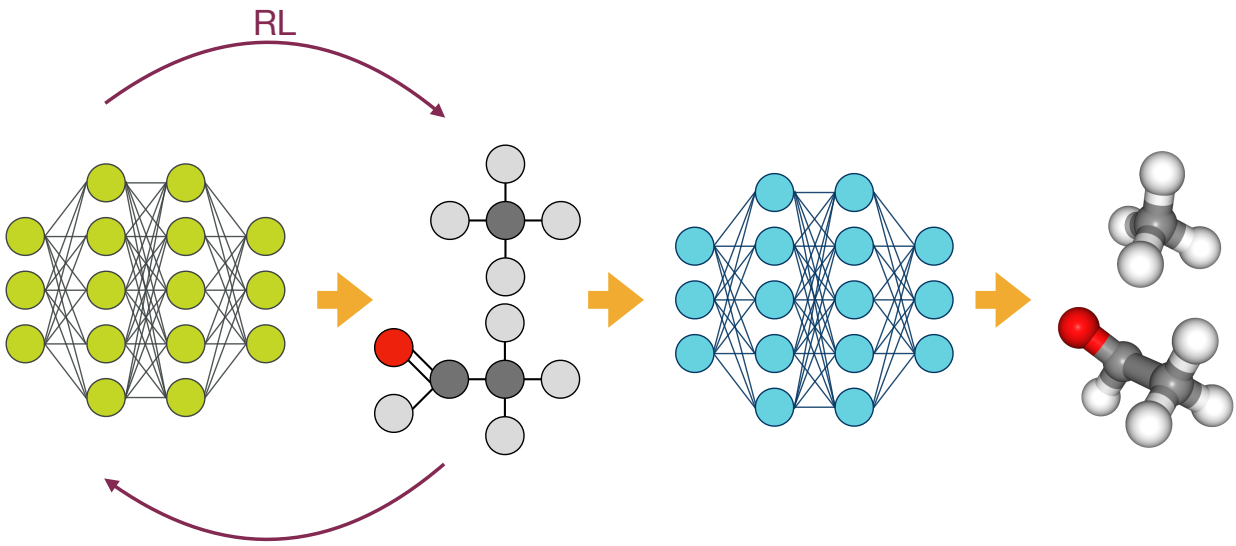
CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG



Towards molecular design with desired property profiles and 3D conformer generation using Deep Generative Models

Searching for target compound design methods capturing all available molecular structure information

Master's thesis in Computer science and engineering

Sara Romeo Atance and Juan Viguera Diez

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2021

Master's thesis 2021

Towards molecular design with desired property profiles and 3D conformer generation using Deep Generative Models

Searching for target compound design methods capturing all available molecular structure information

Sara Romeo Atance
Juan Viguera Diez

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2021

Towards molecular design with desired property profiles and 3D conformer generation using Deep Generative Models
Searching for target compound design methods capturing all available molecular structure information
Sara Romeo Atance
Juan Viguera Diez

Supervisor: Simon Olsson, Department of Computer Science and Engineering
Advisor: Rocío Mercado, AstraZeneca
Examiner: Devdatt Dubhashi, Department of Computer Science and Engineering

Master's Thesis 2021
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Schematic of the two models proposed in this work: a graph-based generative model for molecules with desired property profiles fine-tuned using reinforcement learning, and a transferable conformer generator based on normalizing flows. While we develop these models in parallel, ideally they could be later combined for the construction of a reinforcement learning-based 3D deep generative model for *de novo* design.

Typeset in LaTeX
Gothenburg, Sweden 2021

Towards molecular design with desired property profiles and 3D conformer generation using Deep Generative Models
Searching for target compound design methods capturing all available molecular structure information
Sara Romeo Atance and Juan Viguera Diez
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

As COVID-19 has made sure to show, drug discovery is a very relevant but also very challenging process: the goal is to find a target compound with some set of desired properties (*de novo* design) among the $10^{60} - 10^{100}$ theoretically possible drug-like molecules. Machine learning methods have proven to be effective to address this problem, allowing for efficient exploration of the vast chemical space. Whilst string-based generative models have been thoroughly studied, there has not been yet a great focus on taking into account either the molecular graph or its 3D structure. That is why we propose two generative models: on the one hand, a graph-based deep generative model for targeted design using reinforcement learning; and, on the other hand, a deep auto-regressive generative model of small-molecules conformations. In this work we show the ability of the reinforcement learning framework to fine-tune the graph-based model towards generation of molecules with different desired sets of properties, even when few molecules have the goal attributes initially. We also present some early results in the search of a transferable molecular conformations generator that is able to learn from sample conformations and known rules of physics. These results suggest the potential of including structural information in molecular deep generative models. And, specially, motivate future work towards developing a reinforcement learning-based 3D model for *de novo* design.

Keywords: Machine learning, drug design, *de novo*, generative models, graph neural networks, normalizing flows, reinforcement learning, molecular modeling.

Acknowledgements

There are many people we would like to thank for their support and for making this thesis possible.

It all started when each of us contacted Ola Engkvist, head of the Molecular AI team at AstraZeneca. We would like to thank him for introducing us to the topic and awakening our interest about it. We are also really grateful for making us feel so welcome and cared from the beginning. And we are specially thankful to him for sharing his time with us and being available when needed, even with short notice, in spite of his surely tight schedule. But, above all, thank you Ola for introducing us to Rocío Mercado and Simon Olsson. They are the best supervisors we could have had.

We are really grateful for having met you both, Simon and Rocío. Thank you for proposing such interesting topics and for your help, guidance and availability through the process. It has been a pleasure to work with you. We would like to thank you Rocío for the good vibes you transmit, always with a smile. Your joy and kindness is contagious! And we would like to thank you Simon for being so attentive and for the nice chats at the fikas. They made us feel closer to you in these pandemic times we live at. We really enjoyed them and we appreciate a lot having been able to get to know you better!

Another person we have been able to get to know thanks to this opportunity is Julio Ponte Hernández, with whom we have worked closely and who has helped us with his feedback about this project. It has been a pleasure to get to know you Julio. Hope we can go climbing together more times!

We would also like to thank everyone in the Molecular AI team for their feedback, guidance and nice chats at lunch time. Specially, Tomas Bastys, who always had an interesting (and probably politically incorrect) topic in mind. Thanks for those fun, friendly and interesting discussions! Also, thank you Samuel Genheden for your seminars. They were really interesting and useful.

Additionally, we want to thank our friends for making hard study times fun and good times even better. And for their love and support even from the distance.

Lastly but not least, we thank our families for their support and sacrifice for our education. We always feel you with us although we are a bit far away. Thank you for being our greatest inspiration and best example to look up to. If this is possible, it is because of you.

Sara Romeo Atance and Juan Viguera Diez, Gothenburg, June 2021

Contents

List of Figures

List of Tables

List of Abbreviations

AI Artificial Intelligence. 1, 21
APD Action probability distribution. xiii, xix, 16, 18, 19, 30, 33, 34, 41, 58

BAR Best Agent Reminder. x, xiv, 31, 43, 44, 58

cfconv Continuous filter convolutional. xiii, 11, 12
CNN Convolutional neural network. 11, 14

DRD2 Dopamine receptor D_2. x, xi, xv, xvi, xix, 23, 30, 32, 33, 34, 35, 47, 49, 50, 57, 58, X, XIV

ENN Edge neural network. xix, 17, 30, V

GAN Generative advesarial networks. 13
GGNN Gated graph neural network. 16, 17, 29, 30, 31
GNN Graph neural network. ix, 2, 14, 15, 16, 17, 18, 58, V
GRU Gated recurrent unit. 16

InChI International chemical identifier. 6

KL Kullback Leibler. xiii, 19, 26, 27, 33, 41

LSTM Long short-term memory. 16

MD Molecular dynamics. xv, 39, 52, 53, 54, 59
MLP Multilayer perceptron. xix, 15, 18, 29, 30
MPNN Message passing neural network. 14, 16, 17
MSE Mean squared error. 21

NeRF Natural extension reference frame. ix, xiii, 8, 10, 37
NVP Non-volume preserving. x, 24, V

PPT Percentage properly terminated. 19
PU Percentage unique. 19, 34
PV Percentage valid. 19, 34
PVPT Percentage valid of those properly terminated. 19

QED Quantitative estimate of drug-likeness. x, xiv, xv, xvi, xix, 23, 32, 33, 46, 47, 49, 50, 57, 58, IX, XIV
QSAR Quantitative structure-activity relationship. 23, 33, 35, 47, 48

RL Reinforcement learning. xi, xiii, 20, 21, 22, 34, 35, 42, 44, 57, 60, 63, I, II, III, XIII, XIV, XV

RNN Recurrent neural network. 2, 13, 14, 17

SELU Scaled exponential linear unit. 30

SMILES Simplified Molecular Input Line Entry System. ix, 2, 6, 13, 21, 30, 32

VAE Variational autoencoder. 13

1

Introduction

1.1 Introduction

We develop this thesis in the context of a world pandemic: COVID-19 has challenged the world-wide social-economic order, highlighting the need of, not only prepared enough health care systems, but also a quick and effective response to these kinds of threats. This past year, the race for developing vaccines has been motivated by both economic and scientific reasons, as well as the hope of our societies getting back to their previous lives and feeling safe again. Even if we did not find ourselves in this challenging situation, many more reasons could be outlined to motivate the research in the field of drug discovery, such as looking for more effective (and/or cheaper to produce) treatments for already curable diseases or improving traditional treatments losing their effectiveness, as it is happening with the use of antibiotics for bacterial infections. The impact of drug discovery in our lives is formidable and thus, the need of more advanced methods in this field is undeniable. However, looking for chemical compounds with desired properties is not at all an easy task.

The process of drug discovery can be compared to finding a needle in a haystack: the aim is to find a compound which has some desired properties (solubility, toxicity...) among the $10^{60} - 10^{100}$ theoretically possible, drug-like molecules [3]. Therefore it is neither feasible to synthesise and test every potential compound experimentally, nor to computationally screen a representative fraction of this chemical space.

Machine learning methods are now used to address this problem by virtually designing candidate compounds with desired property profiles (e.g. *de novo* design), instead of screening libraries of millions of molecules looking for the goal attributes. In this manner, a larger fraction of the chemical space can be explored with a smaller computational footprint while keeping the molecules to synthesise and later test small.

Our project aims to take natural steps in the field of AI (Artificial Intelligence)-driven drug discovery. *De novo* design has been successfully explored using tools which encode molecules as strings [4]. However, it has not been fully explored using graphs, which are a more natural representation of most molecules, since their atomic constituents interact with each other through bonds and their overall structure can be represented as a graph. Moreover, the use of 3D and spatial information has not been thoroughly explored in *de novo* design. Therefore, the goal of this project is to

study the reinforcement learning and 3D aspects of deep generative models. While we tackle each of these points in parallel, ideally they could be later combined for the construction of a reinforcement learning-based 3D deep generative model for *de novo* design.

1.2 Context

As previously introduced, the concept of generating molecules given an input set of desired molecular properties is known as *de novo* drug design. Currently, the best way to generate targeted molecules using graph-based methods is to pretrain a model on a general set of molecules (such as the ChEMBL database [5]), and then subsequently fine-tune the model on a smaller set of molecules (such as a set of known 'actives'). Although such approaches work well, a disadvantage is that this sort of fine-tuning process requires the existence of at least a small dataset which contains examples of molecules with the desired properties. Alternatively, one could generate molecules and then filter them, looking for the desired molecular properties. However, in the ideal molecular generative model, one would only generate molecules that have the desired properties, thus eliminating the need to filter them in a subsequent step. Efficient *de novo* design models have the potential to speed up the drug development process significantly by making the hit discovery and lead optimisation phases in drug discovery more efficient and increasing our chances of success in later (e.g. pre-clinical and clinical) phases.

Machine learning models are data-driven and provide a means to estimate probability distributions from data. This is useful for generative chemistry for the following reasons. Appropriately trained models can generate new molecules which follow the chemical rules without necessarily imposing those restrictions in the model – in other words, state-of-the-art machine learning methods may 'learn chemistry' from data. Such models facilitate the exploration process of the space of possible compounds exactly for this reason – they 'understand' chemistry and can take rational actions, analogous to an 'artificial chemist'. The first approaches [6] to drug design using machine learning used molecular string representations (SMILES) and recurrent neural networks (RNNs). More recently, graph neural networks (GNNs) [7] and other graph-based architectures [8, 9, 10, 11, 12] have been used for this purpose. GNNs are specially tailored neural network architectures for learning representations for graph-structured data and are thus nicely suited for handing chemical structures. Both string- and graph-based methods work great for molecular generation, but there is still the limitation that many deep generative models do not take into account 3D information and therefore often fail to capture stereochemistry, which is critical in pharmaceutical applications.

In solution, small molecules will adopt a broad distribution of distinct microscopic configurational states. From statistical mechanics, we know that, at thermal equilibrium, the probability of a microscopic configuration, $\boldsymbol{X}$, is proportional to the Boltzmann factor $\exp(-\beta U(\boldsymbol{X}))$, where $\beta = 1/k_b T$ is the 'inverse temperature' and $U(\boldsymbol{X})$ is the potential energy of the configuration. Many molecular properties

are functions of the distribution of conformations such as the binding free energy. Consequently, sampling this distribution is critical for accurate property prediction. However, efficiently generating configurations following the Boltzmann distribution remains a challenging problem. Therefore, establishing new methods which allow us to directly generate configurations of a molecule with probabilities proportional to the Boltzmann factor is of great interest.

1.3 Goals and Challenges

In this project we will take two direct steps towards the end-goal of *de novo* design:

For our first goal, we want to build a model which allows us to generate graphs of molecules with some given set of desired properties (*de novo* drug design). To do this, we can use graph-based generative models together with reinforcement learning. We propose to use graph-based methods because, even though the performance of string-based approaches is remarkable, they do not take into account the internal structure of the molecule and the interactions between atoms. Therefore, we believe by working with graphs we could boost the performance of the generative model even more. Furthermore, graph-based methods will provide a better integration with 3D generative models at a later time.

For our second goal, we want to develop a model that generates physically realistic configurations for a given a molecule. We describe a molecule by its chemical graph, which contains information about the atomic composition and bonds between atoms. The objective is, given a model for the potential energy of the molecule, to generate conformations in thermal equilibrium and which therefore follow a Boltzmann distribution.

These two aims are the key components of a 3D conditional generative model. In particular, it could be plausible to incorporate the configurational information generated by the second project in the first one and construct a reinforcement learning-based 3D generative model, creating an integrated model. As such, our goal is to make both aims as compatible as possible initially, to enable subsequent integration.

2

Theory

2.1 Molecular representations

In this section we will describe how we can represent molecules in a 'friendly' syntax for both computers and scientists [13].

2.1.1 Classical notations

Many different notations have been developed during the past to be specially suitable for different situations. The most well-known one is the empirical formula, such as H_2O for water. This notation includes information about how many atoms of each kind there are in the molecule, but it does not indicate how atoms are linked nor includes information about the molecular geometry. For these reasons, and as cheminformatics started to gain popularity, new computer-readable representations were developed.

2.1.2 Graph representations

A graph is a set of nodes V and edges E, where each edge in E connects a pair of nodes in V. In the case of a molecular graph, each atom in the molecule will be associated to a node and each bond with a edge. A molecular graph is a 2D object which can be used to represent 3D information by encoding spatial coordinates as node features together with other properties such as atom types, formal charges or chirality. The graph representation will also include an adjacency matrix which will indicate how nodes are connected and a set of edge features such as the kind of bond. It must be noted the adjacency matrix can be given in the form of edge features indicating if a certain type of bond exists or not. Graph representations are node-order dependent and different graph traversal algorithms such as depth-first or breadth-first can be used to determine the order.

Some disadvantages of working with molecular graphs are that they are not compact (they can be memory expensive) and that there are certain types of molecules which cannot be encoded using graphs, such as those which contain ionic or metal-metal bonds.

2.1.3 Linear notations

The large memory usage of matrix representations make them not suitable for simple cheminformatic analysis. For that reason, molecules are commonly encoded using strings of characters which can be interpreted following their corresponding set of rules. Some popular linear representations are SMILES [14] and InChI [15].

2.1.3.1 SMILES

Simplified Molecular Input Line Entry System (SMILES) is a non-unique (there are multiple notations for the same compound) and unambiguous (there is only one compound associated to each representation) notation. The representation is generated by assigning a number to each atom and then traversing the molecule following the specified order. RDKit [16] uses depth-first search as graph traversal algorithm. Having different representations for the same molecule can be used for introducing noise into the model as well as for data augmentation [17]. The different representations are generated by choosing different starting nodes an then following the same graph traversal algorithm.

2.2 Conformation representations and internal coordinates

Now that we know how to codify a molecular graph, we present how molecular conformations can be encoded. A molecular conformation is a spatial arrangement of the atoms of a molecule. The ensemble of 3D conformers accessible to a molecule has an important impact on the properties of the molecule [18], similar to how proteins exhibit different properties when folded or unfolded [19]. The most straightforward way to codify molecular conformations is to annotate the Cartesian 3D coordinates of the atoms in the molecule in a table. However, this approach is not convenient if we want a univocal representation since the same spatial arrangement can be rotated or translated. Indeed, from the $3N$ degrees of freedom of the Cartesian coordinates for atoms in a molecule (N is the number of atoms in a molecule), 6 are fixed for a conformation: three translational coordinates (e.g. from the center of mass) and three rotational elements corresponding to the Euler angles. Therefore, a convenient representation has $3N-6$ degrees of freedom. This can be achieved by computing $N-1$ bond lengths, $N-2$ bond angles and $N-3$ torsion angles. We mathematically define in the following section these quantities.

2.2.1 Geometric definitions

Having understood the advantages of using internal coordinates we can now proceed to formally define the previously introduced magnitudes. Let $\boldsymbol{r}_i$ with $i \in [1,4]$ be the positions of four atoms and $\boldsymbol{r}_{i,j} = \boldsymbol{r}_j - \boldsymbol{r}_i$. Then $\hat{\boldsymbol{n}}_1 = \frac{\boldsymbol{r}_{1,2} \times \boldsymbol{r}_{2,3}}{\|\boldsymbol{r}_{1,2}\| \|\boldsymbol{r}_{2,3}\|}$ and $\hat{\boldsymbol{n}}_2 = \frac{\boldsymbol{r}_{2,3} \times \boldsymbol{r}_{3,4}}{\|\boldsymbol{r}_{2,3}\| \|\boldsymbol{r}_{3,4}\|}$ are unitary perpendicular vectors to the plane formed by atoms $(1,2,3)$ and $(2,3,4)$ respectively. In the previous expressions, $\times$ denotes the cross product and $\|\boldsymbol{v}\|$ denotes the norm of $\boldsymbol{v}$. We define:

- The distance between atom i and j is

$$d_{i,j} = \|\boldsymbol{r}_{i,j}\| \,. \tag{2.1}$$

- The angle described by atoms i, j and k is

$$\theta_{i,j,k} = \arccos \frac{\boldsymbol{r}_{j,i} \cdot \boldsymbol{r}_{j,k}}{\|\boldsymbol{r}_{i,j}\| \, \|\boldsymbol{r}_{j,k}\|} \tag{2.2}$$

  with $\theta_{i,j,k} \in [0, 180]$. Here, '$\cdot$' denotes the dot product.
- The torsion/dihedral angle described by atoms i, j, k and l is defined by

$$\cos \varphi_{i,j,k,l} = \frac{\boldsymbol{r}_{i,j} \times \boldsymbol{r}_{j,k}}{\|\boldsymbol{r}_{i,j}\| \, \|\boldsymbol{r}_{j,k}\|} \cdot \frac{\boldsymbol{r}_{j,k} \times \boldsymbol{r}_{k,l}}{\|\boldsymbol{r}_{j,k}\| \, \|\boldsymbol{r}_{k,l}\|} \tag{2.3}$$

  and

$$\sin \varphi_{i,j,k,l} = \frac{\boldsymbol{r}_{i,j} \times \boldsymbol{r}_{j,k}}{\|\boldsymbol{r}_{i,j}\| \, \|\boldsymbol{r}_{j,k}\|} \times \frac{\boldsymbol{r}_{j,k} \times \boldsymbol{r}_{k,l}}{\|\boldsymbol{r}_{j,k}\| \, \|\boldsymbol{r}_{k,l}\|} \cdot \frac{\boldsymbol{r}_{j,k}}{\|\boldsymbol{r}_{j,k}\|} \tag{2.4}$$

  Therefore,

$$\cos \varphi_{1,2,3,4} = \hat{\boldsymbol{n}}_1 \cdot \hat{\boldsymbol{n}}_2 \tag{2.5}$$

  and

$$\sin \varphi_{1,2,3,4} = \hat{\boldsymbol{n}}_1 \times \hat{\boldsymbol{n}}_2 \cdot \frac{\boldsymbol{r}_{2,3}}{\|\boldsymbol{r}_{2,3}\|}. \tag{2.6}$$

  Equation 2.5 comes from simple properties of the dot product (since both vectors are unitary). Additionally, Equation 2.6 can be derived from the norm of the cross product ($\|\boldsymbol{r}_{j,i} \times \boldsymbol{r}_{j,k}\| = \|\boldsymbol{r}_{j,i}\| \|\boldsymbol{r}_{j,k}\| \sin \theta_{i,j,k}$), given that $\hat{\boldsymbol{n}}_1 \times \hat{\boldsymbol{n}}_2$ and $\boldsymbol{r}_{2,3}$ are parallel vectors by construction. Therefore, we can also define the torsion as the angle between the planes formed by atoms $(1, 2, 3)$ and $(2, 3, 4)$. It measures the twisting of a set of 4 atoms as can be observed in Figure 2.1.

2.2.2 Internal coordinates and Z-matrix

The previously introduced quantities use a set of already placed atoms to construct a reference frame. As we compute these new coordinates with respect to other atoms in the molecule, they are called internal coordinates. Moreover, this way of codifying molecule conformations is known as Z-matrix, and is constructed as follows:

1. We first compute the distance from the first to the second atom.
2. Then we compute the distance from e.g. the second atom to the third one and the angle that these three atoms describe.
3. For the fourth and next atoms, we compute a distance, an angle and a torsion using three already placed atoms as a reference.

This process is illustrated in Figure 2.2, in which the values of distances, angles, torsions and reference atoms are annotated.

Given its Z-matrix, a conformation is uniquely defined. However, given a single conformation, there exist several Z-matrices that can describe it.

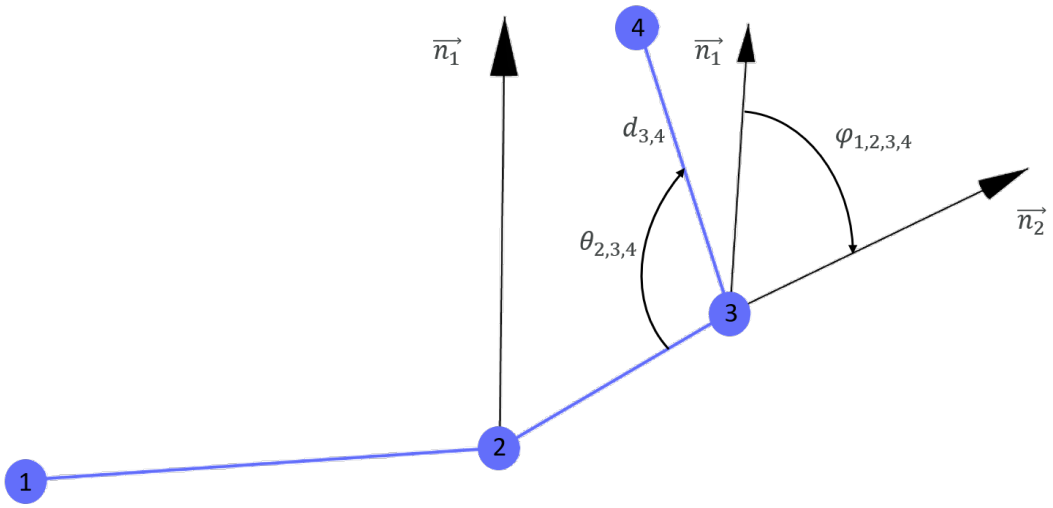
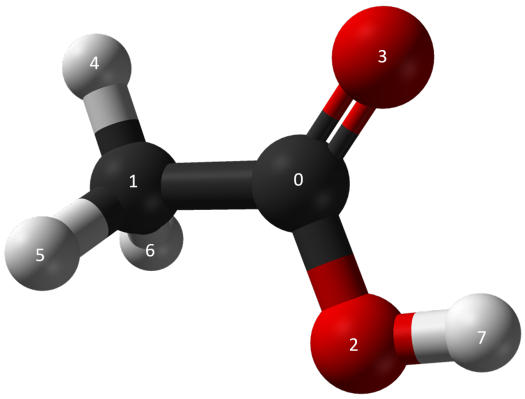


Figure 2.1: Visualisation of the definitions of distance, angle and torsion. The distance between atom 3 and 4 is $d_{3,4}$, $\theta_{2,3,4}$ is the angle described by atoms 2, 3 and 4 and $\varphi_{1,2,3,4}$ is the torsion described by atoms 1, 2, 3 and 4.

