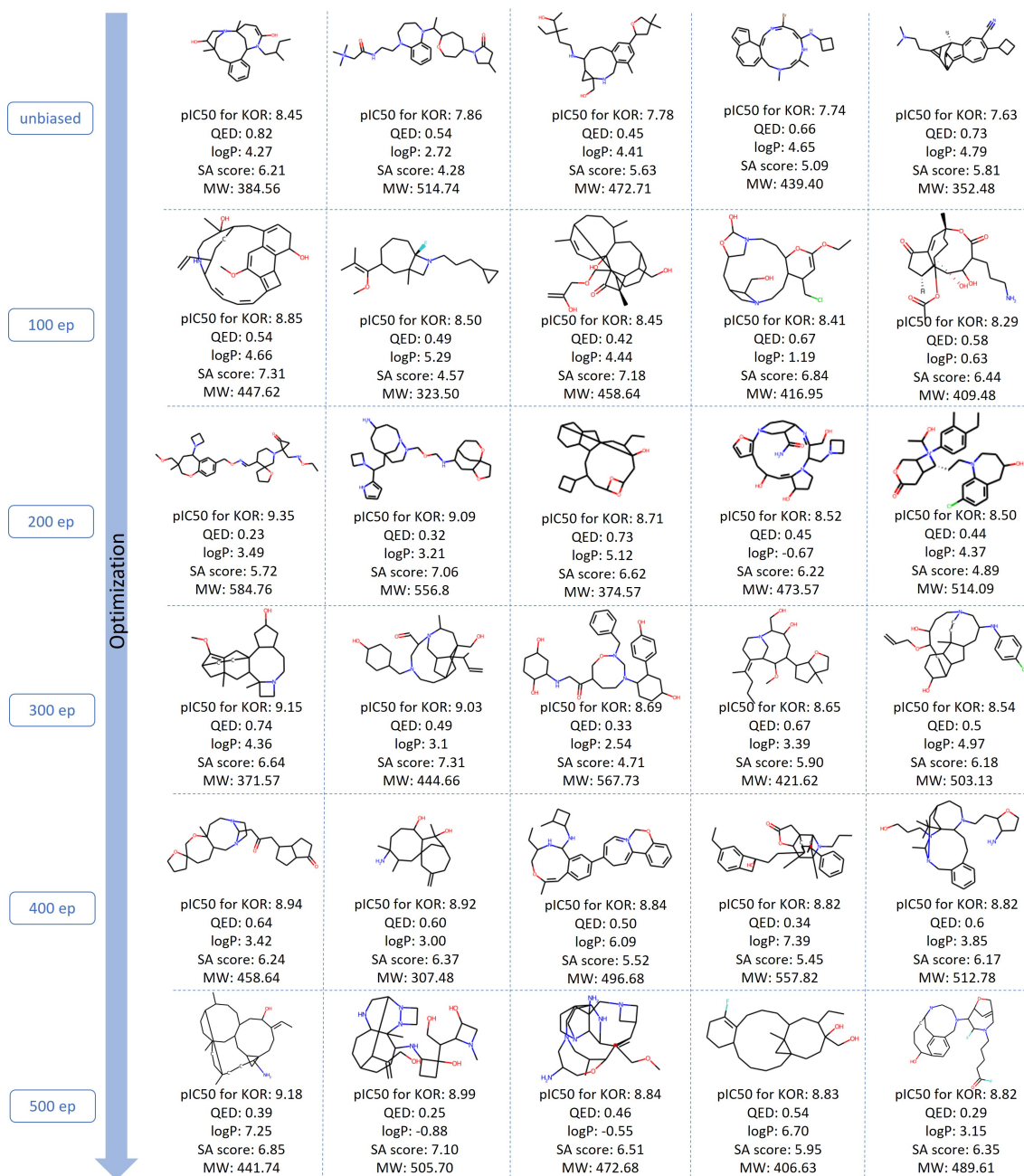


Figure 5.11: Best scoring examples in terms of predicted pIC_{50} generated by the proposed framework for every 100th epoch of optimization (maximization).

30.2% (unbiased) to 15.2% (epoch 50) which is then followed by a constant increase as the model adapts to the new training data.

The internal and external diversity remain high throughout the optimization reaching the highest values for the 500th epoch with 0.904 and 0.906, respectively. This

implies that this optimization framework is capable of generating new valid molecules while maintaining high values of uniqueness which is constantly close to 100.0%.

The overall shift between the unbiased and the optimized model (500 epochs) is significantly less than the one observed for the maximization experiment. This can be largely attributed to the fact that the unbiased distribution is positively skewed which makes it less likely to sample a high number of molecules with low $pIC50$ therefore slowing down the optimization process. In this sense, to achieve better optimization results, one would either have to train the model for a higher number of epochs or sample more molecules at each optimization epoch.

Table 5.22: Comparison of $pIC50$ distribution measures throughout the optimization process (minimization of KOR affinity).

