UNIVERSITY OF AMSTERDAM

MSc Artificial Intelligence
Master Thesis

Deep Generative Models for Graphs: VAEs, GANs, and reinforcement learning for *de novo* drug discovery

by
Nicola De Cao
11286571

September 2018

36 ECTS
September 2017 - July 2018

Supervisor:
Thomas Kipf, MSc

Assessors:
Prof. Max Welling
Prof. Efstratios Gavves

University of Amsterdam
Informatics Institute

Acknowledgements

I would first like to thank my thesis supervisor Thomas Kipf who was always find time for me when needed in the past year and a half. He always supported my work and ideas even during my honors programme.

I would also like to thank Luca Falorsi, Tim R. Davidson, Fabrizio Ambrogi, Marco Federici, Luca Simonetto, Pim de Haan, Jonas Köhler, and Jose Daniel Gallego for passionately sharing with me the past two years of their lives. We shared our passions and dreams, and we had a lot of fun together as well.

I cannot forget Wilker Aziz, Patrick Forré, and Efstratios Gavves who always helped me develop critical thinking and inspired my work.

Finally, I thank my beloved Luisa as well as my parents for their never-ending love and support for my dreams.

Abstract

Deep generative models are a rapidly advancing area of research in machine learning. They recently have been proven to effectively deal with continuous Euclidean data such as sounds, images, and videos. However, how to generate non-Euclidean and structured abstract data, such as graphs, is still an open question. In this work, we propose and investigate how to approach this problem through the use of three popular techniques: Variational Auto-Encoders, Generative Adversarial Networks, and Reinforcement Learning. We show what the main advantages and drawbacks of each of these techniques are. In particular, we explore this problem within the setting of *de novo* drug discovery, i.e., generation of chemical drugs for finding new medications. We propose techniques to overcome some of the disadvantages of each method and to generate high-quality molecule structures as labeled graphs.

Contents

List of Figures

List of Tables

Part I

Main

Chapter 1

Overview

Drug design is the process of finding new drugs based on the knowledge of a biological system. In particular, drug discovery is one of its earliest steps which consists in selecting a number of candidates which will be further considered for experimentation. Computer systems frequently guide modern design processes. Usually, two direct methods select potential drug candidates: screening large compound libraries searching for existing molecules with some desired properties (Merz Jr et al., 2010), and the designing of new unknown ones. The latter is commonly referred as *de novo* drug discovery (Klebe, 2000). With this work, we mainly aim to explore modern deep learning generative models such as variational auto-encoders (Kingma and Welling, 2013) and generative adversarial networks (Goodfellow et al., 2014) as novel tools for generating new drug candidates. Additionally, we study the use of reinforcement learning as a method to bias the generative process towards having some desirable properties (e.g., molecules with high solubility in water).

1.1 Motivation

The drug design process is well-known to be expensive, both in terms of monetary and time resources. Even small improvements at any of its stages could save much human effort. In Figure 1.1 we give a rough intuition of the time scale that is required from the earliest stage of research to consumer sales to have a final medication ready for the market. We also approximately indicate four principal phases of drug design, and how many compounds are considered in each of them. Notice that on average, it requires 12-15 years of research and experimentation to have a single drug ready to be used. Drug discovery is the first step in this long process, and it aims to result in a set of

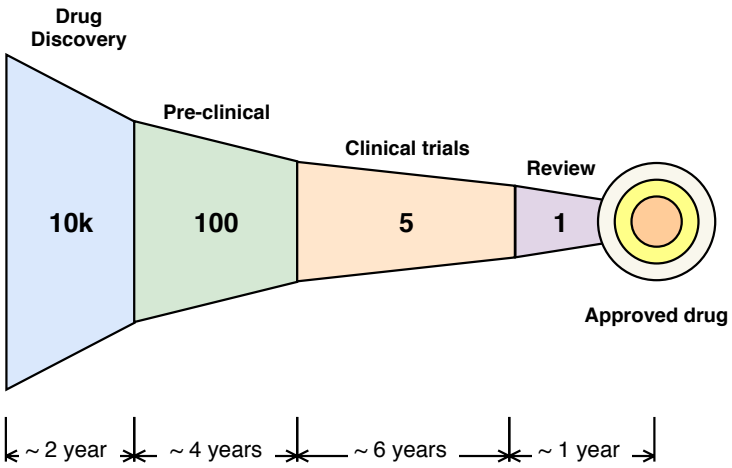


Figure 1.1: The drug design process. We show approximately how many compounds are considered in each phase, and how many years are expected to be spent on each of them[1].

≈10k compounds to be considered as potential candidates in future stages.

There are two major approaches to drug discovery, and they are commonly identified as ligand-based and structure-based (Merz Jr et al., 2010). Ligand-based drug discovery (or indirect drug discovery) is based on the knowledge of other molecules that bind to the biological target of interest. The structure-based drug discovery (or direct drug discovery) is based on virtual screening (i.e., searching large databases of molecules to find those fitting) or designing new unknown ligands (referred as *de novo*). These approaches are usually applied in combination resulting in a large number of candidates ready for pre-clinical tests.

When *de novo* methods are used, researchers first study the properties of the compounds they are looking for. Usually, they manually design part of the molecule since it has to match a known target. Subsequently, computer-aided systems identify/generate new molecules that can potentially interact with such target (Borhani and Shaw, 2012). These systems are usually rule-based, and they use knowledge from biology and chemistry developed in years of human-guided drug design. Although these systems are well-established, modern advances in machine learning, especially in generative models, proved to be useful in many areas, including chemistry and pharmaceutical sciences.

Generative models approach the problem of creating new data. Recently, they have been proven to effectively deal with continuous Euclidean data such as sounds, images, and videos (Kingma and Welling, 2013; van den

[1]Data sourced from fragmented information in http://www.cancerresearchuk.org

Oord et al., 2015; Radford et al., 2016; Vondrick et al., 2016). Unfortunately, molecular structures are not so simple to deal with. Molecules are complex structures that can be represented in multiples ways. Two popular approaches are the use of SMILES (simplified molecular-input line-entry system) strings or labeled graphs. The SMILES is a language-based molecular representation that can encode every compound in a string of characters. A compound can also be represented with a labeled graph where nodes are atoms and edges are bonds. The labels indicate which type of atoms and which type of bonds. In general, the generation of non-Euclidean and structured abstract data, such as text or graphs, is considered to be hard, and still an unsolved problem.

Recent works (Gómez-Bombarelli et al., 2016; Kusner et al., 2017; Dai et al., 2018; Guimaraes et al., 2017) have explored the direction of generative models capable of dealing with structured languages. In particular, these work framed this problem as a generation of SMILES strings. However, models that use SMILES have to face the issue of generating valid strings that correspond to plausible molecules. Generating small molecules is easy, but these methods may not scale well in real-world industrial applications. Indeed, other approaches (Simonovsky and Komodakis, 2018; You et al., 2018; Jin et al., 2018; Samanta et al., 2018) explored the promising direction of generating graph structures that represent compounds. These approaches result in much higher valid outputs. On the other hand, framing the problem on graphs might present other issues such as graph isomorphism under permutation of the nodes (e.g., when evaluating the likelihood of a graph in probabilistic models). Notice that any serialization of a graph (also using SMILES) will suffer from this problem.

In this work, motivated by the discussion above, we study how to approach *de novo* drug design using generative models for labeled graphs.

1.2 Contributions

We identify two main contributions in this thesis. The first contribution is presented in Sections 3.2, 3.3, and 3.4 where we propose a general framework (not only for molecules) for encoding and decoding discretely labeled graphs. In particular, graphs encoding is permutation invariant with respect to nodes. We also propose multiple decoding function variations. Each of them uses a graph-level vector that represents a graph as a whole (i.e., it is not decomposed in a set of nodes). Additionally, we show how we use discretely labeled graphs to represent molecules. We use these building blocks to construct both variational auto-encoder and generative adversarial network

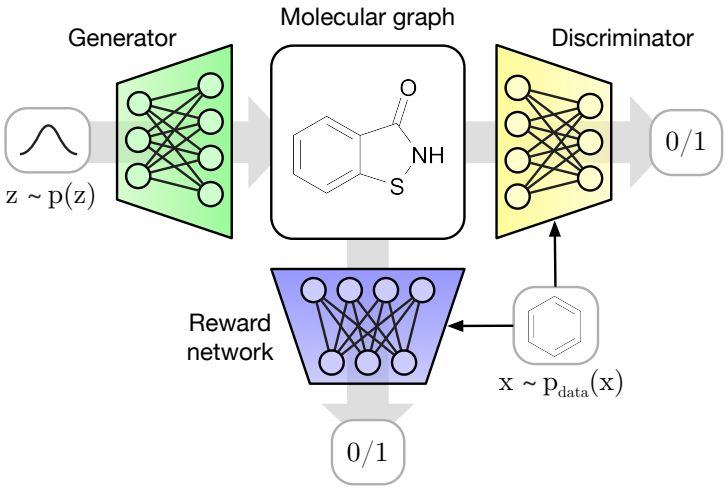


Figure 1.2: Schema of MolGAN from our previous preliminary work (De Cao and Kipf, 2018). In a GAN setting, a vector sampled from a prior is fed to the generator which outputs a graph representation of a molecule. The discriminator classifies whether the molecular graph comes from the generator or the dataset. In addition, a reinforcement learning component bias the process. The reward network estimates the reward (i.e., the chemical properties of a particular molecule provided by an external software) providing a policy gradient to the generator.

for generating such structures.

The second main contribution is presented in Section 3.5 where we show how to use reinforcement learning (RL) in combination with a graph generative process. In particular, we employed a deterministic policy gradient to overcome the high-dimensional action space complexity. Through extensive experimentation, we show the ability of reinforcement learning to bias the generative process towards desirable properties. Reinforcement learning provides additional feedback to the generation resulting in an advantageous convergence. Indeed, the generation of a discrete structure imposes a serious learning challenge due to its indirect differentiability (i.e., generally, we cannot compute the exact gradient of a discrete output by a neural network). Moreover, in the space of graphs, only a tiny amount are valid molecules. With experimental results, we show the benefits, but also the limitations, of combining reinforcement learning with both variational auto-encoder and generative adversarial network for graphs.

In a previous preliminary work (De Cao and Kipf, 2018), we showed a limited study on the combination between WGANs and RL. In Figure 1.2, we show the proposed model with a brief introduction. In this thesis,

we aim in exploring more advanced techniques (e.g., the use of variational auto-encoders or feature matching for WGANs) as well as presenting a more extensive introduction to all employed methods. We will also present more model variations, experiments, evaluations, and discussions.

1.3 Organization

This thesis is organized as follows: we first provide some background knowledge in Chapter 2 to introduce the reader to the main concepts needed to understand our work. These include variational auto-encoders, generative adversarial networks, reinforcement learning, and graph convolution networks.

Chapter 3 exposes several methods and variations to address *de novo* drug design. We propose the use of both graph-based variational and adversarial models in combination with reinforcement learning. We show which issues we want to address, how we build such models, and how we intend to evaluate them. In Chapter 4 we briefly list several related works to ours as well as presenting relevant literature to the methods we use.

Continuing to Chapter 5, we provide a series of experiments to evaluate our methods. We discuss results obtained by these experiments as well as present an extensive discussion. Additionally, we show how our work compares against recent state-of-the-art models.

Finally, in Chapter 6, we draw final considerations of this work, and we propose possible future directions in this field of research. We report additional derivations and plots in the Appendix.

Chapter 2

Background

This chapter aims to give a brief introduction to the main concepts that are used throughout this thesis. However, we assume that the reader is familiar with machine learning, probability theory, and neural networks. We first present two core generative models namely the variational auto-encoder and generative adversarial network in Sections 2.1 and 2.2 respectively. Then, we introduce the learning paradigm of reinforcement learning in Section 2.3, and graph convolution network, an essential deep-learning framework used to deal with graphs, in Section 2.4. Definitions and notations of this chapter are fundamental to the understanding of later chapters.

2.1 Variational Auto-Encoders

First introduced by Kingma and Welling (2013), the variational auto-encoder (VAE) is an unsupervised generative model that allows performing variational inference using an auto-encoding architecture.

An auto-encoder (AE) is a particular neural network architecture used for unsupervised learning of efficient representations (Hinton and Salakhutdinov, 2006). The objective of an AE is to learn two functions: a function e, that produces an encoding/representation in an embedded space $\mathcal{Z}$ from a data space $\mathcal{X}$, and a function d, that decodes such representations back to the original space. AEs are unsupervised learning models since the learning task is inferring functions, from unlabeled data, that describe hidden structures/patterns. AEs are typically used for dimensionality reduction since the size of latent representations is usually smaller than the original data space.

An auto-encoder model can be formalized as follows:

$$
\begin{aligned}
e_\phi &: \mathcal{X} \to \mathcal{Z} \\
d_\theta &: \mathcal{Z} \to \mathcal{X} \\
\phi^*, \theta^* &= \arg\min_{\phi,\theta} \sum_{i=1}^{N} \mathcal{L}(\mathbf{x}^{(i)}, \tilde{\mathbf{x}}^{(i)})\big|_{\tilde{\mathbf{x}}^{(i)}=d_\theta\,(e_\phi \mathbf{x}^{(i)}))} \,,
\end{aligned} \tag{2.1}
$$

where e_ϕ and d_θ are respectively the encoder and decoder functions parameterized by ϕ and θ, $\mathcal{X}$ and $\mathcal{Z}$ are the data space and the latent space. $\mathcal{L} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}^+$ is any loss function that assigns a positive penalty (loss) to a reconstruction with respect to the original observation (zero loss corresponds to perfect reconstruction). Notice that in general a loss function can be also negative, but a positive loss is more interpretable. $X = \{\mathbf{x}^{(i)}\}_{i=1}^N$ and $\tilde{X} = \{\tilde{\mathbf{x}}^{(i)}\}_{i=1}^N$ are the dataset and the reconstructed dataset respectively. This general formulation can be instantiated as a fully differentiable model when both e_ϕ and d_θ are implemented as neural networks and then parameters can be learned and optimized with stochastic gradient descent.

AEs are very effective in dimensionality reduction and anomaly detection (Hinton and Salakhutdinov, 2006; Sakurada and Yairi, 2014), but their usefulness as generative models is limited. Indeed, one may use d_θ to generate new data-points feeding with elements form $\mathcal{Z}$. However, there is no underlying nor pre-defined distribution over the space $\mathcal{Z}$ to sample from. Fortunately, recently, auto-encoders in combination with variational inference have been shown to be useful for learning generative models of data.

Variational learning Variational auto-encoder models make use of the auto-encoder architecture while additionally modeling observed and latent variables using a probabilistic framework. In addition to an AE, in VAEs, assumptions concerning the distribution of latent variables are made. This addition allows VAEs to be used as generative models. Let X be a random variable that models the observation (data), and Z be a hidden random variable that generates data. The relationship between these two variables can be represented using the graphical model in Figure 2.1 where θ and ϕ are parameters of the distributions of the generative model and the variational approximation (inference model) respectively. In this setting, the joint probability distribution $p_\theta(\mathbf{x}, \mathbf{z})$ can be decomposed using $p_\theta(\mathbf{x}, \mathbf{z}) = p_\theta(\mathbf{x}|\mathbf{z})\, p_\theta(\mathbf{z})$ such that the latent variables are drawn independently from a prior $\mathbf{z} \sim p_\theta(\mathbf{z})$ while data are drawn from a likelihood that is conditioned on latent variables $\mathbf{x} \sim p_\theta(\mathbf{x}|\mathbf{z})$. Moreover, we can calculate the posterior in order to make in-

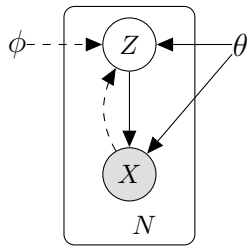


Figure 2.1: Solid lines indicate the generative model $p_\theta(\mathbf{x}, \mathbf{z}) = p_\theta(\mathbf{x}|\mathbf{z})\, p_\theta(\mathbf{z})$ where dashed lines indicate the variational approximation $q_\phi(\mathbf{z}|\mathbf{x})$ of the intractable posterior $p_\theta(\mathbf{z}|\mathbf{x})$.

ference on Z after observing X:

$$p_\theta(\mathbf{z}|\mathbf{x}) = \frac{p_\theta(\mathbf{x}, \mathbf{z})}{p_\theta(\mathbf{x})} = \frac{p_\theta(\mathbf{x}, \mathbf{z})}{\int p_\theta(\mathbf{x}, \mathbf{z})\, d\mathbf{z}} \,. \tag{2.2}$$

Moreover, in order be able to learn, we assume that both the prior and the likelihood come from a well-known parametric family of distributions with almost everywhere differentiable probability density functions. The objective is then to learn such parameters approximating probability functions with neural networks and optimizing them via stochastic gradient descent (SGD). More precisely, we are interested in minimizing the reconstruction error, similar to the AE, but here it is formulated as maximizing the log-evidence $\log p_\theta(\mathbf{X})$. The log-evidence is computed over all the observed data $\mathbf{X} = \{\mathbf{x}^{(i)}\}_{i=1}^N$ to obtain $\theta^* = \arg\max_\theta \log p_\theta(\mathbf{X})$. The log-evidence is decomposed into a sum over the marginal probabilities of individual data-points: $\log p_\theta(\mathbf{X}) = \sum_{i=1}^N \log p_\theta(\mathbf{x}^{(i)})$. Then we can express the evidence of a single data-point as:

$$\log p_\theta(\mathbf{x}^{(i)}) = \log \int p_\theta(\mathbf{x}^{(i)}, \mathbf{z})\, d\mathbf{z} \,. \tag{2.3}$$

Unfortunately, since probability densities are parameterized by neural networks, marginalizing over the latent variables is generally intractable. Furthermore, in practice, even with a Monte Carlo estimate, it would be too slow since it typically involves a sampling loop per data-point which is expensive for large datasets. One way to overcome this issue is to introduce an approximation of the intractable true posterior $q_\phi(\mathbf{z}|\mathbf{x}) \approx p_\theta(\mathbf{z}|\mathbf{x})$ and maximize a lower bound of the log-evidence. This bound (see full derivation in Appendix B.1) is the evidence lower bound (ELBO):

$$\log p_\theta(\mathbf{x}^{(i)}) \geq \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}^{(i)})}\left[\log p_\theta(\mathbf{x}^{(i)}|\mathbf{z})\right] - D_{KL}\left[\, q_\phi(\mathbf{z}|\mathbf{x}^{(i)}) \parallel p_\theta(\mathbf{z}) \,\right] \,. \tag{2.4}$$

where D_{KL} is the Kullback–Leibler divergence (Kullback and Leibler, 1951). The ELBO is generally easier to compute and much more convenient to optimize than the true evidence. In practice, in order to have a gradient estimator, Kingma and Welling (2013) introduced the Stochastic Gradient Variational Bayes (SGVB) algorithm, and they applied that using Normal distributions for both prior and posterior. They used a mean-field assumption in order to make the problem tractable, i.e. a standard multivariate Normal distribution $p_\theta(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$ as a prior, and a fully-factorized multivariate Normal distribution $q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}, diag(\boldsymbol{\sigma}^2))$ as posterior. SGVB optimizes the ELBO using a single point estimation to approximate expected values. A Monte Carlo estimate would lead to less variance but also to an overhead in computation. Additionally, they introduced the *reparameterization trick*[1] to sample from the posterior in order to make the ELBO differentiable with respect to ϕ. The loss function per data-point to minimize with respect to both θ and ϕ with stochastic gradient descent is then:

$$\mathcal{L}(\mathbf{x}^{(i)}; \theta, \phi) = \underbrace{-\log p_\theta(\mathbf{x}^{(i)}|\mathbf{z}^{(i)})}_{\text{reconstruction loss}} + \underbrace{D_{KL}\left[\, q_\phi(\mathbf{z}^{(i)}|\mathbf{x}^{(i)}) \parallel p(\mathbf{z}^{(i)}) \,\right]}_{\text{variational loss}}, \tag{2.5}$$

with $\mathbf{z}^{(i)} \sim q_\phi(\mathbf{z}|\mathbf{x}^{(i)})$.

This loss is very similar to the formalism in Equation 2.1. As a matter of fact, a VAE can be seen as AE, where $e_\phi = q_\phi(\mathbf{z}|\mathbf{x})$ and $d_\theta = p_\theta(\mathbf{x}|\mathbf{z})$, with an additional loss component that acts regularizing the latent space structure with respect to a prior distribution. In this way, a VAE does not only learn efficient encoding and decoding functions but also allows sampling from $\mathcal{Z}$. Therefore, after training, one can use the encoder $e_\phi : \mathcal{X} \to \mathcal{Z}$ to obtain latent representations, the decoder $d_\theta : \mathcal{Z} \to \mathcal{X}$ to obtain reconstructions, and $d_\theta(\mathbf{z})\big|_{\mathbf{z}\sim p(\mathbf{z})}$ to generate new samples.

2.2 Generative Adversarial Networks

VAEs do not optimize a generative process directly. Indeed, VAEs are trained using the ELBO which maximize a lower bound of the evidence while aligning the outputs of the encoder to match a prior distribution. One may desire to have a model which optimizes some generative objective to learn to generate samples that resemble data-points form the dataset directly. This kind of objective should assess the quality of the generative process. VAEs do not address that since the characteristics of the generation process are assessed

[1] $\mathbf{z} \sim \mathcal{N}(\boldsymbol{\mu}, diag(\boldsymbol{\sigma}^2)) \equiv \mathbf{z} \sim \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \mathcal{N}(\mathbf{0}, \mathbf{I})$ where $\odot$ is element-wise product.

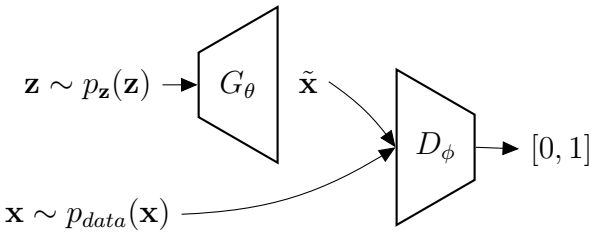


Figure 2.2: Schema of naïve GAN architecture. *Top:* $\mathbf{z}$ is drawn from the prior distribution $p(\mathbf{z})$ and fed to the generator function $\tilde{\mathbf{x}} = G_\theta(\mathbf{z})$. *Bottom:* $\mathbf{x}$ is drawn from the empirical distribution (dataset). *Right:* the discriminator function D_ϕ is a classifier trained to discriminate real data $\mathbf{x}$ (label 1) from generated data $\tilde{\mathbf{x}}$ (label 0).

by the *reconstruction loss*, which is well-defined, but they do not optimize the generation of new samples.

Conversely, first introduced by Goodfellow et al. (2014), the generative adversarial network (GAN) architecture is designed to be a pure generative model. Differently from a VAE, there is no approximate posterior distribution $q(\mathbf{z}|\mathbf{x})$, and the model directly learns to generate samples. Moreover, GANs are likelihood-free models since there is no direct computation of the likelihood. The main idea behind GANs is to have a part of the model, called discriminator, specifically designed and trained to provide feedback, in form of a gradient, to the generated samples. In particular, the discriminator learns to disambiguate real from generated samples allowing optimizing the generator towards matching the empirical data distribution.

Hence, a GAN consist of two main components: a generative model G_θ and a discriminative model D_ϕ. The generator G_θ, parameterized by θ, learns the data distribution to sample new data-points. The discriminator D_ϕ, parameterized by ϕ, learns to classify whether samples came from the empirical distribution $p_{data}(\mathbf{x})$ (dataset) rather than from G_θ to provide credit assignment in form of a gradient to the generator. Those two models are implemented as neural networks and trained simultaneously with SGD. The model is outlined in figure 2.2.

Generator The objective of the generator is to learn a transformation from a noise vector drawn from a prior distribution $p_{\mathbf{z}}(\mathbf{z})$ into a sample $\tilde{\mathbf{x}}$ that resembles the data distribution closely enough to fool the discriminator into classifying generated samples as real samples from the dataset. More formally, the loss function with respect to G_θ is the negative log-probability

of generated samples to belong to the empirical distribution:

$$\mathcal{L}(\mathbf{z}^{(i)};\theta) = -\log D_\phi(G_\theta(\mathbf{z}^{(i)})) \ , \tag{2.6}$$

where $\mathbf{z}^{(i)} \sim p_\mathbf{z}(\mathbf{z})$. Popular choices for $p_\mathbf{z}(\mathbf{z})$ are either an multivariate Uniform $\mathcal{U}(-1,1)$ or a standard multivariate Normal distribution $\mathcal{N}(\mathbf{0},\mathbf{I})$.

Discriminator The purpose of the discriminator is to provide credit assignment to generated samples. The discriminator is indeed needed to provide a gradient to the generator function allowing optimization via SGD. The objective of the discriminator is to classify samples as generated rather than real. Its output is the probability of a sample of being drawn from the dataset. Then, the loss function with respect to D_ϕ is the sum of two negative log-probabilities:

$$\mathcal{L}(\mathbf{x}^{(i)},\tilde{\mathbf{x}}^{(i)};\phi) = -\log D_\phi(\mathbf{x}^{(i)}) - \log(1 - D_\phi(\tilde{\mathbf{x}}^{(i)})) \ , \tag{2.7}$$

where $\mathbf{x}^{(i)} \sim p_{data}(\mathbf{x})$, and $\tilde{\mathbf{x}} = G_\theta(\mathbf{z}^{(i)})$ with $\mathbf{z}^{(i)} \sim p_\mathbf{z}(\mathbf{z})$. Notice that, the term $\log D_\phi(\mathbf{x}^{(i)})$ indicates the log-probability of a sample drawn from the empirical distribution to be classified as drawn from it. Conversely, $\log(1 - D_\phi(\tilde{\mathbf{x}}))$ indicates the log-probability of a generated sample to have not been drawn from the empirical distribution, therefore, to have been generated.

Optimization Since those two model have different objectives, they can be seen as two players in a minimax game: $\min_\theta \max_\phi V(G_\theta, D_\phi)$ where

$$V(G_\theta, D_\phi) = \mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}\left[\log D_\phi(\mathbf{x})\right] + \mathbb{E}_{\mathbf{z}\sim p_\mathbf{z}(\mathbf{z})}\left[\log(1 - D_\phi(G_\theta(\mathbf{z}))\right] \ , \tag{2.8}$$

is the value function. The discriminator tries to maximize V in order to identify all real and generated samples correctly. The generator tries to minimize V in order to generate samples that resemble real ones. Goodfellow et al. (2014) proved that the global optimum exists and the only solution is when $p_{G_\phi} = p_{data}$ where p_{data} is the empirical distribution and p_{G_ϕ} is the model distribution such that $\tilde{\mathbf{x}} \sim p_{G_\phi}(\tilde{\mathbf{x}})$. Therefore, the algorithm converges when the generative model perfectly replicates the data generating process.

Notice that, within the assumption of a naïve GAN model, it is essential to balance the training between G and D. Indeed, when using backpropagation, the generator updates depend on the gradient received by the discriminator. A poorly trained discriminator would not provide a meaningful gradient so neither the generator would update well. On the other hand, an optimal discriminator might lead to a vanishing gradient for the generator

that would never update towards convergence. Thus, a parameter $n_{critic} \geq 1$ is chosen such that the discriminator is updated for n_{critic} times more than the generator at every training step.

In practice, a GAN might be hard to train for multiple reasons and Salimans et al. (2016) have suggested practical techniques such as mini-batch discrimination and feature matching to prevent undesired behaviors. Other works have explored the theoretical properties of the GAN divergence proposing a more general framework (Nowozin et al., 2016) and a different probability distance measure (Arjovsky et al., 2017). Such formulations provided new tools to ensure empirical convergence in some problems.

