NEVAE: A Deep Generative Model for Molecular Graphs[*]

Bidisha Samanta[1]   BIDISHA@IITKGP.AC.IN
Abir De[2] [†],   ABIR@CSE.IITB.AC.IN
Gourhari Jana[3]   GOUR2015HARI@IITKGP.AC.IN
Vicenç Gomez[4]   VICEN.GOMEZ@UPF.EDU
Pratim Kumar Chattaraj[3]   PKC@CHEM.IITKGP.AC.IN
Niloy Ganguly[1]   NILOY@CSE.IITKGP.AC.IN
Manuel Gomez-Rodriguez[5]   MANUELGR@MPI-SWS.ORG

[1]*Department of Computer Science and Engineering, IIT Kharagpur, Kharagpur, India*

[2]*Department of Computer Science and Engineering, IIT Bombay, Mumbai, India*

[3]*Department of Chemistry, IIT Kharagpur, Kharagpur, India*

[4]*DTIC, Universitat Pompeu Fabra, Barcelona, Spain*

[5]*Max Planck Institute for Software Systems, Kaiserslautern, Germany*

Editor: Ryan Adams

Abstract

Deep generative models have been praised for their ability to learn smooth latent representations of images, text, and audio, which can then be used to generate new, plausible data. Motivated by these success stories, there has been a surge of interest in developing deep generative models for automated molecule design. However, these models face several difficulties due to the unique characteristics of molecular graphs—their underlying structure is not Euclidean or grid-like, they remain isomorphic under permutation of the nodes' labels, and they come with a different number of nodes and edges. In this paper, we first propose a novel variational autoencoder for molecular graphs, whose encoder and decoder are specially designed to account for the above properties by means of several technical innovations. Moreover, in contrast with the state of the art, our decoder is able to provide the spatial coordinates of the atoms of the molecules it generates. Then, we develop a gradient-based algorithm to optimize the decoder of our model so that it learns to generate molecules that maximize the value of certain property of interest and, given any arbitrary molecule, it is able to optimize the spatial configuration of its atoms for greater stability. Experiments reveal that our variational autoencoder can discover plausible, diverse and novel molecules more effectively than several state of the art models. Moreover, for several properties of interest, our optimized decoder is able to identify molecules with property values 121% higher than those identified by several state of the art methods based on Bayesian optimization and reinforcement learning.

Keywords: Drug design, Molecule discovery, Deep generative models, Variational autoencoders, Geometric deep learning.

---

[*]Preliminary version of this work appeared in Samanta et al. (2019).

[†]This work was done when Abir De was a post doctoral researcher at MPI-SWS, Germany.

1. Introduction

Drug design aims to identify (new) molecules with a set of specified properties, which in turn results in a therapeutic benefit to a group of patients. However, drug design is still a lengthy, expensive, difficult, and inefficient process with a low rate of new therapeutic discovery (Paul et al., 2010), in which candidate molecules are produced through chemical synthesis or biological processes. In the context of computer-aided drug design (Merz et al., 2010), there is a great interest in developing automated, machine learning techniques to discover sizeable numbers of plausible, diverse, and novel candidate molecules with various desirable properties in the vast ($10^{23} - 10^{60}$) and unstructured molecular space (Polishchuk et al., 2013).

In recent years, there has been a flurry of work devoted to developing deep generative models for automatic molecule design (Dai et al., 2018; De Cao and Kipf, 2018; Gómez-Bombarelli et al., 2016; Guimaraes et al., 2017; Jin et al., 2018; Kusner et al., 2017; Simonovsky and Komodakis, 2018; You et al., 2018a), which has predominantly followed two strategies. The first strategy (Dai et al., 2018; Gómez-Bombarelli et al., 2016; Guimaraes et al., 2017; Kusner et al., 2017) consists of representing molecules using a domain specific textual representation—SMILES strings—and then leveraging deep generative models for text generation for molecule design. Unfortunately, SMILES strings do not capture the structural similarity between molecules and, moreover, a molecule can have multiple SMILES representations. As a consequence, the generated molecules lack in terms of diversity and validity, as shown in Tables 1–2 and Figure 11. The second strategy (De Cao and Kipf, 2018; Jin et al., 2018; Simonovsky and Komodakis, 2018; You et al., 2018a) consists of representing molecules using molecular graphs, rather than SMILES representations, and then developing deep generative models for molecular graphs, in which atoms correspond to nodes and bonds correspond to edges. However, current generative models for molecular graphs share one or more of the following limitations, which preclude them from realizing all their potential:

I. They can only generate (and be trained on) molecules with a given maximum number of atoms (Liu et al., 2018) while, in practice, molecules having similar properties often come with a different number of atoms and bonds.

II. They are not invariant to permutations of their node labels (Dai et al., 2018; De Cao and Kipf, 2018; Gómez-Bombarelli et al., 2016; Guimaraes et al., 2017; Kusner et al., 2017), however, molecular graphs remain isomorphic under permutation of their node labels.

III. Their training procedure suffers from a quadratic complexity with respect to the number of nodes in the graph (Simonovsky and Komodakis, 2018), which makes it difficult to leverage a sizeable number of large molecules during training.

IV. They generate molecular graphs by combining a small set of molecular *graphlets* (Jin et al., 2018), which constrain the diversity of the generated molecules, as shown in Table 1 and Figure 11.

V. They do not provide the spatial coordinates of the atoms they generate (Dai et al., 2018; De Cao and Kipf, 2018; Gómez-Bombarelli et al., 2016; Guimaraes et al., 2017; Jin et al., 2018; Kusner et al., 2017), whereas in practice, a molecule is a three-dimensional object in which the coordinates of its atoms significantly influence its chemical properties, as shown in Figure 6.

VI. To identify molecules that maximize the value of certain property (*e.g.*, solubility in water), they resort to either traditional Bayesian optimization (Gómez-Bombarelli et al., 2016; Kusner et al., 2017; Jin et al., 2018) or reinforcement learning over the continuous latent representa-

tion of molecules they find (De Cao and Kipf, 2018; Guimaraes et al., 2017; You et al., 2018a). Unfortunately, such procedures are unable to discover a sizeable set of candidate molecules with high property values, as shown in Table 3 and Figure 11.

To address the first five shortcomings (I-V), we develop NEVAE, a deep generative model for molecular graphs based on variational autoencoders. Our model relies on several technical innovations, which distinguish us from previous work:

(i) Our probabilistic encoder learns to aggregate information (*e.g.*, bond features, atoms and their coordinates) from a different number of hops away from a given atom and then map this aggregate information into a continuous latent space, as in inductive graph representation learning (Hamilton et al., 2017; Lei et al., 2017). However, in contrast with inductive graph representation learning, the aggregator functions are learned via variational inference so that the resulting aggregator functions are especially well suited to enable the probabilistic decoder to generate new molecules rather than other machine learning tasks such as, *e.g.*, link prediction. Moreover, by using (symmetric) aggregator functions, it is invariant to permutations of the node labels (Kipf and Welling, 2016b) and can encode graphs with a variable number of atoms.

(ii) Our probabilistic decoder jointly represents all edges as an unnormalized log probability vector (or 'logit'), which then feeds a single multinomial edge distribution. Such scheme allows for an efficient inference algorithm with $O(l)$ complexity, where l is the number of true edges in the molecules, which is also invariant to permutations of the node labels. In contrast, previous work typically models the presence and absence of each potential edge using a Bernoulli distribution and this leads to inference algorithms with $O(n^2)$ complexity, where n is the number of nodes, which are not permutation invariant.

(iii) Our probabilistic decoder is able to guarantee a set of local structural and functional properties in the generated molecules by using a *mask* in the edge distribution definition, which can prevent the generation of certain *undesirable* edges during the decoding process. While masking have been increasingly used to account for prior (expert) knowledge in generative models (Gómez-Bombarelli et al., 2016; Kusner et al., 2017) based on SMILES, their use in generative models for molecular graphs has been lacking.

(iv) Our probabilistic decoder is able to provide the spatial coordinates of the atoms of the molecules it generates. To do so, it models the position of each atom using a Gaussian distribution whose mean and variance depend on its latent representation as well as that of each of its neighbors.

To address the last shortcoming (VI), we develop a gradient-based algorithm to optimize the decoder of our model for property oriented molecule generation, *i.e.*, to optimize the decoder so that it learns to generate molecules that maximize the value of certain property (*e.g.*, solubility in water). Note that, in contrast with recent reinforcement learning methods for property oriented molecule generation (De Cao and Kipf, 2018; Guimaraes et al., 2017; You et al., 2018a), our gradient-based algorithm benefits from the inductive bias provided by the original decoder, which in turns enable us to identify *better* molecules, as shown in Table 3. Moreover, given a molecule of interest, our gradient-based algorithm can also be used to optimize the spatial configuration of its atoms for greater stability. We believe our algorithm is of independent interest since it may be adapted to other deep generative models designed for other data types such as graphs, images, text or audio, that maximizes certain property value.

We experiment with molecules from two publicly available datasets, ZINC (Irwin et al., 2012) and QM9 (Ramakrishnan et al., 2014). First, we show that NEVAE beats the state of the art in terms of several relevant quality metrics, *i.e.*, validity, novelty and uniqueness, and the resulting latent space representation of molecules exhibits powerful semantics—we can smoothly interpolate between molecules—and generalization ability—we can generate (valid) molecules that are larger than any of the molecules in the datasets. Then, we demonstrate that, for several properties of interest (*e.g.*, solubility in water), our gradient-based algorithm is able to successfully optimize NEVAE's decoder for property oriented molecule generation. In particular, the optimized decoder is able to identify molecules with property values 121% higher than those identified by several state of the art methods based on Bayesian optimization and reinforcement learning and, given a molecule of interest, it is able to optimize the spatial configuration of its atoms for greater stability, *i.e.*, lower potential energy. To facilitate research in this area, we are releasing an open source implementation of our model in Tensorflow as well as synthetic and real-world data used in our experiments[1].

2. Background on Variational Autoencoders

Variational autoencoders (Kingma and Welling, 2013; Rezende et al., 2014) are characterized by a probabilistic generative model $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ of the observed variables $\boldsymbol{x} \in \mathbb{R}^N$ given the latent variables $\boldsymbol{z} \in \mathbb{R}^M$, a prior distribution over the latent variables $p(\boldsymbol{z})$ and an approximate probabilistic inference model $q_\phi(\boldsymbol{z}|\boldsymbol{x})$. In this characterization, p_θ and q_ϕ are arbitrary distributions parametrized by two (deep) neural networks θ and ϕ and one can think of the generative model as a probabilistic *decoder*, which *decodes* latent variables into observed variables, and the inference model as a probabilistic *encoder*, which *encodes* observed variables into latent variables.

Ideally, if we use the maximum likelihood principle to train a variational autoencoder, we should optimize the marginal log-likelihood of the observed data, *i.e.*, $\mathbb{E}_\mathcal{D}\left[\log p_\theta(\boldsymbol{x})\right]$, where $p_\mathcal{D}$ is the data distribution. Unfortunately, computing $\log p_\theta(\boldsymbol{x})$ requires marginalization with respect to the latent variable $\boldsymbol{z}$, which is typically intractable. Therefore, one resorts to maximizing a variational lower bound or evidence lower bound (ELBO) of the log-likelihood of the observed data, *i.e.*,

$$\max_\theta \max_\phi \mathbb{E}_\mathcal{D}\left[-\mathrm{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z})) + \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\right].$$

Finally, note that the quality of this variational lower bound depends on the expressive ability of the approximate inference model $q_\phi(\boldsymbol{z}|\boldsymbol{x})$, which is typically assumed to be a normal distribution whose mean and variance are parametrized by a neural network ϕ with the observed data $\boldsymbol{x}$ as input.

3. NEVAE: A Variational Autoencoder for Molecular Graphs

In this section, we first give a high-level overview of the design of NEVAE, our variational autoencoder for molecular graphs, starting from the data it is designed for. Then, we describe more in-depth the key technical aspects of its individual components. Finally, we elaborate on the training procedure, scalability and implementation details.