The last ingredient needed for having a univocal representation is setting a rule to choose the reference atoms. Since in this project we will construct conformations placing one atom at each generation step (auto-regressive model), it is very convenient to choose the reference atoms so that they are connected among themselves and also to the atom to place. One way of doing this is to generate an atom ranking in a deterministic fashion and choose the reference atoms with a breadth-first algorithm based on this ranking. In particular, in this project we use the canonical ranking `rdkit.Chem.rdmolfiles.CanonicalRankAtoms` [16] and a standard breadth-first algorithm [20]. These algorithms traverse graphs choosing the highest ranked node when there is ambiguity. Then, the last three preceding atoms (i.e. the 'parent' atoms) to the the current atom are chosen as reference atoms. For example, in Figure 2.2, we can observe how, given an atom order based on a ranking, the atoms connected to atom 0 $\{1, 2, 3\}$ are placed first (coloured in yellow in Figure 2.2). In the case of atom 3, atoms 1 and 2 are not connected but $\{0, 1, 2\}$ are the only set we can use as a reference at this point. Then, coloured in green, the neighbours of the neighbours of 0 are placed. Again, when there is ambiguity with what atom to choose as the next reference atom, the one with the highest ranking (smaller order index) is chosen.

2.2.3 Natural extension Reference Frame (NeRF): Local and global reference coordinate frames.

With the previous definitions, we have enough tools for transforming Cartesian to internal coordinates. In this section, we will describe how to transform internal to Cartesian coordinates using the Natural extension Reference Frame.



Element	Atom	Distance reference	Angle reference	Torsion reference	Distance(Å)	Angle (°)	Torsion (°)
C	1	0 (C)	-	-	1.504	-	-
O	2	0	1	-	1.359	111.060	-
O	3	0	1	2	1.210	126.239	-179.989
H	4	1	0	2	1.088	109.500	179.955
H	5	1	0	2	1.092	109.523	-59.068
H	6	1	0	2	1.092	109.530	58.968
H	7	2	0	1	0.968	105.881	-179.999

Figure 2.2: Illustration of the construction of the Z-matrix of the acetic acid using atom rankings.

Given a set of 3 atoms, we can define a reference frame in which the third atom lies on the origin, the second on the negative x-axis and the first one on the xy-plane. Under these conditions, the coordinates of the fourth atom can be indicated using spherical coordinates, say,

$$\boldsymbol{r} = (r\cos\theta, r\sin\theta\cos\varphi, r\sin\theta\sin\varphi)\,, \tag{2.7}$$

where $r = d_{3,4}$, $\theta = \pi - \theta_{2,3,4}$ and $\varphi = \varphi_{1,2,3,4}$ (as shown in Figure 2.3). This is the 'trick' that NeRF uses.

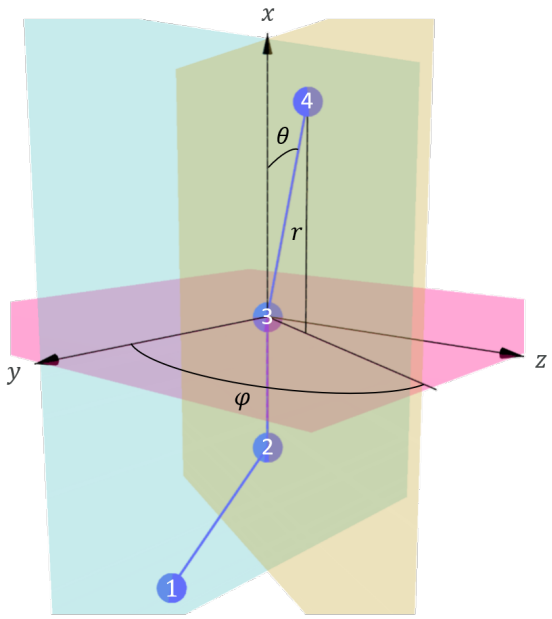


Figure 2.3: Natural extension Reference Frame (NeRF) illustration. Choosing atom 3 as origin, $\hat{\boldsymbol{r}}_{2,3}$ as x-axis and atoms $\{1, 2, 3\}$ defining the x-y plane, the coordinates of atom 4 can be written in spherical coordinates (as shown in Equation 2.7).

Under this (local) coordinate system defined by the other atoms in a molecule (as we do when we use internal coordinates), the transformation from internal to Cartesian is trivial and given by Equation 2.7. The only detail remaining is how to express these coordinates in some reference frame, such as a global reference frame used to specify the coordinates of all the atoms in a molecule. Given the coordinates of atoms 1, 2 and 3 in this global frame, we can make use of the change of basis theorem to compute a rotation matrix defined by column vectors,

$$\boldsymbol{R} = [\hat{\boldsymbol{r}}_{2,3},\quad \hat{\boldsymbol{n}}_1 \times \hat{\boldsymbol{r}}_{2,3},\quad \hat{\boldsymbol{n}}_1]\,, \tag{2.8}$$

where $\hat{\boldsymbol{r}}_{2,3} = \frac{\boldsymbol{r}_{2,3}}{\|\boldsymbol{r}_{2,3}\|}$. In this last equation, the columns of $\boldsymbol{R}$ are the unitary coordinates of the NeRF axes in the global reference frame. Finally, we can transform a vector in the NeRF to the global frame by rotating and adding the coordinates of the local frame origin,

$$\boldsymbol{r}_{global} = \boldsymbol{R}\boldsymbol{r}_{local} + \boldsymbol{r}_3, \tag{2.9}$$

where the subscripts are added to stress the basis which should be used when writing the coordinates of the vectors. As previously stated, if not specified, then the global frame coordinates should be used. In this work we will use the reference frame defined by the four first atoms as the general reference frame.

These kinds of transformations are especially important when dealing with protein simulations. In particular, the change of reference frame consumes an important fraction of the computing time and efficient implementations of these methods exist (such as pNeRF[21]).

2.2.4 SchNet and sub-conformations representations

Convolutional neural networks (CNNs) [22] have been extensively used for processing image, audio and video. However, the atoms in a molecule are not arranged in a grid fashion and, therefore, cannot be correctly treated by models of this kind. SchNet[1], a continuous-filter convolutional (cfconv) neural network for modelling quantum interactions, is a possible generalisation of CNNs which is able to deal with (sub)molecular data. The input of this model is a set of atomic numbers for N atoms (Z_1, ..., Z_N) and a set of their corresponding positions ($\boldsymbol{r}_1$, ..., $\boldsymbol{r}_N$). Based on this input, the model generates a numerical representation which can be thought as a (sub)conformational embedding. In particular, in this work we use SchNet for summarising the information about previously placed atoms in the generative process of a conformation.

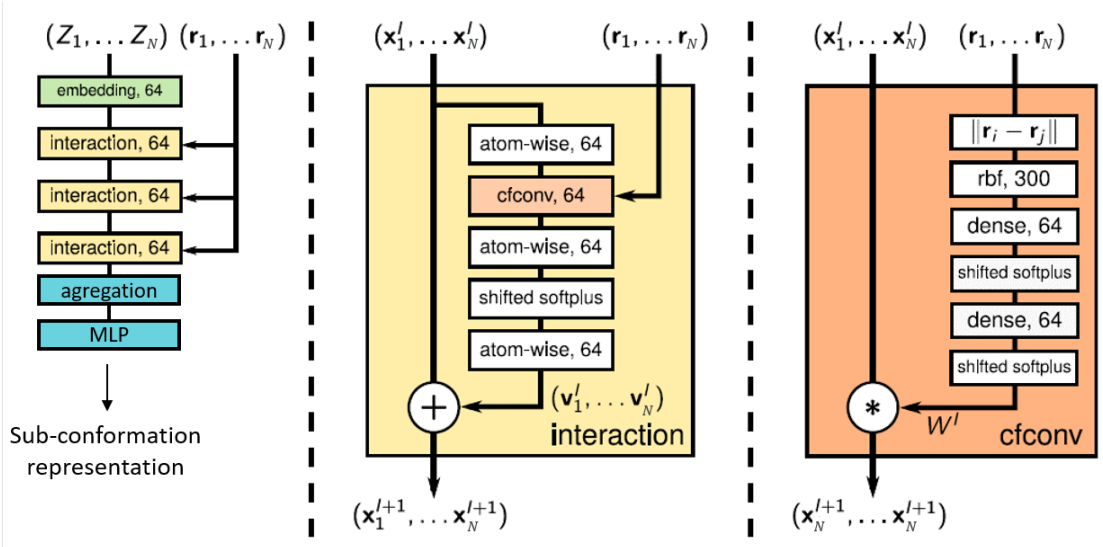


Figure 2.4: Representation of the SchNet architecture (left) and detail of the interaction (centre) and the cfconv (right) blocks. The inputs, atomic numbers of N atoms (Z_1, ..., Z_N) and its corresponding positions ($\boldsymbol{r}_1$, ..., $\boldsymbol{r}_N$), are processed to generate atom-wise feature vectors ($\mathbf{x}_1^l$, ..., $\mathbf{x}_N^l$) that are aggregated to generate a sub-conformational representation. Figure adapted from [1].

The model creates first a feature representation of fixed length, F (64 in our example) of the different N atoms. An atomic number-dependent embedding layer is used for that purpose. Then, a number of interaction blocks are concatenated (3 in Figure 2.4). The idea behind these blocks is that they allow both intra- and inter-atomic feature vectors recombination. In particular, intra-atomic recombination is performed in the atom-wise layers. These are linear dense layers that are applied separately to the representation of atom i, $\boldsymbol{x}_i^l$, and can be written as

$$\mathbf{x}_i^{l+1} = \boldsymbol{D}^l \mathbf{x}_i^l + \boldsymbol{b}^l, \tag{2.10}$$

where l is an identifier of the layer and $\boldsymbol{D}^l$ and $\boldsymbol{b}^l$ are its trainable parameter arrays. Inter-atomic recombination occurs in the cfconv layers. Given some filter-generating function

$$\boldsymbol{W}^l : \mathbb{R}^3 \to \mathbb{R}^F, \tag{2.11}$$

mapping from some real space coordinates to the atom feature space, cfconv layers compute

$$\mathbf{x}_i^{l+1} = \left(\boldsymbol{X} \cdot \boldsymbol{W}^l\right)_i = \sum_j \mathbf{x}_j^l \odot \boldsymbol{W}^l(\boldsymbol{r}_i - \boldsymbol{r}_j). \tag{2.12}$$

For this work, the filter-generating function is constructed in a way it respects molecular translational and rotational invariance. One way of achieving this is by making the interaction layer depend on the inter-atomic distances, $d_{i,j}$. Without further processing, the filters would be highly correlated since a neural network after initialisation is close to linear. This leads to a plateau at the beginning of training that is hard to overcome. To help solve this problem, radial basis functions,

$$e_k(\boldsymbol{r}_i - \boldsymbol{r}_j) = \exp(-\gamma \left\| d_{i,j} - \mu_k \right\|^2), \tag{2.13}$$

are introduced. In particular, filter centers positioned at $0 < \mu_k < 30$Å every 0.1Å with $\gamma = 10$Å are used such that all distances in drug-like compounds are covered by the filters. After that, a number of dense layers (2 in Figure 2.4) with shifted softplus non-linearity, $\mathrm{ssp}(x) = \log(0.5e^x + 0.5)$, functions are added.

One final feature not discussed yet is that the interaction block has a residual structure, inspired by Google's ResNet [23]. That is, some quantities $\boldsymbol{v_i}$ are computed to be summed to the inputs, say,

$$\mathbf{x}_i^{l+1} = \mathbf{x}_i^l + \boldsymbol{v}_i^l. \tag{2.14}$$

Where $\boldsymbol{v}_i^l$ are the residues, generated with a combination of atom-wise and cfconv layers and non-linearities such as shifted softplus.

Finally, after the atom feature vectors are generated, an aggregation function is used for getting a fixed size output (usually sum or average are used in available implementations) followed by a multilayer perceptron in charge of the final sub-conformational representation. Here we use a popular PyTorch implementation of this model, SchNetPack [24].

2.3 Deep generative models

Deep generative models exploit the ability of machine learning models to estimate probability distributions from data to generate new samples which resemble the training set. These kind of models have successfully been applied to generate text [25], music [26] and images [27].

Additionally, they can be very useful in generative chemistry for efficiently exploring the vast chemical space and finding compounds with target properties [28].

2.3.1 Molecular deep generative models

RNN [29], VAE [6] and GAN [30] have successfully been used as generative models for *de novo* design encoding molecules as SMILES.

While the performance of these approaches is remarkable, they do not take into account the internal structure of the molecule and the interactions between atoms. That is why it is interesting to explore if by working with graphs and introducing information to the deep generative models about molecular connectivity and structure, the performance of the models could improve further.

2.3.2 Conformational deep generative models

In the same fashion as in the previous section, we will introduce some relevant previous works in the field of generative models for molecular conformations.

On the one hand, although strings and graph-based generative models have been investigated and even used within industrial applications, they are restricted by a lack of spatial information. For this reason, 3D structure-generating models which are based on deep learning architectures which can model quantum chemical properties [1, 31, 32] have been proposed [33].

On the other hand, normalizing flows [34] have successfully been used to fit underlying probability distributions from a large variety of data. Moreover, they can be used to generate samples from the learned distributions. Therefore, a variety of generative models for molecular conformations based on normalizing flows have been proposed [35, 36, 37]. This last work introduces the Boltzmann Generators framework. Apart from fitting the underlying probability distribution for a set of sample conformations, a method for using domain knowledge from physics during the learning process is introduced; this is the fact that the probability distribution of possible molecular conformations follows the Boltzmann rule. However, this model can only be trained and used on one molecule at a time, meaning that there is no transferability among molecules. This is an specially interesting property from the point of view of *de novo* design since the spatial structure of molecules encodes some of their properties.

In this work we propose an auto-regressive deep generative model based on normalizing flows. The normalizing flows in the model are conditioned on 1) deep learning architectures modelling spatial interactions (i.e. SchNet) and 2) a molecular representation. In this way, our model attempts to address the lack of transferability previously mentioned. Besides, our model is a Boltzmann Generator, incorporating the advantages of the previously described methods.

2.4 Graph-based deep generative models

In this section we will give an overview over graph neural networks and their use for generative purposes in the field of drug discovery.

2.4.1 Graph neural networks: message passing neural networks

Graphs are structures which encode elements (nodes) and the relationships between them (edges). There is an increasing interest in them, as they are useful representation of various systems such as social interactions (e.g. social networks) or physical systems (e.g. interactions between atoms in a molecule). Nonetheless, standard neural networks such as CNNs and RNNs are not capable of taking graph-structured data as input, making it necessary to develop a new kind of neural network.

The idea behind graph neural networks (GNNs) is to combine the ideas of both CNNs (local connections, shared weights and many layers) and *graph embedding* (representing graphs in a low dimensional vector and learning a transformed representation of the nodes) [7]. Message passing neural networks (MPNNs) [38] have been proposed as a unification of different graph neural and convolutional network approaches.

The aim of MPNNs is to learn a node-order (permutation) invariant embedded representation of graphs, g, together with a transformed representation of the node feature vectors at message passing step L, H^L, that takes into account the interactions in the graph.

The forward pass of MPNNs has two different phases: the message passing phase and the readout phase, shown in Figure 2.5

The message passing phase consists in repeating L times the following computation of messages m_i^{l+1} and update of the hidden node states (which are initially set equal to the node features):

$$m_i^{l+1} = \sum_{v_j \in \mathcal{N}(v_i)} M_l(h_i^l, h_j^l, e_{ij}), \tag{2.15}$$

$$h_i^{l+1} = U_l(h_i^l, m_i^{l+1}). \tag{2.16}$$

In the expression above, h_i^l are the hidden states of node i at the message passing step $l \in \{0, 1, .., L-1\}$, $\mathcal{N}(v_i)$ represents the set of vertexes (nodes) which are neighbours of vertex v_i, and e_{ij} represents the edge features of the connection between vertexes i and j. We will denote the set of initial hidden states $\{h_0^0, h_1^0, ..., h_{N-1}^0\}$ as H^0 and the set of final hidden states, as H^L. It is important to highlight this update rule follows the idea of shared parameters and taking into account local connections. Moreover, the message function M_l and, especially, the update function U_l are the ones which will determine the type of GNN.

The readout phase consists of two MLP (multilayer perceptrons); these are in charge of computing the embedding and the attention weights, respectively. They take as input the initial and the final hidden node states H^0 and H^L, and together return the node-order invariant embedding g of the input graph.

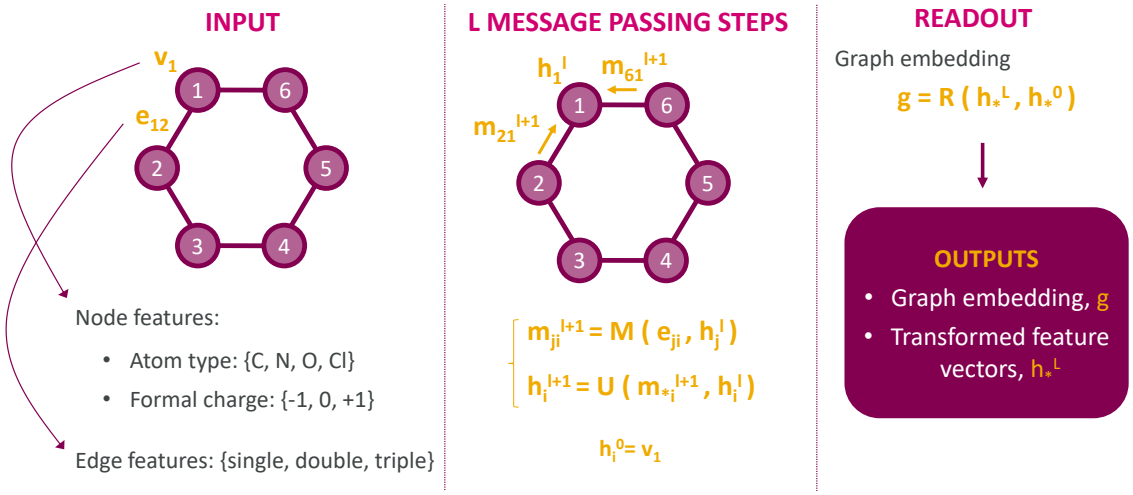


Figure 2.5: General schematic of the message passing neural network. Examples of node and edge features are given for the case where we identify the graph with a molecule. In the message passing phase, only the arriving messages for the first node are pictured. Nonetheless, messages must be computed for every node. A '$*$' as a subscript means the set constituted by the variables corresponding to all possible indices replacing it ($h_*^L \equiv H^L \equiv \{h_0^L, h_1^L, ...\}$).

As a summary, the general architecture of a GNN consists of L propagation blocks which make use of a non-linear propagation rule

$$H^{l+1} = f_{prop}(H^l, E) \quad \forall\, l \in L. \tag{2.17}$$

Where H^l are the hidden node states at block l and H^0 is initialised to V. This propagation block is followed by a readout block which calculates a node-order invariant representation of the molecular graph, g, which can later be used for various tasks, such as property prediction among others.

$$g = f_{readout}(H^L, H^0) \quad \forall\, l \in L. \tag{2.18}$$

The outputs of the GNN block are the transformed node feature vectors H^L and the node-order invariant embedding g.

2.4.2 Gated graph neural networks

Gated graph neural networks (GGNNs) are characterised by using gated recurrent units (GRUs) as the update function (Equation 2.16) of the message passing phase of MPNNs. By using GRUs, the message passing phase becomes more robust. Their use prevents the short-term memory problem, which will otherwise cause the model to forget both the input features and things learnt in the first message passing steps. In the case of graph neural networks, the number of message passing steps L will typically not be large. Because of this feature, GRUs are preferred in this setting when compared to LSTMs (which are more complex and usually exhibit better performance for longer sequences/more message passing steps).

GRUs make use of three additional variables for updating the hidden node state h_i^l to h_i^{l+1}: the update gate vector z_i^{l+1}, the reset gate vector r_i^{l+1} and the candidate hidden node state $\hat{h}_i^{l+1}$. The update rule is the following:

$$z_i^{l+1} = \sigma\left(\mathbb{W}_z^l m_i^{l+1} + \mathbb{U}_z^l h_i^l + b_z^l\right), \tag{2.19}$$

$$r_i^{l+1} = \sigma\left(\mathbb{W}_r^l m_i^{l+1} + \mathbb{U}_r^l h_i^l + b_r^l\right), \tag{2.20}$$

$$\hat{h}_i^{l+1} = \tanh\left(\mathbb{W}_h^l m_i^{l+1} + \mathbb{U}_h^l (r_i^{l+1} \odot h_i^l) + b_h^l\right), \tag{2.21}$$

$$h_i^{l+1} = (1 - z_i^{l+1}) \odot h_i^l + z_i^{l+1} \odot \hat{h}_i^{l+1}. \tag{2.22}$$

Where $\mathbb{W}_*^*$, $\mathbb{U}_*^*$ and b_*^* are learnable parameters by the model. A '$*$' as a subscript means the set constituted by the variables corresponding to all possible indices replacing it. On the one hand, we can observe the update gate vector will determine the contribution of the previous and the candidate hidden node states to the new one. On the other hand, the reset gate vector has the power to determine how much the previous hidden node state contributes to the computation of the candidate one.

2.4.3 Graph-based molecular deep generative model: GraphINVENT

Graph-based deep generative models generate new graphs by iteratively adding nodes and edges to incomplete subgraphs. This is done by sampling from learned distributions of possible actions (referred to as APDs, action probability distributions). Examples of possible actions include adding a new node to a graph, connecting existing nodes in a graph, or terminating a graph under generation. GraphINVENT [2] is a platform for training deep generative models directly on the molecular graph representation and we will take it as a base for our reinforcement learning model. The architecture of the model is shown in Figure 2.6 and is constructed by a GNN block and a global readout block.

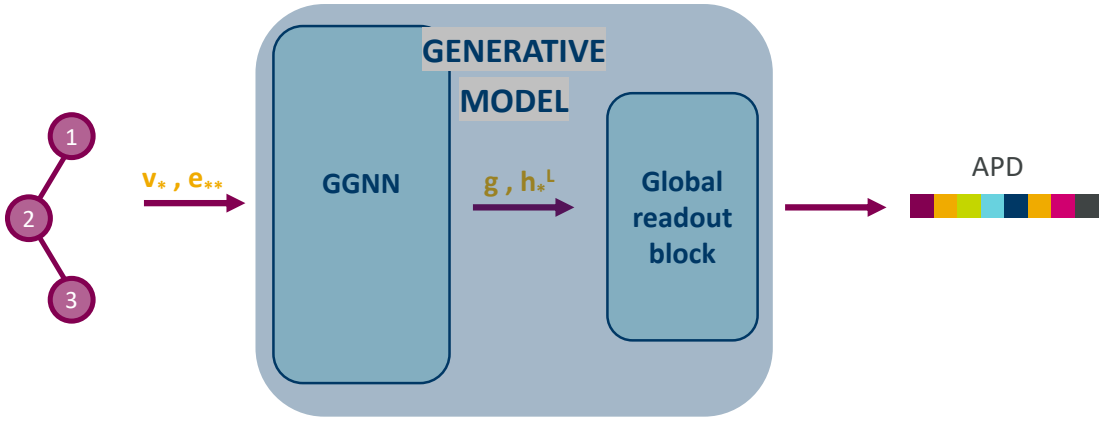


Figure 2.6: General schematic of a graph deep generative model such as GraphINVENT [2]. A single molecule is shown but in practice mini-batches are used in training and generation.

2.4.3.1 GNN block

The GNN block can consist of any type of MPNN. However, here we will focus only on the model that uses a GGNN block, since that was the one which showed the best performance over all the different ones analysed in [2]. The GGNN block used here is as the one described in Section 2.4.2, with the simplification that in this case $M_l = M$ and $U_l = U$, meaning that $\mathbb{W}_*^l = \mathbb{W}_*$, $\mathbb{U}_*^l = \mathbb{U}_*$ and $b_*^l = b_*$; in other words, weights are shared between message passing 'layers'. This makes sense in our setting, where it is reasonable that the weights are the same for all message passing steps, in the same way as they are constant in time in RNNs. Additionally, the number of learnable parameters of the model is considerably reduced, helping to avoid overfitting. Moreover, we will denote as edge neural network (ENN) the network that computes the transformation M to compute the messages. These messages will depend only on the hidden features of the 'sending' node and on the type of edge. In order to take into account the type of edge, there will be one different ENN for each type of bond possible.

The input to the GNN block is an adjacency tensor E and node features matrix V. E uses one-hot encoding for the bond type (Single, Double, Triple or Aromatic) and V makes use of the one-hot encoding for representing the different features such as atom type ({C, N, O, S, Cl, ...}), formal charge ({-1, 0, +1}), implicit H ({0, 1, 2, 3, ...}) and chirality ({None, S, R}). We illustrate this notation via the example shown in Figure 2.7, where the E and V tensors for the sets of bond types and features described will be

$$E = \left[\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \right], \tag{2.23}$$

$$V = \left[\begin{array}{ccccc|ccc|cccc|ccc} 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{array}\right]. \tag{2.24}$$

The vertical lines in V represent the separation between the one-hot encodings of the different features. It must be noted these tensors will be padded with zeros to match the dimensions of those of the largest molecule. The resulting dimensions will then be: $\dim(E) = [n_{\text{nodes max}}, n_{\text{nodes max}}, n_{\text{edge feat.}}]$ and $\dim(V) = [n_{\text{nodes max}}, n_{\text{node feat.}}]$.