Mini-batch discrimination Among the most common failures that can occur when training GANs there is *mode collapse*. This undesired behavior occurs when the generator collapses to the point of always emitting the same sample no matter what its input is. Usually, the discriminator processes each sample independently and therefore there is no guarantee that the combination of multiple gradient signals is coordinated. Salimans et al. (2016) proposed to address this issue by making the discriminator able to look at multiple data points together and then avoiding uncoordinated generator updates. This technique called mini-batch discrimination is quite general, and it may vary depending on modeling choices. In particular, it depends on the discriminator architecture. In general, it consists in the use of a feature aggregation method (e.g., sum or mean of non-linear transformations of the input feature vector) in combination with individual features provided as inputs to the discriminator. In this way, at some point of the discriminator forward pass, each sample intermediate representation would be in part composed of a combination of features from the entire minibatch.

Feature matching Another proposed technique for preventing *mode collapse* is feature matching. This method relies on the idea of training the generator to match statistics of the real data instead of maximizing the GAN objective directly. The discriminator is still trained as a classifier, but its purpose is then to provide data statistics in form of representations from hidden layers. Let f be an intermediate layer of the discriminator or a combination of more, then the new generator objective becomes

$$\mathcal{L}(\theta) = \| \mathbb{E}_{\mathbf{x} \sim p_{data}} [f(\mathbf{x})] - \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}} [f(G_\theta(\mathbf{x})] \|^2 . \tag{2.9}$$

All other GAN training steps and model details remain unchanged. In this way, the generator is pushed to create samples that have the same statistics of the ones from the dataset.

Wasserstein GAN First introduced by Arjovsky et al. (2017), the Wasserstein GAN (WGAN) is an effective and powerful alternative to the original GAN objective optimization. WGAN minimizes a reasonable and efficient approximation of the earth mover's distance (EMD), also know as Wasserstein-1 distance, defined between two probability distributions. Formally, the Wasserstein distance between p and q, using the Kantorovich-Rubinstein duality (Villani, 2008), is defined as

$$D_W [\, p \parallel q \,] = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x \sim p(x)} [f(x)] - \mathbb{E}_{x \sim q(x)} [f(x)] \ , \tag{2.10}$$

where the supremum is over all the 1-Lipschitz functions $f : \mathcal{X} \to \mathbb{R}$. In general, a function $f : \mathcal{X} \to \mathcal{Y}$ is called K-Lipschitz continuous if given two metric spaces $(\mathcal{X}, d_\mathcal{X})$ and $(\mathcal{Y}, d_\mathcal{Y})$, there exists a $K \in \mathbb{R}_+$ such that $d_\mathcal{Y}(f(x_1) - f(x_2)) \leq K d_\mathcal{X}(x_1 - x_2)\ \forall x_1, x_2 \in \mathcal{X}$ (O'Searcoid, 2006). In the case of a GAN setting, f is the discriminator D_ϕ and it is parameterized with ϕ by a neural network, thus the optimization problem becomes

$$\min_\theta \max_\phi \ \mathbb{E}_{\mathbf{x} \sim p_{data}(\mathbf{x})} [D_\phi(\mathbf{x})] - \mathbb{E}_{\mathbf{z} \sim p_\mathbf{z}(\mathbf{z})} [D_\phi(G_\theta(\mathbf{z}))] \ , \tag{2.11}$$

where, in the original WGAN paper, weights ϕ are clamped to a fixed box $[-0.01, 0.01]$ in order to lie in a compact space and to make D_ϕ to be K-Lipschitz for some K. Note that if we replace $\|f\|_L \leq 1$ with $\|f\|_L \leq K$ then we end up with $K \cdot D_W [\, p(x) \parallel q(x) \,]$. Then, the loss function with respect to the generator parameters becomes

$$\mathcal{L}(\mathbf{z}^{(i)}; \theta) = -D_\phi(G_\theta(\mathbf{z}^{(i)})) \ , \tag{2.12}$$

and the loss function with respect to the discriminator parameters becomes

$$\mathcal{L}(\mathbf{x}^{(i)}, \tilde{\mathbf{x}}^{(i)}; \phi) = -D_\phi(\mathbf{x}^{(i)}) + D_\phi(\tilde{\mathbf{x}}^{(i)}) \ . \tag{2.13}$$

Empirically, there are several advantages of using WGAN instead of the standard GAN divergence. Namely WGANs i) do not require maintaining a careful balance in training of the discriminator and the generator (conversely to normal GAN models), ii) do not require a particular carefully design of balanced generator, and discriminator networks and iii) do prevent the *mode collapse* phenomenon better than normal GANs.

Improved WGAN Gulrajani et al. (2017) argued that the parameters clipping in WGAN can lead to undesired behavior and non-convergence. Moreover, it drastically reduces the expressiveness of the neural networks.

Therefore, they introduced a gradient penalty which is a soft constraint on the 1-Lipschitz continuity that does not suffer from the same problems. They then proposed WGAN with gradient penalty (WGAN-GP) which have been proven to be more stable and do not require any clipping. In order to satisfy the 1-Lipschitz continuity, the new loss function with respect to the discriminator is

$$\mathcal{L}(\mathbf{x}^{(i)}, \tilde{\mathbf{x}}^{(i)}; \phi) = \underbrace{-D_\phi(\mathbf{x}^{(i)}) + D_\phi(\tilde{\mathbf{x}}^{(i)})}_{\text{original WGAN loss}} + \underbrace{\lambda \left(\|\nabla_{\hat{\mathbf{x}}^{(i)}} D_\phi(\hat{\mathbf{x}}^{(i)})\| - 1 \right)^2}_{\text{gradient penalty}}, \quad (2.14)$$

where $\lambda > 0$ controls the weight of the gradient penalty.

Gulrajani et al. (2017) proved that the optimal discriminator has a gradient with norm equals to 1 almost everywhere. Therefore, it would be convenient to optimize while having a hard constraint $\|\nabla_{\mathbf{x}} D_\phi(\mathbf{x})\| = 1 \; \forall \mathbf{x} \in \mathcal{X}$. However, it is unfeasible to ensure while using neural networks and classic SGD. Instead, this is done with a point estimation, using random datapoints between real and generated samples. The mean squared error forces the model to have a gradient close to 1 over the manifold $\mathcal{X}$ and therefore approximately satisfying the 1-Lipschitz continuity. WGAN-GP demonstrates the same advantages of WGAN but allows more flexibility during learning avoiding weight clipping.

2.3 Reinforcement Learning

Reinforcement learning (RL) is a broad term in machine learning that indicates several techniques inspired by behaviorist psychology where intelligent agents learn interacting with an environment which provides positive or negative rewards according to their behavior. In RL, rewards are the fundamental component of learning since the latter is performed based on reward signals. In general, reinforcement learningstudies the problem setting of an artificial agent that takes actions to maximize a particular definition of a cumulative reward function (Sutton and Barto, 1998; Li, 2017).

Most of reinforcement learning problems are modeled as Markov Decision Processes (MDPs) where RL agents interact with environments over discrete time steps (an RL schema is shown in Figure 2.3). The Markov property states that the future depends only on the current state and action, but not on the past. More formally, let $\mathcal{S}$ be the space of environmental states and $\mathcal{A}$ be the action space, at each time step t, the agent receives a state $s_t \in \mathcal{S}$ as input and selects an action $a_t \in \mathcal{A}$ according to a policy. The policy is a function $\mathcal{S} \to \mathcal{A}$ that has to be learned through some optimization algorithm. A policy can be defined either on a continuous or discrete action space and be either

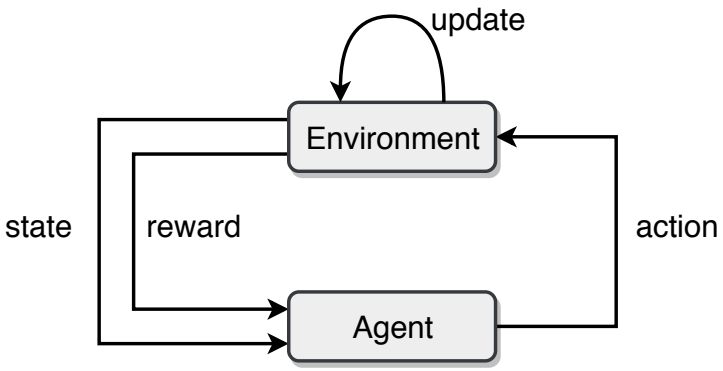


Figure 2.3: Reinforcement Learning schema: an *agent* in an *environment* acts through *actions* changing the environment. Subsequently, the modified environment updates its internal representation providing to the agent its new *state* as well as a reward. The reward is a scalar real number which indicates feedback on how good/bad was the previous action or how good/bad is the new state. Eventually, the agent acts again based on the new state.

stochastic or deterministic. A stochastic policy is represented by $\pi_\theta(s) = p_\theta(a|s)$ which is a parametric distribution in θ that assigns a probability density to actions $\in \mathcal{A}$ conditioned on a state s. The resulting action is then sampled $a \sim \pi_\theta(s)$. Conversely, a deterministic policy is represented by $a = \mu_\theta(s)$ which deterministically outputs an action.

After an agent acts, the environment changes according to its dynamics. In particular, it updates its observable state to s_{t+1} according to a, possibly stochastic, transition function $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$. Notice that the internal environmental status may differ for its observable state. In a full observable environment, the agent is assumed to observe the current environmental state which coincides with its internal status. In a partially observable environment, the agent has access to a restricted part of its status. Additionally, after transitioning from s_t to s_{t+1} with action a_t, the environment provides an immediate reward signal $r_t \in \mathbb{R}$ from a reward function $r(s_t, a_t)$. This process continues until it reaches a terminal state.

From each state, the agent objective is to maximize the discounted future expected return $\mathbb{E}[R_t]$ with $R_t = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k \, r_{t+k}\right]$ which is an accumulated episodic reward with a discount factor $\gamma \in (0, 1]$ that regulates the importance assigned to future rewards. The value function $V^\pi(s_t) = \mathbb{E}[R_t|s_t]$ of the policy π_θ is defined as the expected return starting at state s_t (Sutton and Barto, 1998). Similarly, the state-action value function $Q^\pi(s_t, a_t) = \mathbb{E}[R_t|s_t, a_t]$ of the policy π is the expected return starting at state s_t and performing the action a_t. An RL algorithm is designed to find a policy that

selects actions with maximum expected future return from all states.

The variety of RL algorithms and settings is vast, and we do not aim to provide a broad overview of them. In this work, we restrict the exploration of RL techniques with the use of a deterministic policy optimized with an off-policy gradient-based method (a detailed explanation will follow). In particular, we employ a simplified version of the off-policy deep deterministic policy gradient (DDPG) introduced by Lillicrap et al. (2016). We additionally present a general introduction to other essential RL concepts needed to understand this work.

Off-policy methods optimize an *estimation* policy, the policy that is evaluated and improved, independently from the policy used to generate behavior, called *behavior* policy. Conversely, on-policy methods estimate the value of a policy while using it. In an on-policy setting, learning algorithms optimize the expected return of the policy including the exploration steps. Off-policy algorithms can benefit from agents that behave with a certain degree of randomness, according to a *behavior* policy, to balance exploration, and meanwhile, still optimizing a deterministic *behavior* policy. Generally, behaving according to a deterministic policy may not provide enough exploration resulting in converging to poor policies.

Stochastic Policy Gradient and REINFORCE Policy gradient (PG) methods optimize a policy descending the gradient of a differentiable loss function. The idea behind these algorithms is to optimize a parameterizable policy function defining a loss based on expected rewards. Let then consider a stochastic policy $\pi_\theta : \mathcal{S} \to \mathcal{A}$, a performance objective $J(\pi_\theta) = \mathbb{E}[R_t|\pi_\theta]$ to maximize, indicating the expected future return using the policy π_θ, and the discounted state distribution ρ^π as a probability distribution over the state space $\mathcal{S}$ following the policy π_θ. We can then write the on-policy performance objective as

$$\begin{aligned} J(\pi_\theta) &= \int_{\mathcal{S}} \rho^\pi(s) \int_{\mathcal{A}} \pi_\theta(a|s)\, Q^\pi(s,a)\, ds\, da \\ &= \mathbb{E}_{s\sim\rho^\pi, a\sim\pi_\theta}\left[Q^\pi(s,a)\right] \ , \end{aligned} \tag{2.15}$$

where $\rho^\pi(s)$ indicates the evaluation of the probability density of state s according to ρ^π. Intuitively, optimizing this objective is equivalent of finding a policy that selects actions that maximize expected future returns from all states. Besides, the expected returns from all states are weighed according to the probabilities of observing such states during episodes. Its gradient is then (Sutton and Barto, 1998):

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s\sim\rho^\pi, a\sim\pi_\theta}\left[Q^\pi(s,a) \cdot \nabla_\theta \log \pi_\theta(a|s)\right] \ . \tag{2.16}$$

Notice that the policy gradient does not depend on the gradient of the state distribution and therefore the computation of this gradient is a simple expectation.

A fundamental issue that policy gradient algorithms address in different ways is how to estimate the action-value function. For instance, Mnih et al. (2013) used a neural network to approximate $Q^\pi(s,a) \approx Q^\pi_\phi(s,a)$ with parameters ϕ. Moreover, one more simple approach is to use a sampled return $Q^\pi(s_t,a_t) \approx R_t$ which lead to the formulation of a variant of the REINFORCE algorithm (Williams, 1992). REINFORCE also subtracts a control variate (also known as baseline) from the expected return to reduce the variance of the gradient estimation which is usually high. Baselines are unbiased in expectation (Williams, 1992) and several alternative have been proposed to choose them optimally (Peters and Schaal, 2008).

Deterministic Policy Gradient algorithms are known to perform well or even better in high-dimensional action spaces compared to their stochastic counterparts (Silver et al., 2014). Let then consider a deterministic policy $\mu_\theta : \mathcal{S} \to \mathcal{A}$, a performance objective $J(\mu_\theta) = \mathbb{E}[R_t|\mu_\theta]$ to maximize, indicating the expected future return using the policy μ_θ, and the discounted state distribution ρ^μ as a probability distribution over the state space $\mathcal{S}$ following the policy μ_θ. Similarly to the stochastic case, following Silver et al. (2014), we can then write the on-policy performance objective as

$$\begin{aligned} J(\mu_\theta) &= \int_{\mathcal{S}} \rho^\mu(s)\; Q^\mu(s,a)\big|_{a=\mu_\theta(s)}\; ds \\ &= \mathbb{E}_{s\sim\rho^\mu}\left[Q^\mu(s,a)\big|_{a=\mu_\theta(s)} \right] , \end{aligned} \tag{2.17}$$

where there is no need to integrate over the action space since the policy is deterministic. Indeed, every state is mapped to only one possible action. The objective J has to be maximized and its gradient with respect to the parameters θ to ascend is (Silver et al., 2014):

$$\nabla_\theta J(\mu_\theta) = \mathbb{E}_{s\sim\rho^\mu}\left[\nabla_\theta \mu_\theta(s)\; \nabla_a Q^\mu(s,a)\big|_{a=\mu_\theta(s)} \right] . \tag{2.18}$$

Similarly, they extended this derivation for off-policy methods that learn a deterministic target policy from trajectories generated by a stochastic behavior policy $\beta(s) \neq \mu_\theta(s)$. Therefore the objective becomes

$$J_\beta(\mu_\theta) = \int_{\mathcal{S}} \rho^\beta(s)\; V^\mu(s)\; ds , \tag{2.19}$$

and eventually, the *off-policy deterministic policy gradient* to ascend is

$$\nabla_\theta J_\beta(\mu_\theta) \approx \mathbb{E}_{s\sim\rho^\beta}\left[\nabla_\theta \mu_\theta(s)\ \nabla_a Q^\mu(s,a)\big|_{a=\mu_\theta(s)}\right] , \tag{2.20}$$

where the approximation is due to a dropped term analogously to Degris et al. (2012). Notice that Silver et al. (2014) provided the full gradient derivation with a proof of existence and necessary conditions.

Deep Deterministic Policy Gradient Lillicrap et al. (2016) employed off-policy deterministic policy gradient in combination with neural network function approximators and other techniques such as *target networks* and *experience replay*. This method is known as Deep Deterministic Policy Gradient (DDPG). They approximated the state-action value function $Q^\mu(s,a) \approx Q^\mu_\psi(s,a)$ with parameters ψ. They used a recurrence relation, known as the Bellman equation (Bellman, 2013), to rewrite the deterministic state-action value function as

$$Q^\mu(s_t,a_t) = \mathbb{E}_{s_{t+1}\sim p(s_{t+1}|s_t,a_t)}\left[r(s_t,a_t) + \gamma Q^\mu(s_{t+1},a_{t+1})\big|_{a_{t+1}=\mu_\theta(s_{t+1})}\right] , \tag{2.21}$$

They also employed Q-learning (Watkins and Dayan, 1992) which uses a greedy policy $\mu_\theta(s) = \max_a Q^\mu_\psi(s,a)$ in combination with a different stochastic behavior policy $\beta_\theta(s) = \mu_\theta(s) + \epsilon$, with ϵ some noise source, to generate transitions. The function Q^μ_ψ is trained with mean squared error minimizing the loss

$$\mathcal{L}(\psi) = \mathbb{E}_{s_t\sim\rho^\beta, a_t\sim\beta, s_{t+1}\sim p(s_{t+1}|s_t,a_t)}\left[\left(Q^\mu_\psi(s_t,a_t) - y_t\right)^2\right] , \tag{2.22}$$

where

$$y_t = r(s_t,a_t) + \gamma Q^\mu_{\psi'}(s_{i+1},a_{t+1})\big|_{a_{t+1}=\mu_{\theta'}(s_{i+1})} . \tag{2.23}$$

Notice that, $\mu_{\theta'} \neq \mu_\theta$ as well as $Q^\mu_{\psi'} \neq Q^\mu_\psi$. The two neural networks share their internal structure, but they are parameterized differently. That is because, also according to Mnih et al. (2013), directly updating neural network parameters is unstable in many noisy environments and it may lead to undesired behavior. When Q^μ_ψ is prone to divergence also μ_θ is, since its updates depend on the gradient provided by the state-action value network. For these reasons, DDPG uses separate *target networks* for calculating updates but, differently from Mnih et al. (2013), it uses soft updates, instead of directly copying the weights after a certain number of iterations. In particular, at the beginning of the training parameters are initialized equally (i.e., $\theta = \theta'$ and $\psi = \psi'$) but in the training loop, the weights of the target networks

are updated slowly from the original ones using $\theta' \leftarrow \tau\theta + (1-\tau)\theta$ and $\psi' \leftarrow \tau\psi + (1-\tau)\psi$ as update rules with $\tau \ll 1$. Lillicrap et al. (2016) empirically showed and argued that such soft updates change the optimization process of the state-action network closer to a problem of supervised learning. Therefore, it is an easier and more stable learning task, with convergence to more robust solutions.

Moreover, similarly to Mnih et al. (2015), DDPG employed a *replay buffer* to stabilize training. This buffer consists of a finite sized memory which stores episode transitions (s_t, a_t, r_t, s_{t+1}) according to the exploration policy. The buffer is circular, meaning that when full, the oldest samples are discarded and replaced. While training, at each step, a certain number of transitions are uniformly sampled from the replay buffer and assembled in a minibatch used to train the networks. In practice, this was observed to be beneficial since it allows learning using a set of uncorrelated transitions.

Eventually, in DDPG, the policy updates are employed using DPG as in Equation 2.20. The training procedure is outlined in Algorithm C.1.

2.4 Graph Convolutions

Convolutional neural networks (CNNs) (LeCun et al., 1998) provide an efficient architecture to extract meaningful information and statistical patterns. They learn local structures and how to compose them to form a hierarchical representation using filters. Convolutional filters are learned from the data to recognize features. Filters are translation-invariant which means that they detect features independently of their spatial locations. CNNs are widely used to deal with images, videos and sound data (LeCun et al., 2015), however, the classic notion of convolution is well-defined on the Euclidean space but not on graphs since there is no unique definition of translation on graphs from a spatial perspective (Bruna et al., 2014). Intuitively, if we imagine a convolution of a 2D signal (e.g., an image), then there exists concepts of *top*, *bottom*, *left* and *right* where, in a graph, that is not the case since taking a node and its neighbors there is no particular definition of spatial locations.

The study of graphs in the spectral domain led to the formulation of spectral graph convolutions (see Section 2.4.1) that provided a way to apply convolutions to graphs. Spectral graph convolutions is a general framework and its direct implementation would suffer from high computational complexity. We present a brief overview of two approaches that showed to be effective approximations of spectral graph convolutions: Fast Localized Spectral Filtering proposed by Defferrard et al. (2016) in Section 2.4.2 and graph convolution network (GCN) by Kipf and Welling (2017) in Section 2.4.3.

2.4.1 Spectral Graph Convolutions

We here provide a spectral graph theoretical formulation of graph convolutions. In particular, we are interested in processing signals defined on an undirected, unweighted, and without self-loop connections. Spectral graph theory provides a well-defined localization operator via convolutions using the Kronecker delta in the spectral domain (Shuman et al., 2013). Graph convolutions are defined as linear operators with the Fourier basis which is represented by the eigenvectors of the graph Laplace operator (Von Luxburg, 2007; Defferrard et al., 2016).

The Laplace operator Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph, where $\mathcal{V} = \{v_i\}_{i=1}^N$ is the set of vertices (or nodes) and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges (or links), and $f : \mathcal{V} \to \mathbb{R}$ be a function of a real signal on this graph. Since f is defined in a finite domain, it can be intended as a vector $\mathbf{f} \in \mathbb{R}^N$, where $N = |\mathcal{V}|$ and $f(v_i) = \mathbf{f}_i$. Therefore, the space of all possible signals on finite graphs is isomorphic to $\mathbb{R}^N$. It follows that the scalar product $\langle f, g \rangle = \sum_{i=1}^N f(v_i) \cdot g(v_i)$ can be defined on this space.

The Laplace operator (or Laplacian), usually denoted by the symbols $\nabla \cdot \nabla f$, $\nabla^2 f$ or Δf, is the divergence of the gradient of a function f on the Euclidean space. In a finite graph, an analog of the continuous Laplacian is the discrete Laplace operator. Since it can be used as a linear operation (matrix multiplication) over the finite vector $\mathbf{f}$, the discrete Laplace operator is more commonly called the Laplacian matrix.

The literature provides several formulations to define the Laplace operator which leads it to have different properties (Chung, 1997). Let $\mathbf{A}$ be the adjacency matrix of a graph $\mathcal{G}$ defined as a squared matrix $N \times N$ where $\mathbf{A}_{ij} = \mathbf{A}_{ji} = 1 \; \forall (v_i, v_j) \in \mathcal{E}$ and 0 elsewhere, and $\mathbf{D}$ be the degree matrix, diagonal, with entries $\mathbf{D}_{ii} = \sum_{j=1}^N \mathbf{A}_{ij}$. Then, i) the un-normalized graph Laplacian matrix is $\mathbf{L} = \mathbf{D} - \mathbf{A}$, ii) the symmetric normalized Laplacian matrix is $\mathbf{L}^{sym} = \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$, and iii) the random-walk normalized Laplacian matrix is $\mathbf{L}^{rm} = \mathbf{D}^{-1}\mathbf{L} = \mathbf{I} - \mathbf{D}^{-1}\mathbf{A}$, where $\mathbf{I}$ is the identity matrix. $\mathbf{L}$, $\mathbf{L}^{sym}$, and $\mathbf{L}^{rm}$ are both real symmetric positive semidefinite matrices. Then, these Laplace operators can be seen as

$$(\mathbf{L}f)(v_i) = (\mathbf{L}\mathbf{f})_i = \sum_{j=1}^N \mathbf{A}_{ij} \left[f(v_i) - f(v_j) \right] , \tag{2.24}$$

$$(\mathbf{L}^{sym} f)(v_i) = (\mathbf{L}^{sym}\mathbf{f})_i = \frac{1}{\sqrt{\mathbf{D}_{ii}}} \sum_{j=1}^N \mathbf{A}_{ij} \left[\frac{f(v_i)}{\sqrt{\mathbf{D}_{ii}}} - \frac{f(v_j)}{\sqrt{\mathbf{D}_{jj}}} \right] , \tag{2.25}$$

$$(\mathbf{L}^{rm} f)(v_i) = (\mathbf{L}^{rm}\mathbf{f})_i = \frac{1}{\mathbf{D}_{ii}} \sum_{j=1}^{N} \mathbf{A}_{ij} \left[f(v_i) - f(v_j) \right] \ . \tag{2.26}$$

These operators acts on the space of signals on the graph. Notice that the relation with the continuous Laplacian is that the quadratic form $\langle f, \mathbf{L}f \rangle_i = \sum_{j=1}^{N} \mathbf{A}_{ij} \left[f(v_i) - f(v_j) \right]^2$ is the discrete version of the quadratic form associated with the Laplace operator in $\mathbb{R}^n$: $\langle f, \Delta f \rangle = \int \|\nabla f\|^2 dx$. Such quadratic form measures the smoothness of the function f in the topology of the graph meaning how much adjacent nodes takes similar signal values.

For simplicity, we will consider $\mathbf{L}$ as the Laplacian matrix for the following augmentations even though they are also valid for both $\mathbf{L}^{sym}$ and $\mathbf{L}^{rm}$.

Graph Fourier Transform In the real line $\mathbb{R}$, the Fourier transform of a signal f is defined as

$$\hat{f}(\xi) = \langle f, e^{2\pi i \xi t} \rangle = \int_{\mathbb{R}} f(t)\ e^{-2\pi i \xi t} dt \tag{2.27}$$

where $e^{2\pi i \xi t}$ are the eigenfunctions of the one dimensional Laplace operator in $\mathbb{R}$. Analogously, the Fourier transform on a graph is defined using the Laplacian matrix (Moon and Spencer, 1961). $\mathbf{L}$ is diagonalizable such that $\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top$ where $\mathbf{U} = [\mathbf{u}^{(1)}, .., \mathbf{u}^{(N)}] \in \mathbb{R}^{N \times N}$ is the graph Fourier basis such that $\mathbf{U}^\top = \mathbf{U}^{-1}$, and $\mathbf{\Lambda} = diag([\lambda_1, .., \lambda_N]) \in \mathbb{R}^{N \times N}$ is a diagonal matrix whose diagonal elements are eigenvalues. The set $\{\mathbf{u}^{(i)}\}_{i=1}^{N} \in \mathbb{R}^N$ is composed of orthonormal eigenvectors, known as the set of graph Fourier modes, and they are associated with $\{\lambda_i\}_{i=1}^{N} \in \mathbb{R}$ which are real nonnegative eigenvalues known as frequencies of the graph. Thus, the graph Fourier transform of a signal in vectorial form $\mathbf{f} \in \mathbb{R}^N$ is then defined as $\hat{\mathbf{f}} = \mathbf{U}^\top \mathbf{f}$ where the inverse Fourier transform is $\mathbf{f} = \mathbf{U}\hat{\mathbf{f}}$ (Shuman et al., 2013). Since the Fourier transform is on the Euclidean space, it allows the formulation of convolutions.

Graph filtering Convolutions beween two signals g and f corresponds to multiplication in frequency domain $(g * f)(\xi) = \hat{g}(\xi) \cdot \hat{f}(\xi)$. Therefore, a graph convolution operator $*_G$ is defined as

$$g *_{\mathcal{G}} f = \mathbf{U}((\mathbf{U}^\top \mathbf{g}) \odot (\mathbf{U}^\top \mathbf{f})) \ , \tag{2.28}$$

where $\odot$ is the element-wise Hadamard multiplication. Since we are interested in learning a convolutional filter parameterized by θ, the convolution of a signal can be rewritten as a filter

$$g_\theta *_{\mathcal{G}} f = g_\theta(\mathbf{L})\mathbf{f} = g_\theta(\mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top)\mathbf{f} = \mathbf{U} g_\theta(\mathbf{\Lambda})\mathbf{U}^\top \mathbf{f} \ , \tag{2.29}$$

where $g_\theta(\mathbf{\Lambda})$ is a diagonal matrix interpreted as a function of the spectrum (eigenvalues) of $\mathbf{L}$. g_θ might be a non-parametric filter if all parameters are free such that $g_\theta(\mathbf{\Lambda}) = diag(\theta)$ with $\theta \in \mathbb{R}^N$.