High-level overview. We observe a collection of N molecular graphs $\mathcal{D} = \{\mathcal{G}_i = (\mathcal{V}_i, \mathcal{E}_i)\}_{i \in [N]}$, where $\mathcal{V}_i$ and $\mathcal{E}_i$ denote the corresponding set of nodes (atoms) and edges (bonds), respectively, and this collection may contain graphs with a different number of nodes and edges. For clarity, in the

---
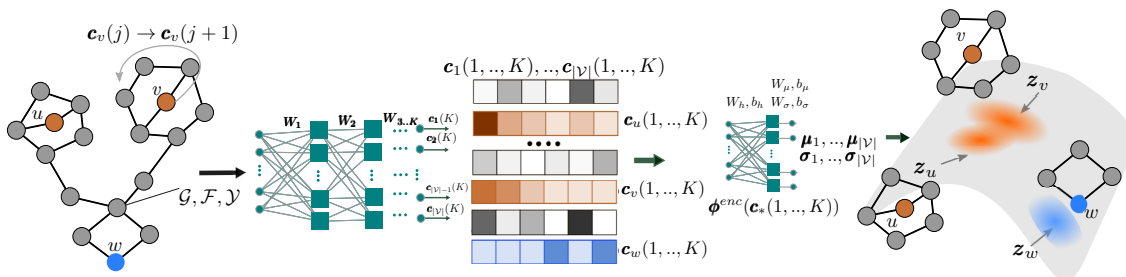
1. https://github.com/Networks-Learning/nevae

Figure 1: The encoder of our variational autoencoder for molecular graphs. From left to right, given a molecular graph $\mathcal{G}$ with a set of node features $\mathcal{F}$ and edge weights $\mathcal{Y}$, the encoder aggregates information from a different number of hops $j \leq K$ away for each node $v \in \mathcal{V}$ into an embedding vector $\mathbf{c}_v(j)$. These embeddings are fed into a differentiable function ϕ^{enc} which parameterizes the posterior distribution q_ϕ over the latent representation of a node, from which we sampled in the generation process.

next two subsections, we will omit graph subindex. Moreover, for each molecular graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, we also observe a set of node features $\mathcal{F} = (\mathcal{T}, \mathcal{X})$, where $\mathcal{T} = \{\mathbf{t}_u\}_{u \in \mathcal{V}}$, $\mathcal{X} = \{\mathbf{x}_u\}_{u \in \mathcal{V}}$, and edge weights $\mathcal{Y} = \{y_{uv}\}_{(u,v) \in \mathcal{E}}$. More specifically, $\mathbf{t}_u$ are one-hot representations of the type of the atoms (*i.e.*, C, H, N or O), $\mathbf{x}_u$ are the coordinates of the atoms in three dimensional space, and y_{uv} are the bond types (*i.e.*, single, double, triple)[2]. Our goal is then to design a variational autoencoder for molecular graphs that, once trained on $\mathcal{D}$, has the ability of creating new plausible molecular graphs, including node features and edge weights. In doing so, it will also provide a latent representation of any graph in $\mathcal{D}$ (or elsewhere) with meaningful semantics.

Following the above background on variational autoencoders, we characterize NEVAE by means of:

— *Prior*: $p_z(\mathbf{z}_1, \ldots, \mathbf{z}_n)$, where $|\mathcal{V}| = |\mathcal{F}| = n \sim \mathrm{Poisson}(\lambda_n)$
— *Inference model (encoder)*: $q_\phi(\mathbf{z}_1, \ldots, \mathbf{z}_n | \mathcal{V}, \mathcal{E}, \mathcal{F}, \mathcal{Y})$
— *Generative model (decoder)*: $p_\theta(\mathcal{E}, \mathcal{F}, \mathcal{Y} | \mathbf{z}_1, \ldots, \mathbf{z}_n)$

In the above characterization, note that we define one latent variable per node, *i.e.*, we have a *node-based* latent representation, and the number of nodes n is a random variable. As a consequence, both the latent representation as well as the graph can vary in size. Next, we formally define the functional form of the inference model, the generative model, and the prior.

Inference model (probabilistic encoder). Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with node features $\mathcal{F}$ and edge weights $\mathcal{Y}$, our inference model q_ϕ defines a probabilistic encoding for each node in the graph by aggregating information from different distances. More formally, for each node $u \in \mathcal{V}$, the inference model is defined as follows:

$$q_\phi(\mathbf{z}_u | \mathcal{V}, \mathcal{E}, \mathcal{F}, \mathcal{Y}) \sim \mathcal{N}(\boldsymbol{\mu}_u, \mathrm{diag}(\boldsymbol{\sigma}_u)) \tag{1}$$

---

2. In our work, we do not explicitly incorporate atomic charge as a node feature neither aromatic bond as an edge feature. It would be interesting to incorporate them in future work.

where $\boldsymbol{z}_u$ is the latent variable associated to node u, $[\boldsymbol{\mu}_u, \mathrm{diag}(\boldsymbol{\sigma}_u)] = \phi^{enc}(\mathbf{c}_u(1), \ldots, \mathbf{c}_u(K))$, and $\mathbf{c}_u(j)$ aggregates information from j hops away from u, *i.e.*,

$$\mathbf{c}_u(j) = \begin{cases} \boldsymbol{r}(\boldsymbol{W}_j^{\mathcal{T}}\boldsymbol{t}_u + \boldsymbol{W}_j^{\mathcal{X}}\boldsymbol{x}_u) & \text{if } j = 1 \\ \boldsymbol{r}\left((\boldsymbol{W}_j^{\mathcal{T}}\boldsymbol{t}_u + \boldsymbol{W}_j^{\mathcal{X}}\boldsymbol{x}_u) \odot \boldsymbol{\Lambda}\left(\cup_{v \in \mathcal{N}(u)} y_{uv}\, \boldsymbol{g}(\mathbf{c}_v(j-1))\right)\right) & \text{if } j > 1. \end{cases} \tag{2}$$

In the above, $\boldsymbol{W}_j^{\mathcal{T}}$ and $\boldsymbol{W}_j^{\mathcal{X}}$ are trainable weight matrices, which propagate information between different search depths, $\boldsymbol{\Lambda}(.)$ is a (possibly nonlinear) symmetric aggregator function in its arguments, $\boldsymbol{g}(\cdot)$ and $\boldsymbol{r}(\cdot)$ are (possibly nonlinear) differentiable functions, ϕ^{enc} is a neural network, and $\odot$ denotes pairwise product. Figure 1 describes our encoder architecture.

The above node embeddings, defined by Eq. 2, are very similar to the ones used in several graph representation learning algorithms such as GraphSAGE (Hamilton et al., 2017), column networks (Pham et al., 2017), and GCNs (Kipf and Welling, 2016a). The main difference with our work is the way we train the weight matrices $\boldsymbol{W}_k^{\bullet}$. Here, we use variational inference so that the resulting embeddings are especially well suited to enable our probabilistic decoder to generate new, plausible molecular graphs. In contrast, the above algorithms use non variational approaches to compute general purpose embeddings to feed downstream classification tasks.

The following proposition highlights several desirable theoretical properties of our probabilistic encoder, which distinguishes our design from most existing generative models of graphs (Jin et al., 2018; Simonovsky and Komodakis, 2018):

Proposition 1 *The probabilistic encoder defined by Eqs. 1 and 2 has the following properties:*
  (i) For each node u, its corresponding embedding j hops away from u, $\mathbf{c}_u(j)$, is invariant to permutations of the node labels of its neighbors.
  (ii) The weight matrices $\boldsymbol{W}_1^{\mathcal{T}}, \ldots, \boldsymbol{W}_K^{\mathcal{T}}$ and $\boldsymbol{W}_1^{\mathcal{X}}, \ldots, \boldsymbol{W}_K^{\mathcal{X}}$ do not depend on the number of nodes and edges in the graph and thus a single encoder allows for graphs with a variable number of nodes and edges.

Proof First, we prove property (i). Consider π a permutation of the node labels, *i.e.* for each u, we have $\pi(u) \in \mathcal{V}$; and the set of all shuffled labels $\{\pi(w) | w \in \mathcal{V}\} = \mathcal{V}$. Let us denote $\widetilde{u} := \pi(u)$. Now we need to prove

$$\mathbf{c}_u(k) = \mathbf{c}_{\widetilde{u}}(k), \quad \forall k \geq 1, \, \forall u \in \mathcal{V}. \tag{3}$$

We proof this by induction. Since the features $\boldsymbol{t}_u$ and $\boldsymbol{x}_u$ are independent of the node label of u, we have that $\boldsymbol{t}_u = \boldsymbol{t}_{\widetilde{u}}$ and $\boldsymbol{x}_u = \boldsymbol{x}_{\widetilde{u}}$, which proves Eq. 3 for $k = 1$, $\forall u \in \mathcal{V}$. Now assume that Eq. 3 is true for $k \leq k' - 1$, with $k' > 1$. That is, we have,

$$\mathbf{c}_{\widetilde{v}}(k' - 1) = \mathbf{c}_v(k' - 1), \quad \forall v \in \mathcal{V}. \tag{4}$$

Also, since the edge-weight y_{uv} between nodes does not depend on their labels, we have $y_{uv} = y_{\widetilde{u}\widetilde{v}}$. This, along with Eq. 4 gives $\{\cup_{v \in \mathcal{N}(u)} y_{uv}\, \boldsymbol{g}(\mathbf{c}_v(k' - 1))\} = \{\cup_{\widetilde{v} \in \mathcal{N}(\widetilde{u})} y_{\widetilde{u}\widetilde{v}}\, \boldsymbol{g}(\mathbf{c}_{\widetilde{v}}(k' - 1))\}$ which, due to the symmetric property of $\boldsymbol{\Lambda}(\cdot)$, implies

$$\boldsymbol{\Lambda}\left(\cup_{v \in \mathcal{N}(u)} y_{uv}\, \boldsymbol{g}\left(\mathbf{c}_v(k' - 1)\right)\right) = \boldsymbol{\Lambda}\left(\cup_{\widetilde{v} \in \mathcal{N}(\widetilde{u})} y_{\widetilde{u}\widetilde{v}}\, \boldsymbol{g}\left(\mathbf{c}_{\widetilde{v}}(k' - 1)\right)\right). \tag{5}$$

The above equation, together with the fact that $\boldsymbol{t}_u = \boldsymbol{t}_{\widetilde{u}}$ and $\boldsymbol{x}_u = \boldsymbol{x}_{\widetilde{u}}$, proves Eq. 3 for $k = k'$.

Second, we prove property (ii). Assume the embedding $\mathbf{c}_v(k-1) \in \mathbb{R}^{D \times 1}$ for all $k > 1$ and $v \in \mathcal{V}$. Moreover, note that, in Eq. 2, all the functions $r(\cdot)$, $g(\cdot)$ and $\mathbf{\Lambda}(\cdot)$ are defined term-wise. To ensure that $\boldsymbol{t}_u \odot \mathbf{\Lambda} \left(\cup_{v \in \mathcal{N}(u)} y_{uv}\, \boldsymbol{g}\left(\mathbf{c}_v(k-1)\right) \right)$ and $\boldsymbol{x}_u \odot \mathbf{\Lambda} \left(\cup_{v \in \mathcal{N}(u)} y_{uv}\, \boldsymbol{g}\left(\mathbf{c}_v(k-1)\right) \right)$ are well defined, we need to have $\mathbf{c}_v(k-1) \in \mathbb{R}^{D \times 1}$ for all $k > 1$ and $v \in \mathcal{V}$. Then, by matching the dimension of vectors in both sides of Eq. 2, we have that $\boldsymbol{W}_k^{\mathcal{T}} \in \mathbb{R}^{D \times D}$ and $\boldsymbol{W}_k^{\mathcal{X}} \in \mathbb{R}^{D \times D}$. ∎

Generative model (probabilistic decoder). We now describe the generative process to synthesize molecules. Let us assume the latent variables $\mathcal{Z}$ and the number of nodes $n = |\mathcal{V}|$ are given. Then, our generative model p_θ is defined as follows:

$$p_\theta(\mathcal{E}, \mathcal{Y}, \mathcal{T}, \mathcal{X} | \mathcal{Z}) = p_\theta(\mathcal{X} | \mathcal{E}, \mathcal{Y}, \mathcal{T}, \mathcal{Z})\, p_\theta(\mathcal{E}, \mathcal{Y} | \mathcal{Z}, \mathcal{T})\, p_\theta(\mathcal{T} | \mathcal{Z}) \tag{6}$$

with

$$p_\theta(\mathcal{T} | \mathcal{Z}) = \prod_{u \in \mathcal{V}} p_\theta(\boldsymbol{t}_u | \mathcal{Z}),$$

$$p_\theta(\mathcal{E}, \mathcal{Y} | \mathcal{Z}, \mathcal{T}) = \left[\prod_{k \in [l]} p_\theta(e_k = (u_k, v_k) | \mathcal{E}_{k-1}, \mathcal{Y}_{k-1}, \mathcal{T}, \mathcal{Z})\, p_\theta(y_{u_k v_k} | \mathcal{Y}_{k-1}, \mathcal{T}, \mathcal{Z}) \right] \times p_\theta(l | \mathcal{Z}),$$

$$p_\theta(\mathcal{X} | \mathcal{E}, \mathcal{Y}, \mathcal{Z}, \mathcal{T}) = \prod_{u \in \mathcal{V}} p_\theta(\boldsymbol{x}_u | \mathcal{E}, \mathcal{Y}, \mathcal{Z}, \mathcal{T}),$$

where l is the number of edges, the ordering for the edge and edge weights is independent of node labels and hence permutation invariant, e_k and $y_{u_k v_k}$ denote the k-th edge and edge weight under the chosen order, and $\mathcal{E}_{k-1} = \{e_1, \ldots, e_{k-1}\}$ and $\mathcal{Y}_{k-1} = \{y_{u_1 v_1}, \ldots, y_{u_{k-1} v_{k-1}}\}$ denote the $k-1$ previously generated edges and edge weights respectively.

Moreover, the model characterizes the conditional probabilities in the above formulation as follows. For each node, it represents all potential values for the atom types $\boldsymbol{t}_u = \boldsymbol{q}$ as an unnormalized log probability vector (or 'logits'), feeds this logit into a softmax distribution and samples the node features. Then, it represents the average number of edges as a logit, feeds this logit into a Poisson distribution and samples the number of edges. Next, it represents all potential edges as logits and, for each edge, all potential edge weights as another logit, and it feeds the former vector into a single softmax distribution and the latter vectors each into a different softmax distribution[3]. Moreover, the edge distribution and the corresponding edge weight distributions depend on a set of binary *masks*, which may depend on the sampled node features and also get updated every time a new edge and edge weight are sampled. By doing so, it prevents the generation of certain *undesirable* edges and edges weights, allowing for the generated graph to fulfill a set of predefined local structural and functional properties. Finally, for each atom, it samples its coordinates $\boldsymbol{x}_u$ from a multidimensional Gaussian distribution whose mean and variance depends on the latent vectors of the corresponding atom as well as its neighbors and the underlying chemical bonds.