	Unbiased	50 epoch	100 epoch	150 epoch	200 epoch	250 epoch	300 epoch	350 epoch	400 epoch	450 epoch	500 epoch
Max $pIC50$	8.454	8.611	8.201	8.110	8.202	7.990	8.025	8.110	7.591	7.603	7.837
Mean $pIC50$	5.984	6.485	6.265	6.177	6.110	6.013	5.925	5.894	5.830	5.814	5.737
Min $pIC50$	5.083	5.133	5.141	5.140	5.091	5.052	4.971	4.995	5.058	5.030	5.001
Int Div	0.887	0.884	0.887	0.886	0.897	0.893	0.896	0.900	0.903	0.897	0.904
Ext Div	0.890	0.894	0.896	0.893	0.899	0.897	0.900	0.901	0.905	0.901	0.906
% Unique	100.0	100.0	100.0	100.0	99.9	99.9	99.7	99.9	99.7	99.7	99.2
% Valid	30.2	15.2	25.5	27.8	32.1	35.5	38.2	39.1	38.1	39.3	41.12

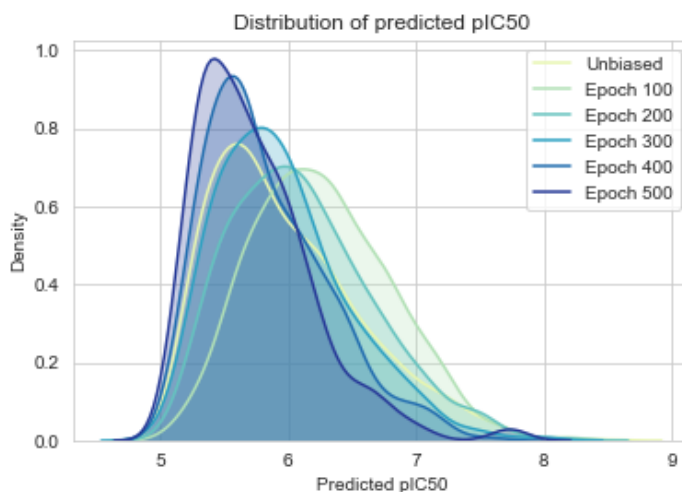


Figure 5.12: Distribution of the predicted $pIC50$ values for the unbiased model and the biased model at every 100 epochs (minimization of KOR affinity).

Figure 5.13 shows the scatter plots and distributions of 1,000 molecules sampled from the optimised model (500 epochs) regarding evaluation metrics such as the QED, SA score, MW and $\log P$ with the red squares representing the region of

desired drug-like properties. When comparing to the maximization experiment, we can conclude that the molecules obtained from the minimization experiment are typically less complex, having lower SA scores, MW and $\log P$ values. The same can be deduced from observing Figure 5.14 which shows the evolution of the samples compounds with the lowest predicted $pIC50$ throughout the optimization process.

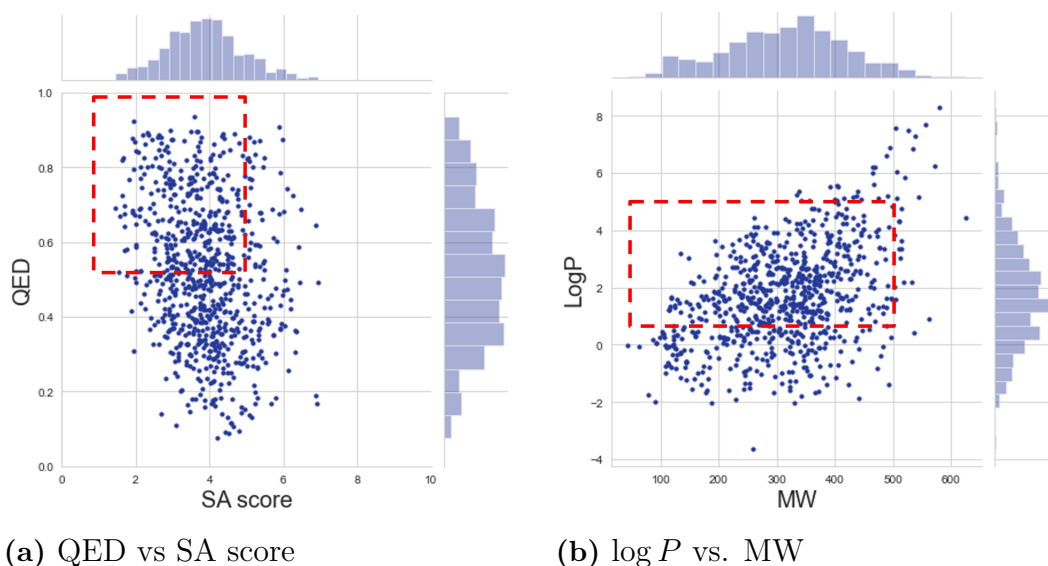


Figure 5.13: Evaluation of properties (QED, SA score, $\log P$ and MW) for the bi-ased model (500 epochs). The red square represents the region of desired properties.

5.4 Conclusions

The devised framework effectively integrates the phases of generation and evaluation of new molecules. We can consider three stages for training the framework, with the first two being generic and the final one being goal-specific. Firstly, an Encoder-Decoder model was trained to map SMILES strings to a latent space with fixed-length vectors. The trained Encoder-Decoder model was able to successfully reconstruct 99.0% of the hold-out test set. Secondly, resorting to this newfound representation as the training data, the WGAN-GP model was trained to learn and mimic its distribution. While the trained WGAN-GP model generates a low percentage of valid SMILES strings, they are constantly diverse and unique which implies that the model did not suffer from mode collapse. Also, by evaluating the distribution of predicted $pIC50$ and scatter plots for QED, SA score, MW and $\log P$, we concluded that the generated distribution follows the original distribution in terms of all of these properties; therefore, effectively learning the underlying characteristics