2.4.2 Fast Localized Spectral Filtering

Non-parametric filters are not localized and their learning complexity is in $\mathcal{O}(N)$ (Defferrard et al., 2016). The use of a polynomial filter can overcome these issues defining

$$g_\theta(\mathbf{\Lambda}) \approx \sum_{k=0}^{K-1} \theta_k \mathbf{\Lambda}^k \,, \tag{2.30}$$

where $\theta \in \mathbb{R}^K$ is a vector of polynomial coefficients. Such filter is K-localized which means that for each vertex, the filter acts only within its K-th order neighbours (nodes that are at maximum K steps away from the considered vertex). The learning complexity is $\mathcal{O}(K)$ which is the size of the filter, and thus it has the same complexity as a classical CNN. The filter kernel centered at vertex v_i is localized with a Kronecker delta function $\delta_i \in \mathbb{R}^N$ so then the value at vertex j is $(g_\theta(\mathbf{L})\delta_i)_j = g_\theta(\mathbf{L})_{ij}$. Notice that $g_\theta(\mathbf{L})_{ij} = 0$ if v_i and v_j are S-th order neighbours for $S > K$.

There are two main drawbacks evaluating Equation 2.29: i) the multiplication with the eigenvector matrix $\mathbf{U}$ is computationally expensive $\mathcal{O}(N^3)$, and ii) computing the eigendecomposition of $\mathbf{L}$ is $\mathcal{O}(N^3)$. For small graphs, these are minor issues, but the complexity is too high for most practical problems. Thus, Hammond et al. (2011) proposed to approximate $g_\theta(\mathbf{\Lambda})$ by a truncated Chebyshev polynomial expansion using

$$g_\theta(\mathbf{\Lambda}) \approx \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\mathbf{\Lambda}}) \,, \tag{2.31}$$

where $\theta \in \mathbb{R}^K$ are Chebyshev coefficients and polynomials $T_k(\tilde{\mathbf{\Lambda}}) \in \mathbb{R}^{N \times N}$ are evaluated at the rescaled eigenvalues diagonal matrix $\tilde{\mathbf{\Lambda}} = 2/\lambda_{max} \cdot \mathbf{\Lambda} - \mathbf{I}$. The Chebyshev polynomial of the first kind of order k, $T_k(x)$, is defined by the recurrence relation $T_0(x) = 1$, $T_1(x) = x$, $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$. Eventually, the filtering operation can be written as

$$g_\theta *_\mathcal{G} f = g_\theta(\mathbf{L})\mathbf{f} \approx \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\mathbf{L}})\mathbf{f} \,, \tag{2.32}$$

where $T_k(\tilde{\mathbf{L}})$ is the Chebyshev polynomial of order k evaluated at the rescaled Laplace matrix $\tilde{\mathbf{L}} = 2/\lambda_{max} \cdot \mathbf{L} - \mathbf{I}$. Using the recurrence relation to compute the filtering operation lead to cost $\mathcal{O}(K|\mathcal{E}|) \ll \mathcal{O}(N^2)$.

2.4.3 Graph Convolutional Networks

Kipf and Welling (2017) proposed to limit the convolution operation in Equation 2.32 with $K = 1$, i.e. a function that is linear with respect to $\mathbf{L}$ and therefore a linear function on the graph Laplacian spectrum. They argued that such formulation can alleviate the problem of overfitting on local neighborhoods for large graphs. Moreover, this approximation allows building deeper neural network models which in practice is known to improve modeling capacity on several domains. With these assumptions, graph convolution networks (GCNs) further approximate $\lambda_{max} = 2$, expecting neural network parameters to adapt to this change in scale during training, leading to formulate

$$g_\theta *_\mathcal{G} f \approx \theta_0 f + \theta_1(\mathbf{A} - \mathbf{I})f \ , \tag{2.33}$$

where θ_0 and θ_1 are scalars. Thus, this can be seen as a sum of a box filter and a self-loop filter. Within the GCN framework we can then generalize the definition of a convolution of a multidimensional signal $\mathbf{X} = [\mathbf{x}^{(1)}, .., \mathbf{x}^{(D)}] \in \mathbb{R}^{N\times D}$ filtered by F feature maps as

$$\tilde{\mathbf{X}} = \mathbf{X}\mathbf{\Theta}_0 + \mathbf{L}\mathbf{X}\mathbf{\Theta}_1 \ , \tag{2.34}$$

where $\mathbf{\Theta}_0, \mathbf{\Theta}_1 \in \mathbb{R}^{D\times F}$ are matrices of parameters and $\tilde{\mathbf{X}} \in \mathbb{R}^{N\times F}$ is the resulting signal. Information propagation through multiple k-th order neighbours is done stacking multiple convolutions and applying nonlinearities between layers.

In the GCN setting, one can alternatively see a convolution as a series of node-to-edge and edge-to-node information propagation operations (Kipf et al., 2018) as we outline in Figure 2.4. Moreover, we can further generalize taking into account an edge annotation and apply a propagation conditioned on edges as well. More formally, let $a_\mathcal{V} : \mathcal{V} \to \mathcal{X}_\mathcal{V}$ be a function that assigns to each vertex an annotation from a set $\mathcal{X}_\mathcal{V}$ and $a_\mathcal{E} : \mathcal{E} \to \mathcal{X}_\mathcal{E}$ a function that assigns to each edge an annotation from a set $\mathcal{X}_\mathcal{E}$, then the triple $(\mathcal{G}, a_\mathcal{V}, a_\mathcal{E})$ is an labeled graph. Then, at each propagation step ℓ, first a node-to-edge propagation function acts transforming each couple of adjacent node signals into an intermediate vector

$$\mathbf{h}^{(\ell)}_{(i,j)} = f^{(\ell)}_{v\to e}(\mathbf{h}^{(\ell)}_i, \mathbf{h}^{(\ell)}_j, \mathbf{x}_{(i,j)}) \ , \tag{2.35}$$

where $\mathbf{h}^(\ell)_i$ and $\mathbf{h}^(\ell)_j$ are hidden node annotations and $\mathbf{x}_{(i,j)} = a_\mathcal{E}(v_i, v_j)$ is the edge annotation between v_i and v_j. Subsequently, for each vertex v_i, an edge-to-node propagation function takes all intermediate vectors of the edges connected to v_i to update the hidden node annotation

$$\mathbf{h}^{(\ell+1)}_i = f^{(\ell)}_{e\to v}(\{\mathbf{h}^{(\ell)}_{(i,j)} \mid v_j \in \mathcal{N}_i\}, \mathbf{x}_i) \ , \tag{2.36}$$

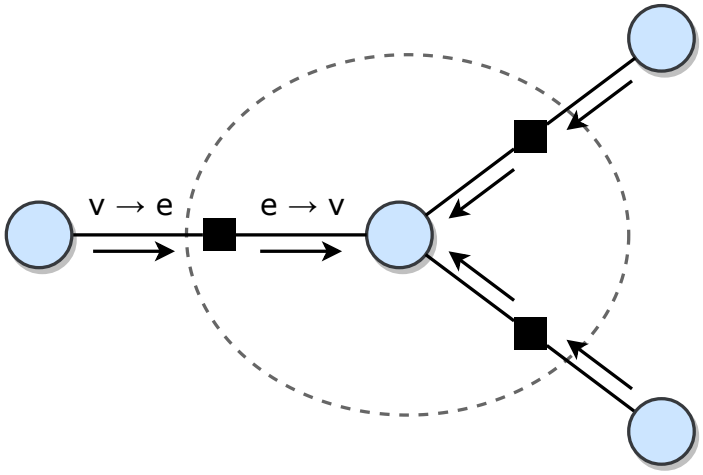


Figure 2.4: The graph convolution network information propagation algorithm consists in a series of node-to-edge and edge-to-node propagation steps. First, a node-to-edge function transforms node signals based on the edge annotation between them. Secondly, a node-to-edge function aggregates all intermediate transformations to update node signals.

where $\mathcal{N}_i$ is the set of neighbour nodes of v_i and $\mathbf{x}_i = a_{\mathcal{V}}(v_i)$ is the node annotation of v_i.

Note that the above formalism is quite general and particular models or applications make more assumptions on the structure of both propagation functions.

Relational graph convolution networks (R-GCNs) As a way to incorporate edge annotations, relational graph convolution network (R-GCN) makes use of GCNs in combination with labeled edges to model relations between entities in a knowledge-graph (Schlichtkrull et al., 2018). In particular, they model entities as nodes and relations as edges. Since relations between entities are fixed and taken from a finite set $\mathcal{R}$, they model the propagation step as a sum (aggregation) of neighbours signals transformed with relation-conditioned functions $f_r \; \forall r \in \mathcal{R}$. The update step is then

$$\mathbf{h}_i^{(\ell+1)} = \sigma\left(f_s^{(\ell)}(\mathbf{h}_i^{(\ell)}) + \sum_{j \in \mathcal{N}_i^r} \sum_{r \in \mathcal{R}} \frac{1}{c_{i,r}} f_r^{(\ell)}(\mathbf{h}_j^{(\ell)})\right) \;, \tag{2.37}$$

where σ is any non-linear function, f_s is the self-loop function, $\mathcal{N}_i^r$ indicates the set of neighbor indices of node v_i under the relation $r \in \mathcal{R}$, $c_{i,r}$ is a normalization constant (e.g., $c_{i,r} = |\mathcal{N}_i^r|$), and each f_* is an affine transformation.

Chapter 3

Method

In this chapter, we first cover several techniques we use to build generative models for molecular graphs. Subsequently, we show how we construct such models and how we intend to evaluate them. We introduce the problem generally, describing how graphs are used as a representation for molecules in Section 3.1. Then, in Section 3.2, we delineate how we use vectorial representations such as matrices and tensors to describe graphs in a suitable way for being processed by neural networks. We also show how we propose to use such numerical representations in combination with R-GCNs and NNs to encode graphs in vectors and decode vectors in graphs. Subsequently, we show how we use these building blocks to construct molecular graph variational auto-encoders in Section 3.3, generative adversarial networks in Section 3.4, and also to combine them with reinforcement learning in Section 3.5. Eventually, we describe and motivate evaluation techniques we intend to use in our experiments in Section 3.6.

3.1 Graph representation of molecules

As we already state in Section 1.1, graphs are a much more appealing way to represent molecules rather than others like the character-based SMILES representation. Therefore, we are generally interested in a generative model for discretely labeled graphs. Recalling the definition of an labeled graph: let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph, where $\mathcal{V} = \{v_i\}_{i=1}^{N}$ is the set of vertices and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges, and let $a_{\mathcal{V}} : \mathcal{V} \to \mathcal{X}_{\mathcal{V}}$ be a function that assigns to each vertex an annotation from a set $\mathcal{X}_{\mathcal{V}}$ and $a_{\mathcal{E}} : \mathcal{E} \to \mathcal{X}_{\mathcal{E}}$ a function that assigns to each edge an annotation from a set $\mathcal{X}_{\mathcal{E}}$, then the triple $(\mathcal{G}, a_{\mathcal{V}}, a_{\mathcal{E}})$ is an labeled graph. We define a discretely labeled graph as an labeled graph where both $\mathcal{X}_{\mathcal{V}}$ and $\mathcal{X}_{\mathcal{E}}$ are finite and discrete sets.

We can always construct a graph representation for a molecule where nodes denote atoms, and edges denote bonds. Indeed, we use annotations, assigning at each node a discrete type from $\mathcal{X}_\mathcal{V}$, which indicates the atom type (e.g., hydrogen, nitrogen, oxygen, etc.), and assigning at each edge an annotation from $\mathcal{X}_\mathcal{E}$ that represent different types of bonds (e.g., single, aromatic, double, etc.). Naturally, a molecule itself has much more additional information that cannot be represented using the formalism described above. Limitations of our graph representation approach are that we do not encode any information about i) the 3D structure of compounds, ii) higher level physical properties such as secondary/tertiary structures, and iii) the energy levels, masses, or other atom properties. Notice that one may extend the edge annotation set to add some extra atom properties. For generative processes like VAEs this might be beneficial, and one may include this additional information as an input of the model and avoid it on the reconstruction. However, in GANs for instance, this could be problematic since the generator, having to predict more values, would face a much harder task. For simplicity as well as aiming for a more general model for also non-molecular graphs, we do not include further information into annotations.

The main objective of our molecular graph generative models is then to learn some function $f : \mathcal{Z} \to \mathcal{X}_\mathcal{G}$ that maps D-dimensional vectors from space $\mathcal{Z} = \mathbb{R}^D$ in the space of all discretely labeled graphs $\mathcal{X}_\mathcal{G}$. Naturally, we also ensure that, from a known distribution defined on $\mathcal{Z}$, we can sample vectors which, through f, are mapped to a proper graph manifold. These issues are addressed using either VAEs or GANs (see Sections 3.3 and 3.4 respectively). Additionally, we are not only interested in sampling general graphs but molecular graphs that correspond to real and valid chemical compounds that resemble the property of a dataset or that are optimized for presenting some useful properties. Optimization towards some chemical scores is done using reinforcement learning (see Section 2.3).

The space of all possible graphs is infinite, and therefore, for practical reasons, it has to be limited in some way. In this work, we limit $\mathcal{X}_\mathcal{G}$ to be the space of graphs of at most $N = 9$ nodes. We also limit the node annotation set to be $|\mathcal{X}_\mathcal{V}| = 5$ (carbon, oxygen, nitrogen, fluorine, and one pad symbol), and the edge annotation set to be $|\mathcal{X}_\mathcal{E}| = 5$ (single, double, triple, aromatic, and one pad symbol that denotes no bond). Notice that these dimensionalities are an arbitrary choice and we choose these particular ones since they are enough to represent all molecules from the dataset we use (see Section 3.6.1 for dataset details). Expanding the number of nodes would be useful in practice since the search for new drugs or materials would benefit from an exploration of larger compounds. In this work, we impose these limitations to avoid a computational bottleneck while doing experiments due to the limited

Nodes	Combinations
$N = 1$	5
$N = 2$	130
$N = 3$	15755
$N = 4$	9781380
$N = 5$	30527359505
$N = 6$	476867685562630
$N = 7$	37253379852304703255
$N = 8$	14551952481746704111343880
$N = 9$	2842172398235648918154908204700 5

Table 3.1: Total number of all possible graphs up to N nodes with $|\mathcal{X}_\mathcal{V}| = 5$ vertex types and $|\mathcal{X}_\mathcal{E}| = 5$ edge types. Such number grows exponentially reaching $2.84 \cdot 10^{31}$ with just 9 vertices.

availability of resources. Indeed, working with bigger molecules, increases the search space of parameters and imposes a severe limitation on the scalability of the algorithm.

Besides, given such limits on the number of nodes, node types, and edge types, the total number of all possible graphs is $\approx 2.84 \cdot 10^{31}$ which is still an extremely high number. Indeed, the number of all possible graphs up to N nodes is given by the combinatorial equation

$$C(N, \mathcal{X}_\mathcal{V}, \mathcal{X}_\mathcal{E}) = \sum_{i=1}^{N} |\mathcal{X}_\mathcal{V}|^i \cdot |\mathcal{X}_\mathcal{E}|^{(i-1)\cdot i/2} , \tag{3.1}$$

which grows exponentially in the number of nodes and polynomially in the number of vertex and edge annotations. In Table 3.1, we show how this number grows in $N \in \{1, .., 9\}$. Notice that the number of valid molecular graphs (i.e., discretely labeled graphs that represent compounds physically possible to exist in nature) is expected to be orders of magnitude less than the total number of possible graphs.

3.2 Vectorial representation of graphs

Neural networks are well known universal functional approximators. For the nature of their inner structure, neural networks receive vectors as inputs, and they produce vectors as outputs. However, neither regular graphs nor labeled graphs are vectors, and therefore, there is the need of defining a

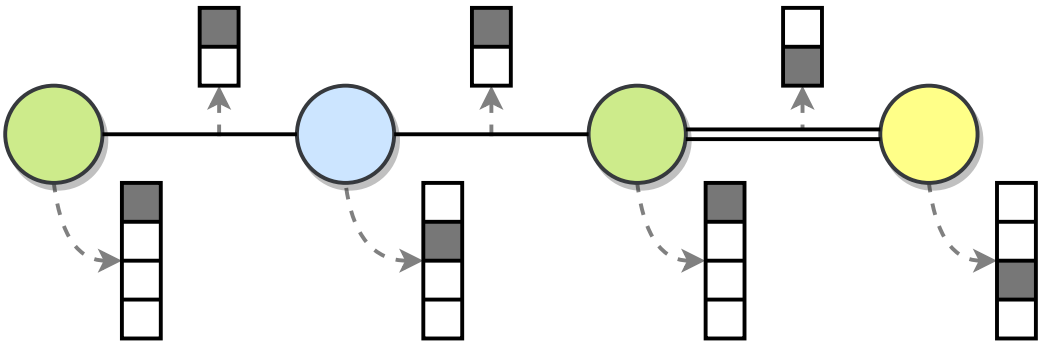


Figure 3.1: One-hot representation of a graph defined on the space of graphs with $|\mathcal{X}_\mathcal{V}| = 4$ vertex types and $|\mathcal{X}_\mathcal{E}| = 2$ edge types. Nodes with the same color denotes vertices with the same type which have the same vectorial representation. The same holds for edge types (single and double line denotes two different edge types).

bijective function to transforms graphs in a vectorial form. In particular, both the encoder of a VAE and the discriminator of a GAN need a graph-to-vector function. Indeed, they both receive graphs as inputs and they need to process them with neural networks and therefore via a vectorial form. Conversely, both the decoder of a VAE and the generator of a GAN output a set of vectors that have to be converted into graphs.

With neural networks, categorical input variables are usually represented through one-hot vectors (i.e., vectors with zeros in all entries but one that indicates which category the variable belongs to). Thus, to encode nodes, we define a function $m_\mathcal{V} : \mathcal{X}_\mathcal{V} \to \mathbb{R}^{|\mathcal{X}_\mathcal{V}|}$ that takes node annotations $\in \mathcal{X}_\mathcal{V}$ and returns one-hot vector representations of them. Afterwards, let $\{\mathbf{x}\}_{i=1}^{N}$ be all one-hot vectors that encode atom types of a molecular graph $\mathcal{G}$ such that $\mathbf{x}_i = m_\mathcal{V}(a_\mathcal{V}(v_i))$ for $i \in \{1, .., N\}$, then, a graph level matrix $\mathbf{X} \in \mathbb{R}^{N \times |\mathcal{X}_\mathcal{V}|} = [\mathbf{x}_1, .., \mathbf{x}_N]^\top$ represents all node types of $\mathcal{G}$ in a compact form. Similarly, another function $m_\mathcal{E} : \mathcal{X}_\mathcal{E} \to \mathbb{R}^{|\mathcal{X}_\mathcal{E}|}$ is used to encode edge annotations into one-hot vector representations. Similarly to the node annotation encoding, these vectors are aggregated into a tensor $\mathbf{A} \in \mathbb{R}^{N \times N \times |\mathcal{X}_\mathcal{E}|}$ where $\mathbf{A}_{ij} \in \mathbb{R}^{|\mathcal{X}_\mathcal{E}|}$ is the vectorial representation of the edge annotation between v_i and v_j, i.e., $\mathbf{A}_{ij} = m_\mathcal{E}(a_\mathcal{E}((v_i, v_j)))$. The tensor $\mathbf{A}$ can also be seen as a series of stacked adjacency matrices where each matrix $\in \mathbb{R}^{N \times N}$ indicates connection of some edge type only: $\mathbf{A} = [\mathbf{A}_1, .., \mathbf{A}_{|\mathcal{X}_\mathcal{E}|}]^\top$.

Using both $\mathbf{X}$ and $\mathbf{A}$, we can then encode a graph in a vectorial representation. In Figure 3.1 we show an example of a graph where both node and edge annotations are encoded in one-hot vectors. In Figure 3.2, we outline the triple representation used in this work: molecule objects, discretely

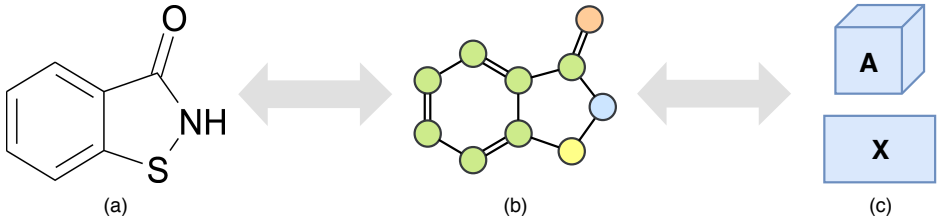


Figure 3.2: Molecules triple representation. The molecule (a) is represented as an labeled graph (b) where nodes denote atoms and edges denote bonds. The graph is represented through a combination of an annotation matrix $\mathbf{X}$ which sparsely encodes node types and the adjacency tensor $\mathbf{A}$ which sparsely encodes edge types. All representations preserve the molecular structure and therefore from all of them the other two can be retrieved.

labeled graphs, and the vectorial form.

Decoding, which consists in starting from $\mathbf{X}$ and $\mathbf{A}$ and constructing an labeled graph, is trivial since each one-hot vector corresponds to a particular node or edge annotation. However, it is not possible to outputs discrete objects directly while using neural networks. Categorical output variables are usually represented by a vector that parameterizes a categorical distribution (i.e., the value of each entry denotes the probability of the variable of belonging to a particular category). For instance, in case of a node annotation it would be a vector in the simplex $\mathbf{x} \in \Delta^{|\mathcal{X}_\mathbf{V}|-1} : \sum_{i=1}^{|\mathcal{X}_\mathbf{V}|} \mathbf{x}_i = 1$ where each entry $\mathbf{x}_i$ indicates the probability of the node of belonging to the i-th annotation/type. Neural network outputs can then be discretized through sampling. Thus, we denote with $\tilde{\mathbf{X}}$ and $\tilde{\mathbf{A}}$ the discrete samples from the continuous $\mathbf{X}$ and $\mathbf{A}$ respectively (see Figures 3.3 and 3.4 for an overview).

Categorical sampling can be performed using the Gumbel distribution via the Gumbel-Max trick (Maddison et al., 2014). Given a vector of probabilities $\mathbf{x} \in \mathbb{R}^D$ then $\tilde{\mathbf{x}}$ is the resulting one-hot vector:

$$\tilde{\mathbf{x}} = \text{one-hot}\left(\arg\max_{i \in \{1,..,D\}} \log(\mathbf{x}_i) + \mathbf{g}_i\right) \quad \text{with} \quad \mathbf{g}_i \sim \mathit{Gumbel}(0,1)\ , \tag{3.2}$$

where $\mathit{Gumbel}(0,1)$ is the standard Gumbel distribution (Gumbel, 1954). Sampling from the standard Gumbel distribution can be easily performed using the standard uniform distribution since, given $u \sim \mathcal{U}(0,1)$, then $g = -\log(-\log(u))$ is a sample from the standard Gumbel distribution.

Unfortunately, categorical sampling is not a differentiable operation which makes it problematic to use in combination with the backpropagation mechanism. We employ a simple gradient estimator for categorical variables also known as the Gumbel-Softmax trick (Jang et al., 2017).

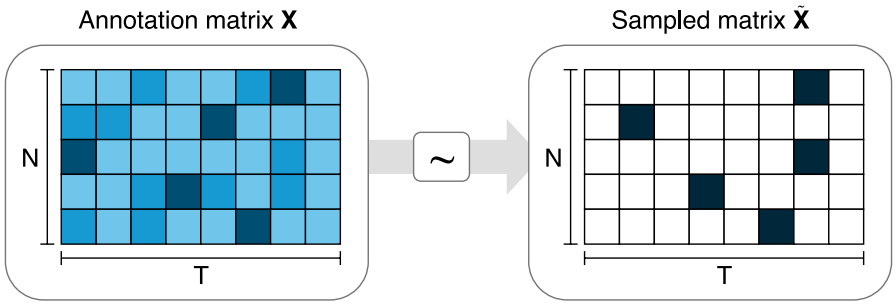


Figure 3.3: The predicted annotation matrix $\mathbf{X} \in \mathbb{R}^{N \times T}$, with $T = |\mathcal{X}_\mathcal{V}|$, is dense, and each row indicates a node $\{v_i\}_{i=1}^{N}$ where columns indicate the predicted probabilities (scales of blue) of each nodes for being of a certain type. Through categorical sampling, the matrix $\tilde{\mathbf{X}}$ is then sparse and one-hot on the rows. Note that sampling is stochastic and not all types with maximum probability are selected.

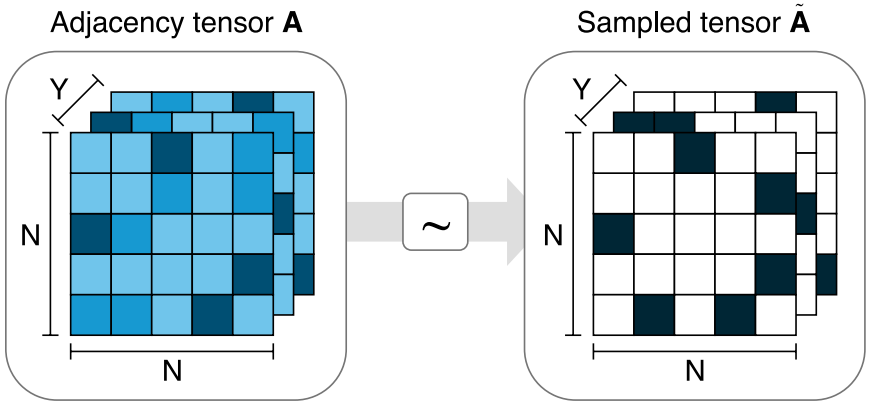


Figure 3.4: The adjacency tensor $\mathbf{A} \in \mathbb{R}^{N \times N \times Y}$, with $Y = |\mathcal{X}_\mathcal{E}|$, is dense, and each vector $\mathbf{A}_{ij} \in \mathbb{R}^{Y}$ indicates the predicted probabilities (scales of blue) of the edge between vertices v_i and v_j of being of a certain type. Through categorical sampling, the adjacency tensor $\tilde{\mathbf{A}}$ is then sparse and one-hot on the last dimension. Note that sampling is stochastic and not all types with maximum probability are selected.

Gumbel-Softmax trick The concrete distribution was independently discovered by Maddison et al. (2017) and Jang et al. (2017), and used as a continuous relaxation of the Gumbel-Max trick. In particular, they used the Gumbel-Max in the forward pass, and they estimated the gradient through its relaxation. This operation was named Gumbel-Softmax trick. They used the softmax function as a differentiable and continuous approximation of the $\arg\max$. Additionally, they estimated the gradient of categorical sampling

through a continuous relaxation of $\tilde{\mathbf{x}}$, as is defined in Equation 3.2, such that the gradient is $\nabla_\theta \tilde{\mathbf{x}} \approx \nabla_\theta \hat{\mathbf{x}}$ where

$$\hat{\mathbf{x}} = \frac{\exp\left((\log \mathbf{x}_i + \mathbf{g}_i)/\tau\right)}{\sum_{j=1}^{D} \exp\left((\log \mathbf{x}_j + \mathbf{g}_j)/\tau\right)} \,. \tag{3.3}$$

The temperature τ controls how smooth the distribution is, and $g_1, .., g_d$ are sampled from the standard Gumbel distribution. The temperature can be annealed according to some schedule. When $\tau \to 0$, samples for the Gumbel-Softmax distribution approaches the generation of one-hot vectors from the categorical distribution.