---

3. Note that the values of the logits are computed only once during the generation process and they do not need to be updated after sampling each edge e_k.

More formally, the distributions of each node feature, the number of edges, each edge and edge weight are given by:

$$p_\theta(\boldsymbol{t}_u = \boldsymbol{q}|\mathcal{Z}) = \frac{e^{\theta_\gamma^{dec}(\boldsymbol{z}_u, \boldsymbol{q})}}{\sum_{\boldsymbol{q}'} e^{\theta_\gamma^{dec}(\boldsymbol{z}_u, \boldsymbol{q}')}}, \tag{7}$$

$$p_\theta(l|\mathcal{Z}) = p_l(e^{\theta_\delta^{dec}(\mathcal{Z})}),$$

$$p_\theta(e = (u,v)|\mathcal{E}_{k-1}, \mathcal{Z}) = \frac{\beta(u,v)e^{\theta_\alpha^{dec}(\boldsymbol{z}_u, \boldsymbol{z}_v)}}{\sum_{(u',v') \notin \mathcal{E}_{k-1}} \beta(u',v')e^{\theta_\alpha^{dec}(\boldsymbol{z}_{u'}, \boldsymbol{z}_{v'})}}, \tag{8}$$

$$p_\theta(y_{uv} = m|\mathcal{Y}_{k-1}, \mathcal{Z}) = \frac{\chi_{uv}(m)e^{\theta_\xi^{dec}(\boldsymbol{z}_u, \boldsymbol{z}_v, m)}}{\sum_{m' \neq m} \chi_{uv}(m')e^{\theta_\xi^{dec}(\boldsymbol{z}_u, \boldsymbol{z}_v, m')}},$$

$$p_\theta(\boldsymbol{x}_u|\mathcal{E}, \mathcal{Y}, \mathcal{Z}) = \mathcal{N}(\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x), \quad \text{with,} \quad [\boldsymbol{\mu}_u^x, \boldsymbol{\Sigma}_u^x] = [\theta_{\mu^x}(\boldsymbol{r}(u)), \theta_{\Sigma^x}(\boldsymbol{r}(u))\theta_{\Sigma^x}^T(\boldsymbol{r}(u))],$$

where p_l denotes a Poisson distribution, $\beta(u,v)$ is the binary mask for edge (u,v) and $\chi_{uv}(m)$ is the binary mask for feature edge value m, $\theta_\bullet^{dec}$ are neural networks, $\theta_{\Sigma^x} \in \mathbb{R}^{3\times3}$, and $\boldsymbol{r}(u) = \boldsymbol{z}_u + \sum_{v \in \mathcal{N}(u)} y_{uv}\boldsymbol{z}_v$. Here, note that the parameters of the neural networks do not depend on the number of nodes or edges in the molecular graph and the dependency of the binary masks $\beta(u,v)$ and $\chi_{uv}(m)$ on the node features and the previously generated edges $\mathcal{E}_{k-1}$ and edge weights $\mathcal{Y}_{k-1}$ are deterministic. More specifically, we set $\beta(u,v) = 1$ if the current degree of both u and v is less than their valency, and $\beta(u,v) = 0$, otherwise, and we set $\chi_{uv}(m) = 1$ if adding an m-ary chemical bond would not increase the degree of both u and v over their valency, and $\chi_{uv}(m) = 0$, otherwise. Figure 2 summarizes our decoder architecture.

Note that, by using a softmax distribution, it is only necessary to account for the presence of an edge, not its absence, and this, in combination with negative sampling, will allow for efficient training and decoding, as it will become clear later in this section. This is in contrast with previous generative models for graphs (Kipf and Welling, 2016b; Simonovsky and Komodakis, 2018), which need to model both the presence and absence of each potential edge. Moreover, we would like to acknowledge that, while masking may be useful to account for prior (expert) knowledge, it may be costly to check for some local (or global) structural and functional properties on-the-fly.

Prior. Given a set of n nodes with latent variables $\mathcal{Z} = \{\boldsymbol{z}_u\}_{u \in [n]}$, $p_z(\mathcal{Z}) \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$.

Training. We now describe the training procedure for NEVAE. Given a collection of N molecular graphs $\{\mathcal{G}_i = (\mathcal{V}_i, \mathcal{E}_i)\}_{i \in [N]}$, each with n_i nodes, a set of node features $\mathcal{F}_i$, set of node coordinates $\mathcal{X}_i$, set of edge weights $\mathcal{Y}_i$, we train our variational autoencoder for graphs by maximizing the ELBO, as described in the previous section, plus the log-likelihood of the Poisson distribution p_{λ_n} modeling the number of nodes in each graph. Hence we aim to solve:

$$\max_{\phi,\theta,\lambda_n} \frac{1}{N} \sum_{i \in [N]} \left(\mathbb{E}_{q_\phi(\mathcal{Z}_i|\mathcal{V}_i, \mathcal{E}_i, \mathcal{F}_i, \mathcal{Y}_i)} \log p_\theta(\mathcal{E}_i, \mathcal{Y}_i, \mathcal{F}_i|\mathcal{Z}_i) - \text{KL}(q_\phi||p_z) + \log p_{\lambda_n}(n_i) \right) \tag{9}$$

Note that, in the above objective, computation of $\mathbb{E}_{q_\phi} \log p_\theta(\mathcal{E}_i, \mathcal{Y}_i, \mathcal{F}_i|\mathcal{Z}_i)$ requires to specify an order of edges present in the graph $\mathcal{G}_i$. To determine this order, we use breadth-first-traversals (BFS) with randomized tie breaking during the child-selection step. Such a tie breaking method makes the
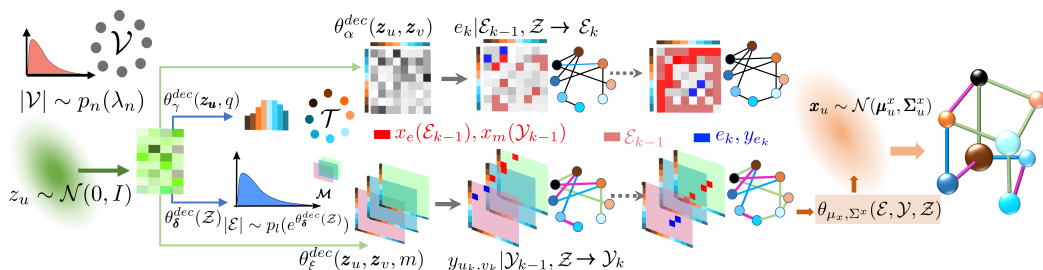
Figure 2: The decoder of our variational autoencoder for molecular graphs. From left to right, the decoder first samples the number of nodes $n = |\mathcal{V}|$ from a Poisson distribution $p_n(\lambda_n)$ and a latent vector $\boldsymbol{z}_u$ per node $u \in \mathcal{V}$ from $\mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. Then, for each node u, it represents all potential node feature values as an unnormalized log probability vector (or 'logits'), where each entry is given by a nonlinearity θ_γ^{dec} of the corresponding latent representation $\boldsymbol{z}_u$, feeds this logit into a softmax distribution and samples the node features. Next, it feeds all latent vectors $\mathcal{Z}$ into a nonlinear log intensity function $\theta_{\boldsymbol{\delta}}^{dec}(\mathcal{Z})$ which is used to sample the number of edges. Thereafter, on the top row, it constructs a logit for all potential edges (u, v), where each entry is given by a nonlinearity $\theta_{\boldsymbol{\alpha}}^{dec}$ of the corresponding latent representations $(\boldsymbol{z}_u, \boldsymbol{z}_v)$. Then, it samples the edges one by one from a soft max distribution depending on the logit and a mask $\beta((u, v)|\mathcal{E}_{k-1})$, which gets updated every time it samples a new edge e_k. On the bottom row, it constructs a logit per edge (u, v) for all potential edge weight values m, where each entry is given by a nonlinearity $\theta_{\boldsymbol{\xi}}^{dec}$ of the latent representations of the edge and edge weight value $(\boldsymbol{z}_u, \boldsymbol{z}_v, m)$. Then, every time it samples an edge, it samples the edge weight value from a soft max distribution depending on the corresponding logit and mask $\chi_{uv}(m)$, which gets updated every time it samples a new $y_{u_k v_k}$. Finally, for each node u, it samples its 3D coordinates $\boldsymbol{x}_u$ from a multidimensional Gaussian distribution whose mean $\mu_{\boldsymbol{x}}$ and variance $\Sigma_{\boldsymbol{x}}$ depend on the latent vectors of the corresponding atom and its neighbors and the underlying chemical bonds.

edge order independent of all node labels except for the source node label. Therefore, to make it completely permutation invariant, for each graph, we sample the source nodes from an arbitrary distribution. More formally, we replace $\log p_\theta(\mathcal{E}_i, \mathcal{Y}_i, \mathcal{F}_i|\mathcal{Z}_i)$ with $\log \mathbb{E}_{s \sim \zeta(\mathcal{V}_i)} p_\theta(\mathcal{E}_i, \mathcal{Y}_i, \mathcal{F}_i|\mathcal{Z}_i)$ for each graph $\mathcal{G}_i$, where s is the randomly sampled source node for the BFS, and ζ is the sampling distribution for s. Note that, after this replacement, we need to take a logarithm of a marginal likelihood, which is hard to compute. Fortunately, by using again Jensen's inequality, we can have a lower bound of the actual likelihood:

$$\log \mathbb{E}_{s \sim \zeta(\mathcal{V}_i)} p_\theta(\mathcal{E}_i, \mathcal{Y}_i, \mathcal{F}_i|\mathcal{Z}_i) \geq \mathbb{E}_{s \sim \zeta(\mathcal{V}_i)} \log p_\theta(\mathcal{E}_i, \mathcal{Y}_i, \mathcal{F}_i|\mathcal{Z}_i)$$

Therefore, to train our model, we maximize

$$\frac{1}{N} \sum_{i \in [N]} \left(\mathbb{E}_{q_\phi(\mathcal{Z}_i|\mathcal{V}_i, \mathcal{E}_i, \mathcal{F}_i, \mathcal{Y}_i), s \sim \zeta(\mathcal{V}_i)} \log p_\theta(\mathcal{E}_i, \mathcal{Y}_i, \mathcal{F}_i|\mathcal{Z}_i) - \mathrm{KL}(q_\phi||p_z) + \log p_{\lambda_n}(n_i) \right). \quad (10)$$

The following theorem points out the key property of our objective function.

Theorem 2 *If the source distribution ζ does not depend on the node labels, then the parameters learned by maximizing the objective in Eq. 10 are invariant to the permutations of the node labels.*

9

Proof For each training graph $\mathcal{G}_i = (\mathcal{V}_i, \mathcal{E}_i)$, we denote each corresponding component in the objective function as

$$\mathcal{L}_{\mathcal{G}_i}(\Theta) = \mathbb{E}_{q_\phi(\mathcal{Z}_i|\mathcal{V}_i,\mathcal{E}_i,\mathcal{F}_i,\mathcal{Y}_i),s\sim\zeta(\mathcal{V}_i)} \log p_\theta(\mathcal{E}_i,\mathcal{Y}_i,\mathcal{F}_i|\mathcal{Z}_i) - \text{KL}(q_\phi||p_z) + \log p_{\lambda_n}(n_i),$$

where Θ is the set of trainable parameters.