Figure 2.7: Example of the chemical graph of formic acid: the nodes are atoms and the edges represent the chemical bonds. When node labels are omitted a carbon atom should be assumed. Note that there is also an implicit hydrogen (not shown) on carbon 2. Figure taken from [2].

The outputs of the GNN block are H^L, the set of final hidden node states, and g, the node-order invariant graph embedding.

2.4.3.2 Global readout block

The global readout block is the one in charge of computing a global property of the graph using H^L and g. It consists of several MLP blocks in charge of computing the different components of the global property desired to be determined. In our case, the global property we are interested in is the action probability distribution (APD). The APD is a vector containing probabilities for all possible actions for growing a graph and it is made up of three components: f_{add}, f_{conn} and f_{term}.

$$\text{APD} = \text{SOFTMAX}(f_{add}, f_{conn}, f_{term}). \tag{2.25}$$

In f_{add}, the probabilities of all different possible ways of adding a node are contained, taking into account to which the new node will be attached, the node to connect to, the new atom type, the new formal charge, the bond type, etc. In f_{conn}, the probabilities of connecting the last appended node are contained, taking into account the node to connect to and the type of the new bond. Finally, in f_{term} we find the probability of terminating the subgraph.

2.4.3.3 Training of GraphINVENT

In order to train GraphINVENT we need to generate the target APDs for each step (subgraph) in the construction of the molecule. This is done during preprocessing, when the target APDs are computed for all subgraphs in the training data using a

graph-decoding route as introduced in [39].

The training loss is defined as the Kullback-Leibler (KL) divergence between the target APD and the predicted APD as shown in Figure 2.8.

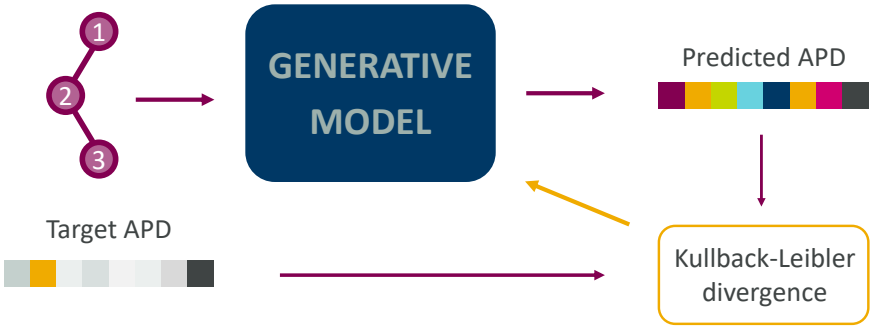


Figure 2.8: The training data consists of subgraphs and target APDs. Subgraphs are given as input to the model, which returns a predicted APD. Training is done by updating the model parameters so that the KL divergence between the target and predicted APD is minimised.

2.4.3.4 Evaluation of GraphINVENT

The model is evaluated by computing following metrics:

- PV: the percentage of valid molecules
- PU: the percentage of unique molecules (when generating an specified molecules)
- PPT: the percentage of properly terminated molecules by sampling the terminate action
- PVPT: the percentage of valid molecules in the set PPT
- $\mathcal{V}_{av}$: the average number of nodes per graph
- ε_{av}: the average number of edges per graph

2.4.3.5 Generation using GraphINVENT

Given a graph G, the action $a_s : G_s \rightarrow G_{s+1}$ takes us from subgraph s to subgraph $s+1$, and is obtained by sampling $p(H_L, g \mid G_s)$, where higher probability actions will lead to more favourable molecules if a model is well trained.

Generation of new molecules starts with an empty graph which is given as input to the model so that it returns an APD, from which an action is sampled and then applied to generate a new graph. This new graph is given again as input to the model to repeat this process until an invalid action is sampled (such as connecting two nodes that are already connected) or until the 'terminate' action is sampled. The generation loop is shown in Figure 2.9.

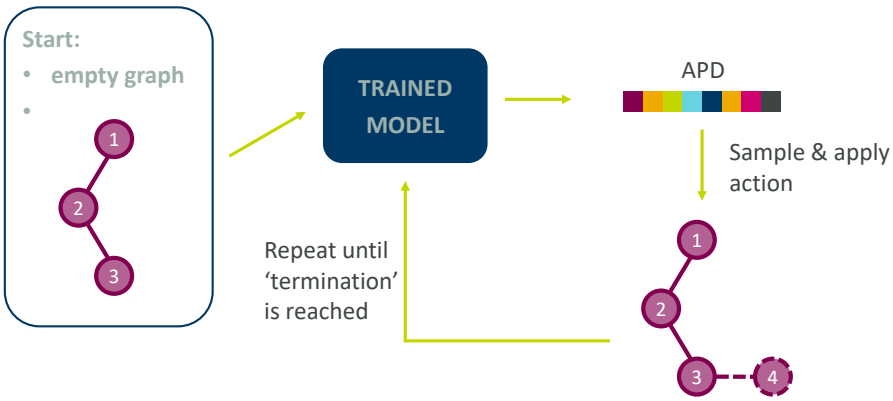


Figure 2.9: Schematic of the generation process.

2.5 Reinforcement learning with graph-based generative models

The goal of reinforcement learning (RL) is to learn which action to take at each possible scenario so that the return (discounted sum of reward signals after every action taken) is maximised [40]. The main characteristic of reinforcement learning is that the agent ('learner') must discover which actions are best to take by trial-and-error.

Reinforcement learning can be used to fine-tune pretrained models for a specific task. More specifically, we will focus on fine-tuning pretrained generative models so that they can learn to generate molecules with some set of desired properties (*de novo* design).

The aim of this section is to explain how the RL methods that we will use work.

2.5.1 Policy gradient methods

Policy gradient methods learn a parameterised policy $\pi(a|s,\theta) = \Pr\{A_t = a|S_t = s, \theta_\mathbf{t} = \theta\}$, where A_t, S_t and θ_t are the action taken, the state visited and the policy parameters at time t respectively. We will now consider methods for learning the policy given a scalar performance measure $G(\theta)$, which the agent aims to maximise. The policy parameters θ are updated so that the loss $J(\theta) = -G(\theta)$ is minimised.

The policy can be parameterised in any way, with the only requirement that $\nabla_\theta \pi(a|s,\theta)$ exists and is finite for all possible states, actions and values of the parameters. This means a neural network can be used to parameterise the policy, and that is what we are going to use.

2.5.2 REINVENT

REINVENT [4] is an AI tool for *de novo* design. It generates targeted chemical compounds by using a SMILES-based generative model together with a RL framework based on [41]. A REINFORCE-type [42] algorithm is used for training.

The goal in REINFORCE is to update the agent policy π_{agent} from the prior policy π_{prior} in a way as to increase the expected score for the action sequences used to build a graph. The loss function proposed in REINVENT is

$$J(\theta) = \left[\log P(A)_{\mathbb{A}} - \log P(A)_{\mathbb{U}}\right]^2 . \tag{2.26}$$

Which is equivalent to using a final step reward of

$$q(t) = \frac{\left[\log P(A)_{\mathbb{A}} - \log P(A)_{\mathbb{U}}\right]^2}{\log P(A)_{\mathbb{A}}}. \tag{2.27}$$

In the loss function, $A = \{a_0, a_1, ..., a_T\}$ is the sequence of actions taken to build a graph G_T such that the model likelihood of the sequence is given by $P(A) = \Pi_{t=0}^{T} \pi(a_t|G_t)$. Further, $\log P(A)_{\mathbb{A}}$ is the agent log-likelihood and $\log P(A)_{\mathbb{U}} = \log P(A)_{prior} + \sigma S(A)$ is the 'augmented' log-likelihood. The prior policy is modulated by the desirability of the sequence of actions in the augmented log-likelihood as the prior policy learns to generate valid molecular graphs based on the training set, while the scoring function guides the policy towards desirable graphs. $S(A) \in [0, 1]$ is a scoring function that rates the desirability of the sequences formed based on the given target properties.

The formulation of this more elaborate loss function (instead of a simpler version only dependent on the score) is motivated by the observation that otherwise the agent would forget what the prior model had learned after a few learning steps. By including the prior log-likelihood we make sure this does not happen. And, finally, by tuning the scale parameter σ, the influence of the score can be set.

The learning process is schematised in Figure 2.10. Both the agent and the prior model are initialised to the chosen pretrained generative model. Then the RL loop is entered, where the agent is then used to generate a batch of molecules (with their associated agent log-likelihoods). Those molecules are next scored and their prior log-likelihood is computed to construct the augmented log-likelihood. After that, the mean squared error (MSE) over all generated molecules between the agent and the augmented log-likelihood is computed. Finally, the agent parameters are updated so that the mean squared error is minimised and a learning step is completed. This process is repeated until the performance of the agent is satisfactory.

2.6 Properties of molecules

There are many attributes which can be measured and properties that can be estimated for each molecule. They will help us understand the applications for which a particular compound can be well-suited.

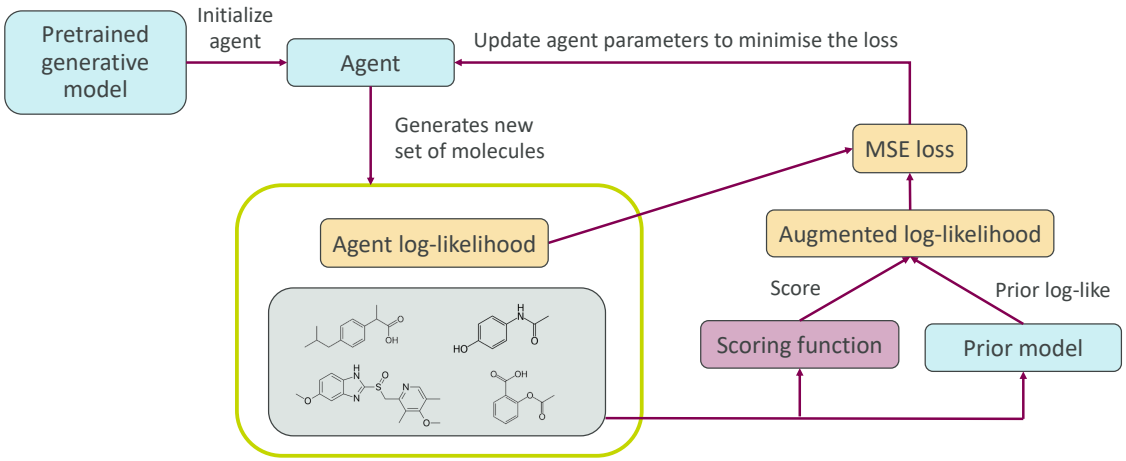


Figure 2.10: Schematic of the RL loop.

2.6.1 Drug-likeness: Lipinski's rule of five

Drug-likeness is a qualitative property which aims to determine whether a certain molecule can be a good drug or not and becomes very relevant during the early stages of drug discovery. In order to answer this question, different properties are inspected. This can be done by checking compliance to Lipinski's rule of five [43].

Lipinski's rule of five tries to determine if a certain compound has the 'right' chemical and physical properties by comparing them to those observed in most *orally* administered drugs. It focuses on the properties relevant for the pharmacokinetics in the human body such as solubility (measured through the octanol-water partition coefficient, also known as logP) or diffusion (directly related to its molecular weight). More specifically, the rule states that, in general, a 'good' oral drug has no more than one violation of the following statements:

- No more than 5 hydrogen bond donors.
- No more than 10 hydrogen bond acceptors.
- A molecular mass of less than 500 Daltons.
- An octanol-water partition coefficient that is smaller than 5.

The name of the rule comes from all the numbers in these constraints being multiples of 5. Additionally, we would like to emphasise that Lipinski's rule of five does not apply to drugs administered by non-oral routes.

As any other rule of thumb, Lipinski's one has many exceptions, variants and extensions. For example, the Ghose filter [44] checks for violations to the following criteria:

- Partition coefficient (logP) in the range -0.4 to $+5.6$.
- Molar refractivity from 40 to 130 $\mathrm{m}^3 \cdot \mathrm{mol}^{-1}$.
- A molecular mass of less from 180 to 480 Daltons.
- 20 to 70 atoms.

2.6.2 Quantitative estimate of drug-likeness (QED)

The QED [45] is a measure of drug-likeness based on the concept of desirability. It is achieved by computing the geometric mean over the desirabilities of different properties:

$$\text{QED} = \exp\left[\frac{1}{n}\sum_{i=1}^{n}\ln d_i\right]. \tag{2.28}$$

The properties taken into account in the estimate are chosen based on their relevance to estimate drug-likeness and to influence the likelihood of attrition. There are eight of them: molecular weight (MW), octanol–water partition coefficient (ALOGP), number of hydrogen bond donors (HBDs), number of hydrogen bond acceptors (HBAs), molecular polar surface area (PSA), number of rotatable bonds (ROTBs), number of aromatic rings (AROMs) and number of structural alerts (ALERTS).

The desirability functions for each property are derived empirically by fitting to asymmetric double sigmoidal (ADS) functions the distribution of values for each property in a set of approved drugs, i.e., a more refined version of Lipinski's rule.

2.6.3 Biological activity

The biological or pharmacological activity of a drug is a property that indicates the possible medical applications of a compound. Quantitative structure–activity relationship (QSAR) models are regression or classification models used to relate a set of compounds to their corresponding biological activities. An example of activity that can be estimated using these models is the DRD2 activity, as done in [46]. The DRD2 gene in humans encodes the dopamine receptor (D_2) protein. Some examples of drugs with high DRD2 activity are bromocriptine, haloperidol and lysergic acid diethylamide (LSD).

2.7 Normalizing flows

Normalizing flows [34] provide a general way of constructing flexible probability distributions over continuous random variables. They operate by pushing an initial density through a series of transformations to produce a richer, more multi-modal distribution. The main idea of flow-based modeling is to express the distribution of a random variable $\boldsymbol{x}$ through a transformation F_{zx} of another random variable $\boldsymbol{z}$ sampled from a *base distribution* $p_z(\boldsymbol{z})$[1]:

$$\boldsymbol{x} = F_{zx}(\boldsymbol{z}), \text{ where } \boldsymbol{z} \sim p_z(\boldsymbol{z}). \tag{2.29}$$

The *base distribution* $p_z(\boldsymbol{z})$ is typically chosen to enable computationally effective i.i.d. sampling. The defining property of flow-based models is that the transformation F_{zx} must be *invertible* and both F_{zx} and $F_{xz} = F_{zx}^{-1}$ must be *differentiable* (and

[1] $\boldsymbol{z}$ is also often referred to as the *latent variable* and therefore the space it lives in as the *latent space*. We will use the two notations interchangeably here. However, there is debate as to whether the latter term is not as appropriate here as it is in latent-variable models.

therefore F_{zx} is *composable*). Under these conditions, the density of $\boldsymbol{x}$ is well-defined and can be obtained by a change of variables:

$$p_x(\boldsymbol{x}) = p_z(\boldsymbol{z})|R_{zx}(\boldsymbol{z})|^{-1}, \tag{2.30}$$

where $\boldsymbol{z} = F_{xz}^{-1}(\boldsymbol{x})$ and $R_{zx}(\boldsymbol{z}) = \det J_{F_{zx}}$ is the determinant of the Jacobian of F_{zx}. Equivalently, we can also write $p_x(\boldsymbol{x})$ in terms of the Jacobian of F_{xz} as

$$p_x(\boldsymbol{x}) = p_z(F_{xz}(\boldsymbol{x}))|R_{xz}(\boldsymbol{x})|. \tag{2.31}$$

In practice, we typically construct a flow-based model by implementing F_{zx} (or F_{xz}^{-1}) with a neural network and taking $p_z(\boldsymbol{z})$ to be a simple density such as a multivariate normal. Usually, several transformations are nested as the repeated application of, even simple, transformations to a uni-modal initial density leads to models of exquisite complexity. Moreover, in the recent years the inclusion of stochastic fluctuations [47] to these kinds of models has been an active area of research, leading to an important improvement in performance. However, due to time constraints on this project, we choose a simpler approach here, the Real Non-Volume Preserving (NVP) [48].

2.7.1 Real NVP and fully masked inputs

Now that the general ideas behind normalizing flows have been introduced, in this section we describe the actual implementations used in this project. In particular, we used a flow constituting of Real NVP [48] transformations as the trainable invertible network. The idea behind these transformations is to separate the inputs of a layer into two groups, $\boldsymbol{z} = (\boldsymbol{z}_1, \boldsymbol{z}_2)$, one that is transformed ($\boldsymbol{z}_2$) and another that is not ($\boldsymbol{z}_1$). The transformations performed on $\boldsymbol{z}$ have the shape

$$f_{zx}(\boldsymbol{z}_1, \boldsymbol{z}_2) = \begin{cases} \boldsymbol{x}_1 = \boldsymbol{z}_1 \\ \boldsymbol{x}_2 = \boldsymbol{z}_2 \odot \exp\left(S(\boldsymbol{z}_1; \boldsymbol{\theta})\right) + T(\boldsymbol{z}_1; \boldsymbol{\theta}) \end{cases}, \tag{2.32}$$

and therefore the Jacobian log-determinant of the transformation is

$$\log R_{zx} = \sum_i S_i(\boldsymbol{z}_1; \boldsymbol{\theta}), \tag{2.33}$$

where $\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2)$ is the output of the layer, $T(\boldsymbol{x}_1; \boldsymbol{\theta})$ is the translation network and $S(\boldsymbol{x}_1; \boldsymbol{\theta})$ is the scaling network. Both T and S are arbitrary non-invertible neural networks. One simple, practical and efficient way to construct these transformations is to use masking [49]: multiplying each weight matrix element-wise with a binary matrix of the same size. With this technique we get to 'mask out' undesired connections while keeping the rest unaltered. We can then trivially invert this kind of transformations as

$$f_{xz}(\boldsymbol{x}_1, \boldsymbol{x}_2) = \begin{cases} \boldsymbol{z}_1 = \boldsymbol{x}_1 \\ \boldsymbol{z}_2 = (\boldsymbol{x}_2 - T(\boldsymbol{x}_1; \boldsymbol{\theta})) \odot \exp\left(-S(\boldsymbol{x}_1; \boldsymbol{\theta})\right) \end{cases}, \tag{2.34}$$

with Jacobian log-determinant defined as

$$\log R_{xz} = -\sum_i S_i(\boldsymbol{x}_1; \boldsymbol{\theta}). \tag{2.35}$$

In this project we are generally generating 3-dimensional (3D) outputs. One way of doing this is using blocks of three of the previously described layers, in which the different coordinates z_i are transformed in different layers. This way, every coordinate is transformed on each of these blocks while maintaining invertibility. Using masking, we can incorporate to the flow inputs that are used for transformations but never transformed: we call these kinds of inputs fully-masked inputs. This magnitudes can be very useful if we want to condition a certain transformation to some context representation $\boldsymbol{C}$.

Putting all this information together, a transformation of the i^{th} coordinate with context representation $\boldsymbol{C}$ can be formulated as

$$f_{zx}(\boldsymbol{z}_{\neq i}, z_i) = \begin{cases} \boldsymbol{x}_{\neq i} = \boldsymbol{z}_{\neq i} \\ x_i = x_i * \exp\left(S(\boldsymbol{z}_{\neq i}|\boldsymbol{C}; \boldsymbol{\theta})\right) + T(\boldsymbol{z}_{\neq i}|\boldsymbol{C}; \boldsymbol{\theta}) \end{cases}, \tag{2.36}$$

with Jacobian log-determinant

$$\log R_{zx} = S(\boldsymbol{z}_{\neq i}|\boldsymbol{C}; \boldsymbol{\theta}), \tag{2.37}$$

and where the subindex '$_{\neq i}$' refers to the non-transformed coordinates. The inverse transformation is finally given by

$$f_{xz}(\boldsymbol{x}_{\neq i}, x_i) = \begin{cases} \boldsymbol{z}_{\neq i} = \boldsymbol{x}_{\neq i} \\ z_i = (x_i - T(\boldsymbol{x}_{\neq i}|\boldsymbol{C}; \boldsymbol{\theta})) * \exp\left(-S(\boldsymbol{x}_{\neq i}|\boldsymbol{C}; \boldsymbol{\theta})\right) \end{cases}, \tag{2.38}$$

with Jacobian log-determinant

$$\log R_{zx} = -S(\boldsymbol{x}_{\neq i}|\boldsymbol{C}; \boldsymbol{\theta}). \tag{2.39}$$

2.8 Likelihood- and energy-based learning

In usual industrial machine learning applications, data is crucial: huge amounts of data are necessary to properly train models. Moreover, more often than not, data is the only 'true' source of information for the learning system. One of the main advantages of working with molecular conformations is that we have physical theories we can use as domain knowledge about the system. In particular, we know from statistical mechanics that, the probability of observing a microscopic configuration, $\boldsymbol{X}$, is proportional to the Boltzmann factor $\exp(-\beta U(\boldsymbol{X})) = \exp(-u(\boldsymbol{X}))$, where $\beta = 1/k_b T$ is the so called inverse temperature (k_b is the Boltzmann constant and T is the temperature), $U(\boldsymbol{X})$ is the potential energy of the configuration and $u(\boldsymbol{X})$ is the dimensionless potential energy. In this project, we take advantage of this fact by training our conformational generative model in such a way that it learns from both examples of conformations and statistical mechanics. We do this by following

the strategy introduced in Boltzmann Generators [37].

Learning from energy can be thought as 'teaching' the model the rules of physics that are relevant to the generation process. In this case, we do this by minimising the Kullback Leibler (KL) divergence of the model distribution, $p_x(\boldsymbol{X};\boldsymbol{\theta})$, and the Boltzmann distribution, $B(\boldsymbol{X}) = \frac{1}{Z_B}\exp(-u(\boldsymbol{X}))$ (where Z_B is the partition function). Therefore, our initial loss function, $J_U(\boldsymbol{\theta})$, is defined as

$$\begin{aligned} J_U(\boldsymbol{\theta}) &= D_{KL}\left[p_x(\boldsymbol{X};\boldsymbol{\theta})||B(\boldsymbol{X})\right] = \mathbb{E}_{p_x(\boldsymbol{X})}\left[\log\frac{p_x(\boldsymbol{X};\boldsymbol{\theta})}{B(\boldsymbol{X})}\right] = \\ &= \mathbb{E}_{p_z(\boldsymbol{Z})}\left[\log p_z(\boldsymbol{Z}) - \log\|R_{zx}(\boldsymbol{Z};\boldsymbol{\theta})\| + \log Z_B + u(F_{zx}(\boldsymbol{Z};\boldsymbol{\theta}))\right], \end{aligned}$$

where $\boldsymbol{Z}$ is the (vertical) concatenation of the latent vectors generating the conformation $\boldsymbol{X}$. If we now drop the terms that do not depend on $\boldsymbol{\theta}$, we get a simpler version, $L_U(\boldsymbol{\theta})$, of the loss function

$$L_U(\boldsymbol{\theta}) = \mathbb{E}_{p_z(\boldsymbol{Z})}\left[u(F_{zx}(\boldsymbol{Z};\boldsymbol{\theta})) - \log\|R_{zx}(\boldsymbol{Z};\boldsymbol{\theta})\|\right]. \tag{2.40}$$

The term $\log\|R_{zx}(\boldsymbol{Z};\boldsymbol{\theta})\| = \sum_{i=1}^{N}\log\|R_{zx}(\boldsymbol{z}_i;\boldsymbol{\theta})\|$ for an auto-regressive model. In other words, if at every generation step, the position of the previously placed atoms is considered as 'known'. Formally, this equality holds if

$$p_x(\boldsymbol{X};\boldsymbol{\theta}) = p_x(\boldsymbol{x}_0;\boldsymbol{\theta})\prod_{i=1}^{N-1} p_x(\boldsymbol{x}_i|\boldsymbol{x}_{<i};\boldsymbol{\theta}). \tag{2.41}$$

Finally, given a set of latent matrices $\{\boldsymbol{Z}_i\}_{i=1}^{N_{conf}}$ generating some molecular conformations $\{\boldsymbol{X}_i = F_{zx}(\boldsymbol{Z}_i)\}_{i=1}^{N_{conf}}$, the loss contribution associated to energy-based learning can be estimated by Monte Carlo sampling as

$$L_U(\boldsymbol{\theta}) \approx \frac{1}{N_{conf}}\sum_{i=1}^{N_{conf}} u(F_{zx}(\boldsymbol{Z}_i;\boldsymbol{\theta})) - \log R_{zx}(\boldsymbol{Z}_i;\boldsymbol{\theta}) \tag{2.42}$$

As mentioned previously, we want our models also to be able to learn from examples of conformations computed or measured by other methods, such as experimentally-measured conformations or conformational states from high precision simulations [50]. Moreover, it has been reported that models only trained on energy find difficulties in exploring very different states [37]. For all of these reasons, we introduce likelihood-based learning. To do so, we add a contribution to the loss, $J_{ML}(\boldsymbol{\theta})$, corresponding to learning from these examples, computed in the same (standard) way as it is done in flow-based models [34], and corresponding to the KL divergence between the data distribution, $p_x^{\text{data}}(\boldsymbol{x})$, and the model distribution, $p_x(\boldsymbol{x};\boldsymbol{\theta})$:

$$\begin{aligned} J_{ML}(\boldsymbol{\theta}) &= D_{KL}\left[p_x^{\text{data}}(\boldsymbol{x})||p_x(\boldsymbol{x};\boldsymbol{\theta})\right] = \mathbb{E}_{p_x^{\text{data}}(\boldsymbol{x})}\left[\log\frac{p_x^{\text{data}}(\boldsymbol{x})}{p_x(\boldsymbol{x};\boldsymbol{\theta})}\right] \\ &= -\mathbb{E}_{p_x^{\text{data}}(\boldsymbol{x})}\left[\log p_x(\boldsymbol{x};\boldsymbol{\theta})\right] + \text{constant} \\ &= -\mathbb{E}_{p_x^{\text{data}}(\boldsymbol{x})}\left[\log p_z(F_{xz}(\boldsymbol{x};\boldsymbol{\theta})) + \log\|R_{xz}(\boldsymbol{x};\boldsymbol{\theta})\|\right] + \text{constant}. \end{aligned}$$

In the expression above, the constant terms denote magnitudes not dependent on $\boldsymbol{\theta}$. In general, we cannot evaluate this expression exactly since we would need an infinite amount of data or a tractable expression for $p_x^{\text{data}}(\boldsymbol{x})$. However, given a collection of samples $\{\boldsymbol{x}_i\}_{i=1}^{N_{samples}}$, one can estimate the expectation over $p_x^{\text{data}}(\boldsymbol{x})$ using Monte Carlo as follows

$$J_{ML}(\boldsymbol{\theta}) \approx -\frac{1}{N_{samples}} \sum_{i=1}^{N_{samples}} \log p_z(F_{xz}(\boldsymbol{x}_i;\boldsymbol{\theta})) + \log\|R_{xz}(\boldsymbol{x}_i;\boldsymbol{\theta})\| + \text{constant}. \quad (2.43)$$

Furthermore, if we choose a standard multivariate Gaussian distribution and drop constant terms, L_{ML} reduces to

$$L_{ML}(\boldsymbol{\theta}) = \mathbb{E}_{p_x^{\text{data}}(\boldsymbol{x})}\left[\frac{1}{2}\|F_{xz}(\boldsymbol{x};\boldsymbol{\theta}))\|^2 - \log\|R_{xz}(\boldsymbol{x};\boldsymbol{\theta})\|\right], \quad (2.44)$$

and can be estimated by

$$L_{ML}(\boldsymbol{\theta}) \approx \frac{1}{N_{samples}} \sum_{i=1}^{N_{samples}} \frac{1}{2}\|F_{xz}(\boldsymbol{x}_i;\boldsymbol{\theta}))\|^2 - \log\|R_{xz}(\boldsymbol{x}_i;\boldsymbol{\theta})\|. \quad (2.45)$$

Please note that in this context, one sample is a set of coordinates for each atom in the molecule. Minimising the Monte Carlo estimate of the KL divergence is equivalent to maximising the likelihood of the model given the set of samples. That is why we call this method likelihood-based learning.