5. GAN-based Framework

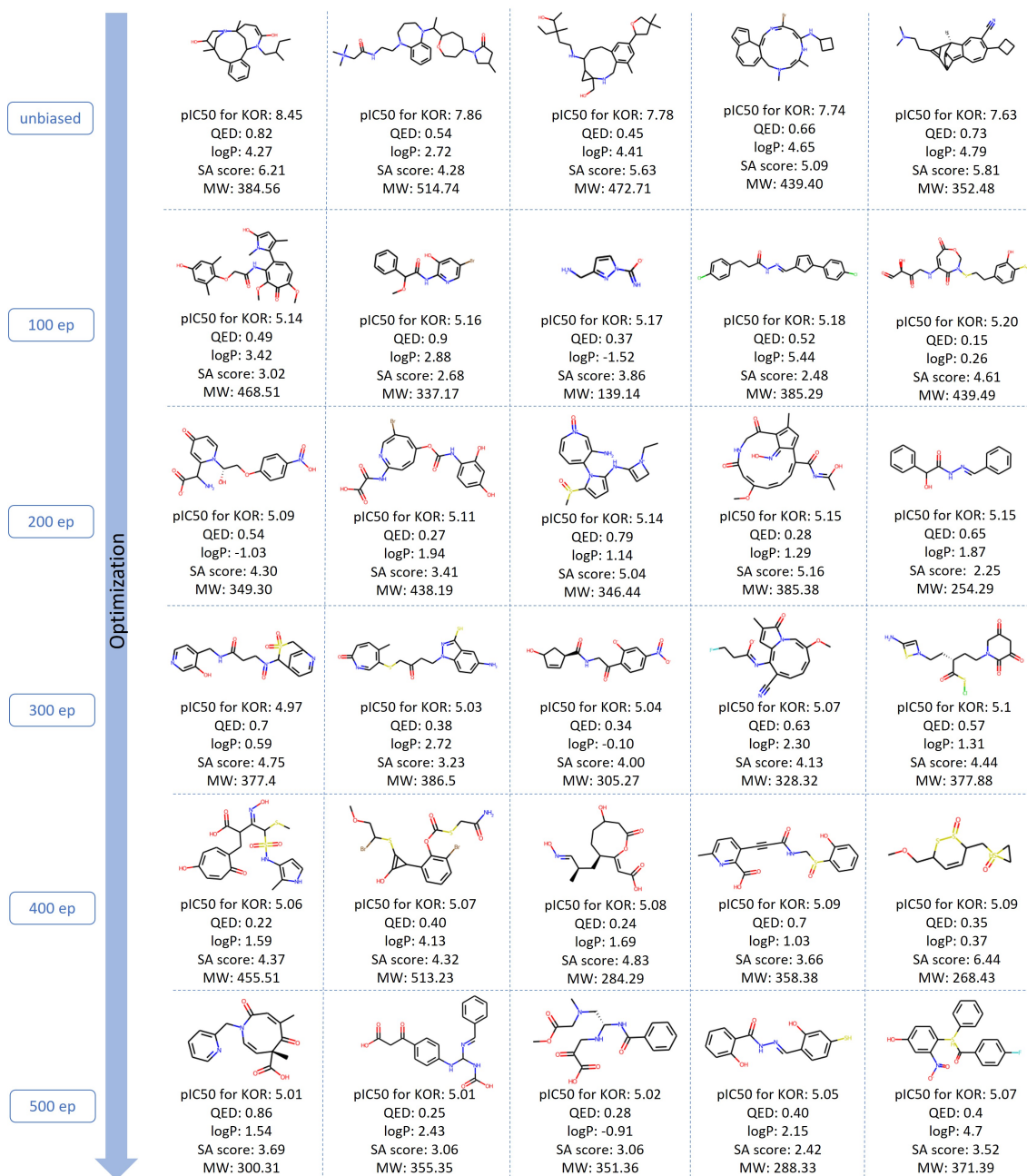


Figure 5.14: Best scoring examples in terms of predicted pIC_{50} generated by the proposed framework for every 100th epoch of optimization (minimization).

of the drug-like molecules that comprise the training data. As the Encoder-Decoder and the WGAN-GP models are generic, they only need to be trained once.

For the optimization step, two approaches were considered: TL and feedbackGAN. While TL obtained the highest predicted pIC_{50} value with 10.44 and a clear shift

of the distribution shape was observed for the maximization experiment, the same didn't occur for the minimization experiment. Overall, we concluded that the feedbackGAN approach was more successful as the shift in the generated distribution was more prominent for both of the experiments while maintaining high levels of diversity and uniqueness. Also, as the optimization process developed, there was an increase in the validity of the generated molecules. Lastly, it is interesting to note that the maximization experiment resulted in more complex molecules while the minimization experiment typically produced less complex compounds.

The overall framework has the advantage that only the optimization step needs to be adapted to different problems and goals, requiring for that a scoring metric function which in this case-study was the KOR Predictor.

Technical notes: The models implemented in this chapter were coded in Python 3.9.4 using Tensorflow 2.5. They were trained on a computer with AMD Ryzen 9 3900x 12-core processor, 64 GB RAM and GPU Nvidia RTX 3080 using CUDA 10.1 and Ubuntu 20.04.3 LTS. The chemistry library used throughout is RDKit 2020_09_2 [35].

Conclusions and Future work

6.1 Conclusions

In this work, a comprehensive study on RNN architectures and its most common hyper-parameters was performed, including a comparison between different types of encoding (OHE and embedding) and datasets (with and without stereo-chemical information). We concluded that LSTM cells clearly outperform the competing types of RNN and drew several comparisons that may serve as guidelines for future experiments. The strategy applied gave rise to a model with as high as 98.7% of valid SMILES with high diversity when not considering stereo-chemical information. When applying it to the Biogenic dataset, we obtained 94.7% of validity and a diversity of 0.90, proving that this is a viable way of generating new compounds, even when considering stereo-chemical information.