3.2.1 Graph encoding

We define as graph encoding the procedure of transforming the vectorial graph representations $\mathbf{X}$ and $\mathbf{A}$ into a graph-level vector $\mathbf{h}_\mathcal{G}$. The encoding consists of two stages: information propagation through graph convolution layers and aggregation of node representations.

Edge-type-conditioned convolutions A series of graph convolution layers convolve node signals $\mathbf{X} = [\mathbf{x}_1, .., \mathbf{x}_N]^\top$ using the graph adjacency tensor $\mathbf{A}$. We base our model on Relational-GCN (Schlichtkrull et al., 2018), a convolutional network for graphs with support for multiple edge types (see Section 2.4.3 for more details). In particular, at every layer ℓ, feature representations of nodes are convolved/propagated according to:

$$\mathbf{h}_i^{(\ell+1)} = \tanh\left(f_s^{(\ell)}(\mathbf{h}_i^{(\ell)}, \mathbf{x}_i) + \sum_{j=1}^{N} \sum_{y=1}^{|\mathcal{X}_\mathcal{E}|} \frac{1}{|\mathcal{N}_i|} \, \mathbf{A}_{ijy} \, f_y^{(\ell)}(\mathbf{h}_j^{(\ell)}, \mathbf{x}_j) \right) \,, \tag{3.4}$$

where $\mathbf{h}_i^{(\ell)}$ is the signal of the node v_i at layer ℓ and $f_s^{(\ell)}$ is a linear transformation function that acts as a self-connection between layers. For each layer, and each edge type, $f_y^{(\ell)}$ is an edge-type specific affine function. $\mathcal{N}_i$ denotes the set of neighbors for node v_i. The normalization factor $1/|\mathcal{N}_i|$ ensures that activations are on a similar scale irrespective of the number of neighbors.

Graph aggregation Zaheer et al. (2017) showed that a function $f(X)$ operating on a set X having elements from a countable universe, is a valid set function, i.e., invariant to the permutation of instances in X, if and only if it can be decomposed in the form $\rho\left(\sum_{x \in X} \phi(x)\right)$, for suitable transformations ϕ and ρ. Therefore, for each graph, after information has propagated through

L convolutional layers, a graph-level vector representation is computed following the same principle. It is an aggregation of node representations transformed with a function j and filtered with an attention mechanism through the function i. Both functions take node state vectors after L convolutions $\{\mathbf{h}_i^{(L)}\}_{i=1}^N$ and node annotations $\{\mathbf{x}_i\}_{i=1}^N$. Both f_g and f_r are implemented a MLPs with a linear output layer. Then, following Li et al. (2016), we define a graph level representation vector as

$$\mathbf{h}_{\mathcal{G}} = \tanh\left(\sum_{i=1}^{N} \sigma\left(f_g\left(\mathbf{h}_i^{(L)}, \mathbf{x}_i\right)\right) \odot \tanh\left(f_r\left(\mathbf{h}_i^{(L)}, \mathbf{x}_i\right)\right)\right), \tag{3.5}$$

where $\sigma(x) = 1/(1+\exp(-x))$ is the logistic sigmoid function, and $\odot$ denotes element-wise multiplication. Then, $\mathbf{h}_{\mathcal{G}}$ is a vector representation of the graph $\mathcal{G}$ that can be further processed by other functions.

3.2.2 Graph decoding

We define as graph decoding the procedure of transforming an embedding vector $\mathbf{h}_{\mathcal{G}}$ into graph vectorial representations $\mathbf{X}$ and $\mathbf{A}$. We identify three steps of decoding: processing $\mathbf{h}_{\mathcal{G}}$ with an MLP, decoding nodes, that is producing $\mathbf{X}$, and decoding edges, that is producing $\mathbf{A}$. We define $\mathbf{h}'_{\mathcal{G}} \in \mathbb{R}^D$ as the graph level vector after processed by the MLP. We further define two variations of node decoding: direct matrix annotation prediction, and incrementally predicting it using a recurrent neural network. We define three variations of edge decoding as well: direct prediction of the adjacency tensor, prediction of intermediate embeddings which decode into the adjacency tensor through a dot-product, and incrementally predicting the adjacency tensor using a recurrent neural network.

Node decoding The direct prediction of the annotation matrix $\mathbf{X}$ is an affine transformation $f : \mathbb{R}^D \to \mathbb{R}^{N\times|\mathcal{X}_\mathcal{V}|}$ followed by a softmax function applied to each row of the output matrix. In this way, $\mathbf{X} = f(\mathbf{h}'_{\mathcal{G}})$, and each row parameterizes a categorical distribution over node types.

Node decoding - recurrent The incremental prediction of the annotation matrix $\mathbf{X}$ is performed first using a recurrent intermediate function $r : \mathbb{R}^D \to \mathbb{R}^M$, for some arbitrary $M \in \mathbb{N}_+$, and secondly using an affine transformation on the top of the recurrent layer $f : \mathbb{R}^M \to \mathbb{R}^{|\mathcal{X}_\mathcal{V}|}$. In the end, a softmax function is applied to each output vector. In this way, the RNN predicts node annotations incrementally. Vectors are stacked together to obtain $\mathbf{X}$.

Edge decoding - adjacency tensor The direct prediction of the adjacency tensor $\mathbf{A}$ is an affine transformation $f : \mathbb{R}^D \to \mathbb{R}^{N \times N \times |\mathcal{X}_\mathcal{E}|}$ followed by a softmax function applied along the last dimension. In this way, $\mathbf{A} = f(\mathbf{h}'_\mathcal{G})$, and each slice $\mathbf{A}_{ij} \in \mathbb{R}^{|\mathcal{X}_\mathcal{E}|}$ parameterize a categorical distribution over edge types.

Edge decoding - dot-product The dot-product edge decoding is performed by first applying an affine transformation $f : \mathbb{R}^D \to \mathbb{R}^{N \times K \times |\mathcal{X}_\mathcal{E}|}$ for some arbitrary $K \in \mathbb{N}_+$, and secondly applying a batched dot-product such that $\mathbf{A}' = \mathbf{Z}\mathbf{Z}^\top$ where $\mathbf{Z} = f(\mathbf{h}'_\mathcal{G})$. A softmax function applied along the last dimension transforms $\mathbf{A}'$ into $\mathbf{A}$. With $\mathbf{Z}^\top$ we denote the batched matrix transpose operator such that if $\mathbf{Z} = [\mathbf{Z}_1, .., \mathbf{Z}_N]$ with $\mathbf{Z}_i \in \mathbb{R}^{K \times |\mathcal{X}_\mathcal{E}|}\ \forall i \in \{1, .., N\}$ then $\mathbf{Z}^\top \in \mathbb{R}^{N \times |\mathcal{X}_\mathcal{E}| \times K} = [\mathbf{Z}_1^\top, .., \mathbf{Z}_N^\top]$. With the batched dot-product we denote a similar semantic: $\mathbf{Z}\mathbf{Z}^\top = [\mathbf{Z}_1\mathbf{Z}_1^\top, .., \mathbf{Z}_N\mathbf{Z}_N^\top]$. The intuition behind this decoding procedure is that for each predicting node, and for each edge type, a neural network predicts a K-dimensional vector. The dot-product between two vectors in this space is a scalar value that denotes the unnormalized probability of a link of some type between two vertices. Since in this setting we generate the whole graph at once, there is the need of using the tensorial forms described above.

Edge decoding - recurrent The incremental prediction of the adjacency tensor $\mathbf{A}$ is performed similar to the recurrent node annotation predictions and the edge dot-product decoding. First we use a recurrent intermediate function $r : \mathbb{R}^D \to \mathbb{R}^M$, for some arbitrary $M \in \mathbb{N}_+$, and secondly we use an affine transformation on top of the recurrent layer $f : \mathbb{R}^M \to \mathbb{R}^{K \times |\mathcal{X}_\mathcal{V}|}$. Subsequently, stacking the outputs from f, we have a tensor $\mathbf{Z} \in \mathbb{R}^{N \times K \times |\mathcal{X}_\mathcal{E}|}$ which is transformed using the dot-product decoder (see above) in $\mathbf{A}$. In this way, the RNN predicts edge annotations incrementally.

3.3 Variational Auto-Encoder models

In this section, we show how we use graph-to-vector encoding and vector-to-graph decoding functions from Section 3.2 to build a molecular graph variational auto-encoder (see Section 2.1 for an overview on VAEs). We outline the architecture of this model in Figure 3.5. Recalling the structure of a VAE, we need to define an encoder e_ϕ and a decoder d_θ. Naturally, e_ϕ makes use of the graph-to-vector encoding function that takes vectorial graph representations $\mathbf{X}$ and $\mathbf{A}$ to output a vector $\mathbf{h}_\mathcal{G}$. Subsequently, we feed $\mathbf{h}_\mathcal{G}$ into two MLPs that compute a mean $\boldsymbol{\mu}_\mathcal{G}$ and a covariance vector $\boldsymbol{\sigma}_\mathcal{G}$

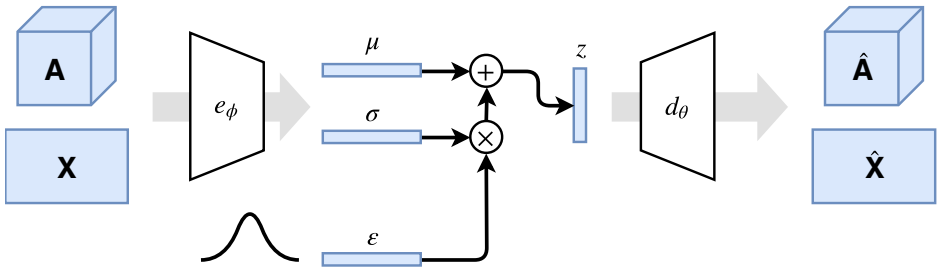


Figure 3.5: Outline of the molecular graph VAE model. Each graph is represented through the adjacency tensor $\mathbf{A}$ and annotation matrix $\mathbf{X}$. The encoder e_ϕ processes the graph to output two graph level vector representations that parameterize a Normal distribution with mean $\boldsymbol{\mu}$ and covariance $diag(\boldsymbol{\sigma})$. Using the reparameterization trick, $\mathbf{z}$ is computed and given to the decoder d_θ which outputs reconstructions $\hat{\mathbf{A}}$ and $\hat{\mathbf{X}}$.

respectively. Notice that as in Kingma and Welling (2013), we use a mean-filed assumption i.e., the covariance matrix is diagonal and $\boldsymbol{\Sigma} = diag(\boldsymbol{\sigma})$. These two vectors are used in combination with the *reparameterization trick* to generate a sample $\mathbf{z}_\mathcal{G}$ from the posterior $q_\phi(\mathbf{z}|\mathbf{x})$. Notice that the prior is a standard Normal distribution $p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$. The vector $\mathbf{z}_\mathcal{G}$ is further processed by a decoder d_θ that has to reconstruct the original inputs. The decoder makes use of one of the previously defined vector-to-graph decoding architectures to process $\mathbf{h}_\mathcal{G}$ and to output $\hat{\mathbf{X}}$ and $\hat{\mathbf{A}}$.

Additionally, to optimize a VAE, we also need to define a *reconstruction loss* to compute the ELBO. In particular, we need to define the negative likelihood $-\log p_\theta(\mathbf{x}^{(i)}|\mathbf{z}^{(i)})$, for a data-point $\mathbf{x}^{(i)}$ given the latent variable $\mathbf{z}^{(i)} \sim q_\phi(\mathbf{z}|\mathbf{x}^{(i)})$. Since our inputs and outputs consist in a series of categorical variables, we make use of the categorical cross entropy. In general, the cross entropy between two probability distributions p and q, defined over on the same domain/support $\mathcal{X}$, is

$$\mathbb{H}[p, q] = \mathbb{E}_{p(x)}[-\log q(x)] = -\int_\mathcal{X} p(x) \log q(x)\, dx\,, \tag{3.6}$$

where the integral is a sum when $\mathcal{X}$ is finite. Then, in our case, we set p as the empirical distribution or the probability of the observations (i.e., the dataset), and q as the parameterized categorical distribution predicted as the output of the decoder. The *reconstruction loss* between a pair of ground

truth labels $\mathbf{X}^{(i)}$ and $\mathbf{A}^{(i)}$, and the predicted $\hat{\mathbf{X}}^{(i)}$ and $\hat{\mathbf{A}}^{(i)}$, is then defined as

$$\begin{aligned}
\mathcal{L}(\mathbf{X}^{(i)}, \mathbf{A}^{(i)}, \hat{\mathbf{X}}^{(i)}, \hat{\mathbf{A}}^{(i)}; \theta, \phi) &= \mathcal{L}(\mathbf{X}^{(i)}, \hat{\mathbf{X}}^{(i)}; \theta, \phi) + \mathcal{L}(\mathbf{A}^{(i)}, \hat{\mathbf{A}}^{(i)}; \theta, \phi) \\
\mathcal{L}(\mathbf{X}^{(i)}, \hat{\mathbf{X}}^{(i)}; \theta, \phi) &= -\sum_{i=1}^{N} \sum_{j=1}^{|\mathcal{X}_\mathcal{V}|} \mathbf{X}_{ij}^{(i)} \log \hat{\mathbf{X}}_{ij}^{(i)} \\
\mathcal{L}(\mathbf{A}^{(i)}, \hat{\mathbf{A}}^{(i)}; \theta, \phi) &= -\sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{k=1}^{|\mathcal{X}_\mathcal{E}|} \mathbf{A}_{ijk}^{(i)} \log \hat{\mathbf{A}}_{ijk}^{(i)} \,,
\end{aligned} \tag{3.7}$$

where $\mathcal{L}(\mathbf{X}^{(i)}, \hat{\mathbf{X}}^{(i)}; \theta, \phi)$ and $\mathcal{L}(\mathbf{A}^{(i)}, \hat{\mathbf{A}}^{(i)}; \theta, \phi)$ are two individual cross entropies that summed up denotes the total loss. Notice that they depends on θ and ϕ since $\hat{\mathbf{X}}^{(i)}$, $\hat{\mathbf{A}}^{(i)}$ are computed using both the encoder and the decoder. Naturally, in a minibatch setting, the loss is computed as an average of a set of losses.

3.4 Generative Adversarial models

In this section, we show how we use graph-to-vector encoding and vector-to-graph decoding functions from Section 3.2 to build a molecular graph generative adversarial network (see Section 2.2 for an overview on GANs). We outline the architecture of this model in Figure 3.6. Recalling the structure of a GAN, we need to define a generator G_θ and a discriminator D_ϕ. Naturally, G_θ is defined as the vector-to-graph decoding that takes random vectors from a standard Normal distribution $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and outputs vectorial graph representations $\mathbf{X}$ and $\mathbf{A}$. Conversely, the discriminator makes use of the graph-to-vector encoding to produce intermediate hidden representations of graphs. Afterwards, an MLP takes such vectors to output scalar values (i.e., logits). We employ WGAN with gradient penalty (WGAN-GP) to improve the learning stability (see Section 3.4).

Since the outputs of the generator are continuous, when we need to generate actual molecules, we employ categorical sampling as defined in Section 3.2, to first discretize $\mathbf{X}$ and $\mathbf{A}$, and then build a graph. We argue that the generator should output discrete objects directly since the discriminator may take advantage of the fact that samples from the dataset are always discrete instead. For these reasons, we additionally explore three model variations that act between the outputs of the generator and the inputs of the discriminator namely: i) we do not apply any discretization, i.e., we use the continuous objects $\mathbf{X}$ and $\mathbf{A}$ directly during both the forward and backward pass, ii) we add standard Gumbel noise to both $\mathbf{X}$ and $\mathbf{A}$ to make the generation stochastic while still forwarding and applying backpropagation with

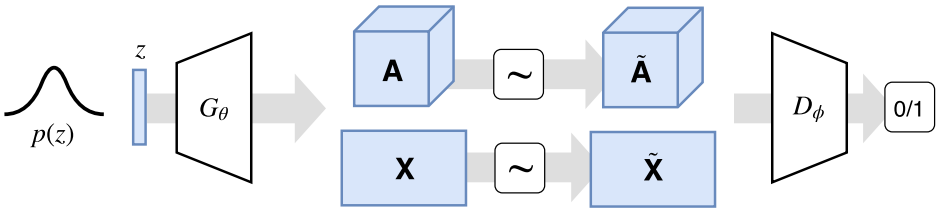


Figure 3.6: Outline of the molecular graph GAN model. A vector $\mathbf{z}$ is sampled from a standard Normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$ and given to the generator G_θ which outputs an adjacency tensor $\mathbf{A}$ and annotation matrix $\mathbf{X}$. Based on some differentiable function (i.e., either the identity function, the addition of Gumbel noise, or the Gumbel-Softmax trick), $\mathbf{A}$ and $\mathbf{X}$ are transformed in $\tilde{\mathbf{A}}$ and $\tilde{\mathbf{X}}$. Subsequently, a discriminator D_ϕ takes these objects to classify whether they comes from the generator or the empirical distribution.

continuous objects, and iii) we employ the Gubmel-Softmax trick to sample form categorical distributions parameterized with $\mathbf{X}$ and $\mathbf{A}$ while being able to use backpropagation with a gradient estimator (see Section 3.2 for more details). When employing categorical sampling, the inputs of the discriminator of generated molecules are discrete, and therefore we address the issue previously stated. However, since we make use of a gradient estimator, it might lead to a harder optimization problem.

3.5 Reinforcement learning models

In this section, we show how we use graph-to-vector encoding from Section 3.2 to enrich both molecular graph VAEs and GANs with reinforcement learning. RL is used as an additional component used during training to optimize the generation of molecules towards desirable properties.

We employ a simplified version of the off-policy deep deterministic policy gradient (DDPG) introduced by Lillicrap et al. (2016) (see Section 2.3). We opted for that algorithm after observing poor results using REINFORCE in combination with a stochastic policy during early stages of this research. DDPG have shown good results in high dimensional spaces like the one we are working with. Indeed, seeing the generation of $\mathbf{X}$ and $\mathbf{A}$ as an action, the action space is $\mathcal{A} = \mathbb{R}^{N \times |\mathcal{X}_\mathcal{V}| \times N \times N \times |\mathcal{X}_\mathcal{E}|}$ which, with the assumptions from Section 3.2, is $\mathbb{R}^{18225}$. Employing DDPG, we do not make use of either the *target networks* and the *replay buffer*. Besides, we introduce two main changes to the state-action value function.

In our case, the policy μ_θ is either the VAE decoder d_θ or the GAN generator G_θ. Both of them take a sample $\boldsymbol{z}$ for the prior as input, instead

of an environmental state s, and they output a molecular graph as an action (i.e., $a = \mathcal{G}$ where $\mathcal{G}$ is internally represented as the tuple $\mathbf{X}, \mathbf{A}$). Though the regular state-action value function $Q^\mu(s, a)$ takes both the environmental state s and the action a, there is no need of providing any state s. Indeed, we do not model episodes, nor do we have a model of the environment, so there is no need to assess the quality of a state-action combination since it does only depend on the action (i.e., the graph $\mathcal{G}$). Therefore, we replace $Q^\mu(s_t, a_t) = \mathbb{E}[R_t|s_t, a_t]$ with $R^\mu(a) = \mathbb{E}[r|a]$ which represents just the expected immediate reward of the generated molecule. We then approximate $R^\mu \approx R^\mu_\psi$ using neural networks trained with mean squared error minimizing the loss

$$\mathcal{L}(\psi) = \mathbb{E}_{\mathbf{z}\sim p_\mathbf{z}} \left[\left(R^\mu_\psi(a) - r(a) \right)^2 \right] \Big|_{a=\mu_\theta(\mathbf{z})} , \tag{3.8}$$

where the function r takes a molecular graph as input and outputs a reward. The policy gradient and how we train the policy remain unchanged.

We will experiment with different rewards namely the quantitative estimate of druglikeness, the octanol-water partition coefficient, and the synthetic accessibility score. In Section 3.6.3 we describe in detail what these metrics are.

When we combine other generative models with reinforcement learning, there is the need of defining a new merged loss. When using VAEs, the decoder d_θ is trained using a linear combination of the VAE loss (i.e., the ELBO), and the rescaled RL loss (i.e., policy gradient). Similarly, when using WGANs the generator G_θ is trained using a linear combination of the WGAN loss (i.e., *fooling* the discriminator) and the rescaled RL loss. We rescale the reinforcement loss $\mathcal{L}_{RL}$ to have the two gradients at same magnitude. Thus, the new combined policy gradient is

$$\nabla_\theta \mathcal{L}(\theta) = \lambda \cdot \nabla_\theta \mathcal{L}_{gen} + (1 - \lambda) \left| \frac{\mathcal{L}_{gen}}{\mathcal{L}_{RL}} \right| \cdot \nabla_\theta \mathcal{L}_{RL} , \tag{3.9}$$

where $\lambda \in [0, 1]$ is a hyperparameter that regulates the trade-off between using the loss of the generative model $\mathcal{L}_{gen}$ (either VAE or WGAN loss), and $\mathcal{L}_{RL}$. Conversely, the encoder e_ϕ and discriminator D_ϕ are trained using only the VAE or the WGAN objective respectively.

3.6 Evaluation techniques

In this section, we describe how we intend to evaluate our models. We will perform experiments to evaluate our model variations in Chapter 5. For all

experiments, we use the QM9 dataset (see Section 3.6.1 for details). We employed quantitative metrics such as importance sampling log-likelihood, the earth mover's distance, and the average reward. These quantities measure training convergence and can assess the overall performance of the generative process. Moreover, we also use qualitative measures to validate molecules generated by our models.

3.6.1 Dataset

The choice of a dataset for computational *de novo* drugs discovery should require an unbiased and large sample of the chemical molecular space. However, the combinatorial nature of such a space imposes a practical trade-off between a wide range of compounds and molecular sizes to approach the problem feasibly. GDB-17 chemical universe is a database of 166.4 billion molecules of up to 17 atoms (Ruddigkeit et al., 2012). It covers many drugs and biologically active compounds. GDB-17 also contains millions of isomers of known drugs, including analogs with high shape similarity to the parent drug. However, for our computational resources, this dataset is too big, so we used QM9, a curated subset of GDB-17 (Ramakrishnan et al., 2014). QM9 is a popular dataset used in computational chemistry which contains 133,885 organic compounds up to 9 heavy atoms: carbon (C), oxygen (O), nitrogen (N) and fluorine (F). In Figure 3.7 we show some samples from it. The predominant stoichiometry in QM9 is $C_7H_{10}O_2$: there are $6,095$ constitutional isomers within the 134k molecules. Constitutional isomers are compounds that have the same molecular formula (same atoms) but different connectivity (different bounds).

This datset does not come with a predefined train, validation and test splits and therefore we employed one by our own. We randomly take apart 10% of the dataset ($13,389$ molecules) for validation and another 10% for test. We split the dataset once and we keep this split for all experiments.

3.6.2 Quantitative evaluation

As quantitative evaluation measures we evaluate three different quantities depending on the model employed namely: the importance sampling log-likelihood of the test set (Burda et al., 2016) when using VAEs, the earth mover's distance (Gulrajani et al., 2017) between the generator distribution and the empirical distribution when using WGANs, and the average rescaled rewards when using reinforcement learning (RL) employing the same measures as Guimaraes et al. (2017).

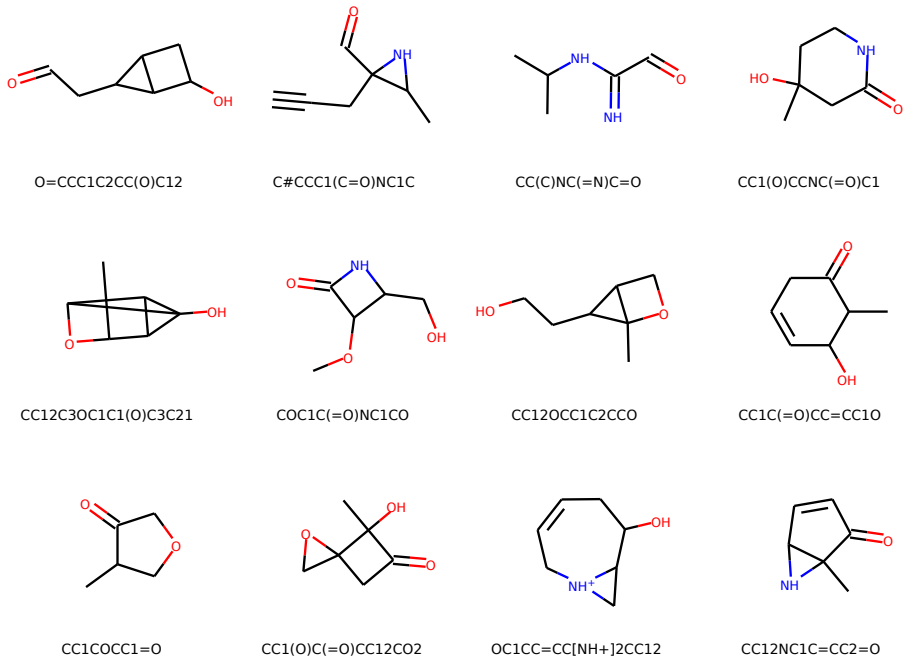


Figure 3.7: Random samples from the datset QM9. For each molecules we show also its SMILES representation.

Log-Likelihood Measuring test set likelihood is a standard measure when evaluating the performance of a probabilistic model with explicit likelihood or evidence. VAEs do have an explicit likelihood computation. However, it is analytically intractable as we showed in Section 2.1. Then, it is possible to compute an approximation of it using Monte Carlo importance sampling (see full derivation in Appendix B.2). The approximated log-likelihood (Burda et al., 2016) per data-point is

$$\log p(\mathbf{x}^{(i)}) \approx \log \frac{1}{M} \sum_{j=1}^{M} p_\theta(\mathbf{x}^{(i)}|\mathbf{z}^{(i)}) \cdot \underbrace{\frac{p(\mathbf{z}^{(i)})}{q_\phi(\mathbf{z}^{(i)}|\mathbf{x}^{(i)})}}_{\text{importance weight}} \Bigg|_{\mathbf{z}^{(i)} \sim q_\phi(\mathbf{z}|\mathbf{x}^{(i)})}, \tag{3.10}$$

when the approximation is done using M samples from the same data-point $\mathbf{x}^{(i)}$ (notice that the sum is over j). The log-likelihood of the entire test set $\mathbf{X} = \{\mathbf{x}\}_{i=1}^{N}$, is the mean over data-point log-likelihoods $\log p(\mathbf{X}) = \frac{1}{N} \sum_{i=1}^{N} \log p(\mathbf{x}^{(i)})$. Notice that, in practice, these quantities can be computed making use of the *log-sum-exp* trick for numerical stability.

Earth mover's distance When using a Wasserstein GAN, the model minimizes the earth mover's distance (EMD). Therefore, we can use that value as a measure of performance and also convergence. Recalling the EMD definition from Section 2.2, we instantiate it for the WGAN model as

$$\text{EMD} \approx \mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}\left[D_\phi(\mathbf{x})\right] - \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}\left[D_\phi(G_\theta(\mathbf{z}))\right] \ , \tag{3.11}$$

where it is an approximation since we do not compute the supremum but we use directly D_ϕ. Notice that in practice, the expected values are computed using Monte Carlo sampling with dimensionality equalling the size of the test set when using $p_{data}(\mathbf{x})$, an arbitrary number when using $p_{\mathbf{z}}(\mathbf{z})$.