As usual, expressions 2.42 and 2.45 can be minimised using standard optimisation methods such stochastic gradient descent. Moreover, the minimisation can be performed on subsets (or batches) of the dataset, which is helpful for avoiding memory constraints and preventing the algorithm from getting stuck at local minima.

3
Methods

3.1 Reinforcement learning with graph-based generative models (1st model)

In this section we will explain in detail the methods used for the model developed to fine-tune graph-based generative models using reinforcement learning.

3.1.1 Model design choices

The model proposed is made up of three main blocks: the graph generative model (GraphINVENT using a GGNN, explained in Section 2.4.3), the reinforcement learning framework (implemented based on Section 2.5.2) and the scoring model (which will change depending on the desired properties).

3.1.1.1 Generative model: GraphINVENT hyperparameters

The GraphINVENT model, as detailed in Section 2.4.3, has two main subparts: the GGNN and the global readout block. The hyperparameter values chosen in both models are those found to work best in the original publication [2].

Taking into account that in the GGNN the message size must be equal to the number of hidden node features, the effective model parameters to choose are then:

- Number of hidden node features (100)
- Graph embedding size (100)
- Hidden size of MLPs-GGNN (250)
- Depth of MLPs-GGNN (4)
- Dropout probability GGNN (0)
- Number of message passes (3)
- Hidden size of MLPs-Global readout (500)
- Depth of MLPs-Global readout (4)
- Dropout probability Global readout(0)

Other parameters which need to be specified in GraphINVENT are those related to the datasets we are going to work with and the features we want to be modelled. We will use a 'simple' version of GraphINVENT in which we will not use aromatic bonds, chirarility or hydrogens (neither explicit nor implicit). Additionally, we will

only use canonical SMILES and allow a maximum number of heavy atoms of 72 (largest number of heavy atoms in the molecules of the dataset of DRD2 actives). The parameter values used in the multi-layer percetrons (MLP) of the GGNN are detailed in Table 3.1. Their weights are initialised from Xavier uniform distributions [51]. The activation functions used are SELUs [52]. And the number of message passes in the message passing phase is also shown in Table 3.1. These parameters relate to the message passing phase and graph readout parameters in the following way:

- Hidden node features = Input features
- Message size = Output features
- Graph embedding size = Output features

Table 3.1: Model hyperparameters used in the multilayer perceptrons (MLPs) and the message passing phase of the gated graph neural network. There are 5 MLPs: 3 MLPs in the edge neural network (one for each type of bond, i.e., single, double, and triple), also referred to as the ENN, which is in charge of computing the messages given some input hidden node features; and 2 MLPs in the graph readout block (one for computing the attention weights and another for computing the embedding for each input feature).

Hyperparameters of the GGNN	
Input features	100
Hidden features	250
Output features	100
Depth	4
Dropout probability	0
Message passes	3

The parameter values chosen in the MLPs of the global readout block are detailed in Table 3.2. Again, weights are initialised from an Xavier normal distribution and the activation functions are SELUs. The remaining parameters of the MLPs are chosen as needed to encode all necessary information for the probabilities of adding an atom, connecting two nodes or terminating a graph. These sizes depend on another hyperparameter: the maximum number of nodes. As we have said, this value depends on the application and the dataset used, and it is 72 in our case.

Table 3.2: Hyperparameters used in the MLPs of the global readout block. There are 5 of these MLPs: 2 in the first tier in charge of computing provisional probabilities for adding and connecting nodes; and 3 in the second tier, where the final components of the APD for adding, connecting and terminating are computed.

MLPs of the Global Readout Block	
Hidden features	500
Depth	4
Dropout probability	0

3.1.1.2 Reinforcement learning framework

We can frame the problem of generating targeted molecular graphs as a Markov decision process, where the agent must make a decision on the best action to choose next given the current graph. As such, we propose to use a REINFORCE-type algorithm introduced in [41] for training a GGNN-based molecular generative model. As previously explained, the goal in REINFORCE is to update the agent policy π from the prior policy π_{prior} in a way as to increase the expected score for the action sequences used to build a graph.

We have based our implementation on the one explained in Section 2.5.2. However, we have also introduced some changes to improve performance in our models. The most notable changes introduced are related to the loss function, where we propose to use a new term which both helps to stabilise learning and to achieve better results. Our new definition of the loss function, Remember Best Agent (BAR) loss, is

$$\begin{aligned} J(\theta) = (1-\alpha) \cdot [\log P(A)_{\mathbb{A}} - (\log P(A)_{\text{Prior}} + \sigma S(A))]^2 \\ + \alpha \cdot [\log P(A')_{\mathbb{A}} - (\log P(A')_{\text{Best } \mathbb{A}} + \sigma S(A'))]^2 . \end{aligned} \tag{3.1}$$

Above, α is a new hyperparameter, which we generally have found to work best when set to 0.5. A' refers to the set of actions taken by the best agent, 'Best $\mathbb{A}$', to generate a set of molecules. The best agent is initialised to the prior model and updated to copy the agent parameters every 5 learning steps if the score obtained by the agent for 1000 molecules at that step is the largest so far.[1] With this new definition of the loss function, we prevent the model from stopping generating highly scored molecules because it will always learn from molecules generated by the best agent. And, if the agent was to assign low likelihoods to the set of actions taken to build those highly scored molecules, it would learn once again how to increase their likelihood. Moreover, this loss also facilitates learning in situations in which few molecules are highly scored, as it will be the case when searching for active molecules. Nonetheless, this new definition will also lead to less diversity in the generated molecules, i.e., a lower percentage of unique molecules.

It must be noted we disregard non-unique molecules in a batch of generated molecules when computing the loss so that we do not update twice in the same direction and push the model towards the generation of duplicate molecules.

The optimal value of σ will depend on our purposes, but it will usually be in the range from 0.1 to 50. The larger its value, the larger the change in likelihood compared to that of the molecules generated initially and, therefore, the larger the change in the molecular properties we are tuning (e.g. the average number of atoms). However, these changes also lead to higher-scored molecules. As such, the value of σ should be chosen depending on what we value most: maintaining similarity to the prior or increasing the score.

[1] The number of molecules is chosen as a trade-off between speed and enough sampling

3.1.1.3 Scoring model

The scoring model can be as complicated as desired and designed for each particular and specific purpose. The scoring function used will be dependent on the application towards which we want to fine-tune our generative model. We must note that we will always assign a score of 0 to non-valid, non-properly terminated or non-unique molecules, as they are not good examples for our model. Moreover, the scoring function will always have a lower bound of 0 and an upper bound of 1, i.e. $S(A) \in [0, 1]$.

We have implemented different versions of the scoring function. First, we implemented some simple scoring functions which aimed to change the average size of the molecules (increase or decrease their size). These were used to check if the reinforcement learning framework was working correctly. Then, we implemented a score which promoted drug-likeness in molecules. Finally, we implemented a score for promoting DRD2 activity as the desired property in the generated molecules.

3.1.1.3.1 Reducing and increasing the average size of the molecules. To start with a 'simple' task, we first tried to shift the distribution of the number of nodes of the generated molecules from around 25 heavy atoms at the beginning a) towards smaller molecules (by defining a scoring function that creates a maximum reward for 10 heavy atoms) and b) towards larger molecules (by defining a scoring function creates a maximum reward for 40 atoms). More specifically, the scoring functions used are

$$S_{\text{reduce}}(A) = \begin{cases} 0 & \text{if not properly terminated or invalid,} \\ 1 - \frac{|n_{nodes} - 10|}{\max_{nodes} - 10} & \text{otherwise;} \end{cases} \tag{3.2}$$

and

$$S_{\text{increase}}(A) = \begin{cases} 0 & \text{if not properly terminated or invalid,} \\ 1 - \frac{|n_{nodes} - 40|}{\max_{nodes} - 40} & \text{otherwise.} \end{cases} \tag{3.3}$$

Where A is the set of actions taken to build the molecule, n_{nodes} is the number of heavy atoms of that molecule and $\max_{nodes}$ is the maximum number of nodes allowed in the model (72 in our case).

3.1.1.3.2 Promoting drug-like molecules. Our next definition for the scoring function is based on the QED [45] implementation from RDKit [16]:

$$S_{\text{QED}}(A) = \begin{cases} 0 & \text{if not properly terminated or invalid,} \\ \text{QED}(\text{SMILES}(A)) & \text{otherwise.} \end{cases} \tag{3.4}$$

Where SMILES(A) refers to the SMILES (see Section 2.1.3.1) of the final generated molecule by the set of actions A.[2]

[2] All generated graphs are converted to canonical SMILES before writing to disk, so as to use less disk-space.

3.1.1.3.3 Promoting DRD2 active molecules. The final, more advanced scoring model aimed to fine-tune our model towards the generation of drug-like DRD2-active molecules. For that purpose, we used both the QED implementation from the previous scoring function and a pretrained DRD2 QSAR model [46] obtained from `https://github.com/pcko1/Deep-Drug-Coder/blob/master/models/qsar_model.pickle`. As the assigned values for both the QED and the activity estimate are not very informative, we will use a threshold of 0.5 (both estimates range from 0 to 1) to either classify molecules as active (QED and activity > 0.5) or inactive (QED or activity < 0.5). If a molecule is classified as active, we will then assign it a score of 1. Otherwise, if the molecule is inactive, the assigned score will be 0.

$$S_{\text{activity}}(A) = \begin{cases} 1 & \text{if properly terminated, valid, QED} > 0.5 \text{ and activity} > 0.5, \\ 0 & \text{otherwise.} \end{cases} \tag{3.5}$$

Using this discrete score means we will only update our model with respect to molecules that fulfil all the constraints and can be thus given a score of 1. Therefore, if not many molecules are active at the beginning, we expect our model to lose ability to generate many different molecules and consequently, we expect the fraction of unique molecules to be reduced while learning.
We note here that this is also an approximation; just because a molecule is valid, has a QED > 0.5, and a predicted activity score > 0.5 does not guarantee it to be active. However, it was a choice that we found to work well in our models.

3.1.2 Training: pretraining and fine-tuning

The training process of our model was carried out in two steps. The first step is a supervised learning phase where the aim is to train the graph generative model so that it learns to propose molecules with similar properties to those in the training set. This is followed by a reinforcement learning phase where the aim is to fine-tune the generative model so that it can learn to generate molecules with targeted properties.

3.1.2.1 Pretraining GraphINVENT

Training of GraphINVENT is done by minimising the Kullback-Leibler (KL) divergence between the target APDs and the predicted ones. Here, the Adam optimiser [53] is used with no weight decay and OneCycleLR [54] learning rate scheduler implemented in PyTorch [55]. In the scheduler, we have used the default parameters but fixing it so as to not increase the learning rate initially by setting the division factor equal to 1 and the maximum learning rate to the desired initial value (i.e. no learning rate 'warm-up'). Furthermore, we take as many steps as epochs and set the fraction of steps for increasing the learning rate to 0.05 (to keep the learning rate constant in our models). Other parameters such as the initial and final learning rates and the batch size must be adjusted for each specific dataset via hyperparameter optimisation. In our setting, we used an initial learning rate of 10^{-4}, a final learning rate of 10^{-7} and a batch size of 1000 subgraphs for optimal training.

3.1.2.1.1 Training data. A subset of ChEMBL 25 [5] was used to train, validate and test our models. This subset is taken from `https://github.com/pcko1/Deep-Drug-Coder/tree/master/datasets` and is preprocessed, as explained in [46], such that only molecules containing {H, C, N, O, F, S, Cl, Br} and less than 50 heavy atoms remain. Moreover, the known DRD2 active molecules were removed so that we can effectively evaluate if the model is able to learn how to generate DRD2 active molecules having seen no previous examples of them. We randomly take $5 \cdot 10^5$ molecules for the training set, $5 \cdot 10^4$ for validation and $5 \cdot 10^4$ for testing.

3.1.2.1.2 Evaluation. The best model is chosen by looking at the evaluation metrics detailed in Section 2.4.3.4. Furthermore, we compare the validation and training losses to avoid overfitting. It must be noted that determining which is the best model is not straightforward. There exists a compromise between the optimisation of percent valid (PV) and percent unique (PU) towards the target value of 100%. This is because the longer we train our model, the greater the percentage of valid molecules generated, but at the expense of a smaller percentage of unique (PU) molecules generated.

3.1.2.2 Fine-tuning GraphINVENT

As we have explained in previous Sections, learning is based on a REINFORCE-type algorithm in which we update the model parameters after each step to minimise the loss function defined in Equation 3.1. We then have two generation steps during learning: one in which the agent generates a batch of molecules, and another in which the best agent so far generates a batch of molecules of the same size. For the first batch, the corresponding agent APD is computed, as well as the APD that the prior model would have assigned. For the second batch, the APD of the best agent is computed, as well as the corresponding APD assigned to the chosen actions by the agent. These APDs are then used for calculating the log-likelihoods needed in the loss function.

The optimiser used here is again PyTorch Adam with no weight decay together with the OneCycle learning rate scheduler (same scheduler settings as before). An initial learning rate of 10^{-4}, together with a final learning rate of 10^{-6}, have been found to work best during RL-based training. A batch size of 64 *molecules* was used in all RL settings, except for models with the scoring function aiming to increase the size of the molecules, for which it was necessary to reduce the batch size to 32. While larger batch sizes lead to less noise, here it was not possible to work with a larger batch size due to memory constraints. However, these batch sizes are comparable to the ones used for training GraphINVENT since we are referring now to full molecules, which have 26 subgraphs per molecule on average; this translates to around 800–1700 subgraphs per batch during RL training (for reference, the batch size used for pretraining GraphINVENT was 1000 subgraphs). Working with full molecules is necessary since the score must be computed for a complete graph.

3.1.2.2.1 Evaluation. As we are not using any dataset in the RL framework, it can be difficult to establish general metrics to evaluate the model apart from the ones defined in Section 2.4.3.4. For this reason, we have defined different scoring functions, such as in Equations 3.2 and 3.3, to test if the framework we have defined is able to make the generative model learn how to increase the score of the generated molecules. Another possibility is to check whether the generated molecules fulfil the desired properties, thus allowing us to assess whether the choice of scoring function is correct for our purposes.
For evaluating our final scoring function (Equation 3.5), we also have a set of known active molecules to compare with. The generative model has not been shown these molecules before and the idea is to check whether the model is able to reproduce molecules from the set of unseen known actives after fine-tuning. More specifically, for this final activity scoring function, we will look at the fraction of reproduced molecules, the fraction of active molecules, the fraction of active and unique molecules, and the average activity score assigned by the QSAR model.

3.1.2.2.2 Active molecules dataset. A DRD2 dataset was downloaded for evaluation of our models; this dataset allows us to check whether our model has learned to generate known active molecules without ever having seen them before. The dataset was downloaded from `https://github.com/pcko1/Deep-Drug-Coder/tree/master/datasets`. The available training, validation and testing splits were combined into a single set. Furthermore, as this dataset also contains inactive compounds, the molecules are filtered so that only those which are scored 1 by the scoring function defined in Equation 3.5 remain. By following these steps, we obtain a set of 3627 active molecules to compare with.

3.1.3 Usage

The steps to follow for generating targeted molecules using the proposed model are:

1. Pretrain GraphINVENT in a supervised learning manner using a general dataset such as ChEMBL [5].
2. Choose a set of desired properties and define a scoring function for that purpose. This score should assign high scores to the set of actions taken to build molecules with the goal attributes.
3. Use the RL framework to fine-tune the pretrained model towards generation of molecules with the desired properties.
4. Analyse the properties of the generated molecules.

To generate molecules with other properties or use a different scoring function, one can go back to step 2 and repeat the process with the same pretrained model.

3.2 Deep auto-regressive generative model for small-molecule configurations (2nd model)

3.2.1 The model

A molecular configuration can be specified by indicating the position of each atom in the molecule M. That is, for a molecule with N atoms, it can be described by $N \in \mathbb{R}^3$ space vectors. We want to generate configurations by sampling the position of each atom $\boldsymbol{x}_i$ in the molecule using a generative auto-regressive model $p_x(\boldsymbol{x}_i|\boldsymbol{x}_{<i};\boldsymbol{\theta})$, where $\boldsymbol{x}_{<i}$ are the positions of the atoms placed before atom i. The model has learnable parameters $\boldsymbol{\theta}$, and the probability of a configuration $\boldsymbol{X}$ is given by

$$p_x(\boldsymbol{X}|\boldsymbol{M};\boldsymbol{\theta}) = p_x(\boldsymbol{x}_1|\boldsymbol{M};\boldsymbol{\theta}) \prod_{i=2}^{N} p_x(\boldsymbol{x}_i|\boldsymbol{x}_{<i},\boldsymbol{M};\boldsymbol{\theta}). \tag{3.6}$$

Furthermore, we encode the relevant information for the model to generate the position of the i^{th} atom in the context vector $\boldsymbol{C}_i$. In particular, it includes a molecular representation $\boldsymbol{M}$, an atomic representation $\boldsymbol{A}_i$ (corresponding to the atom to place), a representation of the previously placed atoms $\mathbf{R}_{<i}$ and information about the local reference frame $\boldsymbol{F}_i$. Formally, $\boldsymbol{C}_i$ is the concatenation of the previously enumerated representations, say,

$$\boldsymbol{C}_i = (\boldsymbol{M}, \boldsymbol{A}_i, \mathbf{R}_{<i}, \boldsymbol{F}_i)\,. \tag{3.7}$$

Moreover we choose $p_x(\boldsymbol{x}_i|\boldsymbol{C}_i;\boldsymbol{\theta})$ to be a normalizing flow (Section 2.7). Our model and the previously described inputs can be visualised in Figure 3.1. Under this assumptions, we can write the probability of generating a conformation $\boldsymbol{X}$ of a molecule with representation $\boldsymbol{M}$ as

$$p_x(\boldsymbol{X}|\boldsymbol{M};\boldsymbol{\theta}) = p_x(\boldsymbol{x}_1|\boldsymbol{C}_1;\boldsymbol{\theta}) \prod_{i=2}^{N} p_x(\boldsymbol{x}_i|\boldsymbol{C}_i;\boldsymbol{\theta}). \tag{3.8}$$

3.2.2 Model inputs and outputs

After providing some high-level description of our model for generating molecule conformations, we proceed to giving a more detailed description of its inputs and outputs.

The molecular representation $\boldsymbol{M}$ and the i^{th} atom representation $\boldsymbol{A}_i$ are generated using a Gated Graph Neural Network (Section 2.4.2). In particular, $\boldsymbol{M}$ is the graph embedding and $\boldsymbol{A}_i$ is the hidden representation of the i^{th} node in the last iteration step h_i^L. However, the representation of the previously placed atoms, $\mathbf{R}_{<i} = (\mathbf{Sch}(\boldsymbol{x}_{<i}, Z_{<i}), \bar{\boldsymbol{x}}_{<i})$, is a concatenation of two elements. The first one, $\mathbf{Sch}(\boldsymbol{x}_{<i}, Z_{<i})$, where $Z_{<i}$ is the atomic number of the previously placed atoms, is generated using SchNet (Section 2.2.4). This is, the positions and atom types of the previously placed atoms are fed into this model to generate a sub-conformational

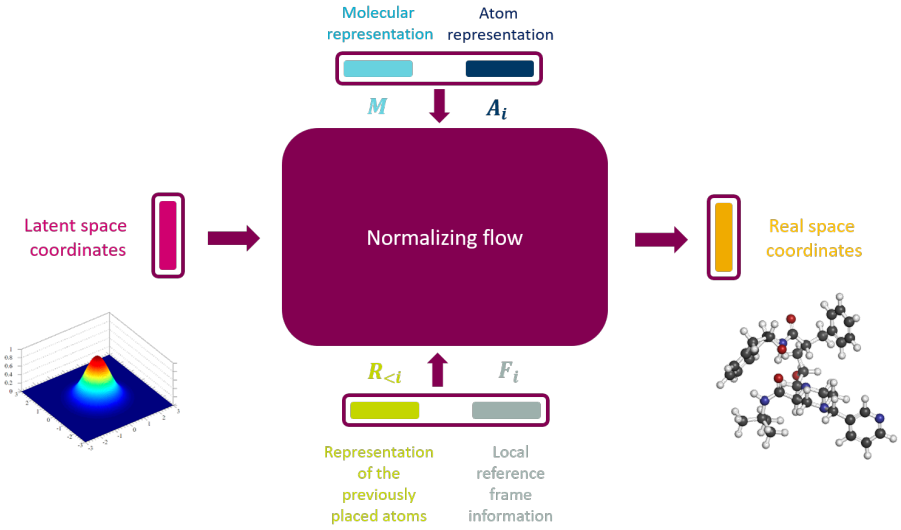


Figure 3.1: Illustration of our auto-regressive generative model for small-molecule configurations. A normalizing flow pushes samples from the latent space to the coordinate space with information about the molecular graph $\boldsymbol{M}$, the atom to place $\boldsymbol{A}_i$, the previously placed atoms $\mathbf{R}_{<i}$ and the local reference frame $\boldsymbol{F}_i$.

representation. Moreover, during the development of this project we found evidence that this information was not enough, since the model was missing information about the positions of the previously placed atoms within the chosen coordinate reference frame (see Section 4.2.1). That is why we add the average position of the previously placed atoms, $\bar{\boldsymbol{x}}_{<i}$, as the second element of the representation acting as a symmetry breaking token. Finally, the information about the local reference frame is encoded as a concatenation of the coordinates of the origin and the axes x, y and z used in the generation step. The origin is the position of the third reference atom; from there, the distance to the new atom is computed. The coordinates of the origin and the axes are described in the NeRF (Section 2.2.3) constructed using the three first atoms in the molecule.

Once this information is given to the model, the generation process works as follows. First, a sample from the base distribution is generated and fed into the normalizing flow. The flow then transforms this latent space vector into a set of three coordinates in the real space. The first atom is placed on the origin, the second one on the x-axes and the third one on the positive y region of the xy-plane. From the fourth and on, the output coordinates are internal coordinates in the NeRF reference frame defined by the reference atoms (Section 2.2).

3.2.3 Training

As explained before, we want our model to be able to take advantage of both examples of configurations found in datasets and our physical knowledge about the system. As such, we performed both likelihood- and energy-based learning (Section 2.8). In practical terms, this means that the model will be trained by minimizing the Kullback-Leibler divergence of the model with respect to both

the data distribution and the Boltzmann distribution. To train on energy, we need a model for the potential energy. For this we used here the SMIRNOFF *openff-1.2.0.offxml* force field from the Open Force Field initiative [56] during training.