Then, to prove that the estimated parameters $\hat{\Theta}$ by maximizing $\mathcal{L}(\Theta)$ are invariant to the permutations of the labels of all $\mathcal{V}_i$'s, it is enough to prove that $\mathcal{L}_{\mathcal{G}_i}(\Theta)$ is invariant to the permutation of $\mathcal{V}_i$ for any $i \in [N]$, and for any Θ. Moreover, note that $\log p_{\lambda_n}(n_i)$ depends on the total number of nodes and edges, and therefore it is node permutation invariant. Hence, it is enough to prove the permutation invariance property of the first two components, *i.e.*, $\text{KL}(q_\phi||p_z)$ and $\mathbb{E}_{q_\phi(\mathcal{Z}_i|\mathcal{V}_i,\mathcal{E}_i,\mathcal{F}_i,\mathcal{Y}_i),s\sim\zeta(\mathcal{V}_i)} \log p_\theta(\mathcal{E}_i,\mathcal{Y}_i,\mathcal{F}_i|\mathcal{Z}_i)$. Since q_ϕ and p_z are both normal distributions, we have:

$$\text{KL}(q_\phi||p_z) = \frac{1}{2}\Big(\text{tr}(\Sigma_p^{-1}\Sigma_q) + (\mu_p - \mu_q)^T \Sigma_p^{-1}(\mu_p - \mu_q) - kn_i + \log\det\Sigma_p - \log\det\Sigma_q \Big) \tag{11}$$

which, in our case, reduces to:

$$\frac{1}{2}\Big(\sum_{u\in\mathcal{V}_i}(\mathbf{1}^T\sigma_u^2 + \mathbf{1}^T\mu_u^2) - kn_i - \mathbf{1}^T\log\sigma_u^2 \Big). \tag{12}$$

Note that, from Proposition 1, we know that the values of $\mathbf{c}_u(k)$ are invariant to the permutation of node labels. Now, since $[\mu_u, \text{diag}(\sigma_u)] = \phi^{\text{enc}}(\mathbf{c}_u(1), \ldots, \mathbf{c}_u(K))$, $\text{KL}(q_\phi||p_z)$ is also invariant to the permutation of node labels. Now, to prove that $\mathbb{E}_{q_\phi(\mathcal{Z}_i|\mathcal{V}_i,\mathcal{E}_i,\mathcal{F}_i,\mathcal{Y}_i)} \log p_\theta(\mathcal{E}_i,\mathcal{Y}_i,\mathcal{F}_i|\mathcal{Z}_i)$, we rely on a reparameterization trick for the normal distribution.

$$\mathbb{E}_{\substack{q_\phi(\mathcal{Z}_i|\mathcal{V}_i,\mathcal{E}_i,\mathcal{F}_i,\mathcal{Y}_i),\\s\sim\zeta(\mathcal{V}_i)}}\Big[\log p_\theta(\mathcal{E}_i,\mathcal{Y}_i,\mathcal{F}_i|\mathcal{Z}_i) \Big] = \mathbb{E}_{\substack{\epsilon_{u\in\mathcal{V}_i}\sim\mathcal{N}(\mathbf{0},\mathbf{I}),\\s\sim\zeta(\mathcal{V}_i)}}\Big[\log p_\theta\Big(\mathcal{E}_i,\mathcal{Y}_i,\mathcal{F}_i\big|(\mu_u + \text{diag}(\sigma_u)\epsilon_u)_{u\in\mathcal{V}_i}\Big) \Big].$$

Note that,

1. $\zeta(\mathcal{V}_i)$ does not depend on node labels (*e.g.*, uniform sampling, degree based sampling, etc.);

2. the edge sequence of $\mathcal{E}_i$ is determined by BFS with randomized tie breaking;

3. the variable ϵ_u does not depend on u since it is sampled from $\mathcal{N}(\mathbf{0}, \mathbf{I})$.

The above facts along with the permutation invariance property of μ_u and $\text{diag}(\sigma_u)$, conclude the proof. ∎

Scalability and implementation details. In terms of scalability, the major bottleneck is computing the gradient of the first term in Eq. 10 during training, rather than encoding and decoding graphs once the model is trained. More specifically, given a source node for a network without masks, an exact computation of the per edge partition function of the log-likelihood of the edges, *i.e.*, $\sum_{e'=(u',v')\notin\mathcal{E}_{k-1}} \exp(\theta_\alpha^{dec}(\mathbf{z}_{u'}, \mathbf{z}_{v'}))$, requires $O(|\mathcal{V}|^2)$ computations, similarly as in most inference algorithms for existing generative models of graphs, and hence is costly to compute even for medium networks. Fortunately, in practice, we can approximate such partition function using negative sampling (Mikolov et al., 2013) which reduces the likelihood computation to $O(l)$, where $l = |\mathcal{E}|$ is the number of (true) edges in the graph. Therefore, for S samples of source nodes, the complexity becomes $O(Sl)$. Here, note that most real-world graphs are sparse and thus $l \ll |\mathcal{V}|^2$.

4. Property Oriented Molecule Generation

In this section, we aim to optimize the probabilistic decoder of our variational autoencoder, described in Section 3, so that it learns to generate molecules that maximize certain molecular property (*e.g.*, solubility in water). To this aim, we approach the problem from the perspective of variational inference and show that the optimal property-oriented decoder can be expressed in terms of the original decoder and the value of the molecular property. This result means that we can obtain molecules from the optimal property-oriented decoder just by applying rejection sampling on the molecules samples from the original decoder. However, in practice, such a naive sampling strategy will be inefficient and impractical, specially given the high-dimensional nature of the data. Therefore, we design a practical method for approximating the optimal property-oriented decoder, which iteratively adapts the parameters of a (parameterized) property-oriented decoder using a stochastic gradient-based algorithm.

Property-oriented decoder design using variational inference. Let p_θ be our original generative model (decoder), which has been trained using a given collection of molecular graphs and $\ell(\cdot)$ be a loss function, which penalizes low values of the molecular property value of interest. Then, we construct the optimal property-oriented decoder p^* by solving the following optimization problem:

$$\underset{p(\cdot|\mathcal{Z})}{\text{minimize}} \quad \mathbb{E}_{\mathcal{Z}\sim p_z(\cdot)}\left[\mathbb{E}_{\mathcal{E},\mathcal{Y},\mathcal{F}\sim p(\cdot|\mathcal{Z})}\left[S(\mathcal{E},\mathcal{Y},\mathcal{F}|\mathcal{Z})\right]\right], \tag{13}$$

with

$$S(\mathcal{E},\mathcal{Y},\mathcal{F}|\mathcal{Z}) = \ell(\mathcal{E},\mathcal{Y},\mathcal{F}) + \rho\log\frac{p(\mathcal{E},\mathcal{Y},\mathcal{F}|\mathcal{Z})}{p_\theta(\mathcal{E},\mathcal{Y},\mathcal{F}|\mathcal{Z})}, \tag{14}$$

where the inner expectation is taken over all molecules generated using the property-oriented decoder $p(\cdot|\mathcal{Z})$ given the latent vectors $\mathcal{Z}$, the outer expectation is taken over all possible latent vectors under the prior distribution[4] $p_z(z_1,\ldots,z_n)$ with $n\sim\text{Poisson}(\lambda_n)$, and we do not assume any specific parametric form for the property-oriented decoder p. In Eq. 14, the first term penalizes molecules with a low value of the property of interest, the second term penalizes property-oriented decoders whose generated molecules differ more from those that the original decoder would generate and the parameter ρ controls the trade off between both terms. Here, note that the second term provides an inductive bias that ensures that the molecules generated by the property-oriented decoder are plausible. Moreover, we can rewrite the inner expectation of the second term in terms of Kullback-Leibler (KL) divergence (Kullback and Leibler, 1951), *i.e.*,

$$\text{KL}\left[p(\cdot|\mathcal{Z})||p_\theta(\cdot|\mathcal{Z})\right] = \mathbb{E}_{\mathcal{E},\mathcal{Y},\mathcal{F}\sim p(\cdot|\mathcal{Z})}\left[\log\frac{p(\mathcal{E},\mathcal{Y},\mathcal{F}|\mathcal{Z})}{p_\theta(\mathcal{E},\mathcal{Y},\mathcal{F}|\mathcal{Z})}\right]$$

which is commonly used as a *distance* measure between distributions.

Then, it is straightforward to show that the above optimization problem is equivalent to the following problem:

$$\underset{p(\cdot|\mathcal{Z})}{\text{minimize}} \quad \mathbb{E}_{\mathcal{Z}\sim p_z(\cdot)}\left[\text{KL}\left[p(\cdot|\mathcal{Z})||g_\theta(\cdot|\mathcal{Z})\right]\right] \tag{15}$$

where

$$g_\theta(\mathcal{E},\mathcal{Y},\mathcal{F}|\mathcal{Z}) = \frac{p_\theta(\mathcal{E},\mathcal{Y},\mathcal{F}|\mathcal{Z})\exp(-\frac{\ell(\mathcal{E},\mathcal{Y},\mathcal{F})}{\rho})}{\mathbb{E}_{\mathcal{E}',\mathcal{Y}',\mathcal{F}'\sim p_\theta(\cdot|\mathcal{Z})}\left[\exp(-\frac{\ell(\mathcal{E}',\mathcal{Y}',\mathcal{F}')}{\rho})\right]}.$$

---

4. If a molecule is given, instead of the prior distribution, one may also consider using the posterior $q_\phi(\cdot|\mathcal{V},\mathcal{E},\mathcal{F},\mathcal{Y})$.

The above objective function achieves its global minimum of zero if the numerator and the denominator are equal. Thus, the optimal property-oriented decoder is just given by:

$$p^*(\mathcal{E}, \mathcal{Y}, \mathcal{F}|\mathcal{Z}) = \frac{p_\theta(\mathcal{E}, \mathcal{Y}, \mathcal{F}|\mathcal{Z}) \exp(-\frac{\ell(\mathcal{E}, \mathcal{Y}, \mathcal{F})}{\rho})}{\mathbb{E}_{\mathcal{E}', \mathcal{Y}', \mathcal{F}' \sim p_\theta(\cdot|\mathcal{Z})}\left[\exp(-\frac{\ell(\mathcal{E}', \mathcal{Y}', \mathcal{F}')}{\rho})\right]} \tag{16}$$

The above result has an important implication. It means that we can use sampling methods to obtain (unbiased) samples from the optimal property-oriented decoder. For example, we can apply rejection sampling on the molecules generated by the original decoder p_θ, where we accept or reject them according to the (exponentiated) property value of interest. However, in practice, these sampling methods may be inefficient if the generated molecules under the original decoder p_θ have low probability under the optimal property-oriented decoder model. Given that molecules are high dimensional objects, this is specially problematic due to the curse of dimensionality. Next, we will design a practical method for approximating $p^*(\mathcal{E}, \mathcal{Y}, \mathcal{F}|\mathcal{Z})$, which iteratively adapts the parameters of a (parameterized) property-oriented model using a stochastic gradient-based algorithm.

A stochastic gradient-based algorithm. In this section, we aim to find a property-oriented decoder $p_{\theta'}$ within the class of parameterized probabilistic decoders defined by Eq. 6 that approximates well the optimal property-oriented decoder p^* that minimizes the objective function in Eq. 13, *i.e.*,

$$\mathbb{E}_{\mathcal{Z} \sim p_z(\cdot)}\left[\mathbb{E}_{\mathcal{E}, \mathcal{Y}, \mathcal{F} \sim p(\cdot|\mathcal{Z})}\left[S(\mathcal{E}, \mathcal{Y}, \mathcal{F}|\mathcal{Z})\right]\right]$$

To this aim, we introduce a general gradient-based algorithm, which iteratively update the parameters θ' of the parameterized property-oriented decoder $p_{\theta'}$ using stochastic gradient descent (SGD) (Kiefer et al., 1952), *i.e.*,

$$\theta'_{j+1} = \theta'_j + \alpha_j \nabla_{\theta'} \mathbb{E}_{\mathcal{Z} \sim p_z(\cdot)}\left[\mathbb{E}_{\mathcal{E}, \mathcal{Y}, \mathcal{F} \sim p_{\theta'}(\cdot|\mathcal{Z})}\left[S_{\theta'}(\mathcal{E}, \mathcal{Y}, \mathcal{F}|\mathcal{Z})\right]\right]|_{\theta'=\theta'_j}$$
$$= \theta'_j + \alpha_j \mathbb{E}_{\mathcal{Z} \sim p_z(\cdot)}\left[\nabla_{\theta'} \mathbb{E}_{\mathcal{E}, \mathcal{Y}, \mathcal{F} \sim p_{\theta'}(\cdot|\mathcal{Z})}\left[S_{\theta'}(\mathcal{E}, \mathcal{Y}, \mathcal{F}|\mathcal{Z})\right]\right]|_{\theta'=\theta'_j},$$

where

$$S_{\theta'}(\mathcal{E}, \mathcal{Y}, \mathcal{F}|\mathcal{Z}) = \ell(\mathcal{E}, \mathcal{Y}, \mathcal{F}) + \rho \log \frac{p_{\theta'}(\mathcal{E}, \mathcal{Y}, \mathcal{F}|\mathcal{Z})}{p_\theta(\mathcal{E}, \mathcal{Y}, \mathcal{F}|\mathcal{Z})},$$

$\alpha_j > 0$ is the learning rate at step j, and $\theta'_0 = \theta$.

In the above, it may seem challenging to compute a finite sample estimate of the gradient of the function $\mathbb{E}_{\mathcal{E}, \mathcal{Y}, \mathcal{F} \sim p_{\theta'}(\cdot|\mathcal{Z})}\left[S_{\theta'}(\mathcal{E}, \mathcal{Y}, \mathcal{F}|\mathcal{Z})\right]$ since the derivate is taken with respect to the parameters of the property-oriented decoder $p_{\theta'}$, which we are trying to learn. However, we can overcome this challenge using the log-derivative trick (Williams, 1992):

$$\nabla_{\theta'} \mathbb{E}_{\mathcal{E}, \mathcal{Y}, \mathcal{F} \sim p_{\theta'}(\cdot|\mathcal{Z})}\left[S_{\theta'}(\mathcal{E}, \mathcal{Y}, \mathcal{F}|\mathcal{Z})\right] = \mathbb{E}_{\mathcal{E}, \mathcal{Y}, \mathcal{F} \sim p_{\theta'}(\cdot|\mathcal{Z})}\left[(S_{\theta'}(\mathcal{E}, \mathcal{Y}, \mathcal{F}|\mathcal{Z}) + \rho)\nabla_{\theta'} \log p_{\theta'}(\mathcal{E}, \mathcal{Y}, \mathcal{F}|\mathcal{Z})\right].$$

The above expression readily yields the following unbiased finite sample Monte Carlo estimator:

$$\nabla_{\theta'} \mathbb{E}_{\mathcal{E}, \mathcal{Y}, \mathcal{F} \sim p_\theta(\cdot|\mathcal{Z})}\left[S_{\theta'}(\mathcal{E}, \mathcal{Y}, \mathcal{F}|\mathcal{Z})\right] \approx \frac{1}{M} \sum_{i=1}^{M} (S_\theta(\mathcal{E}_i, \mathcal{Y}_i, \mathcal{F}_i|\mathcal{Z}) + \rho)\nabla_{\theta'} \log p_{\theta'}(\mathcal{E}_i, \mathcal{Y}_i, \mathcal{F}_i|\mathcal{Z}),$$

$$\tag{17}$$

where M is total number of sampled molecules generated from the property-oriented decoder $p_{\theta'}$. Algorithm 1 summarizes the overall procedure.