A GAN-based framework that aims at generating and optimizing new molecules was also developed. As GANs cannot be straightforwardly applied to discrete data, which is the case of SMILES strings, an Encoder-Decoder model was implemented in order to obtain a new and continuous representation of the molecules. This model reached 98.8% of correctly reconstructed SMILES strings on an hold-out test set and 99.9% on the train set, which shows an improvement when compared to similar approaches presented in the literature. By passing the training data, that comprises SMILES strings, through the Encoder, an equivalent dataset is obtained that, instead of SMILES, contains fixed length continuous vectors. It's this new-found dataset that is then used to train the GAN, more specifically a WGAN-GP. This type of model is able to implicitly learn the distribution of the training data and generate new points following that same distribution by pairing up two competing neural networks: a Generator and Discriminator. Once the WGAN-GP is trained, the Generator network is used to generate new vectors that are then

converted into SMILES strings by the Decoder network. We showed that the trained WGAN-GP was able to replicate the distribution of the training data in terms of the predicted binding affinity for the KOR while still generating molecules with high levels of diversity (0.890 for the external diversity and 0.887 for the internal diversity). Even though the low percentage of valid generated molecules (30.2%) and the time required to train the models is clearly a drawback of this method, the diversity of the generated compounds and the percentage of their uniqueness (100%) makes up for it.

Once the WGAN-GP was trained on the dataset created by the Encoder, an optimization strategy was employed that consisted on continuing the training of the model and generating new molecules at every epoch. These new molecules were then evaluated according to their binding affinity for the KOR and the best scoring generated molecules replaced the worst scoring entries in the training data resorting to a feedback loop. We proved that this strategy successfully resulted in a shift of the generated distribution with its mean moving from 5.984 for the unbiased model to 7.383 when aiming to maximize the predicted pIC_{50} of the generated molecules. Along with this, there was also an increase in the validity of the generated compounds from 30.2% to 62.3% with the internal diversity oscillating around 0.88 which implies that the model did not suffer from mode collapse. Interestingly, the external diversity increased as the optimization process proceeded meaning that the framework was indeed able to generate novel compounds with binding affinities as high as 9.18. In this sense, the devised framework was effectively maximized though it should be noted that this was accompanied by an increase in the complexity of the generated compounds proved by the synthetic accessibility scores of the generated molecules. Moreover, we also showed that this optimization framework can be successfully applied to a different goal such as the minimization of the binding affinity.

In summary, this work contributed with an improved RNN generator and with a novel GAN-based framework that integrates drug generation and evaluation and is therefore able to successfully optimize the desired property. Moreover, as stereochemical information, which is imperative when working with potential drug compounds, was considered in both cases, we showed that, even though most works overlook it, it can be successfully included in the models without significantly reducing their performance.

6.2 Future Work and Open Issues

One of the main limitations of the proposed GAN-based framework lies in the fact that we are only optimizing the model towards a single property (in this case, the binding affinity for the KOR) whereas it has been explained that, for a molecule to be considered a candidate lead, it must not only bind to the chosen target but also exhibit a set of desired drug-like properties. In light of this, it would be interesting to create and evaluate the viability of a Predictor that would produce a scoring that takes in consideration several other metrics such as the SA score, QED, $\log P$ and/or MW. By optimising the model in such a way, we would likely be able to generate molecules with high binding affinities but also strong drug-like properties and easily synthesizable.

It would also be important to compare the devised framework with other published works on DL applied to drug design. In order to do so, we could take advantage of freely available evaluation frameworks such as Guacamol [111] which provides a dataset, benchmarks for evaluating the distribution learning and the goal-directed optimization, and the results obtained by the most widely known works. In doing so, we would be able to rank and compare our model with the competing strategies. It is worth mentioning that we believe that our model will perform particularly well on most of the evaluated metrics as the results presented in this dissertation were obtained with datasets that include stereo-chemical information and are, therefore, more complex, while the Guacamol dataset doesn't include this type of information.

In addition, a comparison between the molecules generated by the unbiased GAN-based framework and the RNN generator could be made in order to conclude on whether different models are able to generate different types of molecules or not. A similar study could be taken for the optimized GAN and an optimized version of the RNN generator (with either TL or RL).

Lastly, and as the continuous representation of the SMILES strings created by the Encoder-Decoder model has shown promising results, we could further explore the information present in this type of representation and attempt to train several predictive models with it and search for specific properties in that latent space.

Bibliography

- [1] J. A. DiMasi, H. G. Grabowski, and R. W. Hansen, “Innovation in the pharmaceutical industry: new estimates of r&d costs,” *Journal of health economics*, vol. 47, pp. 20–33, 2016.
- [2] M. B. M. A. Rashid, “Artificial intelligence effecting a paradigm shift in drug development,” *SLAS TECHNOLOGY: Translating Life Sciences Innovation*, vol. 26, no. 1, pp. 3–15, 2021.
- [3] H. Xue, J. Li, H. Xie, and Y. Wang, “Review of drug repositioning approaches and resources,” *International journal of biological sciences*, vol. 14, no. 10, p. 1232, 2018.
- [4] T. Rodrigues, D. Reker, P. Schneider, and G. Schneider, “Counting on natural products for drug design,” *Nature chemistry*, vol. 8, no. 6, p. 531, 2016.
- [5] V. D. Mouchlis, A. Afantitis, A. Serra, M. Fratello, A. G. Papadiamantis, V. Aidinis, I. Lynch, D. Greco, and G. Melagraki, “Advances in de novo drug design: from conventional to machine learning methods,” *International journal of molecular sciences*, vol. 22, no. 4, p. 1676, 2021.
- [6] G. Schneider and U. Fechner, “Computer-based de novo design of drug-like molecules,” *Nature Reviews Drug Discovery*, vol. 4, no. 8, pp. 649–663, 2005.
- [7] B. Ramsundar, P. Eastman, P. Walters, and V. Pande, *Deep learning for the life sciences: applying deep learning to genomics, microscopy, drug discovery, and more*. O’Reilly Media, Inc., 2019.
- [8] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial networks,” *arXiv preprint arXiv:1406.2661*, 2014.