The way we approach training, inspired by [37], is by first (pre)training the model using likelihood-based learning and then by trying to improve the performance using energy-based learning. In this second stage we simultaneously used likelihood- and energy-based learning, which aims to minimise the combined loss

$$L_T(\boldsymbol{\theta}) = w_{ML} L_{ML}(\boldsymbol{\theta}) + w_U L_U(\boldsymbol{\theta}). \tag{3.9}$$

Above, w_{ML} and w_U are two extra hyperparameters. Using the combined loss helps to keep a balance between exploration and exploitation, as at least, the sample conformation(s) will not be completely forgotten during the learning process. However, tuning these hyperparameters has proven to be very influential in the result of the training. These hyperparameters are specially dataset- *and* method-dependent since the magnitude of the energies of different molecules can be very different and are affected by the choice of the model for computing the potential energy.

Apart from minimising the combined loss while performing energy-based learning, there are some other practical considerations to take into account. First, energy-based learning is much more computationally expensive than likelihood-based learning. Second, the optimisation space is very high dimensional, since we need to generate the positions for every atom in a molecule to be able to evaluate its energy, in contrast to likelihood-based learning, which uses an atom-wise learning procedure. Third, the model usually produces a number of high-energy conformations that add noise to the direction in which the back-propagation gradient points out, especially during the early stages of learning. To solve this problem, we add an (per-molecule) energy threshold and only back-propagate using configurations with energies below this threshold; we treat this new energy threshold as a hyperparameter. Finally, the number of conformations generated per molecule and epoch also proved to be very influential in our experiments. Although a small number of simulations produce a noisier update, a very large number of conformations severely increases the necessary training time. Energy-based learning incorporates some practical difficulties and new hyperparameters, but usually leads to a performance boost when compared to models trained using only likelihood-based learning.

3.2.4 Data and preprocessing: QM9 dataset

For both likelihood- and energy-based learning, we need a set of molecules to learn from. Moreover, for likelihood-based learning we also need example conformations for each molecule. In this project we used the QM9 dataset [50], which includes information about computed geometric, energetic, electronic, and thermodynamic properties for 134k stable small organic molecules made up of the elements {C,

Table 3.3: Parameters used for MD simulations.

Temperature	300 K
Friction coefficient	1 ps^{-1}
Integration time step size	1 fs
Discarded steps	1000
Production steps	1000
Steps between samples	10
Constraint tolerance	10^{-4}

H, O, N, and F}. We choose this dataset because of the high quality of the molecular conformers. The geometric information is obtained from simulations using quantum chemistry models. We use all the molecules that RDKit is able to read from this dataset (131081) to perform likelihood-based learning. Energy-based learning is much more expensive, and therefore we perform it with a much smaller subset of QM9. In particular, we use a set of 8 molecules with QM9 indexes $1, 2, 4, 5, 6, 7, 11$ and 13 as the training set and molecules $0, 9$ and 10 as validation set.

Molecules in QM9 are rather small if we compare them with drug-like molecules. Therefore, although we use this dataset in a first approach, for future work more extensive datasets, with larger molecules, should be considered.

3.2.5 Evaluation

We want to be able to generate conformations that are as similar as possible to the ones that could be observed in a real physical system. Therefore, a suitable evaluation metric would involve comparing the potential energies $U_M(x)$ of molecular configurations generated by the flow model to those generated by molecular dynamics (MD) simulations. We used the package OpenMM [57] for performing the MD simulations. Specifically, we used the force field *openff-1.2.0.offxml* and the Langevin Integrator with the set of parameters indicted in Table 3.3. This is, we first minimise the energy, then we equilibrate the system and, finally, we discard 1 ps of simulation and afterwards save the state of the system every 10 fs.

3.3 Code availability

Code is available on GitHub under the following links:

- `https://github.com/olsson-group/RL-GraphINVENT`,
- `https://github.com/olsson-group/transBG`.

4

Results

4.1 Reinforcement learning with graph-based generative models (1st model)

4.1.1 Training results of GraphINVENT

In this subsection we show the best training results after hyperparameter optimisation of the learning rate scheduler parameters (parameters which were kept fixed are discussed in Section 3.1.1.1). The best parameter set we found was as follows:

- initial learning rate: 10^{-4},
- final learning rate: 10^{-7},
- 30 training epochs.

For more details about hyperparameter optimisation please refer to Appendix A.1.

The evolution of the training and validation losses (KL divergence between the target and predicted APDs) during training is shown in Figure 4.1. Here we also show the evolution of the fraction of 1) valid, 2) properly terminated 3) valid of those properly terminated and 4) unique molecules generated (as defined in Section 2.4.3.4). Indeed we observe that training for 30 epochs leads to a model which has learned to generate a diverse set of chemically valid molecules.

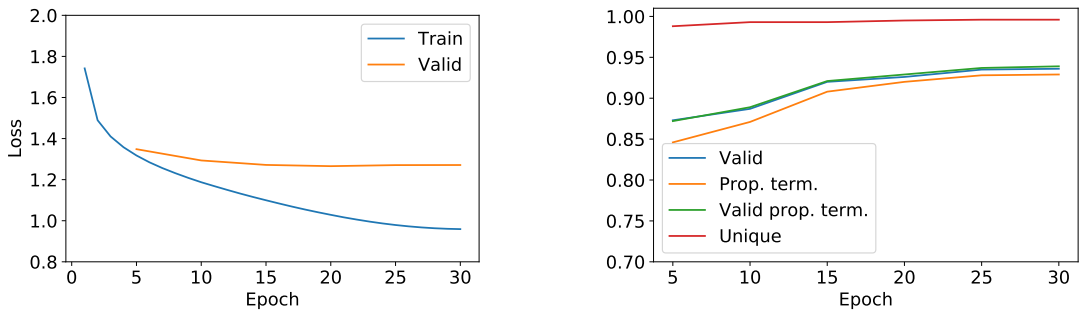


Figure 4.1: Left: Evolution of the training and validation losses during training. Right: Evolution of different evaluation metrics during training for 10^5 molecules.

The performance of these models is comparable to REINVENT. It must also be noted that better results for the percentage of valid and properly terminated molecules can be obtained by training with a smaller learning rate for a larger

number of epochs. However, we believe this training is good enough, as these metrics will improve further during the RL phase, and because the training time is considerably shorter for this model compared to that needed to train a model using a learning rate 10 times smaller.

The model we chose for subsequent RL-based training was that at epoch 20, at which the lowest validation loss (1.2657) is observed. The comparison of the features of the generated molecules by this model and those in the training set is shown in Figure 4.2. There, it can be observed that the generated molecules replicate almost perfectly the properties shown by those in the training set (as it is desired). Furthermore, examples of molecules in the training set and examples of molecules generated by the trained model at this epoch are shown in Figures 4.3 and 4.4, respectively.

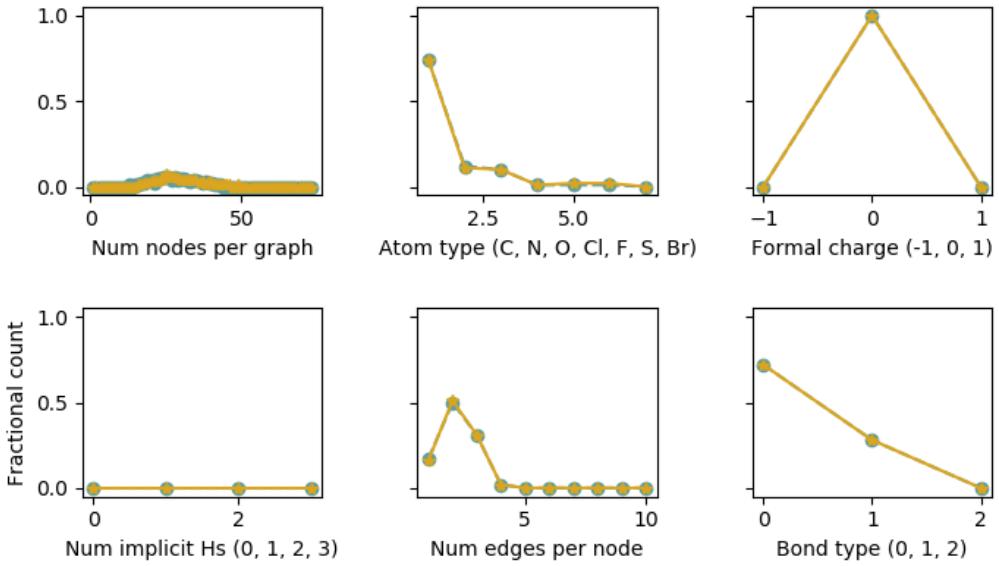


Figure 4.2: Comparison of features of the molecules in the training set (orange) and those generated by the model at epoch 20 (blue).

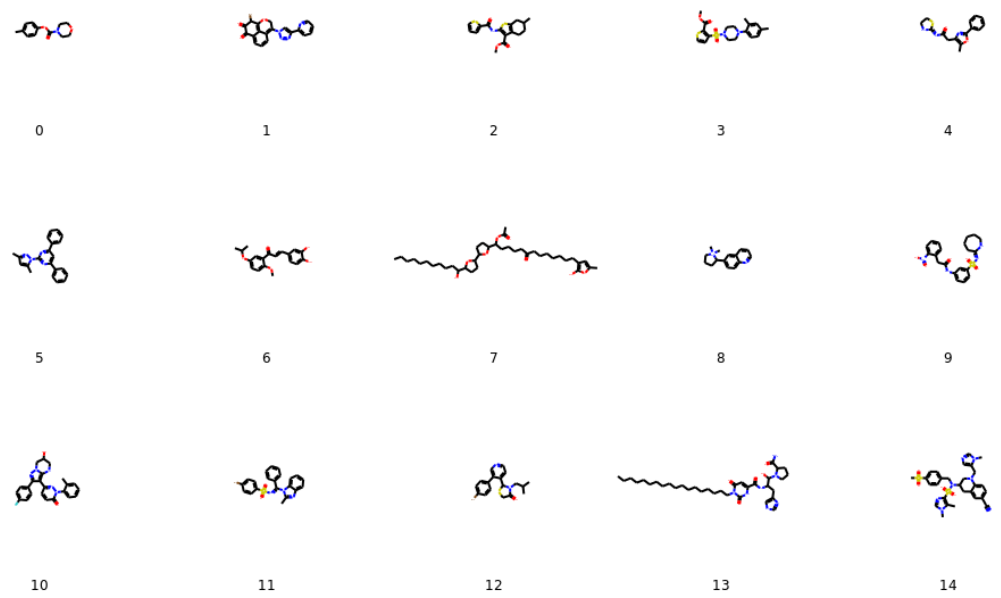


Figure 4.3: Examples of molecules in the training set, a subset of ChEMBL (see Section 3.1.2.1.1).

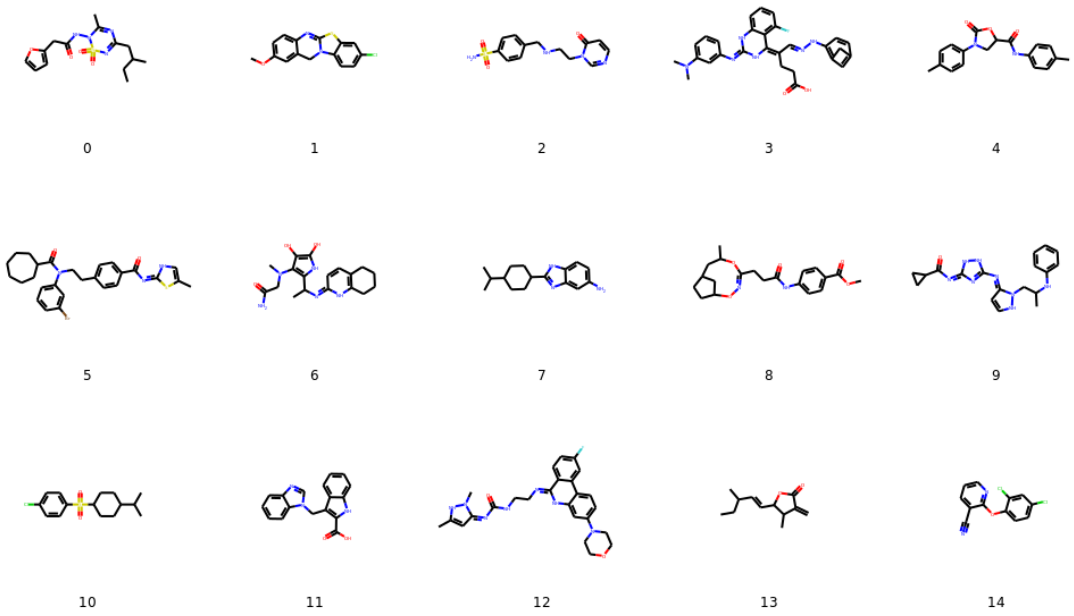


Figure 4.4: Examples of molecules generated by the trained model at epoch 20.

4.1.2 Using the BAR loss function

When analysing the behaviour of the reinforcement learning framework for the loss function proposed in REINVENT (recovered from the loss function proposed herein for $\alpha = 0$), and comparing that to the loss function with $\alpha = 0.5$, all in combination

with the activity scoring function defined in Equation 3.5, we observe that the new loss definition helps to stabilise learning. In this case, the score is binary, either 0 (for 'inactive' molecules) or 1 (for 'active' molecules). As the model only learns from the good examples and there are not many of them generated in the beginning, it is specially desirable in this case to keep track of the model which was able to generate the largest amount of active molecules; this will make learning easier by reminding the agent of the set of actions taken to build them. Otherwise, we sometimes observed that the model was not able to start learning how to generate potential actives, getting lost in the process and therefore leading to larger noise and lower scores (Figure 4.5). We found that using $\alpha = 0.5$ not only accelerated learning, but also provided better and more robust results. See Appendix C for more details.

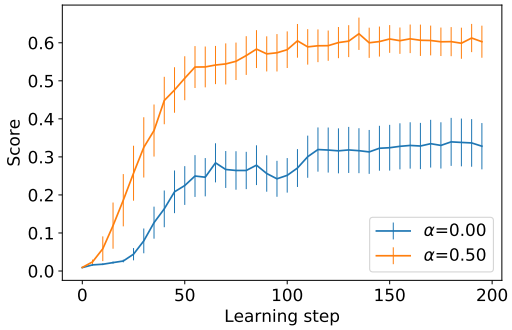


Figure 4.5: Comparison of the average score of the generated molecules as a function of learning step. The results in blue are analogous to using the old loss function proposed in REINVENT (Equation 2.26), which is recovered when $\alpha = 0$, whereas the results in orange correspond to using the new proposed BAR loss with $\alpha = 0.5$ (Equation 3.1).

4.1.3 Reducing and increasing the average size of the molecules

In this first set of experiments using our model, we aimed to prove the ability of the RL framework to fine-tune the generative model towards generating molecules with some set of desired properties. To do so, we used the scoring functions defined previously in Equations 3.2 and 3.3.

In Figure 4.6 we show how, for $\alpha = 0.5$ and $\sigma = 10$ (for more details about hyperparameters please refer to Appendix A.2), the model is able to both decrease and increase the average size of the molecules towards the goal size defined as the maximum of the scoring function (10 and 40 heavy atoms in our setting, respectively). And that it does so by increasing the average score of the generated molecules. In Figures 4.7 and 4.8, some examples of generated molecules are shown for both scenarios. It can be observed that the molecules look reasonable and that there is a remarkable change in the shape of the molecules in both scenarios when compared to the molecules generated by the original GraphINVENT model (shown in Figure 4.4). Results showing the evolution of metrics related to the percentage of valid,

unique and properly terminated molecules, as well as a comparison of the properties of the generated and training sets, can be found in Appendix B.1.

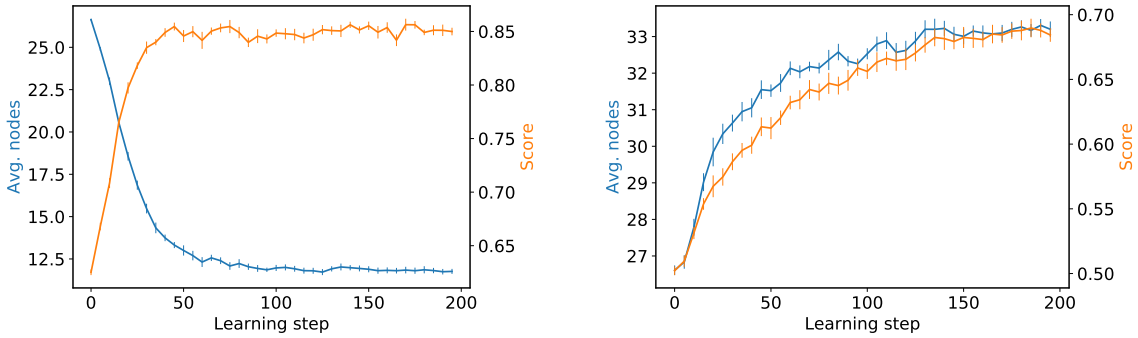


Figure 4.6: Evolution of the average number of nodes (blue) and of the score (orange) for fine-tuning corresponding to reducing the size of the molecules on the left and increasing the size of the molecules on the right. The values are computed for 1000 molecules, taking averages over 10 runs, and the error bars correspond to the standard deviation.

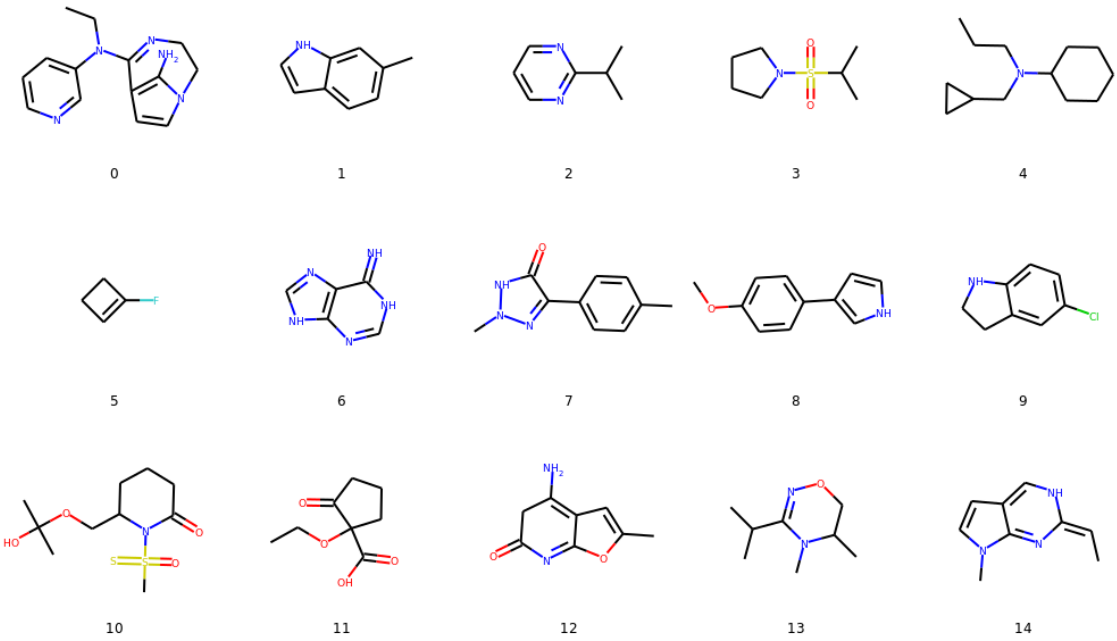


Figure 4.7: Examples of molecules generated by the model after fine-tuning with the score defined in Equation 3.2 for reducing the size of the generated molecules.

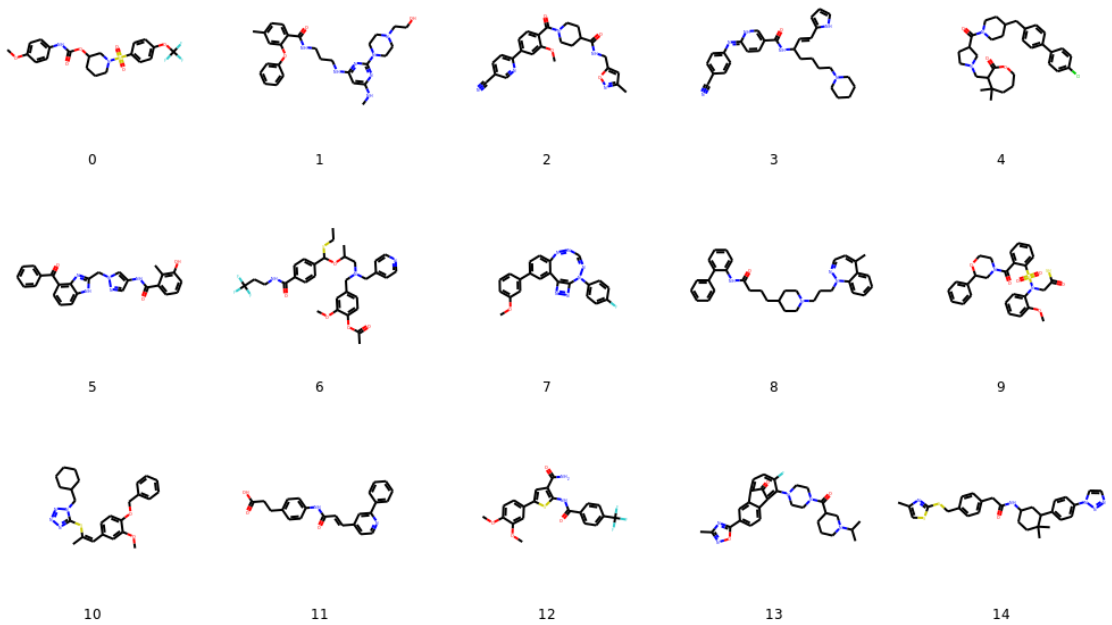


Figure 4.8: Examples of molecules generated by the model after fine-tuning with the score defined in Equation 3.3 for increasing the size of the generated molecules.

4.1.4 Promoting drug-like molecules

In this next subsection, we show the results obtained when training a model using a score based on the QED (Equation 3.4).

In Figure 4.9, we show the evolution of the generated molecules score over the learning process for $\alpha = 0.5$ and $\sigma = 10$, together with other evaluation metrics (see Appendix A.2 for more details about hyperparameters). It can observed that our model is indeed able to improve the average score of the generated molecules. The percentage of valid and properly terminated molecules improves during the learning process as we penalise invalid and non-properly terminated molecules. Furthermore, the average number of heavy atoms in the molecules is found to decrease during learning. This behaviour can be understood when looking at the QED of the generated molecules against the number of nodes (Figure B.4), since higher values are generally achieved by molecules with a number of nodes around 20. Finally, we must also note the fraction of unique molecules (calculated for 1000 molecules) also decreases during learning. This behaviour is not desirable but it is expected since we are updating towards a small number of molecules by weighting them using their score (which is only 0.5 on average at the beginning). Some examples of generated molecules are shown in Figure 4.10. Most of these molecules indeed look drug-like. A comparison of the properties of the molecules generated by the fine-tuned model and those in the training set can be found in Appendix B.2.

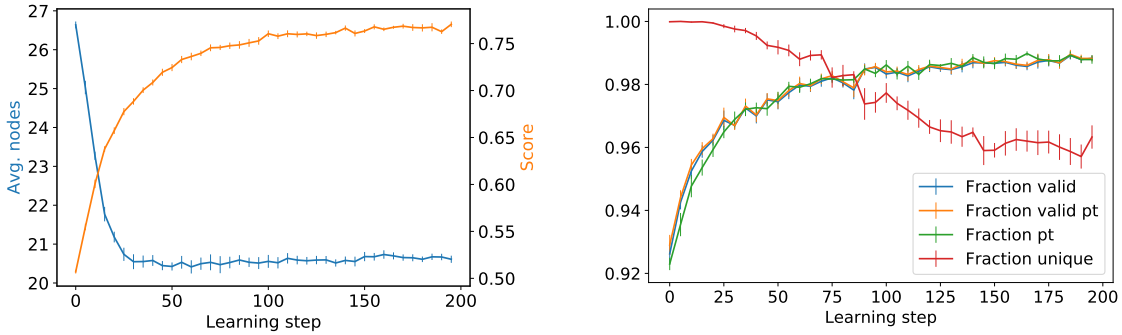


Figure 4.9: Left: Evolution of the average number of nodes (blue) and of the score (orange) in the generated set. Right: Evolution of the fraction of valid, properly terminated, valid of those properly terminated and unique molecules during training. The values are computed for 1000 molecules, taking averages over 10 runs, and the error bars correspond to the standard deviation of the mean.