Average rescaled rewards When optimizing a model using reinforcement learning, measuring rewards is a natural way to assess quality of resulting models. The reward indicates how we either recompense or penalize a generative model based on the quality of generated samples. We measure and average rewards at the end of the training of a generative model to compare or to show how effectively it optimizes some metrics. Rewards are implemented using chemical scores following Guimaraes et al. (2017). Similarly to them, we rescale these scores to have signals $\in [0, 1]$ where 0 indicates the worst reward (e.g., used to indicate invalid compounds) and 1 indicates an optimal one. In particular, we first define a rescaling function

$$\text{remap}(x, x_{min}, x_{max}) = \frac{x - x_{min}}{x_{max} - x_{min}} \ , \tag{3.12}$$

and a clipping function $\text{clip}(x) = \max(\min(x, 0), 1)$. Then, the rescaled reward functions, based on scores defined in Section 3.6.3, are

$$\text{druglikeliness} = score_{QED} \ , \tag{3.13}$$

$$\text{solubility} = \text{clip}\left(\text{remap}(score_{logP}, -2.12, 6.04)\right) \ , \tag{3.14}$$

$$\text{synthesizability} = \text{clip}\left(\text{remap}(score_{SA}, 5, 1.5)\right) \ . \tag{3.15}$$

The only score that is not rescaled is $score_{QED}$ which is already defined in the interval $[0, 1]$. Other non chemical scores such as validity and uniqueness, defined in Section 3.6.3, are not used as rewards. Besides, we also use novelty as reward since in *de novo* drug discovery we are interested in unknown compounds. When we combine rewards, the resulting signal is a product of independent rewards.

3.6.3 Qualitative evaluation

As a qualitative evaluation of the generative process of a model, we measure two sets of metrics. The first indicates general properties of desirable

convergence of a model. For that, we measure the following statistics as defined in Samanta et al. (2018): validity, novelty, and uniqueness. Following Guimaraes et al. (2017), the second set indicates chemical properties of generated compounds which represent qualities typically desired for drug discovery: the quantitative estimate of druglikeness, the octanol-water partition coefficient, and the synthetic accessibility score. Additionally, when employing reinforcement learning, we use these metrics alone or in combination to provide the reward signal to the generative model.

Validity score As defined in Samanta et al. (2018), the validity provides an empirical score that evaluates to which degree a method generates chemically valid molecules. It is a useful measure of performance since we would like to guarantee methods to sample valid compounds as much as possible. Let $\mathcal{S} = \{s^{(i)}\}_{i=1}^{N}$ be a multiset (since it can contain duplicates) of sampled molecules, and let $v : \mathcal{S} \to \{0, 1\}$ be a function[1] that assess whether a molecule is valid (1) or not (0), then, the validity score is defined as

$$\text{validity}_v(\mathcal{S}) = \frac{1}{|\mathcal{S}|} \sum_{i=1}^{N} v(s^{(i)}) \ . \tag{3.16}$$

Moreover, in order to define the uniqueness score and novelty score, it is useful to define the multiset of valid samples as $\mathcal{D} = \{s : s \in \mathcal{S} \wedge v(s) = 1\}$. Note that $\text{validity}_v(\mathcal{S}) \in [0, 1]$ but we consider more convenient to report it in percentage. The individual validity score for molecules is used as a first step when computing rewards since invalid molecular graphs do not map to real molecules. Thus, we cannot compute any other reward.

Uniqueness score The uniqueness score indicates how much a generative algorithm exploits diversity when sampling compounds. In particular, it is a statistical estimator on how many molecules are unique among a sampled set. High uniqueness score is desirable since we would like methods to generate samples from a wide range of possibilities. Let $\mathcal{D}$ be the multiset of valid samples as previously defined, the uniqueness score is then defined as

$$\text{uniqueness}_v(\mathcal{D}) = \frac{|set(\mathcal{D})|}{|\mathcal{D}|} \ , \tag{3.17}$$

where the function $set(\bullet)$ transforms a multiset into a set removing duplicates. Note that $\text{uniqueness}_v(\mathcal{D}) \in [0, 1]$ but we consider more convenient to report it in percentage.

[1] We use the opensource cheminformatics suite RDkit (http://www.rdkit.org) to check whether a generated molecules is chemically valid.

Novelty score Novelty is a desirable property in *de novo* drug discovery since we would like to have methods able to generate sets of new unexplored drugs. Therefore, following Samanta et al. (2018), we define the novelty score as a statistical measure of the degree of how much an algorithm is able to generate valid and novel molecules. Let $\mathcal{X}$ be the training set, and let $\mathcal{D}$ be the set of valid samples as previously defined, then

$$\text{novelty}_v(\mathcal{D}, \mathcal{X}) = 1 - \frac{|set(\mathcal{D}) \cap \mathcal{X}|}{|set(\mathcal{D})|} , \tag{3.18}$$

is the novelty. Note that $\text{novelty}_v(\mathcal{D}, \mathcal{X}) \in [0, 1]$ but we consider more convenient to report it in percentage. Intuitively, since a generative model tries to match the data distribution, some generated samples are similar or equal to some data-points used for training. Thus, the novelty score approximately indicates the proportion of generated compounds that are not in the dataset.

Quantitative estimate of druglikeness To quantify the quality of compounds, Bickerton et al. (2012) applied the concept of desirability to develop a quantitative metric for assessing druglikeness called the quantitative estimate of druglikeness (QED). Desirability is quantified with multiple numeric or categorical desirability functions, and it is measured on different scales. Unlike rule-based metrics, these desirability functions capture the underlying data distribution of several drug properties. In particular, they used eight widely-used molecular properties: molecular weight, octanol-water partition coefficient, number of hydrogen bond donors, number of hydrogen bond acceptors, molecular polar surface area, number of rotatable bonds, the number of aromatic rings and number of structural alerts. The final QED score is then computed as an aggregation of this set of measures as a weighted geometric mean of individual scores. In this way, such a measure is more tolerated to unfavorable properties when the value of other scores is high enough. QED values can range between zero (all properties are undesired) and one (all properties are desired). We show the distribution of this score in the QM9 dataset in Figure 3.8.

Octanol-water partition coefficient The octanol-water partition coefficient (logP), is defined as the logarithm of the ratio of the concentrations between two solvents of a solute (Comer and Tam, 2001). Usually, in pharmaceutical sciences, one of the solvents is water while the second is hydrophobic such as 1-octanol. The octanol-water partition coefficient (logP) provides a thermodynamic measure on how hydrophilic (*water-loving*) or hydrophobic (*water-hating*) a chemical substance is (Sangster, 1997). The logP is useful

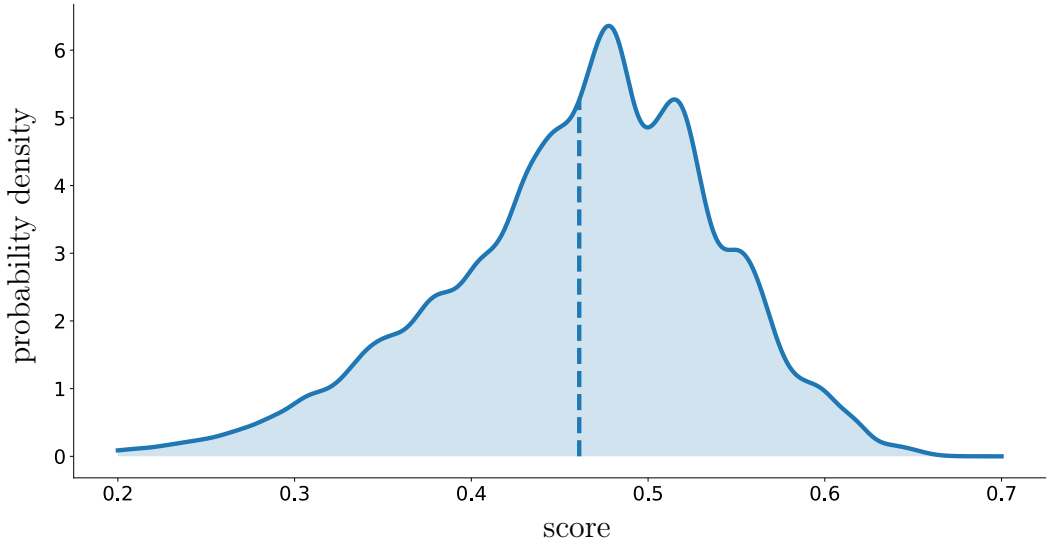


Figure 3.8: Distribution of the quantitative estimate of druglikeness score in QM9. The mean score is 0.46 with a std of 0.08.

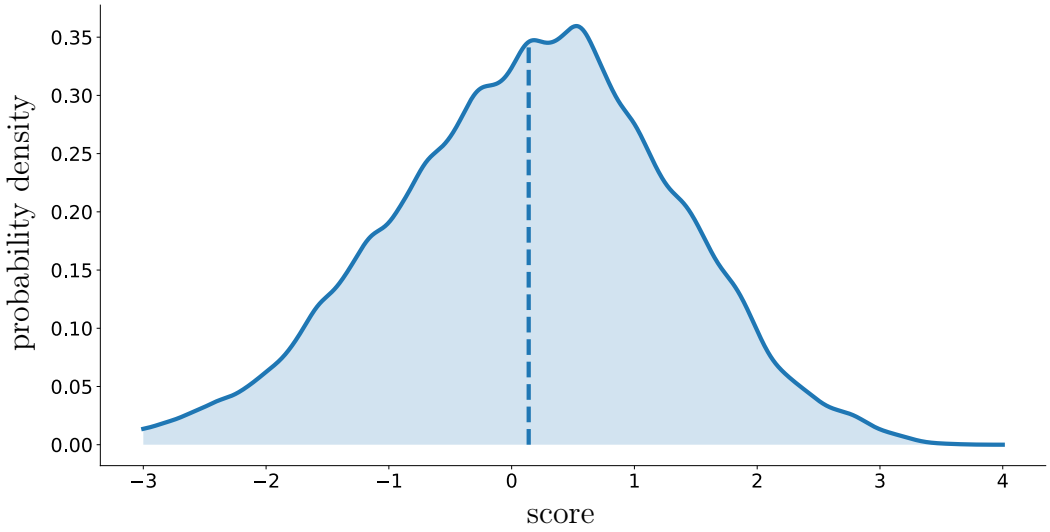


Figure 3.9: Distribution of the octanol-water partition coefficient in QM9. The mean score is 0.14 with a std of 1.16.

while estimating the distribution of drugs within the body since hydrophobic drugs (high logP) would mainly distribute to hydrophobic areas such as phospholipid bilayers of cells while hydrophilic drugs (low logP) would concentrate mostly in aqueous regions (e.g., blood). This coefficient is widely used in medicinal chemistry and drug design. In some works (Gómez-Bombarelli et al., 2016; Kusner et al., 2017; Guimaraes et al., 2017), the logP is used

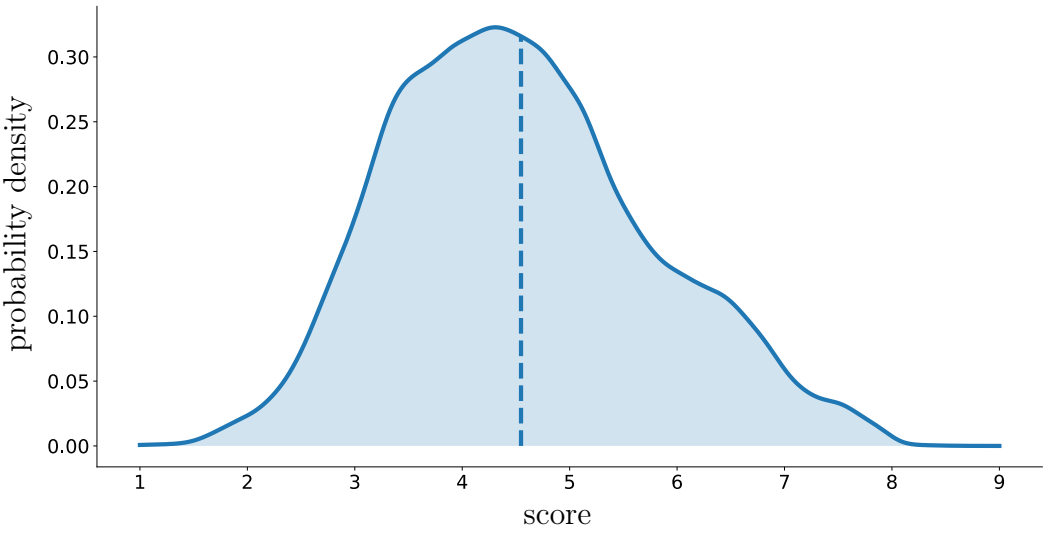


Figure 3.10: Distribution of the synthetic accessibility score in QM9. The mean score is 4.55 with a std of 1.22.

as a metric to be optimized since solubility is a desirable property of drugs. The distribution of this score in the QM9 dataset is shown in Figure 3.9.

Synthetic accessibility score A *de novo* drug discovery process requires an experimental validation of the synthesis of compounds. Compounds that are easier to produce are more likely to be considered as possible candidates. In the case of virtual exploration, there is the need of having a measure to estimate the ease of synthesis before experimentation. Ertl and Schuffenhauer (2009) introduced a method to estimate this ease of synthesis called the synthetic accessibility score (SAS). The SAS is based upon a combination of fragment contributions but also a complexity penalty. Fragments have been extracted from PubChem[2], a public compounds database. Their contributions have been computed based on the analysis of one million of already synthesized chemicals, therefore, capturing historical knowledge about the difficulty of their synthesization. The SAS is calculated as a sum of contributions of all fragments in the molecule divided by the total number of fragments. Then, the molecular complexity penalty takes into account the presence of non-standard structural features, such as large rings, non-standard ring fusions, stereocomplexity, and molecular size. This score relies on the range from 1 (easy to synthesize) and 10 (very difficult to synthesize). The distribution of this score in the QM9 dataset is shown in Figure 3.10.

[2] https://pubchem.ncbi.nlm.nih.gov

Chapter 4

Related work

Here we present some work related to ours. We present both variational and adversarial approaches similar to the one we use. Moreover, we will cover both other generative approaches for molecular generation and some of the literature on the use of neural networks within graphs.

Reinforcement learning for molecular design Molecular generation and synthesization are widely approached from many angles by the machine learning community. For instance, Segler et al. (2018) showed how to combine Monte Carlo tree search (MCTS) with reinforcement learning planning chemical syntheses from well-known compounds.

GAN and RL combination Objective-Reinforced Generative Adversarial Networks (ORGAN) by Guimaraes et al. (2017) is probably the most related work to ours. Their model relies on SeqGAN (Yu et al., 2017) to adversarially learn to output sequences while optimizing towards chemical metrics. SeqGAN is applied to generate SMILES sequences that resemble molecular data. They additionally optimize the generative process using REINFORCE (Williams, 1992) towards desirable chemical properties such as the quantitative estimate of druglikeness (QED), the synthetic accessibility score (SAS), and the octanol-water partition coefficient (logP) (see Section 3.6.3). ORGAN also explored the use of Wasserstein GANs (WGANs).

(W)GAN variations As we mention in Section 3.4, we employed WGAN with gradient penalty (WGAN-GP) (Gulrajani et al., 2017). Wei et al. (2018) showed further techniques to improve the training of WGANs. Moreover, Nowozin et al. (2016) showed that GAN approach is a special case of an existing and more general variational divergence estimation approach proposing

a series of alternative adversarial objectives to optimize. Boundary-seeking generative adversarial network (Hjelm et al., 2018) is an attempt to use GANs with discrete data that uses a gradient estimation similar to a policy gradient for training the generator. Note that some works have been proposed for gradient estimators of discrete variables. Both Tucker et al. (2017) and Grathwohl et al. (2018) provided a more sophisticated and accurate estimation of the gradient from a categorical sampling than straight through or the Gumbel-Softmax trick (Jang et al., 2017). Their works aim at providing unbiased and low variance gradient estimators.

Notice that some work has also been done towards the combination of variational approaches with adversarial learning (Dumoulin et al., 2017; Mescheder et al., 2017; Rosca et al., 2017). These methods address problems such as intractable density for variational Bayes using adversarial/implicit models.

VAEs with SMILES Several approaches that address *de novo* drug design follow the line of using SMILES strings as a representation for molecules. CharacterVAE by Gómez-Bombarelli et al. (2016) proposed one of the first attempt of generating compounds using variational auto-encoders. They used the character based SMILES representations in combination with RNNs. GrammarVAE (Kusner et al., 2017) is a more advanced approach that overcomes the complexity of learning a syntax constraining the output to be a string generated from a context-free grammar (CFG). They propose a variational approach which encodes and decodes directly to and from parse trees and they applied it to both arithmetic expressions and molecular structures. On the same line, syntax-directed variational auto-encoder (SD-VAE) by (Dai et al., 2018) restricted the language space of operation. In this way, they managed to generate both programs and molecules not only syntactically valid but also semantically.

Graph Convolution Networks Another line of research consists of training deep models that work with graph-structured data directly. The work by Scarselli et al. (2009) was one of the first of proposing a combination of neural networks on graph structures. Li et al. (2016) developed a class of neural network models that uses graph-structured inputs in combination with gated recurrent units to output sequences.

Several works utilized graph convolution networks (GCNs) in combination with VAEs for link prediction within graphs (Kipf and Welling, 2016; Grover et al., 2017; Davidson et al., 2018). These models use a node level embedding to encode topological information of the graph and use that to predict links.

Differently, Johnson (2017); Li et al. (2018a); You et al. (2018); Li et al. (2018b) employed likelihood-based methods to learn generative models for graphs of arbitrary size using recurrent models. Johnson (2017) introduced the Gated Graph Transformer Neural Network (GGT-NN) as an extension of the Gated Graph Sequence Neural Network (GGS-NN) (Li et al., 2016). Their model can learn to construct and modify a graph through subsequent steps based on a textual input. GraphRNN (You et al., 2018) and the model proposed by Li et al. (2018a) learn to generate graphs by decomposing the generation process into a sequence of node, and edge additions, conditioned on the structure created so far. Starting with an empty molecular graph, Li et al. (2018b) also modeled graph transition (append node, connect nodes or the terminate process) modeling probability distribution with neural networks on these actions. At each step, they conditioned the generation not only on previous steps but also on chemical objectives including the generation of compounds containing a given scaffold.

Recent work by Velickovic et al. (2018) proposed an extension of the GCN framework that predict attentions over neighbour contributions. Another contemporary work (Battaglia et al., 2018) extensively analyses and discuss GCN in combination with relational inductive biases. Neural Relational Inference (Kipf et al., 2018) showed how an unsupervised graph-based model could learn an infer interactions from a dynamical system only from observational data.

VAEs with graphs Within the chemistry domain, graph-based methods were proved to be useful in many tasks. Neural message passing (Gilmer et al., 2017) used graph-based neural models within quantum chemistry to predict molecular fingerprints. GraphVAE (Simonovsky and Komodakis, 2018) uses a very similar approach to ours to encode and decode graphs directly within a VAE setting. They used edge conditioned convolution (ECC) (Simonovsky and Komodakis, 2017) to propagate information in a graph while conditioning on edges. Junction Tree VAE (Jin et al., 2018) extends the GraphVAE approach with a dual tree-structured representation. They aim to capture the complexity of molecular graphs learning a higher level tree representation of them. NeVAE by (Samanta et al., 2018), was shown to learn a generative process for graphs imitating random graph models and overcoming the issue of graph isomorphism under permutation of the nodes. However, this method samples a number of node embeddings where we sample a single graph-level vector directly.

Chapter 5

Experiments

In this section, we present a series of experiments to analyze the behavior of different model variants for molecular graph generation. For each of them, we report an experimental setup followed with results and visual representations when needed. In Section 5.2, we investigate variational auto-encoders only whereas in Section 5.3 we study their combination with reinforcement learning. We aim to observe models behavior in the presence of additional atom features, with different latent space dimensionalities, and different decoding functions. Similarly, we present molecular graph generative adversarial networks experiments in Section 5.4 and their combination with reinforcement learning in Section 5.5. When studying both approaches in combination with RL, we focus on the efficacy of the introduced bias, and the effect of the parameter that controls the trade-off between the generative objective and the RL component. In Section, 5.6 we analyze overall results from both VAEs, GANs, and their combination with reinforcement learning.

5.1 Shared setup

For all experiments on both VAEs and GANs, we run our models for 500 epochs, on the QM9 dataset (see Section 3.6.1) using minibatches of size 128. Molecules are represented using a fixed canonical order of atoms (similar to SMILES). We optimize these models using the Adam optimizer (Kingma and Ba, 2014) with a learning rate of 10^{-3}.

Encoder and discriminator The encoder e_ϕ and the discriminator D_ϕ share most of the architectural design. They both use a 2-layered R-GCN of dimensionalities [128, 64] to process the graph, followed by an aggregation operation with dimensionality 128, as described in Equation 3.5, to have a

graph-level vector representation. Subsequently, this vector is processed by a 2-layered MLP of dimensionalities [128, 64]. The last layer of D_ϕ consists of an affine transformation $\mathbb{R}^{64} \to \mathbb{R}$. Instead, the last layer of e_ϕ consists of two affine transformations, which depend on the dimensionality of the latent space $\mathcal{Z}$, that output the mean $\boldsymbol{\mu}_\mathcal{G}$ and the diagonal of covariance $\boldsymbol{\sigma}_\mathcal{G}$ respectively. All nonlinearities between layers in both the encoder and discriminator networks are tanh functions.

Decoder and generator Similarly, also the decoder d_θ and the generator G_θ share the same architecture. Notice that they do not share parameters. We use different architectures and number of parameters depending on the type of decoding function used (see Section 3.2.2). We choose the size of hidden layers to have the total number of parameters of all three decoders as equal as possible to avoid unfair comparisons. For instance, an RNN might outperform simpler methods with more parameters, but in that case, it would be erroneous to claim that such RNN is a better model without considering the number of parameters.

The *direct prediction* decoding uses a 3-layered MLP of dimensionalities [128, 256, 512] followed by two linear maps to match the size of the annotation matrix $\mathbf{X}$ and the adjacency tensor $\mathbf{A}$ respectively. The *dot-product decoding* consists a 2-layered MLP of dimensionalities [128, 256] followed by a linear map to predict $\mathbf{X}$, and a linear map to predict the intermediate tensor $\mathbf{Z}$ with $K = 32$. Subsequently, the adjacency tensor $\mathbf{A}$ is $\mathbf{A} = \text{softmax}(\mathbf{Z}\mathbf{Z}^\top)$.

The *recurrent decoding* is implemented using a 2-layered MLP of dimensionalities [128, 256] followed by two long short term memory (LSTM) (Hochreiter and Schmidhuber, 1997) networks that predict the annotation matrix and the adjacency tensor respectively. The LSTM cell used to output $\mathbf{X}$ has a hidden layer of 128 units, and it is followed by an affine transformation $\mathbb{R}^{128} \to \mathbb{R}^{|\mathcal{X}_\mathcal{V}|}$ to predict node types. Also, the LSTM cell used to output $\mathbf{A}$ has a hidden layer of dimensionality 128, and it is followed by a linear map to predict the intermediate tensor $\mathbf{Z}$ with $K = 32$. Similarly to the *dot-product decoding*, $\mathbf{A} = \text{softmax}(\mathbf{Z}\mathbf{Z}^\top)$. Nonlinearities between layers in all decoder/generator variations are tanh functions, and the last ones are softmax functions.

5.2 Variational Auto-Encoders

In this section, we describe a sequence of experiments to analyze the behavior of several variations of molecular graph VAEs. For all experiments, we train using settings and model architectures as described above. Additionally, we

evaluate the model every 10 epochs, and we estimate the log-likelihood on the test set with importance sampling using 500 samples per data-point.

We first show how models perform when adding additional atom features. Subsequently, we compare performance of different decoding functions (as explained in Section 3.2) as well as different latent space dimensionalities. We also analyze the quality of our models plotting a latent space visualization of an $\mathbb{R}^2$ model as well as showing interpolations between samples.

5.2.1 Additional features

In this experiment, we want to investigate the effect of adding additional features as an input to a VAE. As we previously discussed in Section 3.1, it is not straightforward to include additional information like molecular properties in the kind of graph generative models we are using. For instance, if one property is *being in a ring*, when predicted, it would be problematic to combine it with the predictions of the edges. Indeed, the model should output both correctly, or otherwise, there would be inconsistencies. However, in the case of VAEs, we can append additional atom features when feeding the encoder without requiring the decoder to predict them. As atom features, we use the number of neighbors, the explicit and implicit valence, the hybridization, the number of explicit and implicit hydrogens, the number of radical electrons, whether it is in a ring, and the size of the ring when available. We expect that models will benefit of such features having a lower reconstruction error.

We test out hypothesis running a series of VAE variations. In particular, we fix the latent space size to $d = 8$, and we run models with and without additional features also varying the decoding function. As a comparison measure, we use the log-likelihood on the test set (see Section 3.6.2). We measure the validity, uniqueness, and novelty scores with a sample size of 1k.

Results In Table 5.1 we show results of the additional features experiment. As expected, all of the three models that use additional atom information outperform the one without in term of log-likelihood on the test set. We notice that the main advantage is due to a lower reconstruction error. Conversely, the KL is slightly higher than the one without features of one or two nats. It seems that with these features, all models can reconstruct the samples better, having more information about the molecular structure, but this cost in terms of KL. The additional features seem to affect neither the uniqueness score nor the novelty score. However, using these features increases the number of valid molecules in a random sample which empirically demonstrates that the latent space structure is slightly better.

Method	*no features*						
	LL	ELBO	r. loss	KL	validity	uni.	novelty
direct	-24.10	-25.86	18.21	**7.65**	52.30	100.00	**71.30**
dot-product	-23.93	-26.11	18.28	**7.82**	48.20	99.38	**75.70**
recurrent	-23.52	-25.24	17.16	**8.08**	52.50	99.62	**79.81**
Method	*features*						
	LL	ELBO	r. loss	KL	validity	uni.	novelty
direct	**-20.51**	**-22.27**	**13.27**	9.00	**62.50**	99.84	70.88
dot-product	**-19.95**	**-22.58**	**12.88**	9.70	**56.20**	100.00	75.09
recurrent	**-19.61**	**-21.81**	**11.87**	9.94	**60.00**	99.67	71.00

Table 5.1: Analysis on the use of additional atom features in VAEs. All measures are computed on the test set using a VAE with $d = 8$ and direct decoding. LL stands for log-likelihood, *r. loss* stand for reconstruction loss, and *uni.* stands for uniqueness. Validity, uniqueness, and novelty scores are computed with a sample size of 1k. We highlight scores when they are better than the respective counterpart. We do not highlight the uniqueness score since it is not relevant (always $\approx 100\%$).

The uniqueness score is always high and approaching 100%. This behavior can be well explained since the VAE objective makes the model to matches the data distribution. Data, i.e. molecules, are embedded in the latent space and spread around the origin (due to the variational loss). Therefore, different parts of the latent space map to different molecular outputs. For the subsequent experiments, we then use VAEs with additional molecular features leading to an overhead of 10% in the parameter space based on the architectural choice stated at the beginning of this section.

5.2.2 Dimensionality and decoding functions

In this experiment, we want to show the effects on performance of both the latent space dimension and the type of decoding function employed. We use results from the previous experiment to choose whether to include features or not. We investigate $d \in \{2, 4, 8, 16, 32, 64\}$ using the log-likelihood on the test set as a comparison measure. We use the three type of decoding as described in Section 3.2.2. We measure validity, uniqueness, and novelty scores with a sample size of 1k. In this setting, we do not have a strong hypothesis on the results. We do not expect any particular decoding function or dimension to outperform the others.