- [9] M. Olivecrona, T. Blaschke, O. Engkvist, and H. Chen, "Molecular de-novo design through deep reinforcement learning," *Journal of cheminformatics*, vol. 9, no. 1, pp. 1–14, 2017.
- [10] S. Zheng, X. Yan, Q. Gu, Y. Yang, Y. Du, Y. Lu, and J. Xu, "Qbmg: quasi-biogenic molecule generator with deep recurrent neural network," *Journal of cheminformatics*, vol. 11, no. 1, pp. 1–12, 2019.
- [11] O. Prykhodko, S. V. Johansson, P.-C. Kotsias, J. Arús-Pous, E. J. Bjerrum, O. Engkvist, and H. Chen, "A de novo molecular generation method using latent vector based generative adversarial network," *Journal of Cheminformatics*, vol. 11, no. 1, pp. 1–13, 2019.
- [12] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: <https://proceedings.neurips.cc/paper/2017/file/892c3b1c6dccd52936e27cbd0ff683d6-Paper.pdf>
- [13] A. Gupta and J. Zou, "Feedback gan (fbgan) for dna: a novel feedback-loop architecture for optimizing protein functions," 2018.
- [14] B. P. Santos, M. Abbasi, T. Pereira, B. Ribeiro, and J. P. Arrais, "Optimizing recurrent neural network architectures for de novo drug design," in *2021 IEEE 34th International Symposium on Computer-Based Medical Systems (CBMS)*, 2021, pp. 172–177.
- [15] E. Rich, "Artificial intelligence and the humanities," *Computers and the Humanities*, pp. 117–122, 1985.
- [16] G. Hessler and K.-H. Baringhaus, "Artificial Intelligence in Drug Design," *Molecules*, vol. 23, no. 10, p. 2520, oct 2018.
- [17] R. Wu, S. Yan, Y. Shan, Q. Dang, and G. Sun, "Deep image: Scaling up image recognition," *arXiv preprint arXiv:1501.02876*, vol. 7, no. 8, 2015.
- [18] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.
- [19] H.-W. Dong, W.-Y. Hsiao, L.-C. Yang, and Y.-H. Yang, "Musegan: Multi-track sequential generative adversarial networks for symbolic music genera-

- tion and accompaniment,” in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
- [20] G. Skinner and T. Walmsley, “Artificial intelligence and deep learning in video games a brief review,” in *2019 IEEE 4th International Conference on Computer and Communication Systems (ICCCS)*. IEEE, 2019, pp. 404–408.
- [21] H. Chen, O. Engkvist, Y. Wang, M. Olivecrona, and T. Blaschke, “The rise of deep learning in drug discovery,” *Drug Discovery Today*, vol. 23, no. 6, pp. 1241–1250, jun 2018.
- [22] E. Alpaydin, *Introduction to machine learning*. MIT press, 2014.
- [23] S. Russell and P. Norvig, “Artificial intelligence: a modern approach,” 2010.
- [24] J. M. De Sa, *Pattern recognition: concepts, methods, and applications*. Springer Science & Business Media, 2001.
- [25] W. Ertel, *Introduction to artificial intelligence*. Springer, 2017.
- [26] C. C. Aggarwal, *Neural Networks and Deep Learning*. Springer International Publishing, 2018. [Online]. Available: <https://doi.org/10.1007%2F978-3-319-94463-0>
- [27] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, <http://www.deeplearningbook.org>.
- [28] D. Foster, *Generative deep learning: teaching machines to paint, write, compose, and play*. O’Reilly Media, 2019.
- [29] S. Sinha and D. Vohora, “Drug discovery and development: An overview,” *Pharmaceutical Medicine and Translational Clinical Research*, pp. 19–32, 2018.
- [30] A. B. Deore, J. R. Dhumane, R. Wagh, and R. Sonawane, “The stages of drug discovery and development process,” *Asian Journal of Pharmaceutical Research and Development*, vol. 7, no. 6, pp. 62–67, 2019.
- [31] R. Gupta, D. Srivastava, M. Sahu, S. Tiwari, R. K. Ambasta, and P. Kumar, “Artificial intelligence to deep learning: Machine intelligence approach for drug discovery,” *Molecular Diversity*, pp. 1–46, 2021.