Algorithm 1: PROPERTYORIENTEDDECODER: it trains a parameterized property-oriented decoder.

1: **Given:** The loss function $\ell(\cdot)$, parameter ρ, original decoder p_θ, # of iterations M, mini batch size B, and learning rate γ
2: $\theta'_0 \leftarrow \theta$
3: **for** $j = 1, \ldots, M$ **do**
4:     $\mathcal{Z}_j \sim p_z(\cdot)$
5:     $\mathcal{D} \leftarrow \text{MINIBATCH}(p_{\theta'_j}(\cdot|\mathcal{Z}_j), B)$
6:     $\nabla \leftarrow 0$
7:     **for** $(\mathcal{E}_i, \mathcal{Y}_i, \mathcal{F}_i) \in \mathcal{D}$ **do**
8:         $S \leftarrow \ell(\mathcal{E}_i, \mathcal{Y}_i, \mathcal{F}_i) + \rho \log \left(p_{\theta'_j}(\mathcal{E}_i, \mathcal{Y}_i, \mathcal{F}_i|\mathcal{Z}_j)/p_\theta(\mathcal{E}_i, \mathcal{Y}_i, \mathcal{F}_i|\mathcal{Z}_j) \right)$
9:         $\nabla \leftarrow \nabla + (S + \rho) \nabla_{\theta'} \log p_{\theta'_j}(\mathcal{E}_i, \mathcal{Y}_i, \mathcal{F}_i|\mathcal{Z}_j)$
10:     $\theta'_{j+1} \leftarrow \theta'_j + \gamma \frac{\nabla}{B}$
11: **Return** θ'_M

---

5. Experiments

In this section, we first show that NEVAE beats several state of the art machine learning models for molecule design (Dai et al., 2018; Gómez-Bombarelli et al., 2016; Kusner et al., 2017; Simonovsky and Komodakis, 2018; Jin et al., 2018; Liu et al., 2018) in terms of several relevant quality metrics, *i.e.*, *validity*, *novelty* and *uniqueness*. Then, we also show that the continuous latent representations of molecules that our model finds are smooth. Finally, we demonstrate that the property-oriented decoder provided by Algorithm 1 is able to generate molecules that maximize certain desirable properties more effectively than several baselines based on Bayesian optimization and reinforcement learning. Appendix D contains additional experiments on synthetic data.

5.1. Experimental setup

We sample $\sim 10{,}000$ drug-like commercially available molecules from the ZINC dataset (Irwin et al., 2012) with $\mathbb{E}[n] = 44$ atoms and $\sim 10{,}000$ molecules from the QM9 dataset (Ramakrishnan et al., 2014; Ruddigkeit et al., 2012) with $\mathbb{E}[n] = 21$ atoms. For each molecule, we construct a molecular graph, where nodes are the atoms, the node features are the type of the atoms *i.e.* $t_u \in \{C, H, N3, N5, O\}$, edges are the bonds between two atoms, and the weight associated with an edge is the type of bonds (single, double or triple)[5] Then, for each dataset, we train our variational autoencoder for molecular graphs using batches comprised of molecules with the same number of nodes[6]. Finally, we sample 10^6 molecular graphs from each of the (two) trained variational autoencoders using: (i) $\mathcal{G} \sim p_\theta(\mathcal{G}|\mathcal{Z})$, where $\mathcal{Z} \sim p(\mathcal{Z})$ and (ii) $\mathcal{Z} \sim p_\theta(\mathcal{Z}|G = G_T)$, where G_T is a molecular graph from the corresponding (training) dataset. In the above procedure, we only use masking on the weight (*i.e.*, type of bond) distributions both during training and sampling to ensure that the valence of the nodes at both ends are valid at all times, *i.e.*, $x_m(u, v) = \mathbb{I}(m + n_k(u) \leq$

---

5. We consider two different types of atoms for Nitrogen, denoted as $N3$ and $N5$, that indicate valency 3 and 5, respectively. We would like to acknowledge that, in our experiments, we only consider atoms with upto two valency values and thus did not include Sulphur (S).

6. We batch graphs with respect to the number of nodes for efficiency reasons since, every time that the number of nodes changes, we need to change the size of the computational graph in Tensorflow.

$m_{max}(u) \wedge m + n_k(v) \leq m_{max}(v))$, where $n_k(u)$ is the current valence of node u and $m_{max}(u)$ is the maximum valence of node u, which depends on its type f_u. Moreover, during sampling, if there is no valid weight value for a sampled edge, we reject it. To assess to which extent masking helps, we also train and sample from our model without masking. Here, we would like to highlight that, while using masking during test does not lead to a significant increase in the time it takes to generate a graph, using masking during training does lead to an increase of 5% in training time.

5.2. Quality of the generated molecules

In this section, we first perform a quantitative analysis of our model where we compare the quality of the molecules generated by our trained models and by several state of the art competing methods. Then, we perform a qualitative analysis where we first show that the property distribution of the molecules generated by our model closely resembles that of the real data and then demonstrate that the latent space of the molecules inferred by our model is smooth.

Quantitative analysis. For the quantitative analysis, we use eight baselines for comparison: (i) GraphVAE (Simonovsky and Komodakis, 2018), (ii) GrammarVAE (Kusner et al., 2017), (iii) CVAE (Gómez-Bombarelli et al., 2016), (iv) SDVAE (Dai et al., 2018), (v) JTVAE (Jin et al., 2018), (vi) CGVAE (Liu et al., 2018), (vii) MOLGAN (De Cao and Kipf, 2018), (viii) ORGAN (Guimaraes et al., 2017), and (ix) GCPN (You et al., 2018a). Among them, GraphVAE, JTVAE, CGVAE, MOLGAN and GCPN use molecular graphs and GrammarVAE, CVAE, SDVAE, JTVAE and ORGAN use SMILES strings, a domain specific textual representation of molecules. Moreover, we use the following evaluation metrics for performance comparison:

(i) *Novelty*: we use this metric to evaluate to which degree a method generates novel molecules, *i.e.*, molecules which were not present in the (training) dataset, *i.e.* Novelty $= 1 - |\mathcal{C}_s \cap \mathcal{D}|/|\mathcal{C}_s|$, where $\mathcal{C}_s$ is the set of generated molecules which are chemically valid, $\mathcal{D}$ is the training dataset, and Novelty $\in [0, 1]$.

(ii) *Uniqueness*: we use this metric to evaluate to what extent a method generates unique chemically valid molecules. We define, Uniqueness $= |\text{set}(\mathcal{C}_s)|/n_s$ where n_s is the number of generated molecules and Unique $\in [0, 1]$.

(iii) *Validity*: we use this metric to evaluate to which degree a method generates chemically valid molecules[7]. That is, Validity $= |\mathcal{C}_s|/n_s$ where n_s is the number of generated molecules, $\mathcal{C}_s$ is the set of generated molecules which are chemically valid, and note that Validity $\in [0, 1]$.

Tables 1–2 compare our trained models to three state of the art methods above in terms of novelty, uniqueness, and validity. For GraphVAE and CGVAE we report the results reported in the paper and, for SDVAE, since there is no public domain implementation of these methods at the time of writing, we have used the sampled molecules from the prior provided by the authors for the ZINC dataset. For CVAE, GrammarVAE, JTVAE, ORGAN, MOLGAN and GCPN, we run their public domain implementations in the same set of molecules that we used. For MOLGAN, ORGAN and GCPN, we only report the validity on the discovered molecules and refrain comparing their performance in terms of novelty and uniqueness given that their focus is on generating molecules maximizing certain property value.

We find that, in terms of novelty, both our trained models and all competing methods except for the GraphVAE, which assumes a fixed number of nodes, are able to (almost) always generate novel molecules. However, we would also like to note that novelty is only defined over chemically

---

7. We used the opensource cheminformatics suite RDkit (http://www.rdkit.org) to check the validity of a generated molecule.

	Novelty							
Dataset	**NEVAE**	**NEVAE***	**GraphVAE**	**GrammarVAE**	**CVAE**	**SDVAE**	**JTVAE**	**CGVAE**
ZINC	1.000	1.000	-	1.000	0.980	1.000	0.999	1.000
QM9	1.000	1.000	0.661	1.000	0.902	-	1.000	0.943
	Uniqueness							
Dataset	**NEVAE**	**NEVAE***	**GraphVAE**	**GrammarVAE**	**CVAE**	**SDVAE**	**JTVAE**	**CGVAE**
ZINC	0.999	0.588	-	0.273	0.021	1.000	0.991	0.998
QM9	0.998	0.676	0.305	0.197	0.031	-	0.371	0.986

Table 1: Novelty and Uniqueness of the molecules generated using NEVAE and all baselines. The sign * indicates no masking. For both the datasets, we report the number over 10^6 valid sampled molecules.

		Validity										
Dataset	**Sampling type**	**NEVAE**	**NEVAE***	**GraphVAE**	**GrammarVAE**	**CVAE**	**SDVAE**	**JTVAE**	**CGVAE**	**ORGAN**	**MOLGAN**	**GCPN**
ZINC	$Z \sim P(Z)$	1.000	0.590	0.135	0.440	0.021	0.432	1.000	1.000	0.240	1.000	1.000
	$Z \sim P(Z\|G_T)$	1.000	0.580	-	0.381	0.175	-	1.000	-	-	-	-
QM9	$Z \sim P(Z)$	0.999	0.682	0.458	0.200	0.031	-	0.997	1.000	-	-	-
	$Z \sim P(Z\|G_T)$	0.999	0.660	-	0.301	0.100	-	0.965	-	-	-	-

Table 2: Validity of the molecules generated using NEVAE and all baselines. The sign * indicates no masking. For both the datasets, we report the numbers over 10^6 sampled molecules.

valid molecules. Therefore, despite having (almost) perfect novelty scores, GraphVAE, Grammar-VAE, CVAE and SDVAE generate significantly fewer novel molecules than our method. In terms of uniqueness, which is defined over the set of sampled molecules, we observe that all baseline methods, except CGVAE (for ZINC and QM9) and JTVAE (for ZINC), perform very poorly in both datasets in comparison with our method. In terms of validity, our trained models significantly out-perform four competing methods—GraphVAE, GrammarVAE, CVAE, SDVAE and ORGAN—even without the use of masking, and achieve comparable performance to JTVAE, CGVAE and GCPN.

We would like to highlight that, in contrast to our model, GrammarVAE, CVAE and SDVAE use SMILES, a domain specific string based representation, and thus they may be constrained by its limited expressiveness. Among them, GrammarVAE and SDVAE achieve better performance by using grammar to favor valid molecules. GraphVAE generates molecular graphs, as our model, however, its performance is inferior to our method because it assumes a fixed number of nodes, it samples edges independently from a Bernoulli distribution, and is not permutation invariant.

Qualitative analysis. In this section, we first compute the empirical distributions of two well-known molecular properties for the molecules generated by NEVAE, JTVAE, CVAE and GrammarVAE. More specifically, we consider:
 (i) the octanol-water partition coefficient, penalized by synthetic accessibility (SA) score and num-ber of long cycles (penalized logP, $\tau_1(m)$); and,
 (ii) the quantitative estimation of drug-likeness (QED, $\tau_2(m)$).