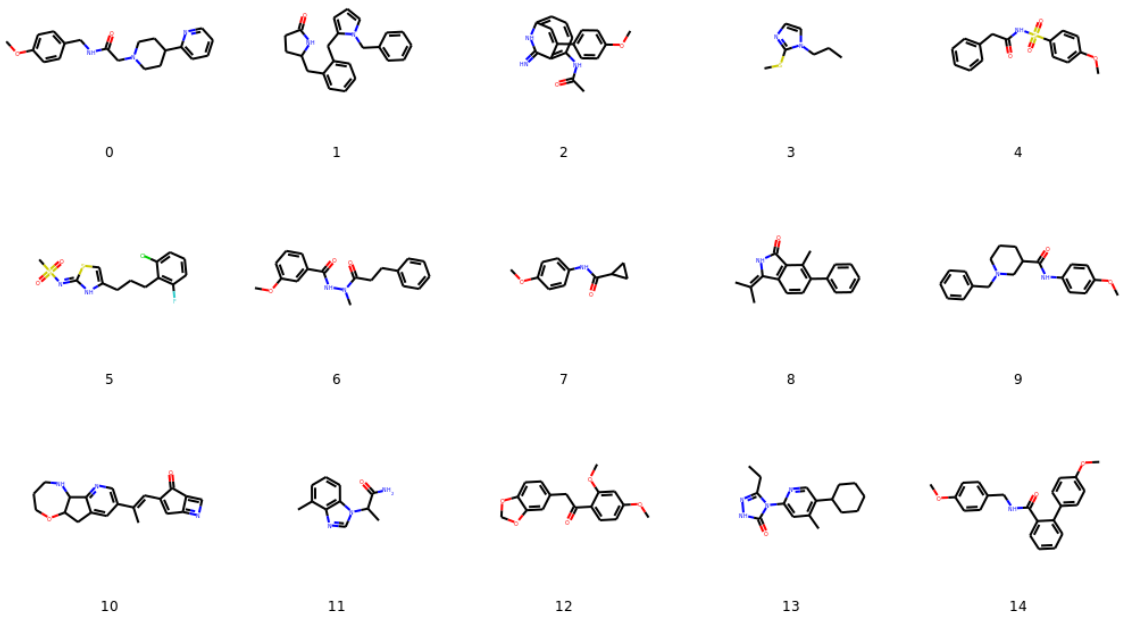


Figure 4.10: Examples of molecules generated by the model after fine-tuning with the score defined in Equation 3.4 for promoting drug-like molecules (high QED).

4.1.5 A scoring function to promote DRD2 active molecules

Finally, a more elaborate scoring function was defined where we both took into account the QED and the QSAR model for the DRD2 activity. In this case, we defined a discrete score in Equation 3.5. Here, molecules which were deemed to be valid, properly terminated and unique, as well as possessing a QED and activity estimate each > 0.5, were given a score of 1; all other generated molecules received a score of 0.

In Figure 4.11 one can observe how the model was able to learn how to generate

active molecules even though there were almost no active molecules generated at the beginning of the learning process. It must be highlighted that although the average score is around 0.6, this is due to the existence of duplicate molecules, since the fraction of active molecules is higher (as it is shown in Table 4.1). One can also observe in Figure 4.11 that the fraction of unique molecules decreases significantly. As we explained for the previous scoring function for promoting drug-like molecules, this does not surprise us since there are few 'good' examples (active molecules) at the beginning, and that is why the model is not able to keep generating a great variety of molecules at the same time as learning to generate more active molecules at each step. We also note that, again, the fraction of valid and properly generated molecules increases during the learning process. Finally, one can observe that the results are robust since most metrics shown exhibit very little noise. Only the average number of nodes shows more variation between different runs; however, this can be explained, once again, due to the strong dependence on the first active molecules generated by the model, from which the agent will learn, thus affecting the whole learning. To summarise, we expect the sets of molecules generated by different models to be quite different from each other.

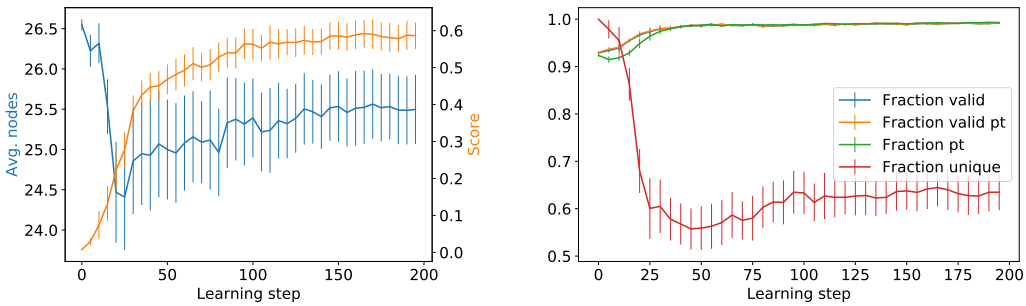


Figure 4.11: Left: Evolution of the average number of nodes (blue) and of the score (orange) in the generated set. Right: Evolution of the fraction of valid, properly terminated, valid of those properly terminated and unique molecules during learning. The values are computed for 1000 molecules, taking averages over 10 runs, and the error bars correspond to the standard deviation of the mean.

In Figure 4.12, some examples of molecules generated by the model are shown. The results are satisfactory since all molecules but one are predicted to be active by the QSAR model. Additional results, such as a comparison of the features between the generated and training sets, can be found in Appendix B.3.

In Table 4.1, we compare to the pretrained model various metrics for two sets of molecules: one in which all 10K molecules are generated by the same fine-tuned model, and another set in which we combine 1K molecules generated by 10 different fine-tuned models into one set of 10K (same set of hyperparameters but different runs). In analysing these molecules, one can calculate metrics such as the average QED, DRD2 activity and the fraction of active molecules. We observe that both sets of molecules show similar values for these metrics. And that they improve substantially when compared to the molecules generated by the

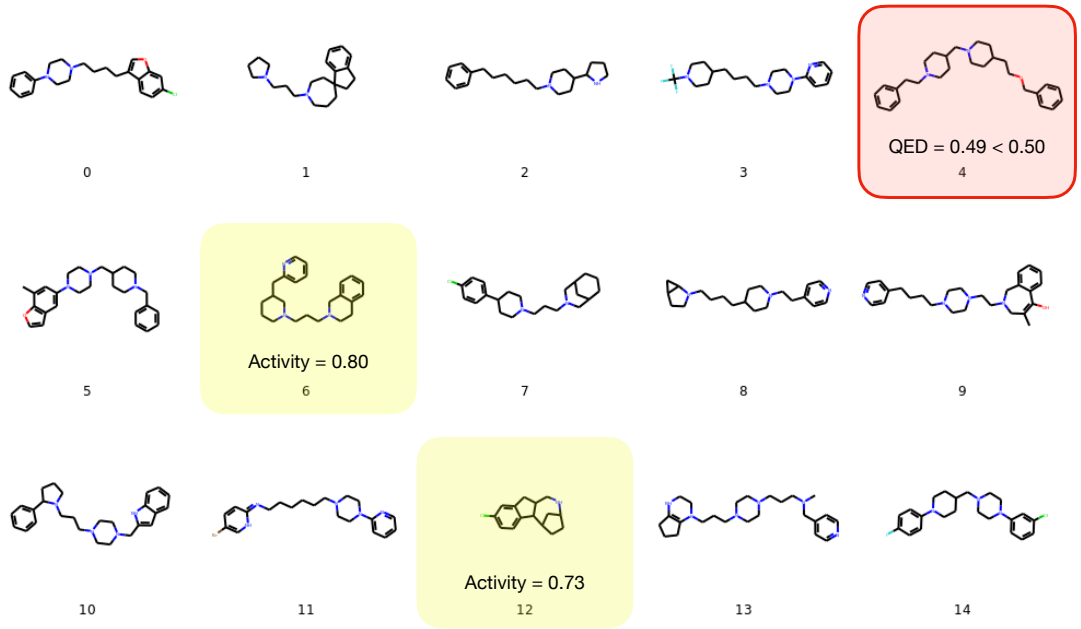


Figure 4.12: Examples of molecules generated by the model after fine-tuning the pre-trained GraphINVENT model using the score defined in Equation 3.5 for promoting the generation of drug-like, DRD2 active molecules. All molecules but number 4 (in red) are predicted to be active (QED $>$ 0.5 and activity $>$ 0.5). Molecules marked in yellow are active but have a lower activity compared to all others, which have associated activity values larger than 0.90 (most of them close to 0.99).

model before fine-tuning. The intersections of the generated sets with the set of known DRD2 active molecules increases for both sets, but especially for the one which combined molecules from different learning processes. This follows from the previous reasoning about the RL-trained models being heavily-dependent on the initial learning steps. As the intersections differ by a factor of 10 (the number of distinct jobs) between 'single' and 'combined', we can infer the sets of molecules generated by the different models generally do not overlap with each other. Finally, we also note that, even though the fraction of active molecules is close to 1 for both fine-tuned models, the percentage of molecules in the DRD2 dataset that our models were able to recover is very small. Nonetheless, this is still a positive result as the generative model has not seen any examples of 'true' active molecules at any point during training and there was no overlap before fine-tuning. This suggests that the model could also learn to generate actives in a new drug discovery task.

Table 4.1: Comparison of various evaluation metrics for three sets of 10,000 generated molecules: one in which all those molecules are generated by one single model, another in which 1000 molecules are generated and combined from 10 different models and a final one in which it is the pretrained GraphINVENT model the one that generates the 10,000 molecules. *Average QED* and *average DRD2 activity* refer to the average scores predicted for QED and DRD2 activity, respectively, using the respective models; *fraction active* refers to the fraction of molecules which have a predicted QED and activity score > 0.5; *fraction unique* and *fraction active and unique* are self-explanatory; *intersection with known actives* refers to the percentage of molecules from the DRD2 dataset which have been re-generated by each model.

	Single job	**Combined jobs**	**Pretr. model**
Average QED	0.72	0.76	0.59
Average DRD2 activity	0.92	0.94	0.03
Active (%)	94	97	1
Unique (%)	48	60	100
Active and unique (%)	45	58	1
Inters. known actives (%)	0.08	0.83	0.00

4.2 Deep auto-repressive generative model for small-molecule configurations (2nd model)

Now that we have presented the results for our graph-based generative model for targeted molecular design, we present the workflow and results of our conformations generator.

4.2.1 Likelihood-based learning results without symmetry breaking token

We evaluate here the results obtained for our conformations generator trained using likelihood-based learning, using *only* the SchNet output as a representation of the previously placed atoms. As previously mentioned, likelihood-based learning is performed on the QM9 dataset. 80% of the readable molecules are used for training and 20% for evaluation. A sample training of the model can be observed in Figure 4.13 (left). The set of used hyperparameters can be found in Appendix A.3. For this model, we observed robust learning behaviour during training, as we saw both the training and validation losses decrease together (except in the final epochs).

In Figure 4.14 we plot an energy histogram of the conformations generated by this model for methane (we will use this molecule throughout as a simple example). As the histogram shows, we observe that the model generates a number of low-energy conformations. However, a remarkable number of high-energy conformations are generated as well. To understand this result, we visualised the conformations (also in Figure 4.14). Although the low-energy conformations seem very reasonable, in the high-energy conformations the model often places two atoms very close to each other (almost in the same position). Additionally, we are not able to see important

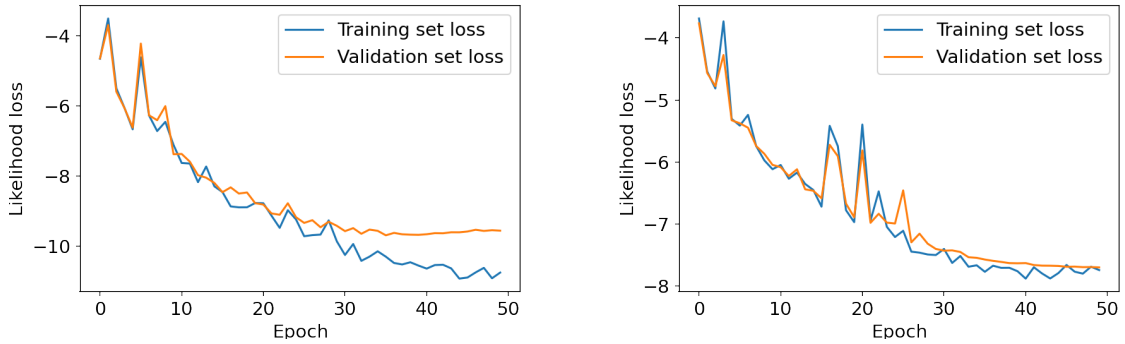


Figure 4.13: Likelihood-based training of our conformation generating model without the symmetry breaking token (left) and with the symmetry breaking token (right).

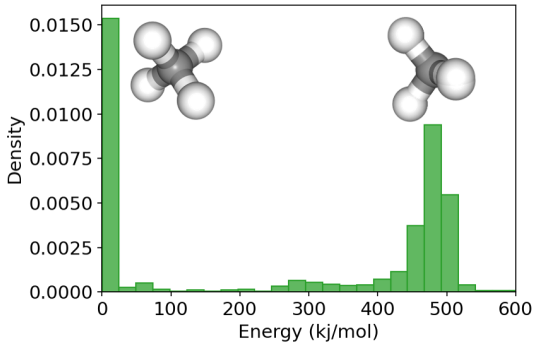


Figure 4.14: Energy histogram generated by a model trained using likelihood-based learning without the symmetry breaking token. Notice in the high-energy conformation the collocation of two atoms, approximately, in the same position.

improvements when we perform energy-based learning only on this molecule or in any other. This two phenomena suggest that it is plausible that the model is not able to correctly interpret the position of the previously placed atoms within a given reference frame. Therefore, it confuses the positions of atoms in molecules that exhibit special symmetries or spatial isomerisms. For these reasons, we took inspiration from G-SchNet[33] and introduced to the representation of the previously placed atoms the average of their positions in the generation reference frame, $\bar{\boldsymbol{x}}_{<i}$, as a symmetry breaking token.

4.2.2 Likelihood-based learning results with symmetry breaking token

In this section, we follow the same training procedure for our molecular conformations generator as in the previous section, only now including the symmetry breaking token. The evolution of training and validation losses is shown in the right plot of Figure 4.13 and the set of chosen hyperparameters are listed in Appendix A.3. Here, we can observe that the validation loss is larger than in the previous setting after adding the token. We do not yet understand the reason for this decrease in the

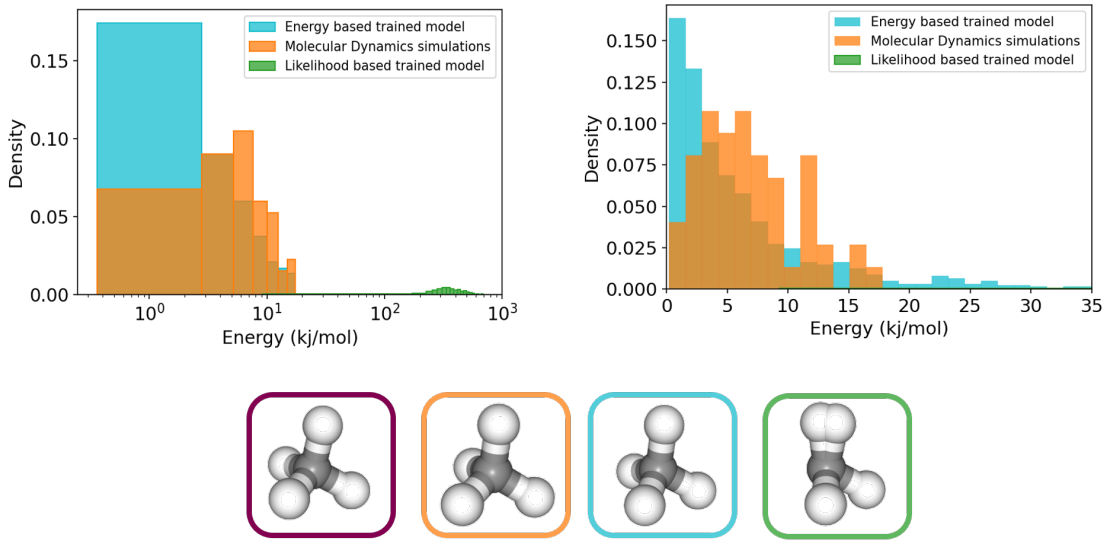


Figure 4.15: Energy histogram and visualisations of the conformations generated by the various methods investigated in this thesis for the molecule of methane. On the left, results are plotted using a logarithmic x-axis; on the right, we focus on the highest overlap region between the MD simulation energies and the conformations energies from the trained energy-based model. Purple colour corresponds to the QM9 conformation.

performance. As such, finding a way to include the information encoded in this token without loss of performance in likelihood-based learning remains as an open research question. However, as we will see in the next Section, the advantages of the use of the symmetry breaking token are clear when we perform energy-based learning.

4.2.3 Energy-based learning on the molecule of methane

As a first approach to energy-based learning, we try to learn to generate methane conformations using our model. Using the previously described techniques (Section 3.2.3), we are able to train our autoregressive model using energy-based learning to fit the energy distribution of methane, illustrated in Figure 4.15. The model had been pretrained using likelihood-based learning on the whole QM9 dataset before the energy-based learning was performed. We observe a clear improvement thanks to the use of energy-based learning on the accuracy of the molecular conformations generated (measured as higher overlap between the energy histogram obtained from MD simulations and the one from our model). However, the low (< 5 kJ/mol) and high (> 17.5 kJ/mol) energy conformations are slightly over-predicted.

4.2.4 Energy-based learning on a set of molecules

Having shown that our model is able to generate reasonable conformations for a small molecule, we wonder if it can learn the conformational space of a set of simple molecules. As mentioned before, energy-based learning is very expensive to perform

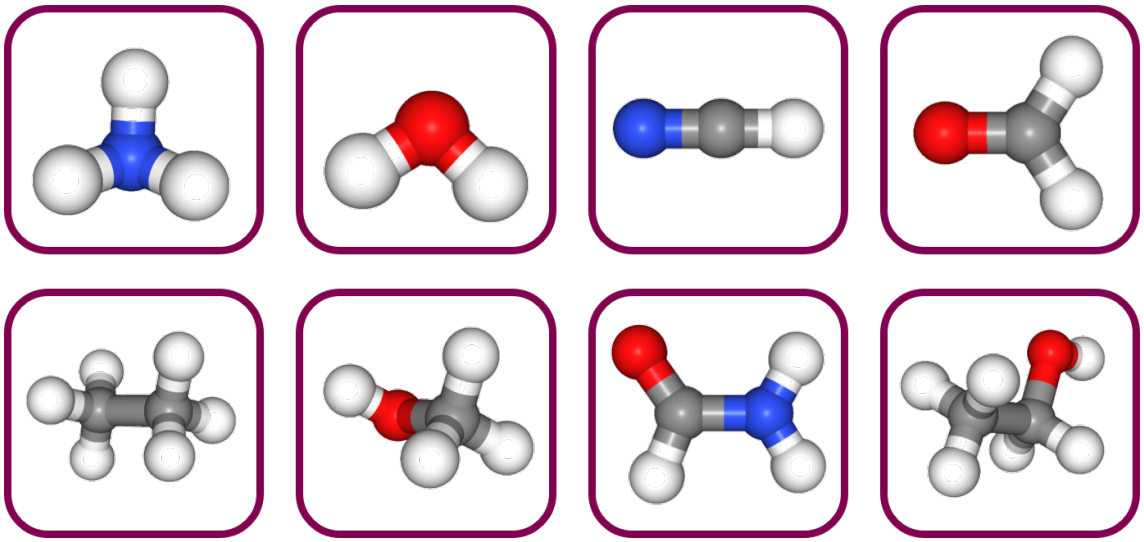


Figure 4.16: Visualization of the QM9 minimal energy conformations for the molecules used as training set for energy-based learning. Their corresponding QM9 indexes are $1, 2, 4, 5, 6, 7, 11$ and 13.

and therefore we are forced to stick to the use of a reduced number of molecules. We use a training set of 8 molecules (with QM9 indexes $1, 2, 4, 5, 6, 7, 11$ and 13 and whose QM9 minimal energy conformations can be visualized in Figure 4.16) and a validation set of 3 molecules (with QM9 indexes $0, 9$ and 10). The motivation for choosing this set is to keep things as simple as possible while learning something that can be general enough to fit the validation set. To do that we have made sure that the molecules in both sets share, approximately, some common characteristics such as the type of atoms and bonds or the average number of atoms.

For training the model on this set we use the previously-explained methods in Section 3.2.3, among others, introducing a different energy-threshold for each molecule in the training set based on observations on the MD histograms. We can see in Figure 4.17 that even with such a small training set, the model is able to learn some general features that minimise the validation loss. This result suggested the possibility of transferability to the task of fitting the conformational space of molecules using energy-based learning. This feature is specially important if we want to predict reasonable conformations for unseen molecules that could be target compounds from some drug-design application.

However, we can see in Figure 4.18 that our model is not able to faithfully reproduce the energies corresponding to conformations generated by MD simulations as effectively as in the previous example, even on the training set. We can distinguish different scenarios. First, for formaldehyde, the energy histogram shifts to the right, thus better matching the MD histograms (overall overlap, however, is still poor). Second, for ethane, it is not clear if the overlap between histograms from MD simulations and the model improves with energy-based learning. Finally, for methanol we observe that the values of the energy in the histogram decrease after

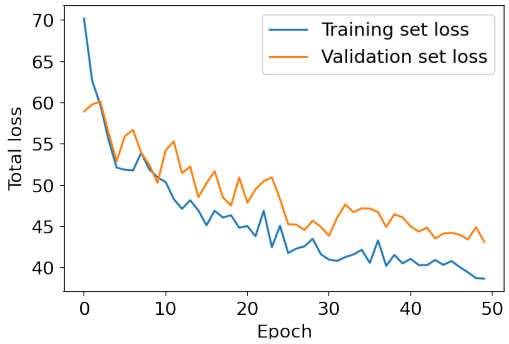


Figure 4.17: Training process using likelihood- and energy-based learning on a small set of molecules.

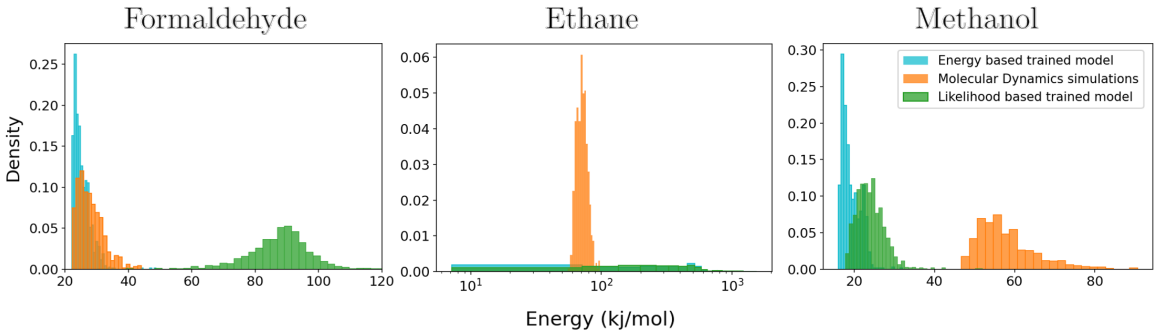


Figure 4.18: Energy histograms for three small organic molecules in the training set generated by the three different methods. The x-axis scale is chosen for each molecule to best visualise the overlap between the different energy histograms.

energy-based learning, even when the MD simulations predict greater energy values. In this case, the model fails to capture the thermal fluctuations and just tends to minimise the energy of the molecule. We do not yet understand this effect with detail.

In addition, we show in Figure 4.19 the equivalent energy histograms and sample conformations in the validation set. For this set, we observe that energy-based learning contributed to improve the overlap of our model's with the MD histograms, although in the case of acetaldehyde the improvement is arguable. We can observe in Figure 4.19 as well how the very high energy states of methane, with two atoms considerably close to each other, are not predominant after energy-based learning and the energy differences with the MD histogram are due to variation of the angles described by the hydrogen atoms. Visually, the conformations generated by the model for the three molecules after performing energy-based learning seem reasonable.

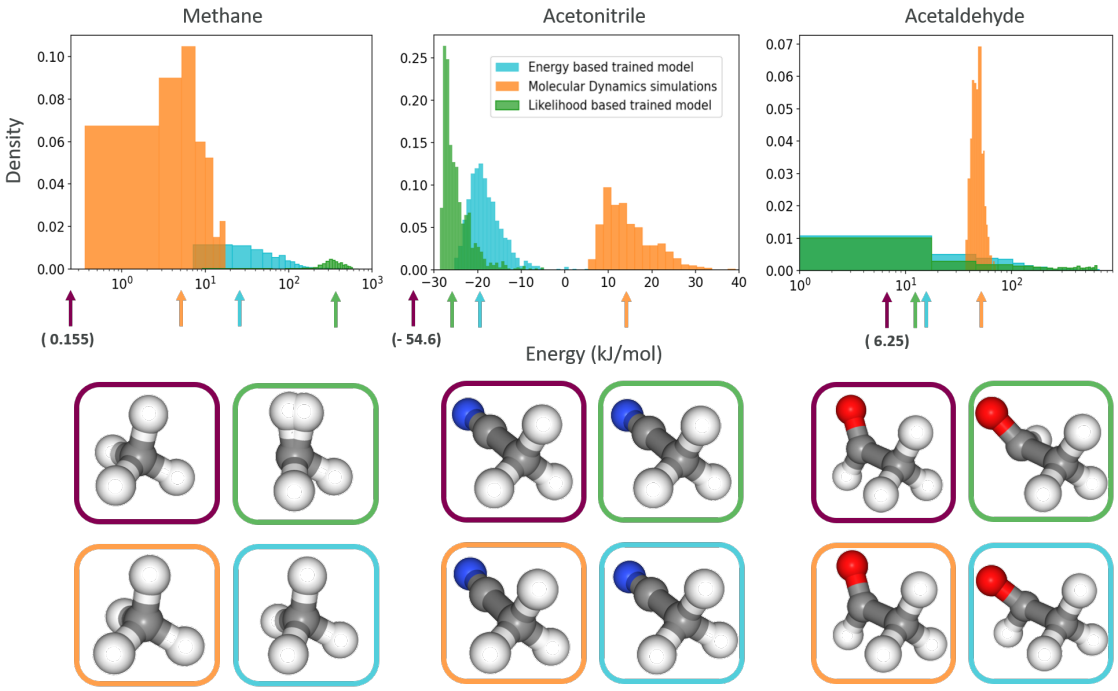


Figure 4.19: Energy histograms and visualisation of the conformations obtained using different methods. Energy of the different visualised conformations are specified with arrows under the histograms. Purple colour corresponds to QM9 conformations.