- [32] L. David, A. Thakkar, R. Mercado, and O. Engkvist, "Molecular representations in ai-driven drug discovery: a review and practical guide," *Journal of Cheminformatics*, vol. 12, no. 1, pp. 1–22, 2020.
- [33] D. Weininger, "Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules," *Journal of chemical information and computer sciences*, vol. 28, no. 1, pp. 31–36, 1988.
- [34] D. C. Elton, Z. Boukouvalas, M. D. Fuge, and P. W. Chung, "Deep learning for molecular design—a review of the state of the art," *Molecular Systems Design & Engineering*, vol. 4, no. 4, pp. 828–849, 2019.
- [35] "Rdkit open-source cheminformatics and machine learning, version 2020.09.01," <https://www.rdkit.org/>.
- [36] Author, "Smiles - a simplified chemical language," <https://www.daylight.com/dayhtml/doc/theory/theory.smiles.html>, last accessed 29 January 2021.
- [37] D. Rogers and M. Hahn, "Extended-connectivity fingerprints," *Journal of chemical information and modeling*, vol. 50, no. 5, pp. 742–754, 2010.
- [38] B. J. Neves, R. C. Braga, C. C. Melo-Filho, J. T. Moreira-Filho, E. N. Muratov, and C. H. Andrade, "Qsar-based virtual screening: Advances and applications in drug discovery," *Frontiers in Pharmacology*, vol. 9, p. 1275, 2018. [Online]. Available: <https://www.frontiersin.org/article/10.3389/fphar.2018.01275>
- [39] U. Muhammad, A. Uzairu, and D. Ebuka Arthur, "Review on: quantitative structure activity relationship (qsar) modeling," *J Anal Pharm Res*, vol. 7, no. 2, pp. 240–242, 2018.
- [40] G. B. Goh, N. O. Hodas, C. Siegel, and A. Vishnu, "Smiles2vec: An interpretable general-purpose deep neural network for predicting chemical properties," *ArXiv*, vol. abs/1712.02034, 2017.
- [41] S. Pushpakom, F. Iorio, P. A. Eyers, K. J. Escott, S. Hopper, A. Wells, A. Doig, T. Guilliams, J. Latimer, C. McNamee, A. Norris, P. Sanseau, D. Cavalla, and M. Pirmohamed, "Drug repurposing: progress, challenges and recommendations," *Nature Reviews Drug Discovery*, vol. 18, no. 1, pp. 41–58, jan 2019.

-
- [42] S. K. Chakravarti and S. R. M. Alla, “Descriptor Free QSAR Modeling Using Deep Learning With Long Short-Term Memory Neural Networks,” *Frontiers in Artificial Intelligence*, 2019.
- [43] C. Cortes and V. Vapnik, “Support-Vector Networks,” *Machine Learning*, 1995.
- [44] V. Svetnik, A. Liaw, C. Tong, J. C. Culberson, R. P. Sheridan, and B. P. Feuston, “Random Forest: A Classification and Regression Tool for Compound Classification and QSAR Modeling,” *Journal of Chemical Information and Computer Sciences*, vol. 43, no. 6, pp. 1947–1958, nov 2003.
- [45] W. Cedeño and D. K. Agrafiotis, “Using particle swarms for the development of QSAR models based on K-nearest neighbor and kernel regression,” *Journal of Computer-Aided Molecular Design*, 2003.
- [46] T. S. Schroeter, A. Schwaighofer, S. Mika, A. Ter Laak, D. Suelzle, U. Ganzer, N. Heinrich, and K.-R. Müller, “Estimating the domain of applicability for machine learning QSAR models: a study on aqueous solubility of drug discovery molecules,” *Journal of Computer-Aided Molecular Design*, vol. 21, no. 12, pp. 651–664, dec 2007.
- [47] X. Liu, K. Ye, H. W. van Vlijmen, A. P. IJzerman, and G. J. Van Westen, “An exploration strategy improves the diversity of de novo ligands using deep reinforcement learning: a case for the adenosine a 2a receptor,” *Journal of cheminformatics*, vol. 11, no. 1, pp. 1–16, 2019.
- [48] Y. Uesawa, “Quantitative structure–activity relationship analysis using deep learning based on a novel molecular image input technique,” *Bioorganic & Medicinal Chemistry Letters*, vol. 28, no. 20, pp. 3400–3403, 2018.
- [49] A. Lusci, G. Pollastri, and P. Baldi, “Deep Architectures and Deep Learning in Chemoinformatics: The Prediction of Aqueous Solubility for Drug-Like Molecules,” *Journal of Chemical Information and Modeling*, vol. 53, no. 7, pp. 1563–1575, jul 2013.
- [50] D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gómez-Bombarelli, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, “Convolutional networks on graphs for learning molecular fingerprints,” in *Advances in Neural Information Processing Systems*, 2015.

- [51] M. Popova, O. Isayev, and A. Tropsha, "Deep reinforcement learning for de novo drug design," *Science advances*, vol. 4, no. 7, p. eaap7885, 2018.
- [52] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.
- [53] A. Gupta, A. T. Müller, B. J. Huisman, J. A. Fuchs, P. Schneider, and G. Schneider, "Generative recurrent networks for de novo drug design," *Molecular informatics*, vol. 37, no. 1-2, p. 1700111, 2018.
- [54] M. H. Segler, T. Kogej, C. Tyrchan, and M. P. Waller, "Generating focused molecule libraries for drug discovery with recurrent neural networks," *ACS central science*, vol. 4, no. 1, pp. 120–131, 2018.
- [55] G. L. Guimaraes, B. Sanchez-Lengeling, C. Outeiral, P. L. C. Farias, and A. Aspuru-Guzik, "Objective-reinforced generative adversarial networks (organ) for sequence generation models," *arXiv preprint arXiv:1705.10843*, 2017.
- [56] L. Yu, W. Zhang, J. Wang, and Y. Yu, "Seqgan: Sequence generative adversarial nets with policy gradient," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 31, no. 1, 2017.
- [57] R. Gómez-Bombarelli, J. N. Wei, D. Duvenaud, J. M. Hernández-Lobato, B. Sánchez-Lengeling, D. Sheberla, J. Aguilera-Iparraguirre, T. D. Hirzel, R. P. Adams, and A. Aspuru-Guzik, "Automatic chemical design using a data-driven continuous representation of molecules," *ACS central science*, vol. 4, no. 2, pp. 268–276, 2018.
- [58] T. Blaschke, M. Olivecrona, O. Engkvist, J. Bajorath, and H. Chen, "Application of generative autoencoder in de novo molecular design," *Molecular informatics*, vol. 37, no. 1-2, p. 1700123, 2018.
- [59] B. Sanchez-Lengeling, C. Outeiral, G. L. Guimaraes, and A. Aspuru-Guzik, "Optimizing distributions over molecular space. an objective-reinforced generative adversarial network for inverse-design chemistry (organic)," *ChemRxiv*, vol. 2017, 2017.
- [60] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling,