Method	*direct decoding*						
	LL	ELBO	r. loss	KL	validity	unique.	novelty
$d = 2$	-22.80	-24.05	17.30	**6.75**	55.90	99.64	**78.71**
$d = 4$	-22.81	-24.07	17.04	7.03	55.20	100.00	73.55
$d = 8$	**-20.51**	**-22.27**	**13.27**	9.00	**64.50**	99.84	70.88
$d = 16$	-20.78	-22.54	13.58	8.96	60.80	99.84	69.90
$d = 32$	-20.84	-22.61	14.01	8.60	61.40	99.66	69.02
$d = 64$	-21.01	-22.79	14.08	8.71	60.20	99.66	67.57
Method	*dot-product decoding*						
	LL	ELBO	r. loss	KL	validity	unique.	novelty
$d = 2$	-23.22	-24.48	17.95	**6.53**	49.10	100.00	**79.63**
$d = 4$	-21.03	-23.25	14.09	9.16	**57.20**	99.65	74.30
$d = 8$	-19.75	-22.58	12.88	9.70	56.20	100.00	75.09
$d = 16$	-19.99	-22.63	12.86	9.77	56.80	99.82	73.59
$d = 32$	**-19.49**	**-22.48**	**12.15**	10.33	56.60	99.82	67.84
$d = 64$	-19.65	-22.66	12.39	10.26	56.10	100.00	75.22
Method	*recurrent decoding*						
	LL	ELBO	r. loss	KL	validity	unique.	novelty
$d = 2$	-21.80	-23.06	15.74	**7.32**	56.60	100.00	**73.50**
$d = 4$	-20.33	-22.07	13.17	8.90	59.60	99.50	65.44
$d = 8$	-19.61	-21.81	11.87	9.94	60.00	99.67	71.00
$d = 16$	**-19.29**	**-21.49**	11.70	9.79	**65.40**	99.85	69.27
$d = 32$	-19.46	-21.67	**11.65**	10.02	62.50	99.52	69.44
$d = 64$	-22.38	-23.97	16.55	7.42	55.40	100.00	72.56

Table 5.2: Comparison of different latent space dimensionalities and decoding functions in a VAE setting with additional features. We highlight the best scores for each decoding function. *r. loss* stands for reconstruction loss where *unique.* stands for uniqueness. We do not highlight the uniqueness score since it is not relevant (always $\approx 100\%$).

Results In Table 5.2 we show results of the dimensionality and decoding functions comparison experiment. Similarly to the previous experiment, we notice that the uniqueness score is quite close to 100% in every setting. In terms of validity scores, *direct decoding* models have 5-10% higher scores than the *dot-product decoding* models. This difference is instead minimal compared to recurrent models. On average, *dot-product decoding* generates slightly more novel compounds comparing to other decodings.

In terms of log-likelihood on the test set, we notice that *dot-product decoding* and *recurrent decoding* models achieve higher values than the *direct decoding* ones. It may be that these two decoders learn better the information about edges. In particular, when learning, both these models have to encode in some way into intermediate hidden vectors the relation between nodes. Instead, in *direct decoding* models, these relations are implicit and encoded in an overall graph representation which is encoded in one step. Moreover, in the recurrent model, predictions are more explicitly conditioned on the previous ones.

Recurrent models outperform the others in terms of reconstruction loss. These values are almost always one nat lower than *dot-product* models and two nats lower that *direct decoding* models. They also have comparable KL values to the *dot-product* models but higher than the *direct decoding* models. Notice that this leads to higher ELBO values compared to the other two settings in almost all recurrent model. However, a better ELBO does not always correspond to better log-likelihood (Rainforth et al., 2018). Indeed, even though recurrent models have a better ELBO, they have comparable log-likelihood values comparing to the *dot-product decoding* ones. Notice that in high dimensions, i.e. $d = 64$, the recurrent VAEs do not achieve good convergence. *Direct decoding* models have worse reconstruction losses than the other decodings, but they have lower KL values. In fact, they result in having comparable ELBO values with *dot-product* models.

When observing the effects of the latent space dimensionality, we see that lower dimensions ($d = 2$ and $d = 4$) perform poorly compared to higher ones. This behavior is unsurprising since the encoder is forced to compress all the information in a smaller representation, and the decoder to learn from it. Conversely, we do not notice benefits of using dimensions higher than $d = 8$. Indeed, unlike classic auto-encoders, variational auto-encoders not always benefit from higher embedding dimensions since there is also the constraint of matching a prior distribution.

In Figures 5.1 and 5.2, we show the evolution during training of the ELBO and the reconstruction loss respectively of *direct decoding* models. From these plots, we see that all models converged since values on the last 100 epochs remain almost unchanged and steadily around the same value. Besides, as previously observed, we can see how lower dimensions perform worse than higher and that $d = 8$ outperforms $d > 8$.

Interpolations In Figure 5.3, we report an interpolation between two random molecules using a VAE with $d = 8$ and *direct decoding*. Since the space of graphs is discrete, it is not possible to have smooth transitions. Besides

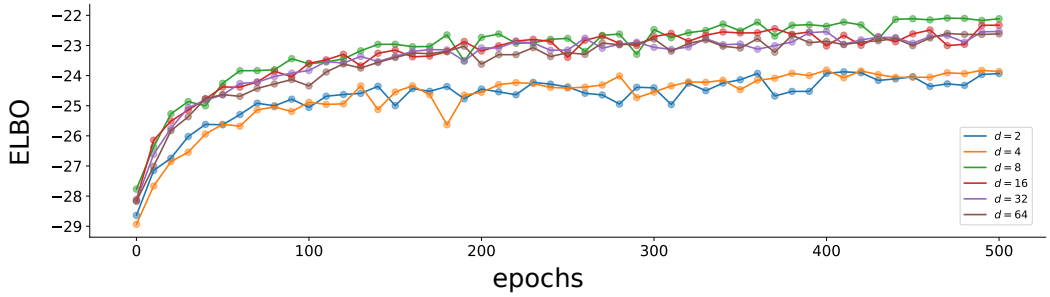


Figure 5.1: Comparison of ELBO during training of VAEs with direct decoding of different latent space dimensions. Notice that higher dimensionality does not always correspond to a higher ELBO.

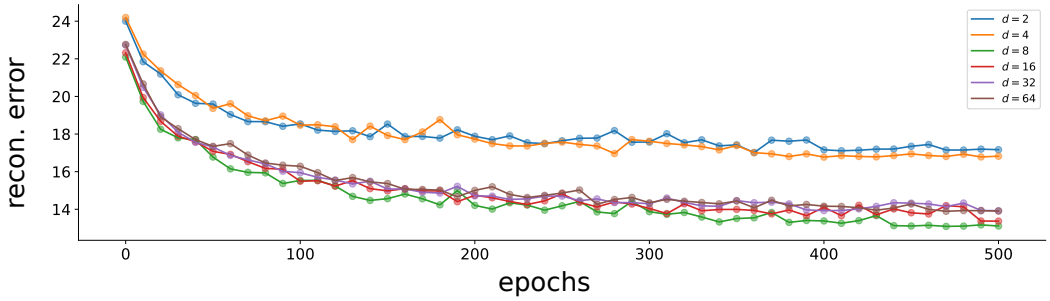


Figure 5.2: Comparison of reconstruction loss during training of VAEs with direct decoding of different latent space dimensions. Notice that higher dimensionality does not always correspond to a lower reconstruction loss.

that, we notice that close molecules tend to be similar to each other as expected. Indeed, close molecules share structural properties. Not all points in the latent space map to molecular graphs and therefore some point between two molecules is missing. This interpolation presets several discontinuities.

Reconstructions In Figure A.1, we plot reconstructions of samples from the test set. For this plot we used a VAE with $d = 32$ and *recurrent decoding* since it is the model with lower reconstruction error. The ability of perfectly reconstructing original samples is desirable but unfortunately extremely hard to achieve. However, from many compounds, the VAE is able to output almost the same structure as the input. It is remarkable that, even when the reconstruction is not correct, the output is still a valid molecule.

Latent space In Figure A.2, we show how the latent space looks like in a model trained with an embedding dimensionality of 2. For this plot, we use a recurrent model since it is the one with the lowest reconstruction error. We

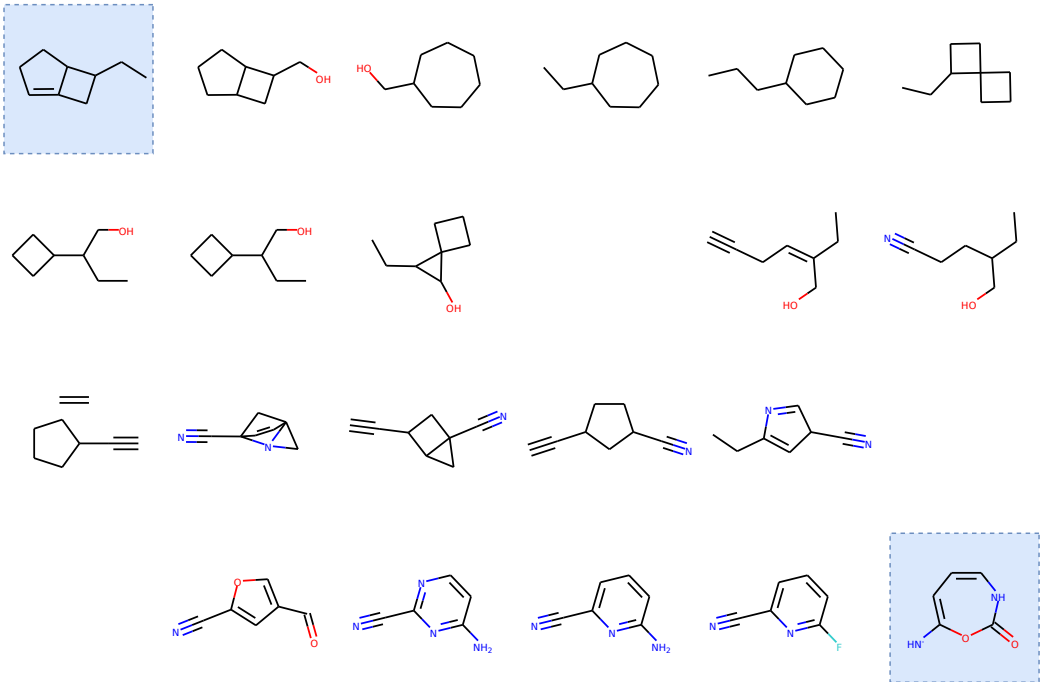


Figure 5.3: Interpolations from two random samples (inside the blue squares) using a VAE with $d = 8$ and direct prediction model. Notice that close molecules tend to be similar to each other. A blank spot indicates that the model outputs an invalid molecular graph from that point of the latent space.

take samples using a symmetric grid 1×1 centered at the origin. Similarly to the interpolation plot, also here not all points map to valid molecules. Moreover, transitions an neighbour similarity are more clear. Most of the compounds have a similar one nearby. This property was expected since it naturally appears with the use of variational auto-encoders.

5.3 VAEs with reinforcement learning

In this section, we study the effects of combining VAEs with reinforcement learning. First we discuss the effect of the hyperparameter λ which regulates the trade-off between the VAE and RL losses. We also study the effect of different ways to forward the outputs of a VAE (as we discussed in 3.2).

For all experiments, similarly to Guimaraes et al. (2017), we start from pre-trained VAE models and then we fine-tune them training using a reinforcement learning objective for further epochs. At early stages of experimentation, we observed that non pre-trained models collapse immediately. They

successfully optimize metrics presented very high scores, but the uniqueness approached zero (i.e., they converged outputting very few samples). Indeed, the reinforcement learning objective does not optimize towards having multiple molecular graphs as outputs. Therefore, there is no constraint to the model in having any degree of diversity in the output. This behavior may indicate that pre-training is fundamental for matching the data distribution before using RL. In this way, we can see the pre-training as a regularizer for the latent space diversity.

For these experiments, we use a pre-trained VAE with $d = 8$ and *direct decoding*, and we further train for 200 epochs. The reward network (see Section 3.5) is implemented using a 2-layered R-GCN of dimensionalities [128, 64] to process the graph, followed by an aggregation operation of dimensionality 128. Subsequently, this vector is processed by a 3-layered MLP of dimensionalities [128, 64, 1] resulting in a scalar value that denotes the predicted reward. Nonlinearities between layers are tanh, and the last ones is a sigmoid function since we normalized all rewards to be $\in [0, 1]$ (see Section 3.6.2). To speed up training and avoid vanishing gradients, we train the reward networks every 5 decoder updates.

5.3.1 The effect of λ

As previously discussed in Section 3.5, we can train our models using a linear combination (with λ as parameter) between the VAE and RL losses. We hypothesize that lower λ values, i.e. the RL loss contributes more, should lead to models that output several molecules with desirable properties. However, these models might collapse since the RL loss does not guarantee the model to outputs samples that resemble the data distribution. Conversely, higher λ values might not have any effects on optimizing towards chemical objectives.

At early stage of this work, we investigated $\lambda \in \{0, 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.825, 1\}$ to evaluate our hypothesis. Unfortunately, we notice that the RL objective does not work well combined with the VAE one. In particular, it seems that any combination of the two losses, i.e., $\lambda \in (0, 1)$, results in collapsed models (i.e., very high ELBO and very low validity and reward scores). We hypothesize that the RL objective cannot be easily used in combination with the reconstruction loss. Indeed, the former pushes towards sampling molecules with some property when the latter penalizes samples not reconstructed correctly. During training, when the RL component acts updating the decoder to output molecules that maximize a reward, the reconstruction error is not minimized at all. Indeed, it appears that during training, the model cannot find an acceptable balance between these two losses, and therefore, it collapses.

Method		validity	uniqueness	novelty	logP	SAS	QED
baseline		64.50	99.84	70.88	0.21	0.32	0.46
no threshold	soft	96.50	1.35	99.90	0.66	0.67	0.40
	noise	97.40	1.23	99.90	**0.69**	0.83	0.39
	GS	**99.90**	**3.10**	99.70	**0.69**	0.89	0.38
threshold uniqueness $> 5\%$	soft	93.20	5.11	99.35	0.62	0.66	0.46
	noise	**98.50**	**6.70**	100.00	0.61	0.65	0.45
	GS	96.70	5.58	99.79	**0.68**	0.87	0.38

(a) Optimizing the octanol-water partition coefficient (logP).

Method		validity	uniqueness	novelty	logP	SAS	QED
baseline		64.50	99.84	70.88	0.21	0.32	0.46
no threshold	soft	93.00	4.95	98.82	0.49	**0.97**	0.52
	noise	**97.60**	1.74	99.90	0.50	0.96	0.49
	GS	92.60	**6.37**	99.03	0.47	0.91	0.55
threshold uniqueness $> 5\%$	soft	91.50	6.23	98.47	0.50	**0.95**	0.53
	noise	**98.10**	6.05	99.90	0.50	0.92	0.54
	GS	92.60	**6.37**	99.03	0.47	0.91	0.55

(b) Optimizing the synthetic accessibility score (SAS).

Method		validity	uniqueness	novelty	logP	SAS	QED
baseline		64.50	99.84	70.88	0.21	0.32	0.46
no threshold	soft	**94.13**	1.83	99.58	0.48	0.62	**0.60**
	noise	92.80	**3.88**	99.78	0.46	0.30	**0.60**
	GS	92.10	3.26	99.78	0.39	0.13	0.56
threshold uniqueness $> 5\%$	soft	**92.62**	5.36	99.27	0.47	0.60	**0.60**
	noise	91.50	**8.63**	98.91	0.49	0.56	**0.60**
	GS	91.70	5.13	99.89	0.40	0.20	0.56

(c) Optimizing the quantitative estimate of druglikeness (QED).

Table 5.3: Different types of feeding the reward network from a VAE. *Soft* denotes feeding directly with the continuous $\mathbf{X}$ and $\mathbf{A}$, *noise* indicates we applied Gumbel noise to $\mathbf{X}$ and $\mathbf{A}$, and *GS* stands for the use of the Gumbel-Softmax trick. Here we used $\lambda = 0$ with $d = 8$ and *direct decoder*. We also show how these scores changes when we apply early stopping with a threshold on the uniqueness. Shaded cell denotes which metric is direct optimized.

5.3.2 Forwarding X and A

In this experiment, we evaluate three different approaches, as previously discussed in Section 3.2, that acts between the output of the generation process and the input of the reward network. These methods take both the annotation matrix $\mathbf{X}$ and adjacency tensor $\mathbf{A}$ and further process them. In particular, they either i) directly feed them into the reward network, we call it *soft* approach, ii) apply Gumbel noise to them, which is equivalent of using a stochastic *behavioral* policy, or iii) use the Gumbel-Softmax Trick (Jang et al., 2017). We train these methods using $\lambda = 0$ (otherwise the model collapses as previously described), evaluating the average rewards every 5 epochs. We also employed early stopping using the uniqueness score for which we set an arbitrary threshold of being >5%. Otherwise, we consider the model to be collapsed. We optimize towards novelty and another score in three different settings: the octanol-water partition coefficient (logP), the synthetic accessibility score (SAS), and the quantitative estimate of drug-likeness (QED). During validation and test, we measure rewards as well as the validity and uniqueness scores with a sample size of 1k (see Section 3.6). We then evaluate the variations as mentioned above using these metrics.

Results In Table 5.3 we show how different forwarding approaches affect scores. No clear pattern indicates that one of the three forwarding methods (i.e., *soft* approach, Gumbel noise, and Gumbel-Softmax) is better than the others. Besides that, we can observe two general trends: i) the RL component effectively influence models to generate molecules with desired properties, and ii) models tend to collapse generating few various molecules.

Shaded cells in Table 5.3 denote which scores are direct optimized. From them, we can observe that they are always higher then the baseline which is the VAE without any RL optimization. This trend is expected and indicates that reinforcement learning is useful to optimize towards metrics. We observe a clear trend towards very high validity score as well. This is likely due to the implicit optimization of valid molecules since invalid ones receive zero reward during training. Therefore, the generator is optimized to generate mostly valid molecular graphs. Novelty score approaches 100% in all settings showing that the model can generalize and learn to output molecules that are not in the training set.

We observe low uniqueness scores in all settings that seems to indicate that models tend to collapse generating only a few samples. Notice that, we set an arbitrary threshold on uniqueness score beyond which we consider a model to be collapsed. Thus, we report results when we either stop training after reaching the threshold or when we keep training regardless. Scores

of collapsed models are not always higher than non collapsed one. Both rewards and validity scores are slightly worse in non collapsed models but never too distant from the collapsed one. This indicated that employing early stopping is useful to avoid undesired behavior and does not compromise the performance of the models.

5.4 Generative Adversarial Networks

In this section, we describe a sequence of experiments to analyze the behavior of several variations of molecular graph GANs. For all experiments on GANs, we employ the WGAN with gradient penalty (WGAN-GP), a more stable variation of generative adversarial network (see Section 2.2). We train WGAN-GP models using settings and architectures as described in Section 5.1. Additionally, we estimate the earth mover's distance using all data-point from the test set and the same amount of samples from the generator. Notice that, for simplicity, we will often refer WGAN-GP as WGAN.

We first investigate the use of mini-batch discrimination and feature matching. Subsequently, we compare performance of different decoding functions (as explained in Section 3.2) as well as different latent space dimensionalities. We also analyze the quality of our models plotting a latent space visualization of an $\mathbb{R}^2$ model.

5.4.1 Feature matching and mini-batch discrimination

As introduced in Section 2.2, in a GAN setting, both feature matching, and mini-batch discrimination are techniques that prevent models from collapsing modes. This experiment is motivated by early stages of work when we observed that our standard GAN and WGANmodels were prone to collapse. Thus, we study the effect of these methods and their combination within our molecular graph GAN. We use feature matching on the last hidden layer of the discriminator. When using mini-batch discrimination, we added an extra component to our model. A 1-layered MLP, with hidden dimensions 8, transforms the output from the R-GCN module. Subsequently, we take the average of these representations are to have a minibatch-level vector. We then append this vector to all intermediate representations of the minibatch in the last hidden layer of the discriminator.

Additionally, as for VAEs, we investigate the effect of using either a *soft* forward of the annotation matrix $\mathbf{X}$ and adjacency tensor $\mathbf{A}$, or adding Gumbel noise to them, or employing the Gumbel-Softmax trick. For this experiment we use a WGAN with dimensionality $d = 8$ and *direct decoding*. To

Method		*no feature matching*			feature matching		
		validity	uni.	novelty	validity	uni.	novelty
no minib. discrim.	soft	**77.40**	**20.32**	63.70	83.40	75.30	63.91
	noise	77.20	12.86	68.52	<u>87.90</u>	**<u>79.64</u>**	<u>65.75</u>
	GS	77.70	16.99	**94.21**	**95.30**	7.87	**86.78**
minib. discrim.	soft	89.20	**67.71**	66.82	**93.20**	42.49	67.81
	noise	89.40	63.87	65.21	85.80	**60.72**	65.38
	GS	**95.80**	17.12	**83.09**	72.10	7.77	**93.48**

Table 5.4: Comparisons between WGAN models that uses or not use mini-batch discrimination and/or feature matching. Models have $d = 8$ and *direct decoding*. We explore *soft* forward, Gumbel noise and Gumbel-Softmax (GS) that act between the generator and the discriminator. *unique.* stands for uniqueness where *minib. discrim.* stands for minibatch discriminator. We highlight the best result for each combination between mini-batch discrimination and feature matching. We underline results of the model with the highest sum of scores.

make a comparison, we measure validity, uniqueness, and novelty, and we take the sum of these scores to pick the best setting for further use in other experiments. We additionally employ model selection using the maximum score during training, evaluating the sum of these scores every 10 epochs.

Results From experimental results, which we show in Table 5.4, we observed that the best setting is the combination of feature matching while forwarding $\mathbf{X}$ and $\mathbf{A}$ adding Gumbel noise. This setting has the maximum sum of validity, uniqueness and novelty scores. Thus, we select this model for further experiments.

Additionally, even though we employ early stopping for model selection, we inspect the behavior of these models after that point. Surprisingly, most of the models eventually collapse before reaching the end of the training. In particular, it seems that the Gumbel-Softmax is quite unstable. We re-ran multiple instances of these experiments, and every time these models collapsed. Indeed, all settings that employ the Gumbel-Softmax tick failed to reach the final epoch of the training without collapsing.

Besides, we observe that both feature matching and mini-batch discrimination prevent collapsing reaching good results. Overall, we do not observe a big gap between using one technique, or the other, or the combination of the two. However, using feature matching only with Gumbel noise results to

be the best setting with respect to our predefined criteria. It is not clear why their combination does not outperform the single use of these techniques. We decided to do not explore more in details this direction, and we will leave this for future work.

5.4.2 Dimensionality and decoding functions

In this experiment, we want to show the effects on performance of both the latent space dimension and the type of decoding function employed. We use results from the previous experiment to choose whether to use feature matching and/or mini-batch discrimination as well as which kind of forwarding approach. We investigate $d \in \{2, 4, 8, 16, 32, 64\}$ using comparison measures such as the earth mover's distance on the test set, as well as validity, uniqueness, and novelty scores calculated as an average from a sample size of 1k.

Additionally, we also explore the three type of decoding as described in Section 3.2.2. After having explored experimental results from VAEs, we expect similar results on GANs namely i) higher dimension are not necessarily better, and ii) *dot-product* and *recurrent decodings* should perform slightly better than the *direct decoding*.

Results Results from this experiment are reported in Table 5.5. Two trends emerge: i) *direct decoding* seems to be slightly better than other decodings in capturing the data distribution and in generating valid molecular graphs, and ii) in high dimensions the *dot-product decoder* does not perform well. Notice that, we employ feature matching with Gumbel noise in these experiments (as a result of model selection from the previous experiment).

Results from *direct decoding* indicates that, within our WGAN settings, it is the best decoding function. Indeed, when using *direct decoding*, most of the dimensionalities present a validity score >85% where they are approximately <70% using the *dot-product decoder* and approximately <80% using the *recurrent decoder*. Moreover, *direct decoding* models present a higher earth mover's distance (EMD) indicating that they tend to match the data distribution better. This behavior is also evident from the novelty score that is much lower than the other decodings. Less novelty indicates that the models also learn to replicate some of the original data.

Models with *dot-product* and recurrent decoding performed worse in term of both EMD and validity. In particular, it seems that *dot-product* is not well suited for high dimensions, resulting in very low validity scores. Additionally, in all three decoding settings, we observe that lower dimension (e.g., $d = 2$ or $d = 4$) have a worse uniqueness score. It seems that, in low dimensional

Method	*direct decoding* validity	uniqueness	novelty	EMD
$d = 2$	71.50	21.40	59.72	-155.83
$d = 4$	85.90	33.22	**64.38**	-121.42
$d = 8$	88.40	40.50	60.36	-128.92
$d = 16$	**94.10**	51.56	56.85	-99.34
$d = 32$	90.10	**63.91**	62.42	-98.28
$d = 64$	89.00	55.51	57.87	**-96.41**
Method	*dot-product decoding* validity	uniqueness	novelty	EMD
$d = 2$	66.20	9.67	93.81	-177.31
$d = 4$	**72.00**	19.17	**98.06**	**-157.59**
$d = 8$	64.90	53.31	95.38	-161.62
$d = 16$	39.60	63.38	96.21	-229.12
$d = 32$	11.30	**80.53**	88.50	-225.52
$d = 64$	12.50	68.80	96.00	-176.31
Method	*recurrent decoding* validity	uniqueness	novelty	EMD
$d = 2$	53.90	12.57	99.55	-174.63
$d = 4$	69.30	10.39	**100.00**	-232.09
$d = 8$	59.20	**67.57**	90.03	**-156.24**
$d = 16$	52.50	34.48	**100.00**	-278.16
$d = 32$	**80.30**	45.95	99.50	-202.13
$d = 64$	53.80	42.94	95.72	-238.47

Table 5.5: Comparison of different latent space dimensionalities and decoding functions in a WGAN setting with feature matching. We highlight the best scores for each decoding function.

spaces, the generator fails to learn a large set of different transformation, but instead it converges outputting only a few samples.

In Figure A.3, we show how the latent space looks like in a model trained with an embedding dimensionality of 2. For this plot, we use a direct prediction decoding since it is the one with the highest validity score. We take samples using a symmetric grid 2×2 centered at the origin. Similarly to the latent space of a VAE, also here not all points map to valid molecules and most of the compounds have a similar one nearby. Differently from a VAE, it is evident that the models learns to output less compounds (lower diversity).

5.5 GANs with reinforcement learning

In this section, we study the effects of combining WGANs with reinforcement learning. We first study the effect of the hyperparameter λ which regulates the trade-off between the WGAN and RL losses. Similarly to the experiments on VAEs with RL, we start from pre-trained WGAN models and we fine-tune them training using a reinforcement learning objective for further epochs. We make this choice for the same reasons previously explained in Section 5.3. Similarly to Section 5.3, we also optimize towards novelty and a single score in three different settings: the logP, the SAS, and the QED (as explained in Section 3.6.3 and defined in Section 3.5).

For these experiments, we use a pre-trained feature matching WGAN with $d = 32$ and *direct decoding* and Gumbel noise, further training for 200 epochs. Additionally, we use the same reward network architecture and hidden dimensions as described for the VAEs models in Section 5.3. To speed up training and avoid vanishing gradients, we train the reward networks every 5 decoder updates (similar to the n_{critic} in Goodfellow et al. (2014)).

5.5.1 The effect of λ

Differently from the combination between VAEs and reinforcement learning, we observed that varying the λ parameter (i.e., the trade-off between the WGAN and RL losses) does not present undesired collapsing behaviors. We hypothesize that within GAN models, the reinforcement learning objective does not contrast the generative objective since does not include any reconstruction loss. Therefore, they can be optimized together with different weights assigned to one or the other (i.e., controlling λ).

In this experiment, we then expect λ to be negatively correlated with the reward since low λ values correspond to high contribution from the RL loss. Conversely, higher λ values might not have any effects on optimizing the model towards chemical objectives. Thus, we also expect λ to be positively correlated with the uniqueness score since high λ values corresponds to optimize towards matching the data distribution more (since most of the gradient contribution would be from the GAN objective).