For each molecular graph $\mathcal{G}_D$ in the real dataset, we first sample 100 molecules $\{m\}$ from our trained model, JTVAE, CVAE and GrammarVAE using $m \sim p(\cdot|\mathcal{Z})$, $\mathcal{Z} \sim q(\cdot|\mathcal{G} = \mathcal{G}_D)$. Then, we compute the mean of the properties, *i.e.*,

$$\bar{\tau}_{1,\mathcal{G}_D} = \mathbb{E}_{m \sim p_\theta(\cdot|\mathcal{Z}), \mathcal{Z} \sim q_\phi(\cdot|\mathcal{G}=\mathcal{G}_D)} \tau_1(m) \text{ and,} \tag{18}$$

$$\bar{\tau}_{2,\mathcal{G}_D} = \mathbb{E}_{m \sim p_\theta(\cdot|\mathcal{Z}), \mathcal{Z} \sim q_\phi(\cdot|\mathcal{G}=\mathcal{G}_D)} \tau_2(m). \tag{19}$$

Finally, we look into how closely the distributions of $\bar{\tau}_{1,\mathcal{G}_D}$ and $\bar{\tau}_{2,\mathcal{G}_D}$ for our model and the state of the art models resemble the distributions of $\tau_1(\mathcal{G}_D)$ and $\tau_2(\mathcal{G}_D)$—the properties of the molecules in real dataset. Figure 3 summarizes the results for the ZINC dataset, which show that the property

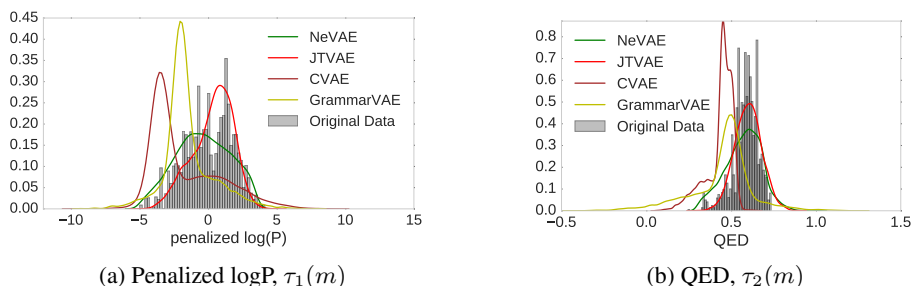(a) Penalized logP, $\tau_1(m)$ (b) QED, $\tau_2(m)$

Figure 3: Empirical distribution of penalized logP and QED scores for the molecules generated by NEVAE, JTVAE, CVAE and GrammarVAE.
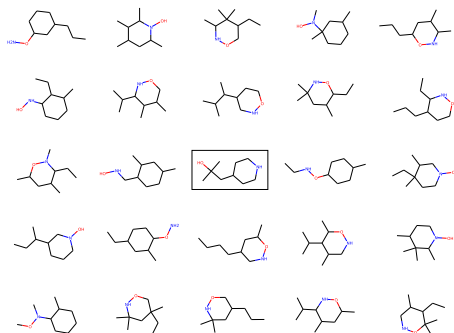

Figure 4: Molecules sampled using the probabilistic decoder $\mathcal{G}_i \sim p_\theta(\mathcal{G}|\mathcal{Z})$, where $\mathcal{Z}$ is the (sampled) latent representation of a given molecule $\mathcal{G}$ (boxed) from the ZINC dataset. The sampled molecules are topologically similar to each other as well the original. This provides qualitative evidence for the smooth latent space of molecules provided by NeVAE.

distributions of the molecules generated by both NEVAE and JTVAE closely resemble those of the molecules in the dataset.

Next, we qualitatively demonstrate that the latent space of molecules inferred by our model is smooth. To that aim, given a molecule, along with its associated graph $\mathcal{G}$, node features $\mathcal{F}$ and edge weights $\mathcal{Y}$, we first sample its latent representation $\mathcal{Z}$ using our probabilistic encoder, *i.e.*, $\mathcal{Z} \sim q_\phi(\mathcal{Z}|G, \mathcal{F}, \mathcal{Y})$. Then, given this latent representation, we generate various molecular graphs by sampling from our probabilistic decoder, *i.e.*, $\mathcal{G}_i \sim p_\theta(\mathcal{G}|\mathcal{Z})$. Figure 4 summarizes the results for one molecule from ZINC dataset, which shows that the sampled molecules are topologically similar to the given molecule. Finally, we also show that our encoder, once trained, creates a latent space representation of molecules with powerful semantics. In particular, since each node in a molecule has a latent representation, we can make fine-grained changes to the structure of a molecule by perturbing the latent representation of single nodes. To this aim, we proceed as follows.

First, we select one molecule with n nodes from the ZINC dataset. Given its corresponding graph, node features and edge weights, $\mathcal{G}$, $\mathcal{F}$ and $\mathcal{Y}$, we sample its latent representation $\mathcal{Z}_0$. Then, we sample new molecular graphs $\mathcal{G}$ from the probabilistic decoder $\mathcal{G} \sim p_\theta(\mathcal{G}|\mathcal{Z})$, where $\mathcal{Z} = \{z_i + a_i z_i | z_i \in \mathcal{Z}_0, a_i \geq 0\}$ and a_i are given parameters. Figure 5 provides several examples across both datasets, which show that the latent space representation is smooth and, as the distance from the initial molecule increases in the latent space, the resulting molecule differs more from the original.
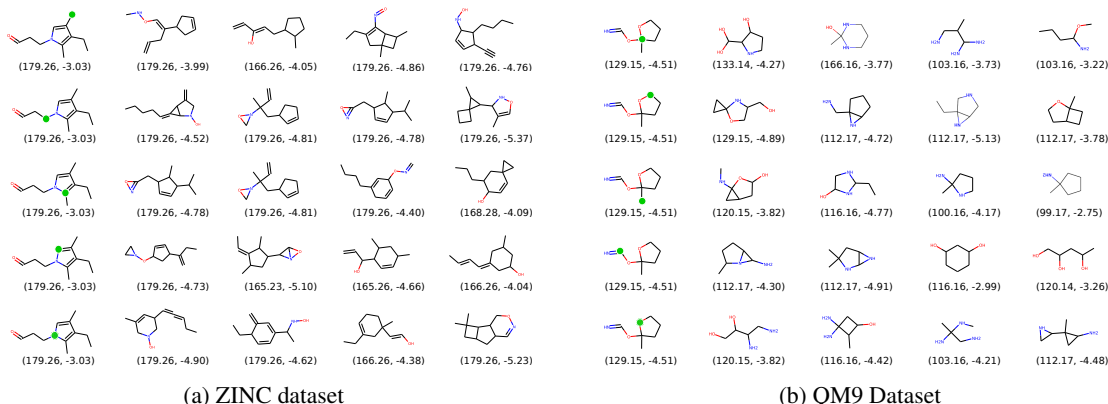
16

| | (a) ZINC dataset | | | | | (b) QM9 Dataset | | | |

Figure 5: Molecules sampled using the probabilistic decoder $\mathcal{G} \sim p_\theta(\mathcal{G}|\mathcal{Z})$, where $\mathcal{Z} = \{z_i + a_i z_i | z_i \in \mathcal{Z}_0, a_i \geq 0\}$ and a_i are given parameters. In each row, we start from the same molecule, set $a_i > 0$ for a single arbitrary node i (denoted as •) and set $a_j = 0, j \neq i$ for the remaining nodes. Under each molecule we report molecular weight and synthetic accessibility score.

Method		Penalized logP			QED		
		1st	2nd	3rd	1st	2nd	3rd
Bayesian optimisation	NEVAE	2.826	2.447	2.299	0.732	0.705	0.704
	GrammarVAE	2.521	2.330	1.893	0.724	0.712	0.698
	CVAE	1.835	1.776	1.234	0.712	0.698	0.508
	JTVAE	3.503	3.224	2.960	0.848	0.831	0.776
Property oriented decoder	MOLGAN	0.259	0.233	0.231	0.398	0.368	0.344
	ORGAN	3.148	2.334	2.145	0.812	0.807	0.745
	GCPN	4.284	3.621	2.902	0.913	0.885	0.775
	NEVAE (Algorithm 1)	**6.82**	6.65	6.55	**0.920**	0.916	0.912

Table 3: Penalized logP and QED scores for the best three molecules generated by our property oriented decoder, NEVAE (Algorithm 1), and all baselines. For our property oriented decoder, we used $\rho = 10^{-4}$ and $\rho = 5 \times 10^{-6}$ for penalized logP and QED scores, respectively.

5.3. Property oriented molecule generation

In this section, we first use our gradient-based algorithm (refer to Algorithm 1) to design property-oriented decoders that maximize penalized logP, $\tau_1(m)$, and QED, $\tau_2(m)$. Then, we use our gradient-based algorithm to design property-oriented decoders that, given a molecule of interest, are able to optimize the spatial configuration of its atoms for greater stability, *i.e.*, lower potential energy. Appendix B contain additional experiments assessing the influence that parameter ρ has on the properties and spatial configuration of the generated molecules.

Discovering molecules with high penalized logP and QED scores. To maximize the logP and QED, we consider the loss function $\ell(m) = -\tau_1(m)$ and $\ell(m) = -\tau_2(m)$, respectively, and compare our proposed property oriented decoder with three reinforcement learning based methods—MOLGAN (De Cao and Kipf, 2018), ORGAN (Guimaraes et al., 2017) and GCPN (You et al., 2018a)—and Bayesian optimization over the latent space of molecules under the encoders of several VAE based models—NEVAE, GrammarVAE, CVAE and JTVAE. Appendix C contains additional details regarding our implementation of Bayesian optimization.
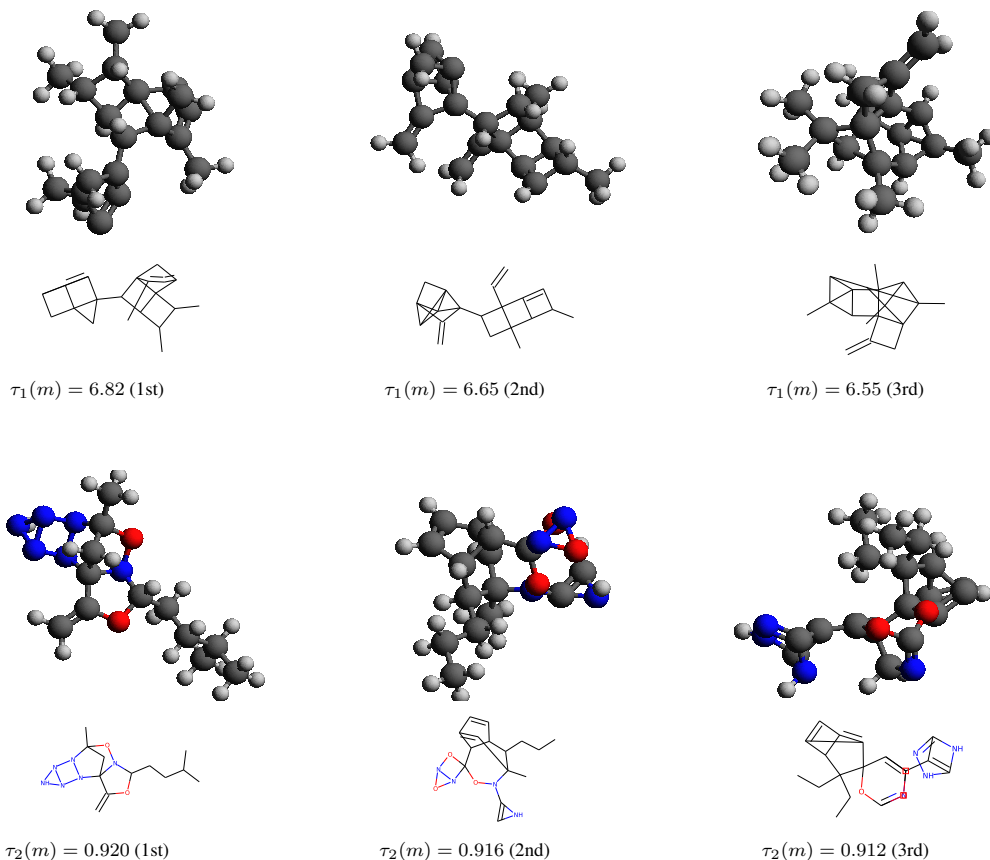
Figure 6: Visualization of the best three molecules generated by our property oriented decoder optimized for penalized logP (top row) and QED (bottom row) scores.

Table 3 shows the values of penalized logP and QED for the best three molecules generated by each method. The results show that our property oriented decoder, NEVAE (Algorithm 1), is able to identify molecules with property values 121% higher than those identified by the best performing competitor, *i.e.*, Bayesian optimization over the latent spaced of molecules generated by JTVAE. Finally, note that, in contrast with all the competitors, our property oriented decoder is also able to provide a (plausible) spatial configuration for the atoms of each of the identified molecules, as shown in Figure 6.

Discovering molecules with low potential energy values. The stability of a molecule depends on its potential energy—a lower value of potential energy indicates higher stability. In this section, our goal is to generate the most stable three dimensional structure for a given two dimensional molecular graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with atoms $\{t_u\}_{u \in \mathcal{V}}$, initial spatial coordinates x'_u and bond-types $\mathcal{Y}$.

To this aim, we only need to optimize the part of the decoder that generates the spatial coordinates rather than the entire decoder. Therefore, given a molecular graph, we solve the following optimization problem using the same gradient-descent algorithm described in Section 4:

$$\underset{p(\cdot|\mathcal{E},\mathcal{Y},\mathcal{Z})}{\text{minimize}} \quad \mathbb{E}_{\mathcal{Z} \sim q_\phi(.|\mathcal{V},\mathcal{E},\mathcal{Y},\mathcal{F}')} \left[\mathbb{E}_{\{x_u\}_{u \in \mathcal{V}} \sim p(\cdot|\mathcal{E},\mathcal{Y},\mathcal{Z})} \left[S(\{x_u\}_{u \in \mathcal{V}} | \mathcal{E}, \mathcal{Y}, \mathcal{Z}) \right] \right], \quad (20)$$

Initial configuration, $SPE = -895.7493$ a.u.　　　Final configuration, $SPE = -898.9146$ a.u.