- C. Cortes, N. Lawrence, and K. Q. Weinberger, Eds., vol. 27. Curran Associates, Inc., 2014. [Online]. Available: <https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf>
- [61] E. Putin, A. Asadulaev, Y. Ivanenkov, V. Aladinskiy, B. Sanchez-Lengeling, A. Aspuru-Guzik, and A. Zhavoronkov, “Reinforced adversarial neural computer for de novo molecular design,” *Journal of chemical information and modeling*, vol. 58, no. 6, pp. 1194–1204, 2018.
- [62] E. J. Bjerrum and B. Sattarov, “Improving chemical autoencoder latent space and molecular de novo generation diversity with heteroencoders,” *Biomolecules*, vol. 8, no. 4, 2018. [Online]. Available: <https://www.mdpi.com/2218-273X/8/4/131>
- [63] D. F. Veber, S. R. Johnson, H.-Y. Cheng, B. R. Smith, K. W. Ward, and K. D. Kopple, “Molecular properties that influence the oral bioavailability of drug candidates,” *Journal of medicinal chemistry*, vol. 45, no. 12, pp. 2615–2623, 2002.
- [64] C. A. Lipinski, F. Lombardo, B. W. Dominy, and P. J. Feeney, “Experimental and computational approaches to estimate solubility and permeability in drug discovery and development settings,” *Advanced drug delivery reviews*, vol. 23, no. 1-3, pp. 3–25, 1997.
- [65] L. Di and E. H. Kerns, *Drug-like properties: concepts, structure design and methods from ADME to toxicity optimization*. Academic press, 2015.
- [66] M. P. Gleeson, P. D. Leeson, and H. van de Waterbeemd, “Physicochemical properties and compound quality,” in *The Handbook of Medicinal Chemistry*, 2014, pp. 1–31.
- [67] R. P. Schwarzenbach, P. M. Gschwend, and D. M. Imboden, *Environmental organic chemistry*. John Wiley & Sons, 2016.
- [68] F. Mao, W. Ni, X. Xu, H. Wang, J. Wang, M. Ji, and J. Li, “Chemical structure-related drug-like criteria of global approved drugs,” *Molecules*, vol. 21, no. 1, p. 75, 2016.
- [69] O. Ursu, A. Rayan, A. Goldblum, and T. I. Oprea, “Understanding drug-likeness,” *Wiley Interdisciplinary Reviews: Computational Molecular Science*, vol. 1, no. 5, pp. 760–781, 2011.

- [70] T. I. Oprea, "Property distribution of drug-related chemical databases," *Journal of computer-aided molecular design*, vol. 14, no. 3, pp. 251–264, 2000.
- [71] G. R. Bickerton, G. V. Paolini, J. Besnard, S. Muresan, and A. L. Hopkins, "Quantifying the chemical beauty of drugs," *Nature chemistry*, vol. 4, no. 2, pp. 90–98, 2012.
- [72] P. Ertl and A. Schuffenhauer, "Estimation of synthetic accessibility score of drug-like molecules based on molecular complexity and fragment contributions," *Journal of cheminformatics*, vol. 1, no. 1, pp. 1–11, 2009.
- [73] H. Öztürk, A. Özgür, and E. Ozkirimli, "DeepDTA: deep drug–target binding affinity prediction," *Bioinformatics*, vol. 34, no. 17, pp. i821–i829, 09 2018. [Online]. Available: <https://doi.org/10.1093/bioinformatics/bty593>
- [74] Y. Parmentier, M.-J. Bossant, M. Bertrand, and B. Walther, "5.10 - in vitro studies of drug metabolism," in *Comprehensive Medicinal Chemistry II*, J. B. Taylor and D. J. Triggle, Eds. Oxford: Elsevier, 2007, pp. 231–257. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/B008045044X001255>
- [75] H. Blockeel, *Machine Learning and Inductive Inference*. Acco, 2018.
- [76] G. Pocock and C. D. Richards, *Human physiology: The basis of medicine*. Oxford university press, 1999.
- [77] K. Suzuki, *Artificial neural networks: methodological advances and biomedical applications*. BoD–Books on Demand, 2011.
- [78] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
- [79] A. Amidi and S. Amidi, "Recurrent neural networks cheatsheet," 2019, <https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks>. [Online]. Available: <https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks>
- [80] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

-
- [81] R. Pascanu, T. Mikolov, and Y. Bengio, “On the difficulty of training recurrent neural networks,” in *International conference on machine learning*. PMLR, 2013, pp. 1310–1318.
- [82] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [83] C. Olah, “Understanding lstms,” 2015, <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>. [Online]. Available: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>
- [84] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using rnn encoder-decoder for statistical machine translation,” *arXiv preprint arXiv:1406.1078*, 2014.
- [85] M. Schuster and K. K. Paliwal, “Bidirectional recurrent neural networks,” *IEEE transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.
- [86] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” *arXiv preprint arXiv:1409.3215*, 2014.
- [87] T. Wang, X. Yuan, and A. Trischler, “A joint model for question answering and question generation,” *arXiv preprint arXiv:1706.01450*, 2017.
- [88] R. Nallapati, B. Zhou, C. Gulcehre, B. Xiang *et al.*, “Abstractive text summarization using sequence-to-sequence rnns and beyond,” *arXiv preprint arXiv:1602.06023*, 2016.
- [89] M. Arjovsky and L. Bottou, “Towards principled methods for training generative adversarial networks,” *arXiv preprint arXiv:1701.04862*, 2017.
- [90] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein generative adversarial networks,” in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. International Convention Centre, Sydney, Australia: PMLR, 06–11 Aug 2017, pp. 214–223. [Online]. Available: <http://proceedings.mlr.press/v70/arjovsky17a.html>
- [91] C. Villani, *Optimal transport: old and new*. Springer Science & Business Media, 2008, vol. 338.