We explore $\lambda \in \{0, 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.825, 1\}$ to evaluate our hypothesis. We compute the validity, uniqueness, and novelty scores as well as all chemical rewards to show differences between the optimized one and the others. We run a validation every 5 epochs as an average from 1k samples.

Results From the experimental results in Table 5.6, it emerges that our hypotheses were correct and that λ regulates the bias between WGAN and RL in the generative process. In particular, we observe the positive and negative correlations, as previously hypothesized, between λ, rewards, and uniqueness values.

The negative correlation between λ and the rewards is clear since lower the λ value is, higher the currently optimized reward is. Indeed, the best results for all rewards are when $\lambda = 0$. As we also reported in Section 5.3, here we observe a trend towards very high validity score as well due to the implicit optimization of valid compounds.

We observe that when we optimize either the SAS or the logP, the other one increases as well. This unexpected behavior can be explained because it might be that more soluble compounds are also easily synthesizable and vice versa. We can also clearly notice the positive correlation between λ and the uniqueness score. Indeed, the latter tend to be very low for $\lambda \to 0$ showing that as much as the RL is higher than the GAN loss, the model tends to do not match the data distribution anymore.

For each reward, we also plot its evolution during training with different λ values in Figure 5.4. From these plots, it is even more clear how λ affect scores. For the first part of the training (pre-training), all scores are stable and around the average of the data distribution. Depending on the value of λ, when we introduce the RL loss, the values of these scores rise according. In particular, for high value, the bias towards metrics is none or small, and the scores tend to be steadily around the same value. Conversely, for low values, the rewards immediately rise.

5.6 Overall analysis

In this section, we do not explore new settings, but we instead do an overall analysis of our models as well as making a comparison of our best ones against works similar to ours. We compare with both variational and adversarial approaches. Additionally, we analyze the distribution of the SAS to show how much RL bias the generative process. We also present the top four molecules by rewards from our models, discussing their properties and the differences between the ones generated by our VAEs or our WGANs.

5.6.1 Comparison with other VAE-based works

We compare our models against recent likelihood-based methods that utilize variational approaches as well. We report a comparison with CharacterVAE

Method	validity	uniqueness	novelty	logP	SAS	QED
$\lambda = 0.0$ (full RL)	**100.00**	2.40	**100.00**	**0.68**	0.73	0.42
$\lambda = 0.125$	99.90	2.90	99.80	0.65	0.65	0.49
$\lambda = 0.25$	**100.00**	3.30	99.90	0.67	0.64	0.47
$\lambda = 0.375$	99.60	4.72	99.40	0.65	0.47	0.50
$\lambda = 0.5$	99.70	12.84	96.09	0.66	0.55	0.48
$\lambda = 0.625$	97.10	45.42	90.11	0.47	0.34	0.50
$\lambda = 0.75$	94.50	57.35	75.03	0.43	0.34	0.49
$\lambda = 0.875$	86.50	**74.10**	53.29	0.36	0.32	0.50
$\lambda = 1.0$ (no RL)	90.10	63.91	62.42	0.30	0.28	0.50

(a) Optimizing the octanol-water partition coefficient (logP).

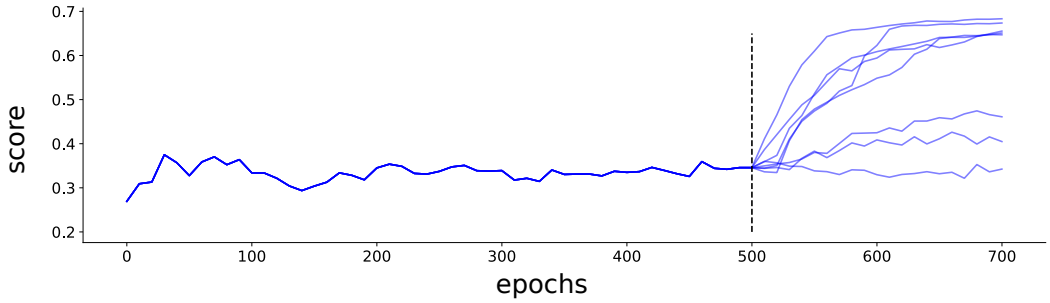
Method	validity	uniqueness	novelty	logP	SAS	QED
$\lambda = 0.0$ (full RL)	**100.00**	1.30	**100.00**	0.51	**0.99**	0.46
$\lambda = 0.125$	**100.00**	1.80	**100.00**	0.49	**0.99**	0.45
$\lambda = 0.25$	**100.00**	1.10	**100.00**	0.50	0.98	0.46
$\lambda = 0.375$	**100.00**	3.50	99.90	0.50	0.98	0.44
$\lambda = 0.5$	99.90	12.61	95.70	0.49	0.92	0.47
$\lambda = 0.625$	98.10	25.28	89.50	0.42	0.82	0.50
$\lambda = 0.75$	95.90	38.79	81.75	0.37	0.68	0.47
$\lambda = 0.875$	88.30	55.15	75.08	0.35	0.53	0.50
$\lambda = 1.0$ (no RL)	90.10	**63.91**	62.42	0.30	0.28	0.50

(b) Optimizing the synthetic accessibility score (SAS).

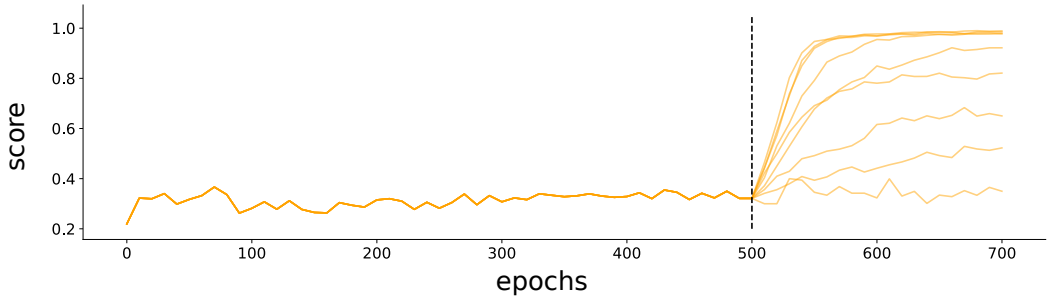
Method	validity	uniqueness	novelty	logP	SAS	QED
$\lambda = 0.0$ (full RL)	**100.00**	3.16	**100.00**	0.47	0.53	**0.61**
$\lambda = 0.125$	**100.00**	7.21	**100.00**	0.37	0.47	**0.61**
$\lambda = 0.25$	99.80	10.16	**100.00**	0.48	0.58	**0.61**
$\lambda = 0.375$	99.90	11.11	99.00	0.42	0.55	0.60
$\lambda = 0.5$	99.40	31.29	94.67	0.37	0.46	0.56
$\lambda = 0.625$	97.20	49.69	88.79	0.32	0.33	0.51
$\lambda = 0.75$	93.70	64.35	83.03	0.33	0.33	0.51
$\lambda = 0.875$	89.40	**69.69**	56.49	0.35	0.31	0.50
$\lambda = 1.0$ (no RL)	90.10	63.91	62.42	0.30	0.28	0.50

(c) Optimizing the quantitative estimate of druglikeness (QED).

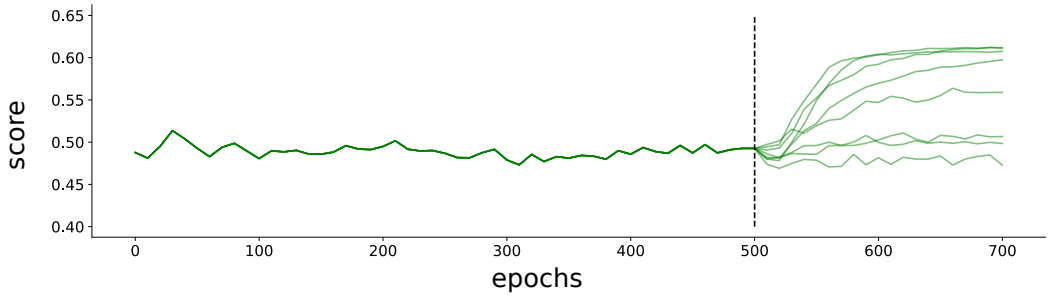
Table 5.6: Trade-off between WGAN and RL objectives varying the λ using feature matching WGAN with $d = 32$, *direct decoder*, and Gumbel noise.



(a) Optimizing the octanol-water partition coefficient (logP).



(b) Optimizing the synthetic accessibility score (SAS).



(c) Optimizing the quantitative estimate of druglikeness (QED).

Figure 5.4: Evolution of metrics during training with different λ values. For this plot, we used feature matching WGAN with $d = 32$, *direct decoder*, and Gumbel noise. The dashed line indicates from when we start introducing the RL loss. Notice that pre-training is the same for every model.

(Gómez-Bombarelli et al., 2016), GrammarVAE (Kusner et al., 2017), and GraphVAE (Simonovsky and Komodakis, 2018). In chapter 4, we presented a very short overview of these works. For this evaluation, we use our best VAE setting, selected from previous experiments, and a VAE trained with RL to optimize validity and novelty. Similarly, we also use our best WGAN setting, selected from previous experiments, and a WGAN trained with RL to optimize validity and novelty as well. We then selected a VAE with $d = 8$,

Method	validity	uniqueness	novelty
CharacterVAE	10.3	67.5	90.0
GrammarVAE	60.2	9.3	80.9
GraphVAE	55.7	76.0	61.6
GraphVAE/imp	56.2	42.0	75.8
GraphVAE NoGM	81.0	24.1	61.0
Our VAE	61.5	**97.6**	69.1
Our VAE with RL	**89.1**	11.1	**92.3**
Our WGAN	**89.2**	26.5	55.7
Our WGAN with RL	**99.6**	14.5	**97.7**

Table 5.7: Comparison of our best models against recent variational approaches for molecular generation. Baseline results are taken from Simonovsky and Komodakis (2018).

and a *direct decoding*, feature matching WGAN with $d = 32$, *direct decoding*, $\lambda = 0$, and Gumbel noise. The final evaluation scores are taken from an average of 10k random samples. The number of samples differs from previous experiments to be the same as baselines in Simonovsky and Komodakis (2018) which we report results from.

Results are presented in Table 5.7. It emerges that training without optimizing any metric except validity and novelty results in a model with a higher uniqueness score compared to the ones in previous experiments. Although the uniqueness scores of our models are slightly higher compared to GrammarVAE, the other baselines are superior in terms of this score. Even if we do not consider our models to be collapsed, such a low score confirms our hypothesis that they are prone to collapsing modes. On the other hand, we observe significantly higher validity scores compared to the VAE-based baselines.

Notice that, differently from our combined approach with RL, standard VAEs optimize the evidence lower bound (ELBO), and therefore, there is no explicit nor implicit optimization of outputs validity. Moreover, since a part of the ELBO maximizes reconstruction of the observations, the novelty in the sampling process is not expected to be high since it is not optimized directly. However, in all reported methods novelty is $> 60\%$ and, in the case of CharacterVAE, 90%. Though CharacterVAE can achieve a high novelty score, it underperforms in terms of validity. Our models, on the other hand, achieve both high validity and novelty scores outperforming all baselines.

5.6.2 Comparison with a GAN-based work

Here, we compare our work against Objective-Reinforced Generative Adversarial Networks (ORGAN) by Guimaraes et al. (2017) since it is the closest related work to ours. Their model relies on SeqGAN (Yu et al., 2017) to adversarially learn to output sequences while optimizing towards chemical metrics with REINFORCE (Williams, 1992). The main differences from our approach are that they model molecules using SMILES sequences instead of graphs, and their RL component uses REINFORCE while we use DDPG.

We compare to them using two of our best models combined with reinforcement learning, namely: a VAE ($d = 8$, *direct decoding*, and Gumbel noise), and a feature matching WGAN ($d = 32$, *direct decoding*, $\lambda = 0$ and Gumbel noise). We performed model selection using previous experiments picking the best performing models (selecting λ), according to the optimized reward. We also choose models with a uniqueness score of at least 10% to prevent an unfair comparison with collapsed ones.

Results for ORGAN are taken from Guimaraes et al. (2017) (Table 1). We report average scores for 6400 sampled compounds to match their setting. Additionally, we re-ran ORGAN experiments to report execution time, since it is not provided in the original work, training all models on same machine[1]. Guimaraes et al. (2017) reported three model variants, ORGAN, that uses a standard GAN with RL, OR(W)GAN, that uses WGAN with RL, and *Naive RL* that uses reinforcement learning only. Notice that these variations are applied after pre-training. They trained on a 5k subset of QM9 since their model is quite computationally expensive. They employed pre-training using maximum likelihood (MLE) for 250 epochs, and then they further train for 100. Differently from us, they use dropout (Srivastava et al., 2014) with a rate of 0.75 to prevent overfitting and to add stochasticity to their models. Following their work, we report three settings where optimize the same objectives using RL. For each of them, we train both our models using the same 5k subset from ORGAN as well as on the entire dataset. We also trained and pre-train for the same amounts of epochs as they did.

In Table 5.8, we show the comparison with ORGAN. From the experimental results, we mainly observe that our models i) require much less time to do the same amount of epochs, ii) perform poorly when trained on the 5k subset, and iii) outperform ORGAN in two out of three settings when trained on the entire dataset.

Indeed, in terms of training time, our models outperform ORGAN by a large margin (training at least ≈45 times faster). We think that the main reasons for the speedup are that we do not rely on sequential generation nor

[1]Intel Xeon CPU E5-2640 v3 @ 2.60GHz and Nvidia GeForce GTX Titan X (Pascal)

discrimination and that we do not evaluate rewards at every iteration (which is a bottleneck since it has to run on CPU by an external software). ORGAN relies on an RNN that outputs long sequences of SMILES characters, and these sequences receive rewards with a roll-out operation. This procedure cannot run in parallel, and it is expensive. Besides, both ORGAN and our models have a comparable number of parameters, with the latter being approximately 20% larger, so we do not consider this factor as a source of additional computational complexity. Computing molecular scores by the external software is a bottleneck to the learning procedure. This is done by both algorithms bu ours avoids evaluating rewards at each iteration.

For those reasons, we believe that our models that directly predict molecular graphs are superior in term of resources needed to train them. Indeed, we can train on the entire QM9 dataset (20 times larger than the subset) and still train in half of the time. However, differently, from ORGAN, it seems that our model is incapable of learning without a large dataset. In fact, our approaches outperform ORGAN in optimizing the QED and the SAS, but not the logP, only when we train on the full dataset.

When we do worse, we hypothesize that *direct decoding* does learn to generalize enough to output new highly scored molecules. When we do better, we think it is mainly due to two factors: i) it should be easier to optimize a molecular graph predicted as a single sample than to optimize an RNN model, and ii) DDPG instead of REINFORCE provides a better gradient, and it improves the sampling procedure towards metrics.

5.6.3 Best samples

In Figure 5.5 and 5.6 we show some among the highest scoring samples (depending on the optimized reward) from our best VAE and WGAN models respectively. Both models are the one selected above as the best ones and optimized with reinforcement learning. Notice that values are not $\in [0, 1]$ since we report the actual chemical scores and not the normalized one.

We observe that both the VAE and the WGAN models converge to different best scoring molecules. However, these molecules have similar scores for the SAS and the logP models. We observe that the QED VAE is more capable of learning to generate molecule with high scores. This is probably due to the pre-training since the VAE decoder might have memorized part of the training set, among which there are high score molecules, and later used that to output molecules with desirable scores. Moreover, we hypotheses that the QED is in general harder to optimize compared to other scores.

We also noticed a recurrent pattern in the molecular structure of these best scoring compounds. For instance, when optimizing the QED, our VAE

Method	validity	diversity	QED	SAS	logP	time
ORGAN	88.2	0.55	0.52	0.32	0.35	9.6*
OR(W)GAN	85.0	0.95	0.60	0.54	0.47	10.1*
Naive RL	97.1	0.80	0.57	0.53	0.50	9.4*
Our VAE	88.8	**0.97**	0.54	0.60	0.63	**0.13**
Our VAE (full QM9)	92.9	0.84	0.60	0.48	0.56	**3.2**
Our WGAN	**99.7**	0.66	0.55	0.48	0.45	**0.21**
Our WGAN (full QM9)	**100.0**	**0.96**	**0.61**	0.54	0.47	**4.1**

(a) Optimizing the quantitative estimate of druglikeness (QED).

Method	validity	diversity	QED	SAS	logP	time
ORGAN	96.5	0.92	0.51	0.83	0.45	8.7*
OR(W)GAN	97.6	**1.00**	0.20	0.75	0.84	9.6*
Naive RL	97.7	0.96	0.52	0.83	0.46	10.6*
Our VAE	89.6	0.99	0.49	0.71	0.49	**0.09**
Our VAE (full QM9)	94.0	**1.00**	0.51	**0.86**	0.46	**2.2**
Our WGAN	**100.0**	0.89	0.51	0.70	0.63	**0.15**
Our WGAN (full QM9)	**99.8**	0.97	0.47	**0.92**	0.48	**3.3**

(b) Optimizing the synthetic accessibility score (SAS).

Method	validity	diversity	QED	SAS	logP	time
ORGAN	94.7	0.76	0.50	0.63	0.55	8.7*
OR(W)GAN	94.1	0.90	0.42	0.66	0.54	9.2*
Naive RL	92.7	0.75	0.49	0.70	**0.78**	10.6*
Our VAE	87.9	**0.98**	0.61	0.49	0.75	**0.09**
Our VAE (full QM9)	**95.2**	**0.98**	0.38	0.86	0.66	**2.1**
Our WGAN	**100.0**	0.72	0.51	0.71	0.57	**0.13**
Our WGAN (full QM9)	**99.8**	**0.93**	0.49	0.59	0.67	**3.1**

(c) Optimizing the octanol-water partition coefficient (logP).

Table 5.8: Comparison of our WGAN and VAE models against a similar work to ours: ORGAN (Guimaraes et al., 2017). For each setting we select our best model based on the score we are optimizing for as well as a uniqueness score of at least 10%. If not indicated, models are optimized on a 5k subset of QM9. Time is reported in hours.

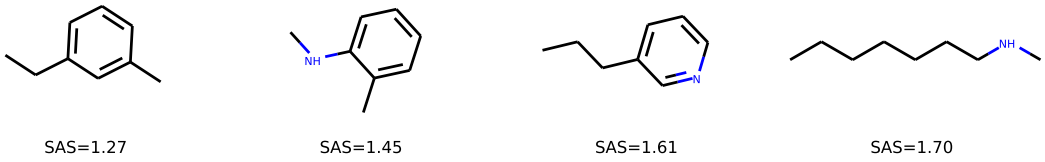
learns that a ring with 5 atoms and a small tail has one of the highest rewards in the search space explored by our model. Indeed, all top four molecules share this structure with small variations. We observe this trend in the other settings as well.

5.6.4 Score distributions

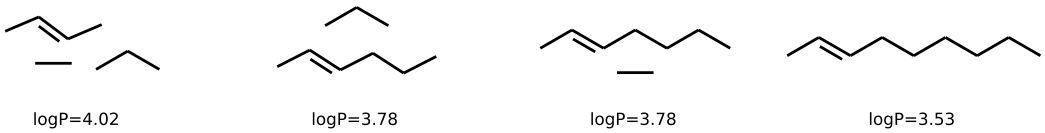
In Figure 5.7 we show a kernel density estimation of the distribution of the synthetic accessibility score (SAS) from the dataset compared to the one learned by two of our models. In particular, in Figure 5.7a, we use feature matching WGAN with $d = 32$ and Gumbel noise, and in Figure 5.7b the same model trained with $\lambda = 0$ to optimize the SAS. We estimate these densities with 10k sample from the models as well as all samples from the dataset to estimate the data one.

We first show how our normal WGAN is able to match the distribution of this score. It is quite surprising that the two distributions almost perfectly match. This indicates that the model learns to reproduce samples that resemble the one from the data. Secondly, we show how adding RL biases a lot the generative process. In particular, the SAS has to be minimized, and the resulting generative distribution of this score is shifted towards low values. The mean has changed from 4.55 to 2.46. Notice that the std of the data distribution is 1.22 so the change is substantial.

(a) Optimizing the quantitative estimate of druglikeness (QED).



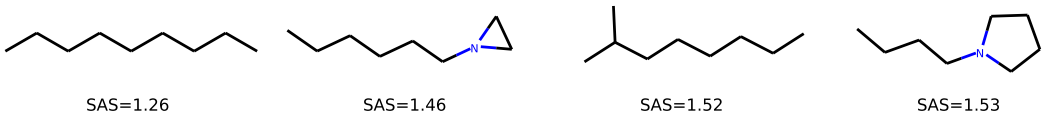
(b) Optimizing the synthetic accessibility score (SAS).



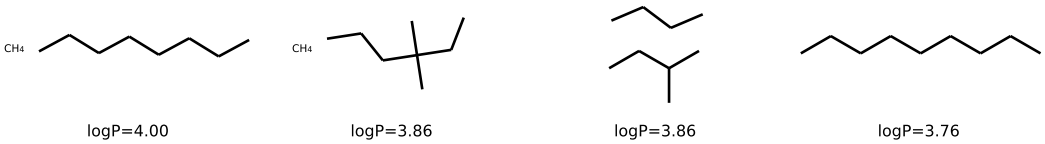
(c) Optimizing the octanol-water partition coefficient (logP).

Figure 5.5: Top four molecules with scores sampled from a VAE with $d = 8$, $\lambda = 0$, *direct decoder*, and Gubmel noise. Notice that values are not $\in [0, 1]$ since we report the actual chemical scores and not the normalized one.

(a) Optimizing the quantitative estimate of druglikeness (QED).

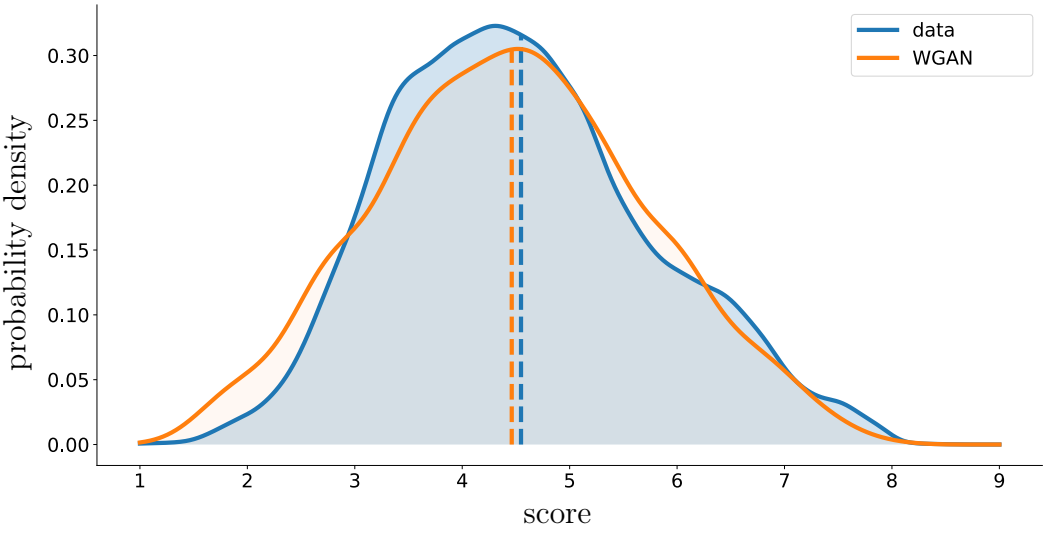


(b) Optimizing the synthetic accessibility score (SAS).

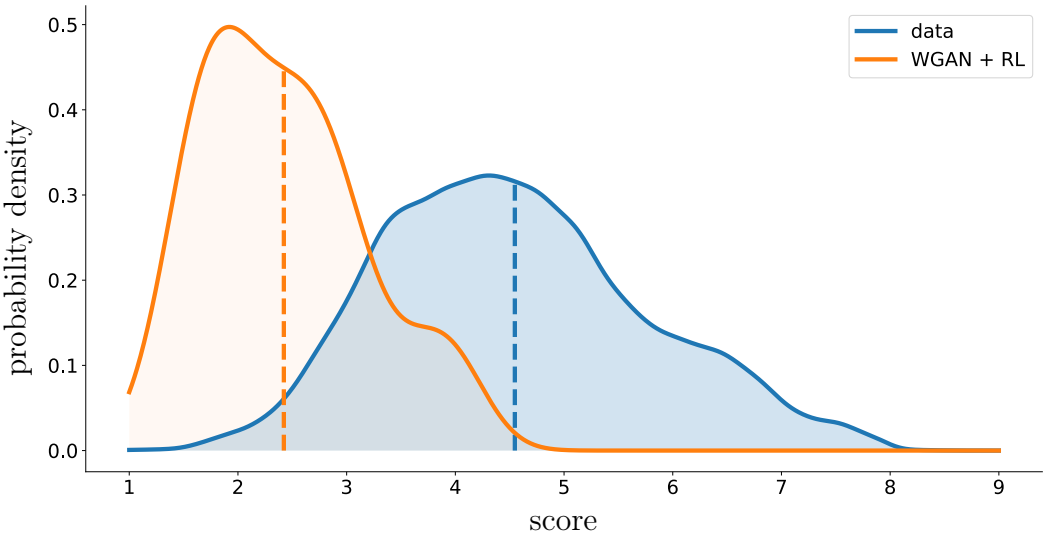


(c) Optimizing the octanol-water partition coefficient (logP).

Figure 5.6: Top four molecules with scores sampled from WGAN with $d = 32$, $\lambda = 0$, *direct decoder*, and Gubmel noise. Notice that values are not $\in [0, 1]$ since we report the actual chemical scores and not the normalized one.



(a) WGAN matches the data distribution .



(b) WGAN in combination with RL push the distribution of the synthetic accessibility score to be as low as possible.

Figure 5.7: The synthetic accessibility score (SAS) distribution of dataset and samples from either WGAN or WGAN with RL. We used $d = 32$ with *direct decoding*. With RL we used $\lambda = 0$.

Chapter 6

Conclusion and Future work

In this chapter, we draw conclusions on this thesis, and we present some directions for future work. One of the central questions we try to give an answer to is which is the best choice between likelihood-based and likelihood-free methods for molecular graph generation. In particular, between variational auto-encoders and generative adversarial networks. Although from experimental results (see Section 5.6) it seems that our WGAN models outperform our VAE ones, we prefer to do not make a strong statement. Indeed, where graph WGANs present high validity scores and high chemical metrics, they also exhibit low uniqueness, outputting a small set of diverse compounds. On the other hand, graph VAEs do not suffer from such problem but they cannot be optimized in combination with reinforcement learning.

6.1 Conclusion

6.1.1 Contributions

In the light of experimental results, we mainly identify contributions of this thesis in showing both variational and adversarial deep generative models for graphs capable of outputting high-quality molecules that optimize some arbitrary non-differentiable reward function (e.g., the quantitative estimate of druglikeness). From theoretical considerations and experimental results, we want to highlight that i) our models can perform equally or better than current state-of-the-art models for molecular generation, ii) we confirm the superiority, in term of computational resources needed to train, of one-shot graph-based prediction compared to the use of a recurrent SMILES prediction, iii) the use of a deterministic policy gradient empirically proved to be useful in biasing the generative process, and iv) we showed that likelihood-

based models are challenging to be optimized in combination of RL.

In some settings, we show that our models outperform current state-of-the-art approaches on similar tasks. Notice that we train them on a larger dataset to achieve such performances resulting in longer training time. Besides that, even with more extended training, our models showed to be even faster in the case of comparison with ORGAN (Guimaraes et al., 2017). Even though we do not compare the computational resources required to train other SMILES-based generative models, based on empirical results, we believe that the use of non-recurrent decoding is beneficial in these terms. Indeed, a graph is predicted as a whole object and then training can be easier parallelized compared to recurrent methods required in case of using SMILES strings. One may decide to use SMILES without a recurrent generation, but this would go against the trend of learning more general models that incorporate both syntax and semantic in structured languages. Besides that, we leave the study of a more in-depth analysis of recurrent and non-recurrent deep molecular graph generation for future work.