5

Discussion

5.1 Reinforcement learning with graph-based generative models (1^{st} model)

The proposed RL framework has shown a remarkable ability for fine-tuning the pretrained graph-based generative model towards production of molecules with desired sets of properties, even in challenging situations where only few examples of compounds with those desired properties were initially present in the generated set. This was the case when we aimed to generate DRD2 active molecules. We have also shown our model is able to perform well in tasks such as reducing and increasing the size of the molecules or promoting molecules with high associated QED values. Finally, we have shown how the RL framework is able to improve the performance of the underlying generative model by also increasing the percentage of valid and properly terminated molecules, making it comparable to the performance exhibited by REINVENT models.

The main drawback of the proposed model is the amount of time and computational power needed to pretrain the graph-based generative models, which is longer than that needed for string-based ones. However, as this process only has to be done once for each dataset (which can be as general as ChEMBL), we believe it is worth the investment due to the remarkable performance and flexibility in the applications of the RL framework. We must also highlight the short time (between $10 - 40$ minutes, with the scoring model being the main bottleneck) needed for the reinforcement learning process. This feature makes our model very competitive, since once we have a pretrained GraphINVENT model, we can quickly and easily fine-tune it towards the desired generation task.

The structural information included by using graphs can be specially important in the fine-tuning process. We believe so since the model can learn to relate which interactions lead to higher scores. In this sense, we also believe that the performance could improve further once the 3D information computed by the second model is also included, motivating further research in the topics presented in this thesis.

We have shown very promising results in which the model has been able to achieve the goals we had set. If we compare the results of generating DRD2 active molecules with those of previous works [46], we observe our model is able to generate a much

greater fraction of (predicted) active molecules: around 95% of active compounds against only 54% in the best result from the previous work mentioned. This proves the remarkable ability of our generative model for targeted molecular design. As future work, it would also be interesting to train REINVENT in the same way and compare the results for the activity scoring function. This way we would indeed be able to analyse if by adding information to the model about the structure of the molecules, the performance improves.

Some molecules generated when increasing their size seem to be unstable due to having unrealistic large rings. This can be explained since the model has not been shown many examples on how to predict APDs for large subgraphs, making it difficult for it to learn which actions can lead to stable molecules. We can prove this reasoning since this stability problem is not observed when reducing the size of the molecules, as there were many small subgraphs in the training set used when pretraining GraphINVENT and therefore the model learned which actions lead to stable molecules. We also observe there are some unstable molecules when favouring drug-like molecules. It is harder to explain the behaviour in this case, but we believe it might be influenced by the QED values not being very informative. We say so since we do not observe unstable molecules with our more complex activity score in which we disregard the exact QED value by using a threshold. Furthermore, we must highlight we only intended to prove the model was able to increase the selected scores in the reduce, increase and QED settings, which it successfully did. We did not take into account other properties in the scoring function such as stability and/or synthesizabilty. By adding them to the score functions, we expect this issue to be solved. Moreover, it is important to highlight that for our more complex scoring function promoting DRD2 active compounds, the model has been able to learn to generate stable compounds, which again suggests that by choosing an appropriate definition for the score, one can obtain 'good' molecules.

Finally, we also want to highlight the fact that we have introduced a slightly different loss function, the BAR loss, to that used in REINVENT. This new function included a second term in what we considered the best agent so far. With this new definition, better and more stable learning results have been achieved, as it has helped the agent to learn how to generate molecules with better scores in situations in which it did not manage to learn before.

5.2 Deep auto-regressive generative model for small-molecule configurations (2^{nd} model)

In this work we have introduced some early results in the search of a transferable molecular conformation generator. We have proposed and built a model that incorporates some of the most promising approaches in the field. In particular, we have constructed an auto-regressive model based on normalizing flows conditioned on a molecular representation generated by GNNs. The positions of the atoms are generated using internal coordinates and are conditioned on a representation of the

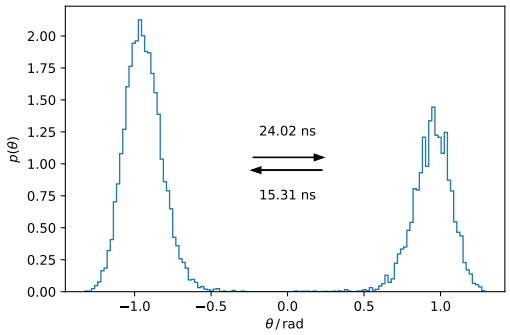


Figure 5.1: Bi-stability of an internal degree of freedom of cyclohexane. The degree of freedom is a torsion θ and the two states are communicated transition times of the order of 20 ns.

atom to place and the interaction patterns of the previously placed ones. Our results show that both likelihood-based and energy-based learning can be used to train deep learning models in this context and that these two methods are complementary.

We have also shown that, even for a set of small molecules, the model does not capture all the necessary information to reasonably match the MD simulations histograms. However, we can clearly see an improvement in the performance of the models when trained with the different, previously mentioned, methods. It is important to remark that the MD simulations are not necessarily representative of the reality since the force-fields they use include a number of approximations. Moreover, even if MD simulations represent the reality, it is possible that in order to fully capture the ensemble of possible states, extremely long simulation times would be needed. This is the case of molecules such as cyclohexane, presenting two different states with very low transition probability between them as we can see in Figure 5.1. In this example, the internal degree of freedom that differentiates the two states is a torsion angle and the transition times are of the order of 20 ns. This is indeed one of the main drawbacks of the use of MD simulations and the limitation we try to address with these methods. Additionally, we have been able to observe transferability when performing both likelihood- and energy-based learning. This is very interesting from the practical point of view since, ideally, a model of this kind would generate physically realistic conformations even for never-before-seen molecules while exploring the chemical space. Our model is far from perfectly representing the conformational space, but it introduces a possible research path and gives a proof-of-concept for the proposed method.

5.2.1 Further work

As previously mentioned, although our conformations generator is far from reproducing real physical systems, some other similar approaches/improvements could be studied to push its performance. Some examples are the use of more expressive flows that have recently been proposed such as [47] or [58]. The same principle applies to the representation of the interaction patterns of the previously placed atom. In

particular, recent methods such as [31] present very promising results. Moreover, as previously discussed it would be beneficial to study other symmetry breaking tokens that have a positive effect on the likelihood-based learning. In addition, further algorithms for generating reference atoms can be investigated, likely having an effect on the performance since they affect the structure of the generated Z-matrices.

5.3 Limitations

The two models presented here can be shown as two steps on the way to develop a *de novo* design tool that takes into account the 3D structure and physical interactions in a molecule. Therefore, one limitation, and a future work motivation, is that this desired model could not be fully developed in the time frame of this thesis. However, both projects can be evaluated independently.

Following a similar line of thought, if we focus on the limitations of the first sub-project (RL with graph-based generative models), we can conclude that one of the most important limitations it has at the moment is that it does not consider the possible 3D configurations of molecules when optimising properties. Therefore, it will not be able to (directly) predict properties which arise due to the 3D conformation of the molecule and take them into account for *de novo* design. Other limitations are related to the time and computational resources needed to pretrain the generative model. However, that only needs to be done once for each dataset (if we want to change the set of atoms for example), as we have discussed earlier.

Additionally, the second model (Generative model for small-molecule configurations) does not explicitly include a physical model of molecular interactions but instead learns a surrogate model based on continuous filter convolutions, inspired by the SchNet architecture. The physical model we use for training this model is subject to some accuracy limitations due to its classical approximation of quantum mechanics. Long-range interactions may not be captured well in the SchNet architecture, possibly leading to a further decline in accuracy. Furthermore, the generation of conformations happens without simulation of the dynamics of the molecule in question. As such, unlike for molecular dynamics simulations, properties that rely on dynamics can not be faithfully computed. Yet, this approach is expected to leave a much smaller computational footprint. The departure from the simple parametric physical models further leads to a loss of interpretability. Notwithstanding, due to the auto-regressive structure of the model, we may dissect how each atom in a molecule is placed.

5.4 Risk analysis and ethical considerations

De novo drug design techniques have the potential to generate very positive effects on society:

- Decrease of the cost and environmental impact of medicines: Efficiently exploring the chemical space could bring the possibility of discovering new medicines with similar properties to the existing ones, but at a lower expense and/or with smaller impact to the environment. Moreover, these tools could help to mitigate the effect of some pathogens developing resistance to treatments, such as antibiotics-resistant bacteria, because they would allow us to find alternative antibiotics faster.

- Combat future pandemics: Last year we have witnessed how the COVID-19 pandemic has shaken the whole world. *De novo* drug design could help to develop effective treatments and vaccines in a briefer period of time than is usually required.

However, these techniques could also threaten public health if used maliciously:

- The ability of searching for target drugs can also be used for the development of dangerous substances: some examples could be powerful and hard to detect poisons, chemical weapons, explosives, environmental toxins, or compounds that generate high levels of dependence and tolerance.

For all the benefits these tools could bring to the world, we believe they should be public domain knowledge. However, we cannot ignore the potential threat of their bad-intentioned use. Therefore, we think the use of this technology, when fully developed, must be regulated.

6
Conclusion

Since the properties a molecule exhibits directly depend on the ensemble of its possible 3D configurations and its molecular graph, it is believed that understanding potential spatial arrangements of candidate compounds and developing graph-based methods will be key for the next generation of *de novo* generation algorithms. Moreover, RL is thought to be the optimal way to fine-tune *de novo* generative models towards target properties. Developments in this field may not only influence new scientific discoveries, but also have a direct impact on societies, due to the speed up of the exploration of the vast chemical space for i.e. designing new materials and drugs.

The motivation for the use of deep learning models comes essentially from the speed gain, without loss of performance, that these models bring. The importance of accelerating *de novo* generation technologies is specially clear in these times of global pandemic. Consequently, deep learning models are believed to be powerful allies for fighting future sanitary crises.

We have shown in this thesis that RL-based methods are effective for shifting *de novo* generation towards molecules that fulfil desired properties using graph-based methods, as it had been shown for string-based methods. Moreover, we have also proposed and characterised a conformations generator conditioned on a molecular representation. These two projects have contributed to the field of drug discovery, both by bringing new ideas and showing proofs-of-concept. Moreover our code will be accessible so that it can be used by the drug-discovery community. Finally, we believe our work motivates the future creation of an integrated model that incorporates the configurational information and can be fine-tuned with RL methods.

Bibliography

[1] Kristof T. Schütt, Pieter-Jan Kindermans, Huziel E. Sauceda, Stefan Chmiela, Alexandre Tkatchenko, and Klaus-Robert Müller. Schnet: A continuous-filter convolutional neural network for modeling quantum interactions. 2017.

[2] Rocío Mercado, Tobias Rastemo, Edvard Lindelöf, Günter Klambauer, Ola Engkvist, Hongming Chen, and Esben Bjerrum. Graph networks for molecular design. 08 2020.

[3] Gisbert Schneider and Uli Fechner. Computer-based de novo design of drug-like molecules. *Nat Rev Drug Discov*, 4:649 – 663, 2005.

[4] Thomas Blaschke, Josep Arús-Pous, Hongming Chen, Christian Margreitter, Christian Tyrchan, Ola Engkvist, Kostas Papadopoulos, and Atanas Patronov. Reinvent 2.0 – an ai tool for de novo drug design. *ChemRxiv.*, Preprint, 2020.

[5] David Mendez, Anna Gaulton, A Patrícia Bento, Jon Chambers, Marleen De Veij, Eloy Félix, María Paula Magariños, Juan F Mosquera, Prudence Mutowo, Michał Nowotka, María Gordillo-Marañón, Fiona Hunter, Laura Junco, Grace Mugumbate, Milagros Rodriguez-Lopez, Francis Atkinson, Nicolas Bosc, Chris J Radoux, Aldo Segura-Cabrera, Anne Hersey, and Andrew R Leach. ChEMBL: towards direct deposition of bioassay data. *Nucleic Acids Research*, 47(D1):D930–D940, 11 2018.

[6] R Gomez-Bombarelli, D Duvenaud, and J Miguel. Automatic chemical design using a data-driven continuous representation of molecules. arxiv. *arXiv preprint arXiv:1610.02415*, 2016.

[7] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. *CoRR*, abs/1812.08434, 2018.

[8] Nicola De Cao and Thomas Kipf. Molgan: An implicit generative model for small molecular graphs. *arXiv preprint arXiv:1805.11973*, 2018.

[9] Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324*, 2018.

[10] Yibo Li, Liangren Zhang, and Zhenming Liu. Multi-objective de novo drug design with conditional graph generative model. *Journal of cheminformatics*, 10(1):33, 2018.

[11] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. *arXiv preprint arXiv:1802.04364*, 2018.

[12] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Hierarchical generation of molecular graphs using structural motifs. *arXiv preprint arXiv:2002.03230*, 2020.

[13] Laurianne David, Amol Thakkar, Rocío Mercado, and Ola Engkvist. Molecular representations in ai-driven drug discovery: a review and practical guide. *Journal of Cheminformatics*, 12(1):1–22, 2020.

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

[15] Stephen R Heller, Alan McNaught, Igor Pletnev, Stephen Stein, and Dmitrii Tchekhovskoi. Inchi, the iupac international chemical identifier. *Journal of cheminformatics*, 7(1):1–34, 2015.

[16] Greg Landrum. Rdkit: Open-source cheminformatics.

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

[18] Simon Axelrod and Rafael Gomez-Bombarelli. Geom: Energy-annotated molecular conformations for property prediction and molecular generation. 2021.

[19] S. Walter Englander and Leland Mayne. The nature of protein folding pathways. *Proceedings of the National Academy of Sciences*, 111(45):15873–15880, 2014.

[20] Steven S. Skiena. *Sorting and Searching*, pages 103–144. Springer London, London, 2008.

[21] Mohammed AlQuraishi. pnerf: Parallelized conversion from internal to cartesian coordinates. *bioRxiv*, 2018.

[22] Asifullah Khan, Anabia Sohail, Umme Zahoora, and Aqsa Saeed Qureshi. A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review*, 53(8):5455–5516, Apr 2020.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[24] Kristof T. Schütt, Pieter-Jan Kindermans, Huziel E. Sauceda, Stefan Chmiela, Alexandre Tkatchenko, and Klaus-Robert Müller. Schnetpack - deep neural networks for atomistic systems.

[25] Kathleen McKeown. *Text generation*. Cambridge University Press, 1992.

[26] Jean-Pierre Briot, Gaëtan Hadjeres, and François-David Pachet. Deep learning techniques for music generation–a survey. *arXiv preprint arXiv:1709.01620*, 2017.

[27] Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Rezende, and Daan Wierstra. Draw: A recurrent neural network for image generation. In *International Conference on Machine Learning*, pages 1462–1471. PMLR, 2015.

[28] Hongming Chen, Ola Engkvist, Yinhai Wang, Marcus Olivecrona, and Thomas Blaschke. The rise of deep learning in drug discovery. *Drug discovery today*, 23(6):1241–1250, 2018.

[29] Marwin HS Segler, Thierry Kogej, Christian Tyrchan, and Mark P Waller. Generating focused molecule libraries for drug discovery with recurrent neural networks. *ACS central science*, 4(1):120–131, 2018.

[30] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *arXiv preprint arXiv:1406.2661*, 2014.

[31] Weihua Hu, Muhammed Shuaibi, Abhishek Das, Siddharth Goyal, Anuroop Sriram, Jure Leskovec, Devi Parikh, and C. Lawrence Zitnick. Forcenet: A graph neural network for large-scale quantum calculations, 2021.

[32] Johannes Klicpera, Janek Groß, and Stephan Günnemann. Directional message passing for molecular graphs, 2020.

[33] Niklas W. A. Gebauer, Michael Gastegger, and Kristof T. Schütt. Symmetry-adapted generation of 3d point sets for the targeted discovery of molecules, 2020.

[34] George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. 2019.

[35] Minkai Xu, Shitong Luo, Yoshua Bengio, Jian Peng, and Jian Tang. Learning neural generative dynamics for molecular conformation generation, 2021.

[36] Victor Garcia Satorras, Emiel Hoogeboom, Fabian B. Fuchs, Ingmar Posner, and Max Welling. E(n) equivariant normalizing flows for molecule generation in 3d, 2021.

[37] Frank Noé, Simon Olsson, Jonas Köhler, and Hao Wu. Boltzmann generators: Sampling equilibrium states of many-body systems with deep learning. *Science*, 365(6457), 2019.

[38] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, pages 1263–1272. PMLR, 2017.

[39] Yibo Li, Langren Zhang, and Zhenming Liu. Multi-objective de novo drug design with conditional graph generative model. *J Cheminform*, 10(1):33, 2018.

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

[41] Marcus Olivecrona, Thomas Blaschke, Ola Engkvist, and Hongming Chen. Molecular de-novo design through deep reinforcement learning. *J Cheminform*, 9:48, 2017.

[42] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[43] Christopher A Lipinski. Lead-and drug-like compounds: the rule-of-five revolution. *Drug Discovery Today: Technologies*, 1(4):337–341, 2004.

[44] Arup K Ghose, Vellarkad N Viswanadhan, and John J Wendoloski. A knowledge-based approach in designing combinatorial or medicinal chemistry libraries for drug discovery. 1. a qualitative and quantitative characterization of known drug databases. *Journal of combinatorial chemistry*, 1(1):55–68, 1999.

[45] G Richard Bickerton, Gaia V Paolini, Jérémy Besnard, Sorel Muresan, and Andrew L Hopkins. Quantifying the chemical beauty of drugs. *Nature chemistry*, 4(2):90–98, 2012.

[46] Panagiotis-Christos Kotsias, Josep Arús-Pous, Hongming Chen, Ola Engkvist, Christian Tyrchan, and Esben Jannik Bjerrum. Direct steering of de novo

molecular generation with descriptor conditional recurrent neural networks. *Nature Machine Intelligence*, 2(5):254–265, 2020.
[47] Hao Wu, Jonas Köhler, and Frank Noé. Stochastic normalizing flows. 2020.
[48] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp, 2017.
[49] Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation, 2015.
[50] L. C. Blum and J.-L. Reymond. 970 million druglike small molecules for virtual screening in the chemical universe database GDB-13. *J. Am. Chem. Soc.*, 131:8732, 2009.
[51] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.
[52] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks, 2017.
[53] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
[54] Leslie N Smith and Nicholay Topin. Super-convergence: Very fast training of neural networks using large learning rates. In *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, volume 11006, page 1100612. International Society for Optics and Photonics, 2019.
[55] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.
[56] David Slochower, Niel Henriksen, Lee-Ping Wang, John Chodera, David Mobley, and Michael Gilson. Binding thermodynamics of host-guest systems with smirnoff99frosst 1.0.5 from the open force field initiative, Jul 2019.
[57] Peter Eastman, Jason Swails, John D. Chodera, Robert T. McGibbon, Yutong Zhao, Kyle A. Beauchamp, Lee-Ping Wang, Andrew C. Simmonett, Matthew P. Harrigan, Chaya D. Stern, Rafal P. Wiewiora, Bernard R. Brooks, and Vijay S. Pande. Openmm 7: Rapid development of high performance algorithms for molecular dynamics. *bioRxiv*, 2017.
[58] Didrik Nielsen, Priyank Jaini, Emiel Hoogeboom, Ole Winther, and Max Welling. Survae flows: Surjections to bridge the gap between vaes and flows. 2020.

A

Appendix 1: Hyperparameter optimisation

A.1 Training of GraphINVENT

During hyperparameter optimisation for training of GraphINVENT many different hyperparameter sets were explored. The main hyperparameters to change were the initial learning rate and the number of epochs, since we mainly kept the final learning rate to be 3 orders of magnitude smaller than the initial one. While search of optimal values for these parameters, we kept in mind the larger the initial learning rate, the smaller the number of epochs needed to converge and therefore the quicker we had to reduce the learning rate. An example of another hyperparameter set with a learning rate 10 times smaller than for the chosen one (see Section 4.1.1) was:

- initial learning rate of 10^{-4},
- final learning rate of 10^{-7}, and
- 200 epochs.

It can be observed in Figure A.1 that, even though training is significantly slower compared to the one showed in Section 4.1.1, the fraction of valid and properly terminated molecules is larger in this case. However, as we motivated in the referred Section, we believe this improvement not to be remarkable enough so as to justify the substantial increase in training time (around three times more, needing almost a week instead of 2 days). Furthermore, we have observed these metrics improve in the later RL phase.

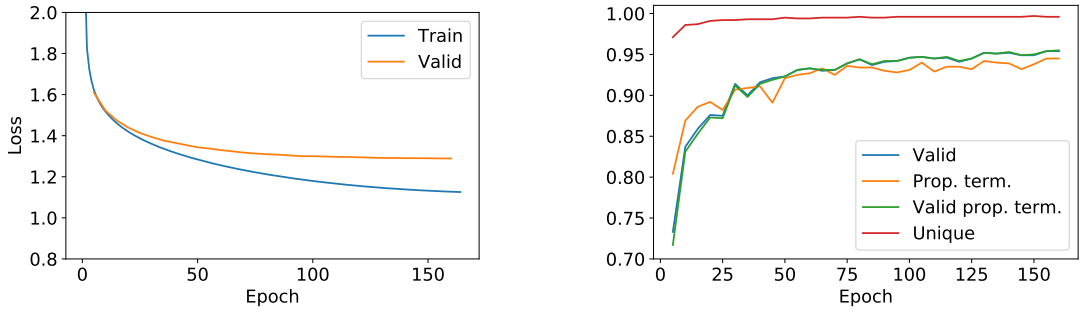


Figure A.1: Left: Evolution of the training and validation losses during training. Right: Evolution of different evaluation metrics during training for 10^5 molecules.

A.2 RL framework

In this section we will show an example of hyperparameter optimisation in the RL phase for the scoring function promoting activity (Equation 3.5). Nonetheless, it must be noted similar optimal hyperparameter values were obtained for all other scoring functions.

In Figures A.2 and A.3 we show the evolution of different evaluation metrics and the average score of the generated molecules over learning for different values of σ. It can be observed the performance is better for σ equal to 20 or 50 than for 10. A smaller value of σ leads to smaller changes in the agent, preventing loosing previously learned knowledge on how to generate 'good' molecules. Furthermore, it also seems to lead to more stable learning, meaning that it monotonically approaches the final value (as in the fraction of unique molecules shown in Figure A.2).

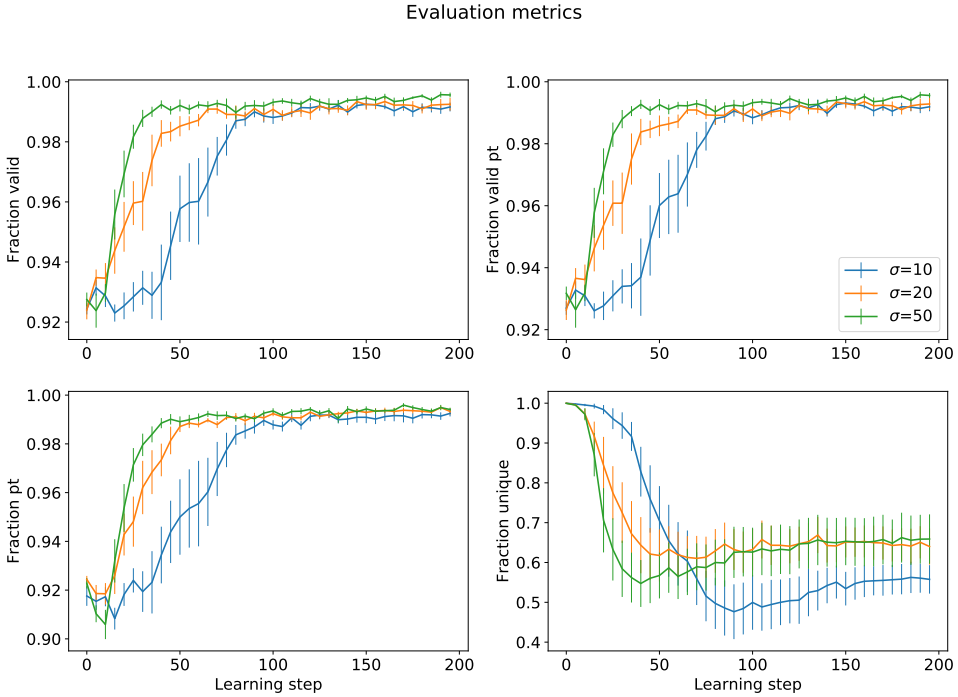


Figure A.2: Evaluation metrics (top left: fraction valid, top right: fraction valid of those properly terminated, bottom left: fraction properly terminated, bottom right: fraction unique) for different values of σ (the parameter which modulates the effect of the scoring function in the loss function) with $\alpha = 0.5$. The metrics are computed for 1000 molecules and averaged over 10 different runs. The error bars shown correspond to the standard deviation of the mean.