Initial configuration, $SPE = -901.4993$ a.u.　　　Final configuration, $SPE = -902.8904$ a.u.
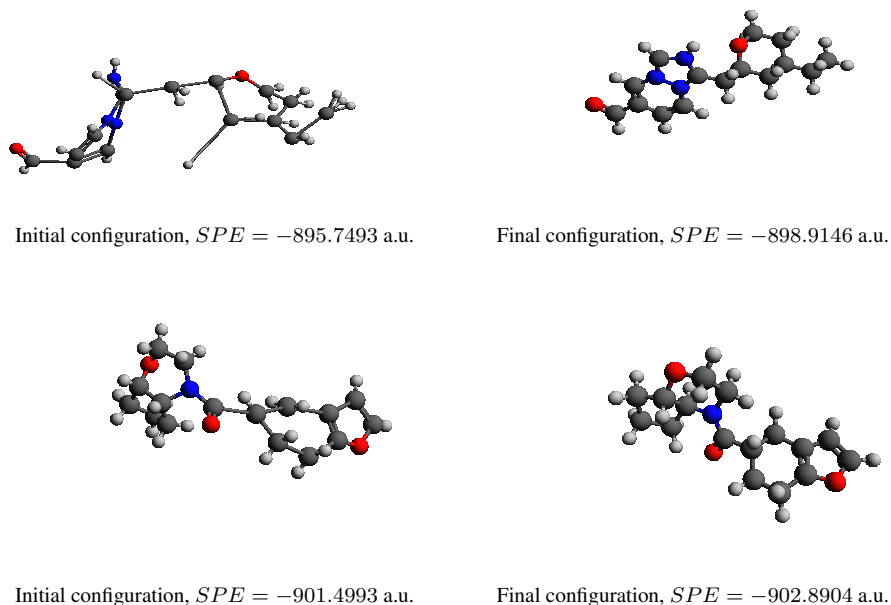
Figure 7: Single point energy (SPE) for the initial spatial configuration (left column) and optimized spatial configuration (right column) for two randomly sampled molecular graphs present in the ZINC dataset, where each row corresponds to each of the molecules.
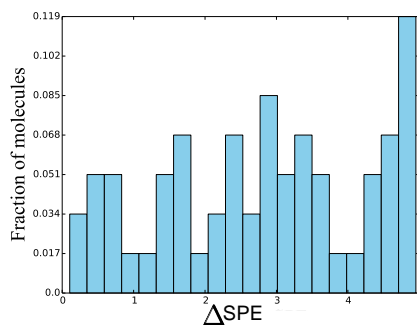

Figure 8: Difference between single point energies between the initial and the optimized spatial configurations (ΔSPE) for 100 randomly sampled molecular graphs present in the ZING dataset. The final optimized spatial configuration has always lower SPE.

with

$$S(\{\boldsymbol{x}_u\}_{u \in \mathcal{V}} | \mathcal{E}, \mathcal{Y}, \mathcal{Z}) = \ell(\{\boldsymbol{x}_u\}_{u \in \mathcal{V}}) + \rho \log \frac{p(\{\boldsymbol{x}_u\}_{u \in \mathcal{V}} | \mathcal{E}, \mathcal{Y}, \mathcal{Z})}{p_\theta(\{\boldsymbol{x}_u\}_{u \in \mathcal{V}} | \mathcal{E}, \mathcal{Y}, \mathcal{Z})}, \tag{21}$$

where the loss $\ell(.)$ is the single point energy (SPE) in terms of Atomic Unit (a.u.), computed using the Gaussian toolbox by Frisch et al. (2015), and the outer expectation in Eq. 20 is computed over all possible latent vectors under the posterior distribution conditioned on the given molecular graph.
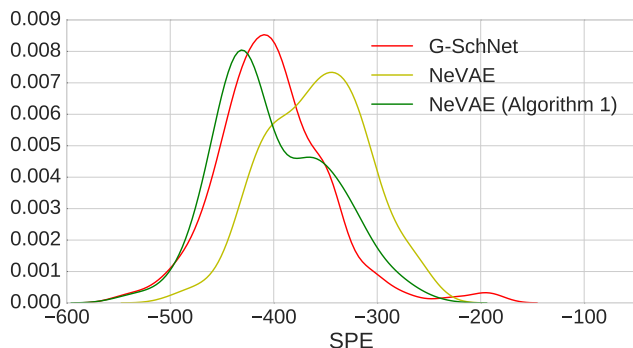
19

Figure 9: Empirical distribution of single point energy (SPE) for the molecules generated by our property oriented decoder (NEVAE, Algorithm 1), NEVAE and G-SchNet. Here, our property oriented decoder is trained to optimally trade off the SPE value of the molecules it generates and their similarity to the molecules generated by NEVAE.

Figures 7 and 8 show that, for a majority of the molecular graphs, our property oriented decoder can effectively modify the spatial configurations of the molecules to improve their stability[8]. Additionally, Figure 9 shows that, in comparison with the original decoder of our variational autoencoder and G-SchNet (Gebauer et al., 2019), our property oriented decoder is able to generate more stable molecules—molecules with lower SPE values. Here, we would like to acknowledge that, unlike G-SchNet, our model does not guarantee the preservation of rotational symmetry, however, it outperforms G-SchNet in terms of the stability of the generated molecules.

6. Conclusion

In this work, we have introduced a variational autoencoder for molecular graphs, that is permutation invariant of the nodes' labels of the graphs they are trained with, and allow for graphs with different number of nodes and edges as well as three dimensional spatial coordinates for atoms. Moreover, the decoder is able to guarantee a set of local structural and functional properties in the generated graphs through masking. Then, we have developed a gradient based algorithm to optimize the decoder of our model so that it learns to generate molecules that maximize the value of certain property of interest. Finally, we have shown that our variational autoencoder is able to discover plausible, diverse and novel molecules more effectively than several state of the art methods and, for several properties of interest, our optimized decoder is able to identify molecules with property values 121% higher than those identified by several state of the art methods.

Our work also opens many interesting venues for future work. For example, in the design of our variational autoencoder, we have assumed graphs to be static, however, it would be interesting to augment our design to dynamic graphs by, *e.g.*, incorporating a recurrent neural network or long short-term memory (LSTM) units. Moreover, we have focused on molecular graphs, however, we believe our methodology could be adapted to other real-world graphs. Finally, there are other problems related to molecular design, such as retro synthesis (Segler et al., 2018), where machine learning may advance the state of the art.

---

8. The molecular graphs were not present in the training set used to train NEVAE.

Acknowledgement

References

Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439): 509–512, 1999.

Hanjun Dai, Yingtao Tian, Bo Dai, Steven Skiena, and Le Song. Syntax-directed variational autoencoder for structured data. In *ICLR*, 2018.

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

MJ Frisch, GW Trucks, HB Schlegel, GE Scuseria, MA Robb, JR Cheeseman, G Scalmani, V Barone, B Mennucci, GA Petersson, et al. Gaussian 09, revision a. 02; gaussian, inc: Wallingford, ct, 2009. *There is no corresponding record for this reference*, 2015.

Niklas Gebauer, Michael Gastegger, and Kristof Schütt. Symmetry-adapted generation of 3d point sets for the targeted discovery of molecules. In *NeurIPS*, 2019.

Rafael Gómez-Bombarelli, David Duvenaud, José Miguel Hernández-Lobato, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *arXiv preprint arXiv:1610.02415*, 2016.

Gabriel Lima Guimaraes, Benjamin Sanchez-Lengeling, Carlos Outeiral, Pedro Luis Cunha Farias, and Alán Aspuru-Guzik. Objective-reinforced generative adversarial networks (organ) for sequence generation models. *arXiv preprint arXiv:1705.10843*, 2017.

William Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. *NIPS*, 2017.

John J Irwin, Teague Sterling, Michael M Mysinger, Erin S Bolstad, and Ryan G Coleman. Zinc: a free tool to discover chemistry for biology. *Journal of chemical information and modeling*, 52(7):1757–1768, 2012.

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

Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

Jack Kiefer, Jacob Wolfowitz, et al. Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*, 23(3):462–466, 1952.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of the 4th International Conference on Learning Representations*, 2016a.

Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016b.

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

Matt J Kusner, Paige, Brooks, and José Miguel Hernández-Lobato. Grammar variational autoencoder. *arXiv preprint arXiv:1703.01925*, 2017.

Tao Lei, Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Deriving neural architectures from sequence and graph kernels. *ICML*, 2017.

Jure Leskovec, Deepayan Chakrabarti, Jon Kleinberg, Christos Faloutsos, and Zoubin Ghahramani. Kronecker graphs: An approach to modeling networks. *Journal of Machine Learning Research*, 11(Feb): 985–1042, 2010.

Qi Liu, Miltiadis Allamanis, Marc Brockschmidt, and Alexander L Gaunt. Constrained graph variational autoencoders for molecule design. *arXiv preprint arXiv:1805.09076*, 2018.

Kenneth M Merz, Dagmar Ringe, and Charles H Reynolds. *Drug design: structure-and ligand-based approaches*. Cambridge University Press, 2010.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, 2013.

Steven M Paul, Daniel S Mytelka, Christopher T Dunwiddie, Charles C Persinger, Bernard H Munos, Stacy R Lindborg, and Aaron L Schacht. How to improve r&d productivity: the pharmaceutical industry's grand challenge. *Nature reviews Drug discovery*, 9(3):203, 2010.

Trang Pham, Truyen Tran, Dinh Q Phung, and Svetha Venkatesh. Column networks for collective classification. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pages 2485–2491, 2017.

Pavel G Polishchuk, Timur I Madzhidov, and Alexandre Varnek. Estimation of the size of drug-like chemical space based on gdb-17 data. *Journal of computer-aided molecular design*, 27(8):675–679, 2013.

Raghunathan Ramakrishnan, Pavlo O Dral, Matthias Rupp, and O Anatole Von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific data*, 1:140022, 2014.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

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

Bidisha Samanta, Abir De, Gourhari Jana, Pratim Kumar Chattaraj, , Niloy Ganguly, and Manuel Gomez-Rodriguez. Nevae: A deep generative model for molecular graphs. *AAAI*, 2019.

Marwin HS Segler, Mike Preuss, and Mark P Waller. Planning chemical syntheses with deep neural networks and symbolic ai. *Nature*, 555(7698):604, 2018.

Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. *arXiv preprint arXiv:1802.03480*, 2018.

Edward Snelson and Zoubin Ghahramani. Sparse gaussian processes using pseudo-inputs. In *NeurIPS*, 2006.

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

Jiaxuan You, Bowen Liu, Zhitao Ying, Vijay Pande, and Jure Leskovec. Graph convolutional policy network for goal-directed molecular graph generation. In *NeurIPS*, 2018a.

Jiaxuan You, Bowen Liu, Zhitao Ying, Vijay Pande, and Jure Leskovec. Graph convolutional policy network for goal-directed molecular graph generation. In *NeurIPS*, 2018b.

Appendix A. Implementation Details

Architecture details. Table 4 provides additional details on the architecture of our variational autoencoder for graphs, where it is important to notice that the parameters to be learned do not depend on the size of the graphs (*i.e.*, the number of nodes and edges). Note that, r and g are linear forms and the aggregator function Λ is a sum, which is a symmetric function, for simplicity[9].

Layer	Architecture	Inputs	Type of non-linearity	Parameters to be learned	Output
Input	Feedforward (K layers)	$\mathcal{E}, \mathcal{F}, \mathcal{Y}$	$r(\cdot)$: Linear $g(\cdot)$: Linear $\Lambda(\cdot)$: Sum	$W_1^{\mathcal{T}}, W_1^{\mathcal{X}}, ..., W_K^{\mathcal{T}}, W_K^{\mathcal{X}}$	$\mathbf{c}_1, \ldots, \mathbf{c}_n$
Encoder	Feedforward (Two layers)	$\mathbf{c}_1, \ldots, \mathbf{c}_n$	Softplus Softplus Softplus	W_h, b_h $W_\mu, b_\mu,$ W_σ, b_σ	$\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_n$ $\boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_n$
Decoder	Feedforward (One layer)	$\mathcal{Z}$	Softplus	W, b	$\mathcal{E}, \mathcal{Y}, \boldsymbol{f}$

Table 4: Details on the architecture of NEVAE.

Hyperparameter tuning. At the very outset, to train NEVAE, we implemented stochastic gradient descent (SGD) using the Adam optimizer. Therein, we had to specify four hyperparameters: (i) D – the dimension of z_u, (ii) K – the maximum number of hops used in encoder to aggregate information, (iii) L – the number of negative samples, (iv) l_r– the learning rate. Note that, all the parameters $\boldsymbol{W}_\bullet$'s and $\boldsymbol{b}_\bullet$'s in the input, hidden and output layers depend on D and K. We selected these hyperparameters using cross validation. More specifically, we varied l_r in a logarithmic scale, *i.e.*, $\{0.0005, 0.005, 0.05, 0.5\}$, and the rest of the hyperparameters in an arithmetic scale, and chose the hyperparameters maximizing the value of the objective function in the validation set. For synthetic (real) data, the resulting hyperparameter values were $D = 7(5)$, $K = 3(5)$, $L = 10(10)$ and $l_r = 0.005(0.005)$. To run the baseline algorithms, we followed the instructions in the corresponding repository (or paper).