6.1.2 Limitations

We also individuate some limitations of our work. We show the performance of all our models while outputting compounds of at most 9 atoms. When using non-recurrent models, the prediction step and the graph convolution networks layers dimensions scale quadratically in the number of nodes. Moreover, we evaluate the generation of small molecules for drug discovery, and there is no guarantee that for larger input/output spaces, our models would learn to generate high-quality molecules. These are limitations of the present study, and we leave further investigations for future work.

We employ reinforcement learning using an off-policy deterministic policy gradient method. We supported this choice from the earliest stages of our work when we observed instability when training with a stochastic policy. DDPG was observed to perform well in high-dimensional spaces in other tasks, and we observe the same trend in our study. The graph predicted as a whole action allowed the use of a much simpler form of RL that avoids the roll-out of multiple re-scaled rewards through several steps. This aspect might be beneficial in larger scale implementations since it has proved to be very fast (even though, as previously mentioned, we showed results with a relatively small molecular size). However, we miss an evaluation of different RL techniques that might influence the outcomes of our experiments.

Although we show that reinforcement learning is needed to optimize towards non-differentiable rewards, we also observe susceptibility to *mode collapse* in most of our models. For instance, both the WGAN and the RL

objective do not encourage the generation of diverse and non-unique outputs whereby the model tends to be pulled towards a solution that only involves little sample variability. The latter ultimately results in the generation of only a handful of different molecules if we do not stop training early. The employment of early stopping is a straightforward way to address the problem, but it does not avoid it. This issue is a specific limitation of our models that might be addressed in future work.

Our likelihood-based models proved to be impossible to be optimized in combination with reinforcement learning. After pre-training, as we explained in Section 5.3, we are able to train with RL only since any other combination with the ELBO objective leads to collapsing models. Conversely, the use of implicit models, such as generative adversarial networks, shows non-collapsing behaviors. Moreover, it is possible to control the trade-off between the WGAN and the RL objective through a parameter (we called it λ). We argue that this difference occurs because the reconstruction error is incompatible with a reinforcement objective and that our variational approach has to learn the node order when the adversarial approach does not have to. Indeed, optimizing the generation of compounds with some properties is opposite to optimizing reconstruction, since when one is optimal, the other is not and vice versa. Moreover, during pre-training, our likelihood-based methods have to learn a particular order of nodes in the reconstruction. The latter is due to the ELBO term that forces the model to output the same sequence of the nodes. Trying to make the ELBO permutation invariant would require matching algorithms or integrating (summing) over all permutations. In general, finding the optimal match corresponds to solving graph isomorphism with is a well-known to be not solvable in polynomial time (i.e., usually exponentially complex in the number of nodes). However, a polynomial time algorithm is known for planar graphs with maximum vertex degree (Pemmaraju and Skiena, 2003) which in the case of molecular graphs. We will leave the study of more sophisticated ways to compute the likelihood for future work.

6.2 Future work

Except for some points discussed above, we identify three principal directions for future work: i) the employment of more sophisticated techniques for generative processes to avoid *mode collapse*, ii) the use of recent variational/adversarial combinations, and iii) straighten experimental results on larger dataset and more realistic reward functions (appropriately designed to incentive the match with a biological target).

As we previously discussed, *mode collapse* is an undesired behavior that emerges from many of our models. A promising area for future research would be investigating more in-depth the nature of this issue and applying specific techniques to address it. For instance, works such as Srivastava et al. (2017) have been proposed to reduce *mode collapse* in GANs.

On a similar line, (Dumoulin et al., 2017; Mescheder et al., 2017; Rosca et al., 2017) combine variational approaches with adversarial learning. These approaches may address some of the issues of our models. In particular, some of them i) train an encoder function (very useful) within an implicit model, ii) might permit the training of semi-variational models that does not collapse when trained in combination with RL (i.e., $\lambda > 0$), and iii) might reduce *mode collapse* as we state above.

It is well-known that the use of a larger dataset is beneficial to deep learning models. As future work, we propose to train our models on ChEMBL (Gaulton et al., 2011). It is a >3 million compounds dataset widely used in chemistry and pharmaceutical sciences. ChEMBL mainly contains bioactive molecules with drug-like properties. It contains much more variety and molecular information than QM9. We think that the use of ChEMBL would increase the performance of our models and the quality of the generated molecules. Additionally, in future work, we should investigate the generation of larger compounds.

Finally, within this work, we aimed to propose and show generative models for small molecules optimized towards simple non-differentiable objectives (e.g., the synthetic accessibility score). As future work, we would propose to explore more sophisticated rewards such as scoring functions to match particular biological targets. These scoring functions are commonly used in drug discovery research, but they are built *ad hoc* depending on the purpose of the drug (e.g., chemotherapy for a particular organ).

Part II

Appendix

Appendix A

Plots

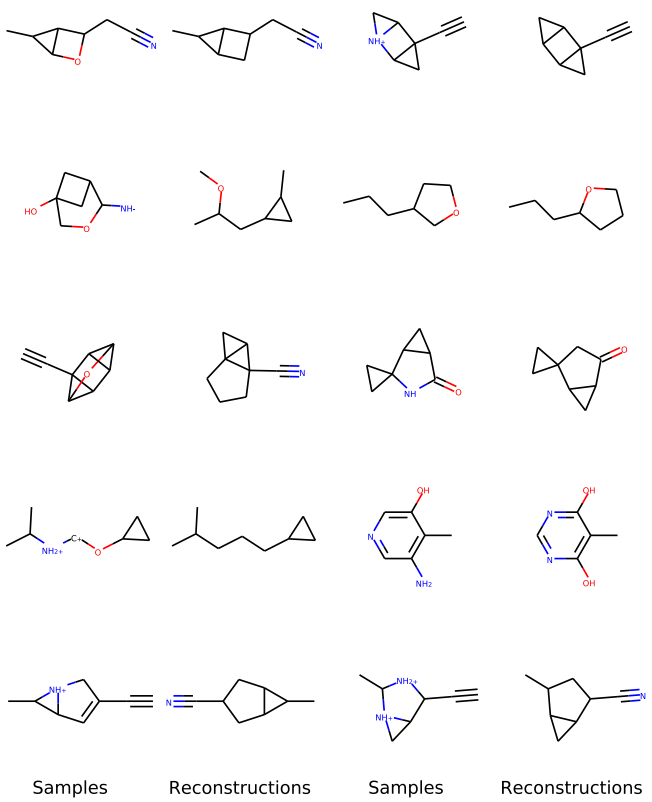


Figure A.1: Plot of original samples and respectively reconstructions. For this plot we used a VAE with $d = 32$ and recurrent prediction model. Notice that most of the reconstructions do not perfectly match the original samples but almost all atoms and bonds are predicted correctly.

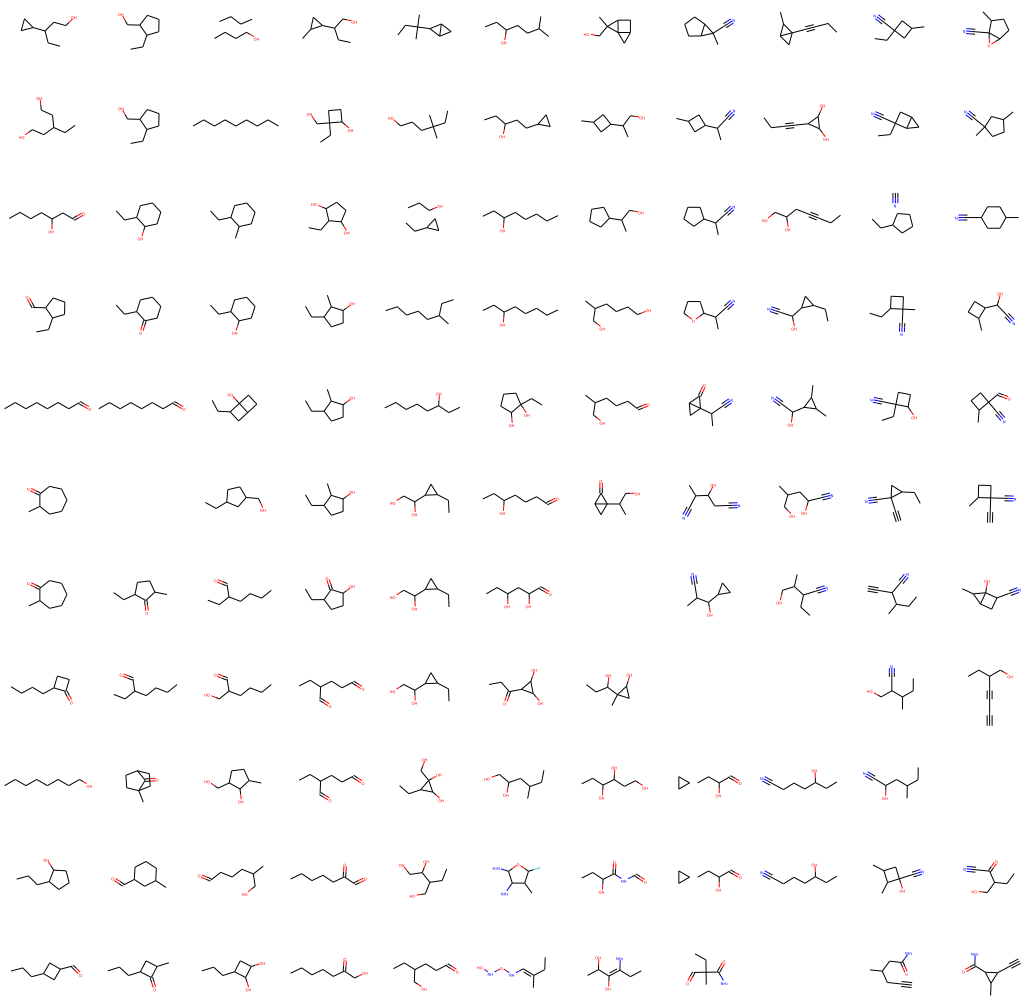


Figure A.2: Plot of the latent space around the origin. For this plot we used a VAE with $d = 2$ and recurrent prediction model. We take samples using a symmetric grid 1×1 centered at the origin of the $\mathbb{R}^2$ space resulting in plotting the interval $[-0.5, 0.5]$ along the x axis and $[-0.5, 0.5]$ along the y axis. Notice that close points map to similar compounds.

Figure A.3: Plot of the latent space around the origin. For this plot we used a WGAN with $d = 2$ and direct prediction model. We take samples using a symmetric grid 4×4 centered at the origin of the $\mathbb{R}^2$ space resulting in plotting the interval $[-2, 2]$ along the x axis and $[-2, 2]$ along the y axis. Notice that close points map to similar compounds.

Appendix B

Derivations

B.1 Evidence Lower Bound

$$
\begin{aligned}
\log p(\mathbf{x}) &\overset{*}{=} \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}\left[\log p(\mathbf{x})\right] \\
&= \int q(\mathbf{z}|\mathbf{x}) \log p(\mathbf{x}) d\mathbf{z} \\
&= \int q(\mathbf{z}|\mathbf{x}) \log \frac{p(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{p(\mathbf{z}|\mathbf{x})} d\mathbf{z} \\
&\overset{**}{=} \int q(\mathbf{z}|\mathbf{x}) \log p(\mathbf{x}|\mathbf{z}) \frac{p(\mathbf{z})}{q(\mathbf{z}|\mathbf{x})} \frac{q(\mathbf{z}|\mathbf{x})}{p(\mathbf{z}|\mathbf{x})} d\mathbf{z} \\
&= \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z})] + \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}\left[\log \frac{p(\mathbf{z})}{q(\mathbf{z}|\mathbf{x})}\right] + \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}\left[\log \frac{q(\mathbf{z}|\mathbf{x})}{p(\mathbf{z}|\mathbf{x})}\right] \\
&= \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z})] - D_{KL}\left[\, q(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z}) \,\right] + D_{KL}\left[\, q(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z}|\mathbf{x}) \,\right] \\
&\overset{***}{\geq} \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z})] - D_{KL}\left[\, q(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z}) \,\right]
\end{aligned}
$$

* expectation with respect to a distribution that does not influence $\log p(\mathbf{x})$
** multiplication and division by the *proposal distribution* $q(\mathbf{z}|\mathbf{x})$, notice that they have to share the same support
*** holds since $D_{KL}\left[\, p \parallel q \,\right] \geq 0 \; \forall p, q$

B.2 Importance sampling

$$
\begin{aligned}
\log p(\mathbf{x}) &= \log \int p(\mathbf{x}, \mathbf{z}) d\mathbf{z} \\
&= \log \int p(\mathbf{x}|\mathbf{z}) p(\mathbf{z}) d\mathbf{z} \\
&\overset{*}{=} \log \int p(\mathbf{x}|\mathbf{z}) \frac{p(\mathbf{z})}{q(\mathbf{z}|\mathbf{x})} q(\mathbf{z}|\mathbf{x}) d\mathbf{z} \\
&= \log \mathbb{E}_{q(\mathbf{z}|\mathbf{x})} \left[p(\mathbf{x}|\mathbf{z}) \frac{p(\mathbf{z})}{q(\mathbf{z}|\mathbf{x})} \right] \\
&\overset{**}{\geq} \mathbb{E}_{q(\mathbf{z}|\mathbf{x})} \left[\log p(\mathbf{x}|\mathbf{z}) \frac{p(\mathbf{z})}{q(\mathbf{z}|\mathbf{x})} \right] \\
&\overset{***}{\approx} \log \frac{1}{N} \sum_{i=1}^{N} p(\mathbf{x}|\mathbf{z}) \frac{p(\mathbf{z})}{q(\mathbf{z}|\mathbf{x})} \Bigg|_{\mathbf{z} \sim q(\mathbf{z}|\mathbf{x})}
\end{aligned}
$$

* multiplication and division by the *proposal distribution* $q(\mathbf{z}|\mathbf{x})$, notice that they have to share the same support
** Jensen's inequality
*** Monte Carlo sampling

Appendix C

Algorithms

C.1 Deep Deterministic Policy Gradient

Algorithm C.1 Deep Deterministic Policy Gradient algorithm[1]

Randomly initialize state-action network $Q^{\mu}_{\psi}(s,a)$ and policy $\mu_\theta(s)$ with weights ψ and θ. Initialize target networks $Q^{\mu}_{\psi'}$ and $\mu_{\theta'}$ with weights $\psi' \leftarrow \psi$, $\theta' \leftarrow \theta$, and empty replay buffer R

for episode = 1, M **do**

Initialize a random process ϵ for action exploration

Receive initial observation state s_1

for t = 1, T **do**

Select action $a_t = \mu_\theta(s_t) + \epsilon_t$ according to the current policy and exploration noise

Execute action a_t and observe reward r_t and observe new state s_{t+1}

Store transition (s_t, a_t, r_t, s_{t+1}) in R

Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R

Set $y_i = r_i + \gamma Q^{\mu}_{\psi'}(s_{i+1}, a_{i+1})\big|_{\mu_{\theta'}(s_{i+1})}$

Update ψ by minimizing the loss: $L = \frac{1}{N}\sum_{i=1}^{N}(Q_\psi(s_i,a_i) - y_i)^2$

Update θ using the sampled policy gradient:

$$\nabla_\theta J(\mu_\theta) \approx \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta \mu_\theta(s_i)\ \nabla_a Q^{\mu}_{\psi}(s_i,a_i)\big|_{a_i=\mu_\theta(s_i)}$$

Update the target networks: $\psi' \leftarrow \tau\psi + (1-\tau)\psi'$ and $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$

end for

end for

[1] from Lillicrap et al. (2016) with modified notation

Acronyms

AE auto-encoder.

CNN convolutional neural network.

ELBO evidence lower bound.

GAN generative adversarial network.

GCN graph convolution network.

logP octanol-water partition coefficient.

NN meural network.

QED quantitative estimate of druglikeness.

R-GCN relational graph convolution network.

RL reinforcement learning.

SAS synthetic accessibility score.

SGD stochastic gradient descent.

VAE variational auto-encoder.

WGAN Wasserstein GAN.

WGAN-GP WGAN with gradient penalty.

Bibliography

Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein generative adversarial networks. In *International Conference on Machine Learning*, pages 214–223.

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al. (2018). Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*.

Bellman, R. (2013). *Dynamic programming*. Courier Corporation.

Bickerton, G. R., Paolini, G. V., Besnard, J., Muresan, S., and Hopkins, A. L. (2012). Quantifying the chemical beauty of drugs. *Nature chemistry*, 4(2):90.

Borhani, D. W. and Shaw, D. E. (2012). The future of molecular dynamics simulations in drug discovery. *Journal of computer-aided molecular design*, 26(1):15–26.

Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. (2014). Spectral networks and locally connected networks on graphs. *International Conference on Learning Representations*.

Burda, Y., Grosse, R., and Salakhutdinov, R. (2016). Importance weighted autoencoders. *International Conference on Learning Representations*.

Chung, F. R. (1997). *Spectral graph theory*. American Mathematical Soc.

Comer, J. and Tam, K. (2001). *Lipophilicity profiles: theory and measurement*. Wiley-VCH: Zürich, Switzerland.

Dai, H., Tian, Y., Dai, B., Skiena, S., and Song, L. (2018). Syntax-directed variational autoencoder for molecule generation. In *International Conference on Machine Learning*.

Davidson, T. R., Falorsi, L., De Cao, N., Kipf, T., and Tomczak, J. M. (2018). Hyperspherical variational auto-encoders. In *Conference on Uncertainty in Artificial Intelligence*.

De Cao, N. and Kipf, T. (2018). MolGAN: An implicit generative model for small molecular graphs. *ICML 2018 workshop on Theoretical Foundations and Applications of Deep Generative Models*.

Defferrard, M., Bresson, X., and Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852.

Degris, T., White, M., and Sutton, R. S. (2012). Linear off-policy actor-critic. In *In International Conference on Machine Learning*. Citeseer.

Dumoulin, V., Belghazi, I., Poole, B., Mastropietro, O., Lamb, A., Arjovsky, M., and Courville, A. (2017). Adversarially learned inference. *International Conference on Learning Representations*.

Ertl, P. and Schuffenhauer, A. (2009). Estimation of synthetic accessibility score of drug-like molecules based on molecular complexity and fragment contributions. *Journal of cheminformatics*, 1(1):8.

Gaulton, A., Bellis, L. J., Bento, A. P., Chambers, J., Davies, M., Hersey, A., Light, Y., McGlinchey, S., Michalovich, D., Al-Lazikani, B., et al. (2011). ChEMBL: a large-scale bioactivity database for drug discovery. *Nucleic acids research*, 40(D1):D1100–D1107.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. *International Conference on Machine Learning*.

Gómez-Bombarelli, R., Wei, J. N., Duvenaud, D., Hernández-Lobato, J. M., Sánchez-Lengeling, B., Sheberla, D., Aguilera-Iparraguirre, J., Hirzel, T. D., Adams, R. P., and Aspuru-Guzik, A. (2016). Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 4(2):268–276.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

Grathwohl, W., Choi, D., Wu, Y. o. U. i. A. I., Roeder, G., and Duvenaud, D. (2018). Backpropagation through the void: Optimizing control variates for black-box gradient estimation. *International Conference on Learning Representations*.

Grover, A., Zweig, A., and Ermon, S. (2017). Graphite: Iterative generative modeling of graphs. In *Conference on Neural Information Processing Systems Bayesian Deep Learning Workshop*.

Guimaraes, G. L., Sanchez-Lengeling, B., Farias, P. L. C., and Aspuru-Guzik, A. (2017). Objective-reinforced generative adversarial networks (ORGAN) for sequence generation models. *arXiv preprint arXiv:1705.10843*.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. C. (2017). Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems*, pages 5769–5779.

Gumbel, E. J. (1954). Statistical theory of extreme valuse and some practical applications. *Nat. Bur. Standards Appl. Math. Ser. 33*.

Hammond, D. K., Vandergheynst, P., and Gribonval, R. (2011). Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150.

Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507.

Hjelm, D., Jacob, A. P., Che, T., Trischler, A., Cho, K., and Bengio, Y. (2018). Boundary-seeking generative adversarial networks. *International Conference on Learning Representations*.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Jang, E., Gu, S., and Poole, B. (2017). Categorical reparameterization with gumbel-softmax. *International Conference on Learning Representations*.

Jin, W., Barzilay, R., and Jaakkola, T. (2018). Junction tree variational autoencoder for molecular graph generation. *arXiv preprint arXiv:1802.04364*.

Johnson, D. D. (2017). Learning graphical state transitions. *International Conference on Learning Representations*.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *Proceedings of the 3rd International Conference on Learning Representations (International Conference on Learning Representations)*.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *International Conference on Learning Representations*.

Kipf, T., Fetaya, E., Wang, K.-C., Welling, M., and Zemel, R. (2018). Neural relational inference for interacting systems. *International Conference on Machine Learning*.

Kipf, T. N. and Welling, M. (2016). Variational graph auto-encoders. In *Conference on Neural Information Processing Systems Bayesian Deep Learning Workshop*.

Kipf, T. N. and Welling, M. (2017). Semi-supervised classification with graph convolutional networks. *International Conference on Learning Representations*.

Klebe, G. (2000). Recent developments in structure-based drug design. *Journal of Molecular Medicine*, 78(5):269–281.

Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86.

Kusner, M. J., Paige, B., and Hernández-Lobato, J. M. (2017). Grammar variational autoencoder. In *International Conference on Machine Learning*, pages 1945–1954.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Li, Y. (2017). Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*.

Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. (2016). Gated graph sequence neural networks. *International Conference on Learning Representations*.

Li, Y., Vinyals, O., Dyer, C., Pascanu, R., and Battaglia, P. (2018a). Learning deep generative models of graphs. *International Conference on Learning Representations*.

Li, Y., Zhang, L., and Liu, Z. (2018b). Multi-objective de novo drug design with conditional graph generative model. *arXiv preprint arXiv:1801.07299*.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. *International Conference on Learning Representations*.

Maddison, C. J., Mnih, A., and Teh, Y. W. (2017). The concrete distribution: A continuous relaxation of discrete random variables. *International Conference on Learning Representations*.

Maddison, C. J., Tarlow, D., and Minka, T. (2014). A* sampling. In *Advances in Neural Information Processing Systems*, pages 3086–3094.

Merz Jr, K. M., Ringe, D., and Reynolds, C. H. (2010). *Drug design: structure-and ligand-based approaches*. Cambridge University Press.

Mescheder, L., Nowozin, S., and Geiger, A. (2017). Adversarial variational bayes: Unifying variational autoencoders and generative adversarial networks. *International Conference on Machine Learning*.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *Conference on Neural Information Processing Systems*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529.

Moon, P. and Spencer, D. E. (1961). The vector Helmholtz equation. In *Field Theory Handbook*, pages 136–143. Springer.

Nowozin, S., Cseke, B., and Tomioka, R. (2016). f-gan: Training generative neural samplers using variational divergence minimization. In *Advances in Neural Information Processing Systems*, pages 271–279.

O'Searcoid, M. (2006). *Metric spaces*. Springer Science & Business Media.

Pemmaraju, S. and Skiena, S. (2003). *Computational Discrete Mathematics: Combinatorics and Graph Theory with Mathematica®*. Cambridge university press.

Peters, J. and Schaal, S. (2008). Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4):682–697.

Radford, A., Metz, L., and Chintala, S. (2016). Unsupervised representation learning with deep convolutional generative adversarial networks. *International Conference on Learning Representations*.

Rainforth, T., Kosiorek, A. R., Le, T. A., Maddison, C. J., Igl, M., Wood, F., and Teh, Y. W. (2018). Tighter variational bounds are not necessarily better. *International Conference on Machine Learning*.

Ramakrishnan, R., Dral, P. O., Rupp, M., and Von Lilienfeld, O. A. (2014). Quantum chemistry structures and properties of 134 kilo molecules. *Scientific data*, 1:140022.

Rosca, M., Lakshminarayanan, B., Warde-Farley, D., and Mohamed, S. (2017). Variational approaches for auto-encoding generative adversarial networks. *arXiv preprint arXiv:1706.04987*.

Ruddigkeit, L., Van Deursen, R., Blum, L. C., and Reymond, J.-L. (2012). Enumeration of 166 billion organic small molecules in the chemical universe database gdb-17. *Journal of chemical information and modeling*, 52(11):2864–2875.

Sakurada, M. and Yairi, T. (2014). Anomaly detection using autoencoders with nonlinear dimensionality reduction. In *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*, page 4. ACM.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. (2016). Improved techniques for training GANs. In *Advances in Neural Information Processing Systems*, pages 2234–2242.

Samanta, B., De, A., Ganguly, N., and Gomez-Rodriguez, M. (2018). Designing random graph models using variational autoencoders with applications to chemical design. *arXiv preprint arXiv:1802.05283*.

Sangster, J. (1997). *Octanol-water partition coefficients: fundamentals and physical chemistry*. John Wiley & Sons.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. (2009). The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80.

Schlichtkrull, M., Kipf, T. N., Bloem, P., van den Berg, R., Titov, I., and Welling, M. (2018). Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, pages 593–607. Springer.

Segler, M. H., Preuss, M., and Waller, M. P. (2018). Planning chemical syntheses with deep neural networks and symbolic AI. *Nature*, 555(7698):604.

Shuman, D. I., Narang, S. K., Frossard, P., Ortega, A., and Vandergheynst, P. (2013). The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 30(3):83–98.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *International Conference on Machine Learning*.

Simonovsky, M. and Komodakis, N. (2017). Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Conference on Computer Vision and Pattern Recognition*.

Simonovsky, M. and Komodakis, N. (2018). GraphVAE: Towards generation of small graphs using variational autoencoders. *arXiv preprint arXiv:1802.03480*.

Srivastava, A., Valkoz, L., Russell, C., Gutmann, M. U., and Sutton, C. (2017). VEEGAN: Reducing mode collapse in GANs using implicit variational learning. In *Advances in Neural Information Processing Systems*, pages 3308–3318.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Tucker, G., Mnih, A., Maddison, C. J., Lawson, J., and Sohl-Dickstein, J. (2017). Rebar: Low-variance, unbiased gradient estimates for discrete latent variable models. In *Advances in Neural Information Processing Systems*, pages 2624–2633.

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. (2015). Wavenet: A generative model for raw audio. In *9th ISCA Speech Synthesis Workshop*, pages 125–125.

Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. (2018). Graph attention networks. *International Conference on Learning Representations*.

Villani, C. (2008). *Optimal transport: old and new*, volume 338. Springer Science & Business Media.

Von Luxburg, U. (2007). A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416.

Vondrick, C., Pirsiavash, H., and Torralba, A. (2016). Generating videos with scene dynamics. In *Advances In Neural Information Processing Systems*, pages 613–621.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4):279–292.

Wei, X., Gong, B., Liu, Z., Lu, W., and Wang, L. (2018). Improving the improved training of Wasserstein GANs: A consistency term and its dual effect. *International Conference on Learning Representations*.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Reinforcement Learning*, pages 5–32. Springer.

You, J., Ying, R., Ren, X., Hamilton, W. L., and Leskovec, J. (2018). GraphRNN: A deep generative model for graphs. In *International Conference on Machine Learning*.

Yu, L., Zhang, W., Wang, J., and Yu, Y. (2017). SeqGAN: Sequence generative adversarial nets with policy gradient. In *Association for the Advancement of Artificial Intelligence*, pages 2852–2858.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., and Smola, A. J. (2017). Deep sets. In *Advances in Neural Information Processing Systems*, pages 3394–3404.