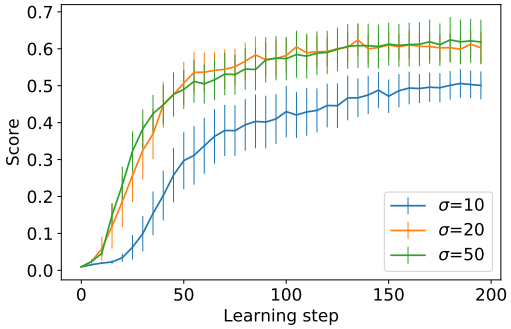


Figure A.3: Average score of the generated molecules for different values of σ with $\alpha = 0.5$. The score is computed for 1000 molecules and averaged over 10 different runs. The error bars shown correspond to the standard deviation of the mean.

In Figures A.4 and A.5 we show the evolution of different evaluation metrics and the average score of the generated molecules over learning for different values of α. It can be observed the performance is better for α equal to 0.5 than for higher (0.75) or smaller (0.25) values. Therefore, the chosen hyperparameter values for the activity scoring function are $\sigma = 20$ and $\alpha = 0.5$. For other scoring functions we have used $\sigma = 10$ and $\alpha = 0.5$ although larger values of σ could lead to higher scores, since our main purpose there was to test the ability of the RL framework to fine-tune the generative model towards a desired goal.

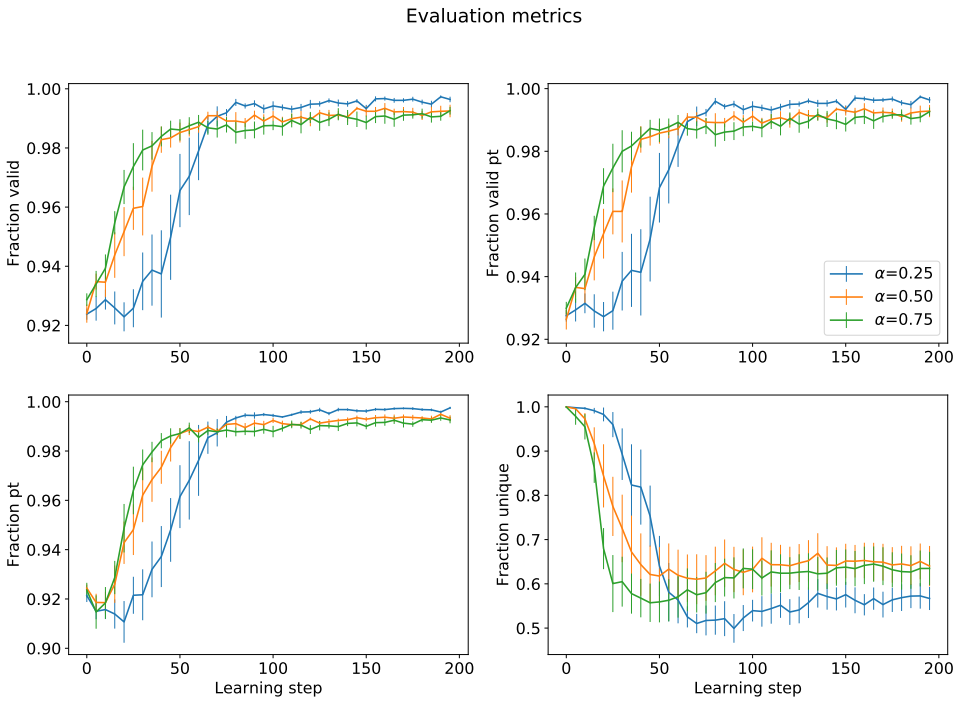


Figure A.4: Evaluation metrics (top left: fraction valid, top right: fraction valid of those properly terminated, bottom left: fraction properly terminated, bottom right: fraction unique) for different values of α (the parameter which modulates the effect of each term in the loss function) with $\sigma = 20$. The metrics are computed for 1000 molecules and averaged over 10 different runs. The error bars shown correspond to the standard deviation of the mean.

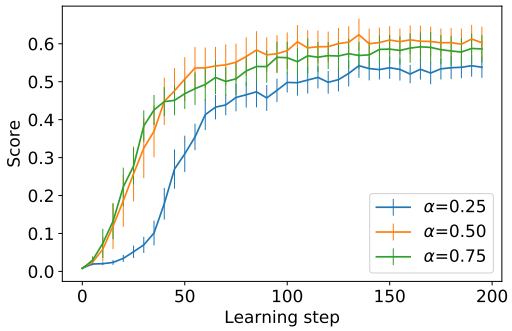


Figure A.5: Average score of the generated molecules for different values of α with $\sigma = 20$. The score is computed for 1000 molecules and averaged over 10 different runs. The error bars shown correspond to the standard deviation of the mean.

A.3 Hyper-parameters for the conformations generator

In each of the NVP transformations we add an NVP block per transformed variable. For example, if we generate a distance, an angle and a torsion, and we have 8 transformations, the flow has 24 NVP blocks. Each NVP block is formed by two feed-forward layers with Leaky ReLu activations followed by a feed-forward layer with tanh activation. In Tables A.1 and A.2 we collect the hyper-parameters used for training the conformations generator without and with symmetry breaking token.

Table A.1: Hyper-parameters used for training the model that did not include the symmetry breaking token in Figure 4.13 (left).

NVP parameters	
Intermediate representation size	256
Number of transformations	8
GNN parameters	
Molecular representation size	64
Atom representation size	64
Depth	2
ENN hidden dimension	128
Gather embedding hidden dimension	128
Message passes	3
Message size	64
SchNet parameters	
Representation size	64
Number of interaction blocks	3
Cutoff radius	5 Å
Radial basis functions	25

Table A.2: Hyperparameters used for training the model that with the symmetry breaking token in Figure 4.13 (right) and the posterior energy-based learning shown in Figure 4.17.

NVP parameters	
Intermediate representation size	256
Number of transformations	10
GNN parameters	
Molecular representation size	64
Atom representation size	64
Depth	2
ENN hidden dimension	128
Gather embedding hidden dimension	128
Message passes	3
Message size	64
SchNet parameters	
Representation size	64
Number of interaction blocks	3
Cutoff radius	5Å
Radial basis functions	25
Energy-based learning parameters	
w_U	0.5
w_{ML}	1
Number of conformations	256

B

Appendix 2: Further results

In this Appendix we will show some additional results which are worth showing but less relevant than those previously commented in the Results (4) Section.

B.1 Reducing and increasing the size of the molecules

In Figure B.1 we show the evolution of different metrics during learning for the scenarios in which we aimed to reduce and increase the size of the generated molecules. It can be observed that in both cases, the fraction of valid and properly terminated molecules increases with the number of learning steps taken by the agent. However, when reducing the size of the molecules, the fraction of unique molecules also decreases. This makes sense, since there are less possible combinations of atoms the smaller the molecules. And, agreeing with this reasoning, the fraction of unique molecules does not decrease when increasing the size of the molecules.

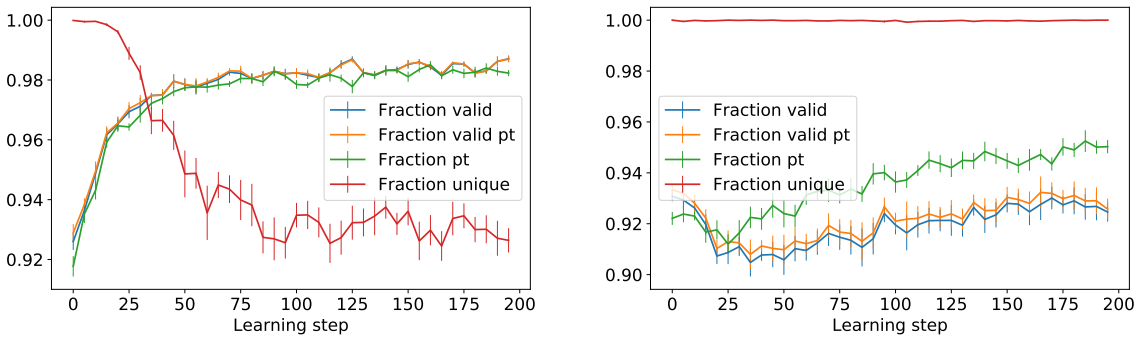


Figure B.1: Evolution of the fraction of valid, properly terminated, valid of those properly terminated and unique molecules during learning. Left: reducing the size of the molecules using Equation 3.2. Right: increasing the size of the molecules using Equation 3.3. The values are computed for 1000 molecules as an average over 10 runs, and the error bars correspond to the standard deviation of the mean.

In Figures B.2 and B.3 a comparison of the features of the generated molecules against those in the training data of GraphINVENT is shown for reducing and increasing the size of the molecules respectively. In can be observed in these Figures all properties remain mostly the same but the number of heavy atoms per molecule,

as we had desired.

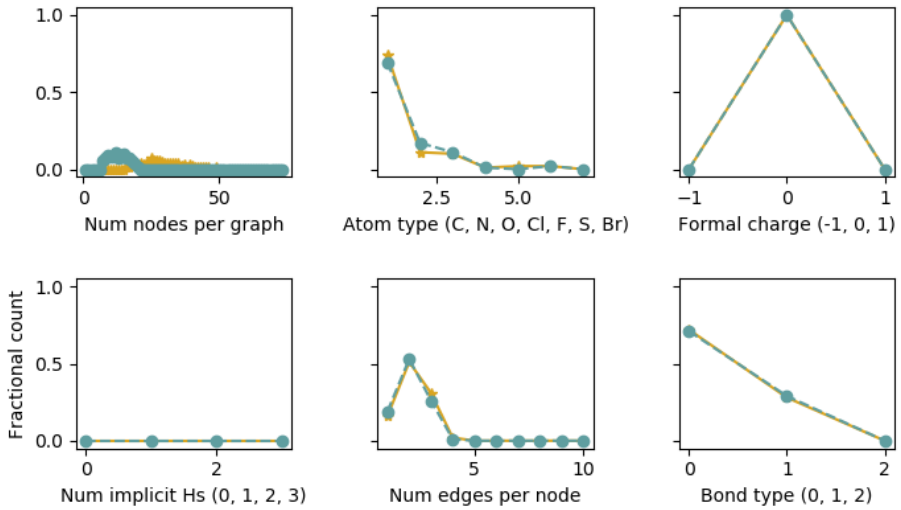


Figure B.2: Comparison of features of the molecules in the training set (orange) and those generated by the fine-tuned model (blue) for generating smaller molecules.

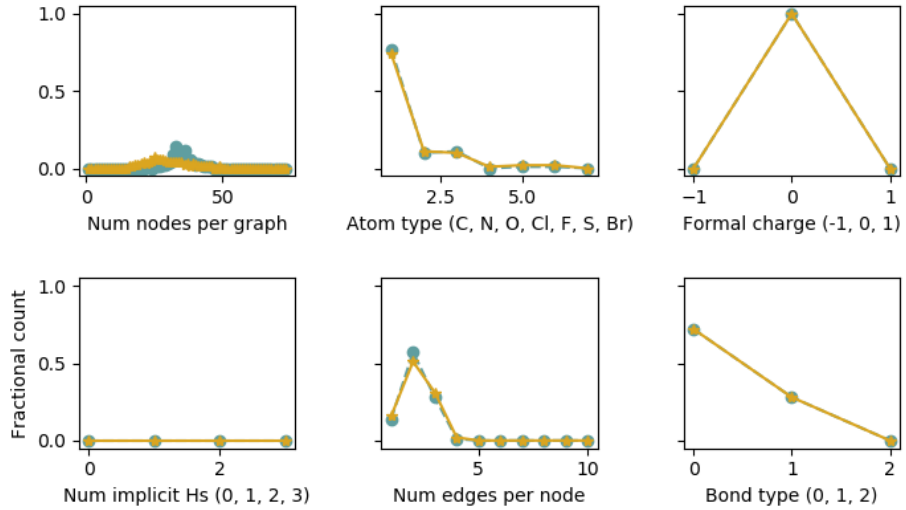


Figure B.3: Comparison of features of the molecules in the training set (orange) and those generated by the fine-tuned model (blue) for generating larger molecules.

B.2 Promoting drug-like molecules

In Figure B.4 the dependence of the QED values with the number of heavy atoms per molecule is shown. It can be seen that, on average, molecules with a number of atoms between 18 and 24 have larger QED and therefore, will have larger associated scores in Equation 3.4.

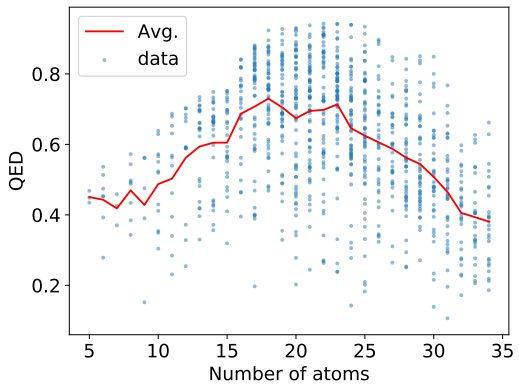


Figure B.4: Scatter plot of the QED of the molecules in the test set of the data used to train GraphINVENT against their number of nodes together with the average QED for each number of nodes.

In Figure B.5 a comparison of the features of the generated molecules against those in the training data of GraphINVENT is shown for the scenario in which we promoted the generation of drug-like molecules. In can be observed in this Figure all properties remain mostly the same but the number of heavy atoms per molecule, due to the dependence of the QED with the number of nodes.

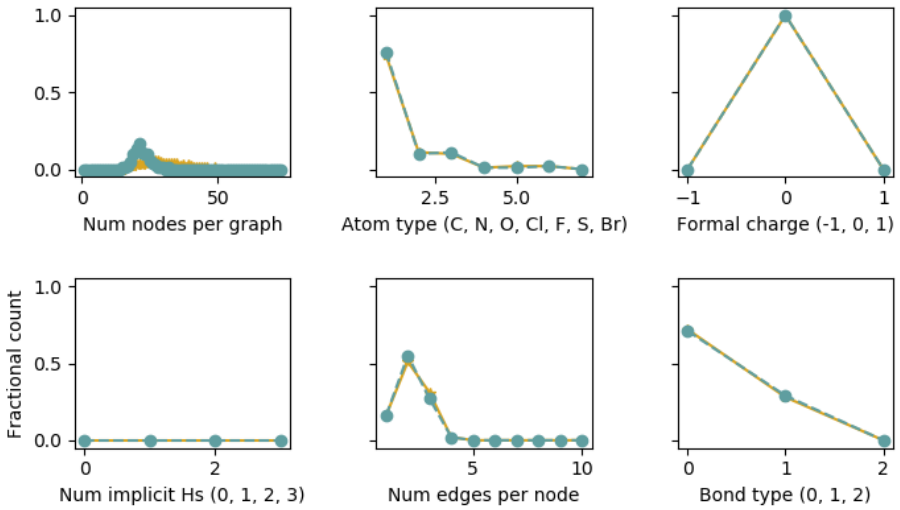


Figure B.5: Comparison of features of the molecules in the training set (orange) and those generated by the fine-tuned model (blue) for generating drug-like molecules.

B.3 Promoting DRD2 active molecules

In Figure B.6 a comparison of the features of the generated molecules against those in the training data of GraphINVENT is shown for the scenario in which we promoted the generation of DRD2 active molecules. In can be observed in this Figure all properties remain mostly the same but the number of heavy atoms per molecule. Nonetheless more differences are observed when compared to the features corresponding to generated molecules after fine-tuning with other scoring functions. This makes sense, since few molecules were active initially a more changes had to be made by the model to the properties of the molecules so that they become active.

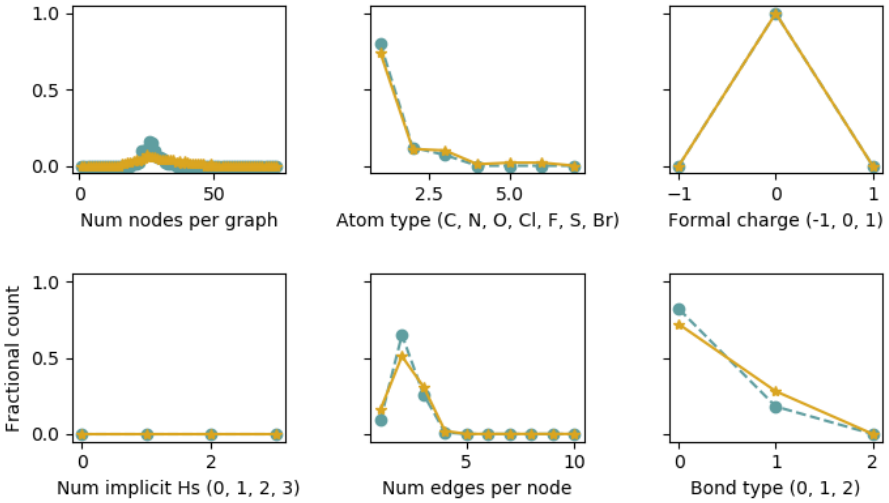


Figure B.6: Comparison of features of the molecules in the training set (orange) and those generated by the fine-tuned model (blue) for generating DRD2 active molecules.

In Figure B.7 the average number of nodes and score is shown for different jobs with the same set of best hyperparameters. It can be observed, different values are obtained for different runs, however we must highlight all of them manage to generate active molecules and they all start doing so in the first few learning steps.

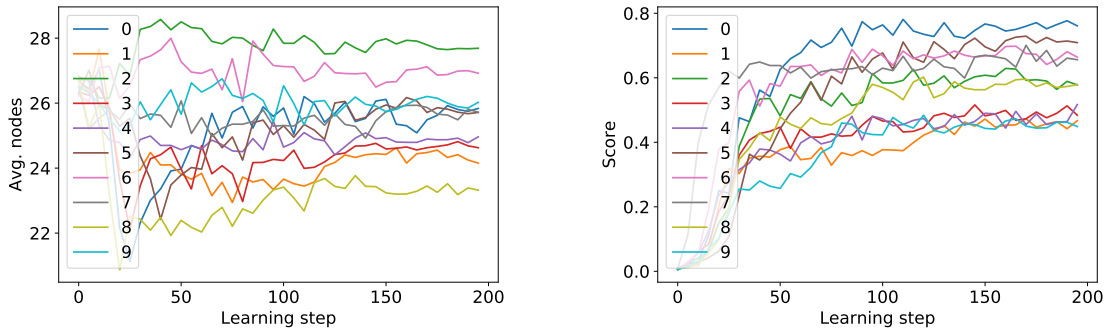


Figure B.7: Average number of nodes (left) and score (right) of the generated molecules for different jobs with $\alpha = 0.5$ and $\sigma = 20$. The number of molecules generated to compute these metrics is 1000.

C

Appendix 3: Justifying the new definition of loss function in the RL framework

The goal of this Appendix is to show further results on how the new term we have added to the loss function (Equation 3.1) adds stability to training and achieves better results compared to the loss defined in REINVENT (which corresponds to $\alpha = 0$ in our setting).

C.1 Reducing the size of the molecules

When reducing the size of the molecules, one can observe in Figure C.1 that including the term in which the best agent generates molecules ($\alpha = 0.5$) significantly improves the results: the average number of nodes is much closer to the goal value of 10 with 12 for $\alpha = 0.5$ whereas it is 18 for $\alpha = 0$.

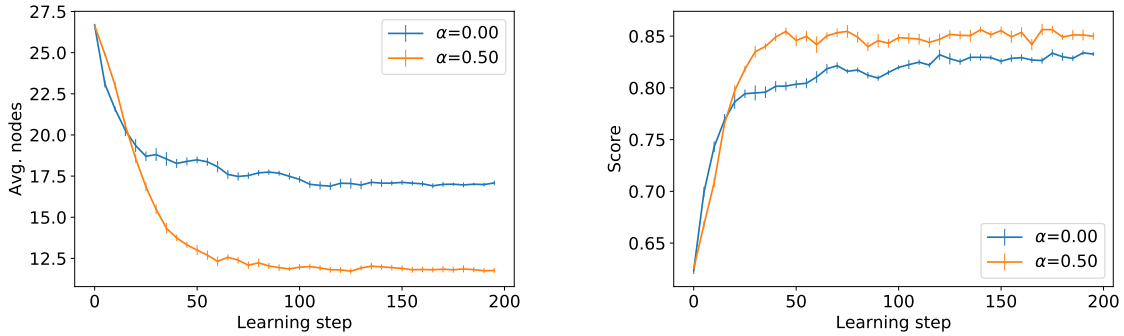


Figure C.1: Average number of nodes (left) and score (right) of the generated molecules for two different values of α with $\sigma = 20$. The score is computed for 1000 molecules and averaged over 10 different runs. The error bars shown correspond to the standard deviation of the mean.

C.2 Increasing the size of the molecules

A similar behaviour is observed in Figure C.2, where the average number of nodes does not change for $\alpha = 0$, whereas it does approach the goal value of 40 for $\alpha = 0.5$.

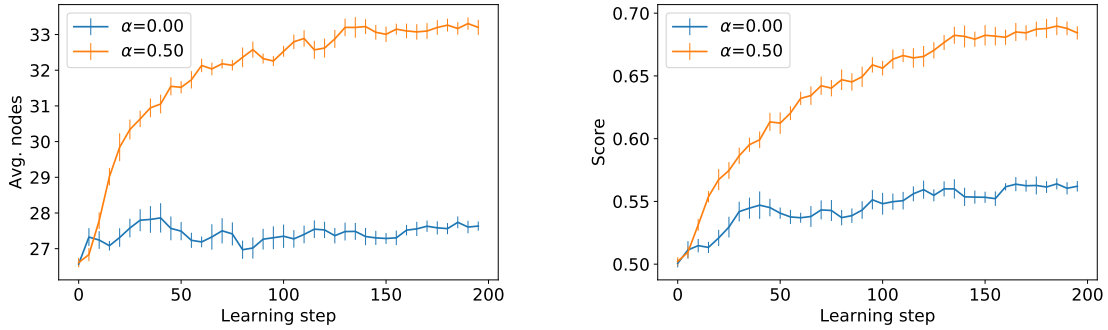


Figure C.2: Average number of nodes (left) and score (right) of the generated molecules for two different values of α with $\sigma = 20$. The score is computed for 1000 molecules and averaged over 10 different runs. The error bars shown correspond to the standard deviation of the mean.

C.3 Promoting drug-like molecules

Again, when using a scoring function based on the QED, the new loss function achieves a better score than the old one, as it can be observed in Figure C.3.

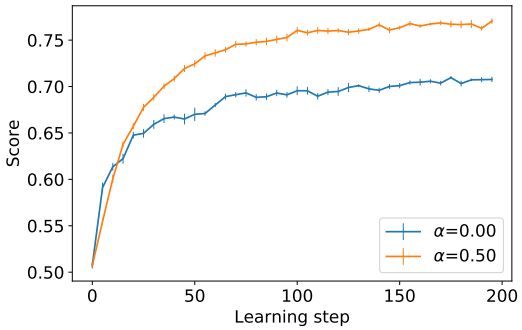


Figure C.3: Average score of the generated molecules for two different values of α with $\sigma = 20$. The score is computed for 1000 molecules and averaged over 10 different runs. The error bars shown correspond to the standard deviation of the mean.

C.4 Promoting DRD2 active molecules

Finally, when analysing the behaviour for both values of α for the scoring function defined in Equation 3.5, we observe the new definition helps to stabilise learning. In this case, the score is discrete, either 0 or 1, and the model only learns from the good examples (active molecules i.e. molecules scored 1). As there are not many of them at the beginning, it is specially good in this case to keep track of the model able to generate the largest amount of active molecules so as to make learning easier by reminding the agent of the set of actions taken to build them. Otherwise, we sometimes observe that the model is not able to start learning how to generate them,

getting lost in the process as it is shown in Figure C.4. Additionally, using $\alpha = 0.5$ is able to accelerate learning as well as achieving better and more robust results.

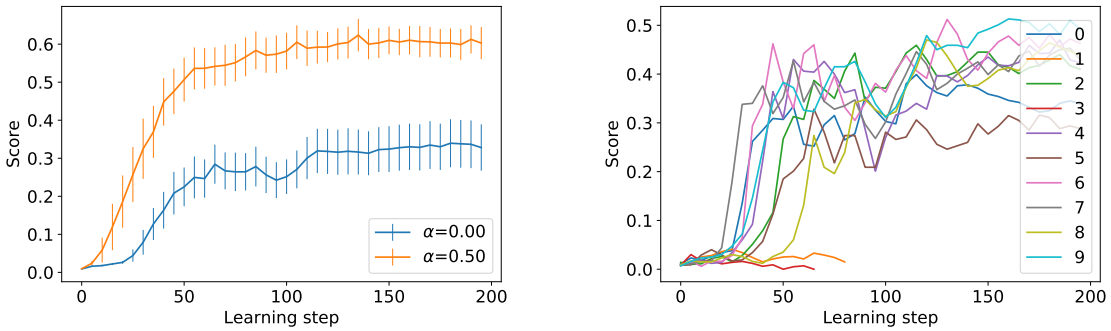


Figure C.4: Average score (left) and scores (right) of the generated molecules for different runs with $\sigma = 20$ and $\alpha = 0$. The score is computed for 1000 molecules.