- [92] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.
- [93] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, “Understanding deep learning requires rethinking generalization,” *arXiv preprint arXiv:1611.03530*, 2016.
- [94] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *International conference on machine learning*. PMLR, 2015, pp. 448–456.
- [95] T. Pereira, M. Abbasi, B. Ribeiro, and J. P. Arrais, “End-to-end deep reinforcement learning for targeted drug generation,” in *In ICCBB 2020: 2020 4th International Conference on Computational Biology and Bioinformatics (ICCBB 2020)*, 2020.
- [96] T. Mikolov, M. Karafiát, L. Burget, J. Černocký, and S. Khudanpur, “Recurrent neural network based language model,” in *Eleventh annual conference of the international speech communication association*, 2010.
- [97] S. Salman and X. Liu, “Overfitting mechanism and avoidance in deep neural networks,” *arXiv preprint arXiv:1901.06566*, 2019.
- [98] A. Gaulton, L. J. Bellis, A. P. Bento, J. Chambers, M. Davies, A. Hersey, Y. Light, S. McGlinchey, D. Michalovich, B. Al-Lazikani *et al.*, “ChEMBL: a large-scale bioactivity database for drug discovery,” *Nucleic acids research*, vol. 40, no. D1, pp. D1100–D1107, 2012.
- [99] T. Sterling and J. J. Irwin, “Zinc 15–ligand discovery for everyone,” *Journal of chemical information and modeling*, vol. 55, no. 11, pp. 2324–2337, 2015.
- [100] E. Jang, S. Gu, and B. Poole, “Categorical reparameterization with gumbel-softmax,” 2017.
- [101] H. Schmidhammer, F. Erli, E. Guerrieri, and M. Spetea, “Development of diphenethylamines as selective kappa opioid receptor ligands and their pharmacological activities,” *Molecules*, vol. 25, no. 21, p. 5092, 2020.
- [102] L. Lalanne, G. Ayranci, B. L. Kieffer, and P.-E. Lutz, “The kappa opioid receptor: from addiction to depression, and back,” *Frontiers in psychiatry*, vol. 5, p. 170, 2014.

- [103] K. L. Mores, B. R. Cummins, R. J. Cassell, and R. M. Van Rijn, “A review of the therapeutic potential of recently developed g protein-biased kappa agonists,” *Frontiers in pharmacology*, vol. 10, p. 407, 2019.
- [104] Y. Shang and M. Filizola, “Opioid receptors: Structural and mechanistic insights into pharmacology and signaling,” *European journal of pharmacology*, vol. 763, pp. 206–213, 2015.
- [105] S. Page, M. M. Mavrikaki, T. Lintz, D. Puttick, E. Roberts, H. Rosen, F. I. Carroll, W. A. Carlezon, and E. H. Chartoff, “Behavioral pharmacology of novel kappa opioid receptor antagonists in rats,” *International Journal of Neuropsychopharmacology*, vol. 22, no. 11, pp. 735–745, 2019.
- [106] M. Valenza, K. A. Windisch, E. R. Butelman, B. Reed, and M. J. Kreek, “Effects of kappa opioid receptor blockade by ly2444296 hcl, a selective short-acting antagonist, during chronic extended access cocaine self-administration and re-exposure in rat,” *Psychopharmacology*, vol. 237, no. 4, pp. 1147–1160, 2020.
- [107] T. Oliveira Pereira, M. Abbasi, B. Ribeiro, and J. P. Arrais, “End-to-end deep reinforcement learning for targeted drug generation,” in *2020 4th International Conference on Computational Biology and Bioinformatics*, ser. ICCBB 2020. New York, NY, USA: Association for Computing Machinery, 2021, p. 7–13. [Online]. Available: <https://doi.org/10.1145/3449258.3449260>
- [108] S. K. Chakravarti and S. R. M. Alla, “Descriptor free qsar modeling using deep learning with long short-term memory neural networks,” *Frontiers in Artificial Intelligence*, vol. 2, p. 17, 2019. [Online]. Available: <https://www.frontiersin.org/article/10.3389/frai.2019.00017>
- [109] T. Pereira, M. Abbasi, J. L. Oliveira, B. Ribeiro, and J. Arrais, “Optimizing blood–brain barrier permeation through deep reinforcement learning for de novo drug design,” *Bioinformatics*, vol. 37, pp. i84–i92, 07 2021. [Online]. Available: <https://doi.org/10.1093/bioinformatics/btab301>
- [110] A. L. Beam, B. Kompa, A. Schmaltz, I. Fried, G. Weber, N. Palmer, X. Shi, T. Cai, and I. S. Kohane, “Clinical concept embeddings learned from massive sources of multimodal medical data,” in *PACIFIC SYMPOSIUM ON BIO-COMPUTING 2020*. World Scientific, 2019, pp. 295–306.

- [111] N. Brown, M. Fiscato, M. H. Segler, and A. C. Vaucher, “Guacamol: Benchmarking models for de novo molecular design,” *Journal of Chemical Information and Modeling*, vol. 59, no. 3, pp. 1096–1108, 2019. [Online]. Available: <https://doi.org/10.1021/acs.jcim.8b00839>