Algorithm 2: Training with Minibatches

1: **Input:** Training graphs $\{\mathcal{G}_i(\mathcal{V}_i, \mathcal{E}_i)_{i \in [N]}\}$, hyperparameters $\psi = \{D, K, L, l_r\}$, Θ
2: **Output:** Inferred parameters $\hat{\Theta}$.
3: $\hat{\Theta} \leftarrow$ Initialize(Θ)
4: $\mathcal{B} \leftarrow$ CreateBatches$(\{\mathcal{G}_i(\mathcal{V}_i, \mathcal{E}_i)_{i \in [N]}\})$
5: **for** $\mathcal{B}_k \in \mathcal{B}$ **do**
6:    NeVAE$_{\hat{\Theta}} \leftarrow$ BuildComputationalGraph(Nodes$(\mathcal{B}_k), \hat{\Theta})$
7:    $\hat{\Theta} \leftarrow$ Train(NEVAE$_{\hat{\Theta}}, \mathcal{B}_k)$

Training with minibatch. We implemented stochastic gradient descent (SGD) using minibatches, where each batch contained graphs with the same number of nodes. More specifically, we first group the training graphs $\mathcal{G}_i$'s into batches $\mathcal{B} = \{\mathcal{B}_k\}$ such that $|\mathcal{V}_i| = |\mathcal{V}_j|$ for all $\mathcal{G}_i, \mathcal{G}_j \in \mathcal{B}_k$. Then, at each iteration, we select a batch at random, build a computation graph for the number of nodes corresponding to the batch using the parameters estimated in the previous iteration, and update the

---

9. We did experiment with other symmetric aggregator functions such as pooling, as in the inductive graph representation learning (Hamilton et al., 2017; Lei et al., 2017), and did not notice significant gains in practice.

parameters using the computation graph and the batch of graphs. Such a procedure helps to reduce the overhead time for building the computational graph, from per sample to per batch. This batching and training process is summarizedd in Algorithm 2, where "CreateBatches(...)" group the training graphs into batches, "BuildComputationalGraph(...)" builds the computation graph "NeVAE" using the parameters from the previous iteration and a given number of nodes, "Nodes(...)" returns the number of nodes of the graphs in a batch, and "Train(...)" updates the parameters given the computation graph and the parameters from the previous iteration.

Hardware and software specifications. We carried out all our experiments for NEVAE using Tensorflow 1.4.1, on a 64 bit Debian distribution with 16 core Intel Xenon CPU (E5-2667 v4 @3.20 GHz) and 512GB RAM.

Appendix B. Additional Results on Property Oriented Molecule Design

In this section, we investigate the influence that the parameter ρ (refer to Eq. 15) has on the properties and the structure of the molecules generated by our property oriented decoder. Figure 10 summarizes the results, which show that, as ρ increases, the generated molecules become structurally more planar and closer to the training molecules and their penalized logP and QED scores decreases, as expected.
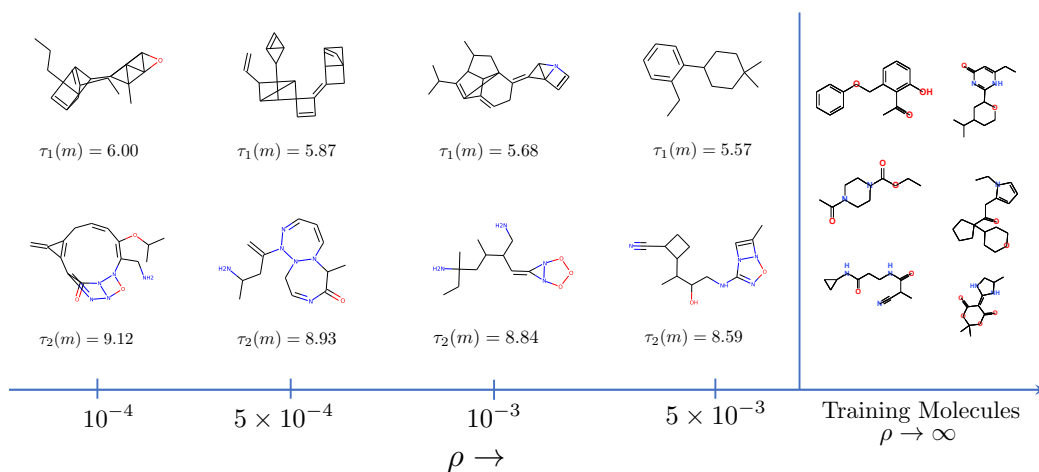

Figure 10: Molecules with the highest penalized logP (top row) and QED (bottom row) among those generated by our property oriented decoder for different values of ρ.

Appendix C. Additional Details on Bayesian Optimization

To implement Bayesian optimization (BO) for property-oriented molecule generation, we proceed similarly as in previous work (Gómez-Bombarelli et al., 2016; Kusner et al., 2017; Jin et al., 2018). More specifically, we first sample 3,000 molecules from our ZINC dataset, which we split into training (90%) and test (10%) sets. Then, for our model and each competing model with public domain implementations, we train a sparse Gaussian process (SGP) (Snelson and Ghahramani, 2006) with the latent representations and $y(m)$ values of 100 inducing points sampled from the training set. The SGPs allow us to make predictions for the property values of new molecules in the latent spaces. Then, we run 5 iterations of batch Bayesian optimization (BO) using the expected improvement (EI) heuristic (Jones et al., 1998), with 50 (new) latent vectors (molecules) per iteration.

In this section, we complement the performance comparison between NEVAE, GrammarVAE, CVAE, and JTVAE from Table 3 using two additional quality measures:

(a) the predictive performance of the trained SGPs in terms of log-likelihood (LL) and root mean square error (RMSE) on the test set; and,

(b) the average value $\mathbb{E}[y(m)]$, fraction of valid molecules and fraction of *good* molecules, *i.e.*, $y(m) > 0$, among the molecules found using EI.

Table 5, Figure 11 and Figure 12 summarize the results. In terms of log-likelihood and RMSE, the SGP trained using the latent representations provided by our model outperforms all baselines. In terms of the property values $\mathbb{E}[y_1(m)]$ of the discovered molecules and fraction of valid and good molecules, BO under NEVAE also outperforms all baselines; however, in terms of $\mathbb{E}[y_2(m)]$ this is the second best after JTVAE. Here, we would like to highlight that, while BO under JTVAE is able to find a few molecules with larger property value than BO under NEVAE, it is unable to discover a sizeable set of unique molecules with high property values.

Objective	Penalized logP				QED			
	NEVAE	GrammarVAE	CVAE	JTVAE	NEVAE	GrammarVAE	CVAE	JTVAE
LL	**-1.45**	-1.75	-2.29	*-1.54*	**-1.27**	-1.75	-1.77	*-1.48*
RMSE	**1.23**	1.38	1.80	*1.25*	**0.85**	1.37	1.39	*1.07*
Fraction of *valid* molecules	**1.00**	0.77	0.53	**1.00**	**1.00**	0.42	0.33	**1.00**
Fraction of *unique* molecules	**0.58**	0.29	0.41	0.32	**0.61**	0.22	0.20	0.42

Table 5: Property prediction performance (LL and RMSE) using Sparse Gaussian processes (SGPs) and property maximization using Bayesian Optimization (BO).

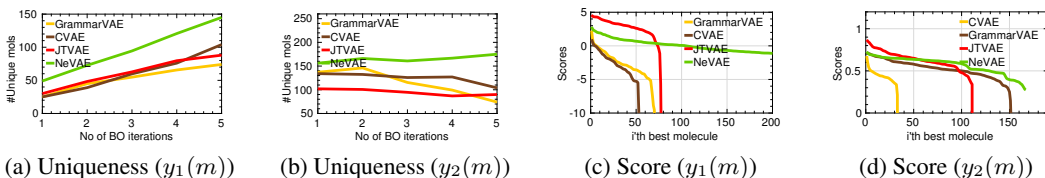(a) Uniqueness ($y_1(m)$) (b) Uniqueness ($y_2(m)$) (c) Score ($y_1(m)$) (d) Score ($y_2(m)$)

Figure 11: Property maximization using Bayesian optimization. Panel (a) and (b) show the variation of Uniqueness with the no. of BO iterations for $y_1(m)$ and $y_2(m)$ respectively. Panel (c) and (d) show the values of $y_1(m)$ and $y_2(m)$ sorted in the decreasing order.

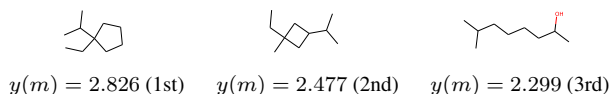$y(m) = 2.826$ (1st) $y(m) = 2.477$ (2nd) $y(m) = 2.299$ (3rd)

Figure 12: Best molecules found by Bayesian Optimization (BO) using our model.

Appendix D. Additional Experiments on Synthetic Graphs

In this section, we first demonstrate that our original model NEVAE is able to generate graphs with a predefined local topological property, *i.e.*, graphs without triangles. Then, we show that our model is able to learn smooth latent representations of a popular type of random graphs, Kronecker graphs (Leskovec et al., 2010). Then, we present additional quantitative results on the ability of our model to learn and mimic the generative processes that determine the absence or presence of nodes and edges in of Kronecker graphs and Barabási-Albert graphs (Barabási and Albert, 1999), a scalability analysis and finally illustrate the effect of node label permutations on the decoder parameter estimation. Finally, we show that the optimal property-oriented decoder designed using variational inference is able to generate synthetic graphs with certain structural properties.

D.1. Experimental setup

We first generate two sets of synthetic networks, each containing 100 graphs, with up to $n = 1000$ number of nodes. The first set contains triangle free graphs and the second set contains a 50%-50% mixture of Kronecker graphs with initiator matrices: $\mathbf{\Theta}_1 = [0.9, 0.6; 0.3, 0.2]$, and $\mathbf{\Theta}_2 = [0.6, 0.6; 0.6, 0.6]$. For each dataset, we train our variational autoencoder for graphs by maximizing the corresponding ELBO. Then, we use the trained models to generate three sets of 1000 graphs by sampling from the decoders, *i.e.*, $\mathcal{G} \sim p_\theta(\mathcal{G}|\mathcal{Z})$, where $\mathcal{Z} \sim p(\mathcal{Z})$.

D.2. Quality of the generated graphs

We first evaluate the ability of our model to generate triangle free graphs by measuring the validity of the generated graphs, *i.e.*, Validity $:= |\{\mathcal{G}_i \in \mathbb{G} \,|\, \mathcal{G}_i$ has no triangles$\}|/|\mathbb{G}|$, where $\mathbb{G}$ is the set of 1000 graphs generated using the trained model. We experiment both with and without masking during training and during test time. We observe that, if we train and test our model with masking, it achieves a validity of 100%, *i.e.*, it always generates triangle free graphs. If we only use masking during training, our model is able to achieve a validity of 68%, and, if we do not use masking at all, our models achieves a validity of 57%. Moreover, while using masking during test does not lead to significant increase in the time it takes to generate a graph, using masking during training does lead to an increase of 18% in training time. Figure 13 shows several example of triangle free graphs generated by our model.

Next, we evaluate the ability of our model to learn smooth latent representations of Kronecker graphs as follows. First, we select two graphs ($\mathcal{G}_0$ and $\mathcal{G}_1$) from the training set, one generated using an initiator matrix $\mathbf{\Theta}_0 = [0.9, 0.6; 0.5, 0.1]$ and the other using $\mathbf{\Theta}_1 = [0.6, 0.6; 0.6, 0.6]$. Then, we sample the latent representations $\mathcal{Z}_0$ and $\mathcal{Z}_1$ for $\mathcal{G}_0$ and $\mathcal{G}_1$, respectively, and sample new graphs from latent values $\mathcal{Z}$ in between these latent representations (using a linear interpolation), *i.e.*, $\mathcal{G} \sim p_\theta(\mathcal{G}|\mathcal{Z})$, where $\mathcal{Z} = a\mathcal{Z}_0 + (1-a)\mathcal{Z}_1$ and $a \in [0, 1]$, and the node labels, which define the matching between pairs of nodes in both graphs, are arbitrary. Figure 14 provides an example, which shows that, remarkably, as $\mathcal{Z}$ moves towards $\mathcal{Z}_0$ ($\mathcal{Z}_1$), the sampled graph becomes similar to that of $\mathcal{G}_0$ ($\mathcal{G}_1$) and the inferred initiator matrices along the way smoothly interpolate between
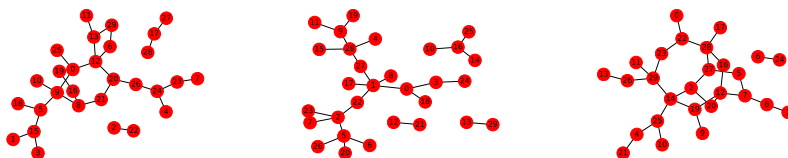
Figure 13: Graphs sampled using our variational autoencoder trained with a set of triangle free graphs. By using masking, our variational autoencoder is able to always generate triangle free graphs.
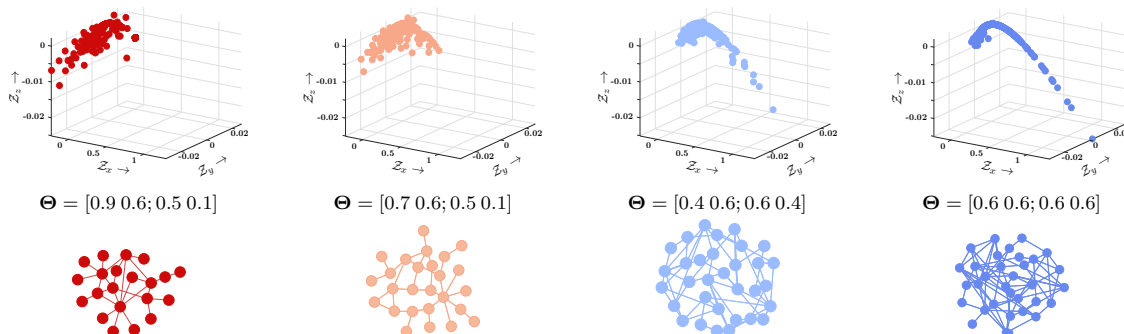

Figure 14: Graph generation by sampling graphs from our probabilistic decoder whose latent representation lies in between the latent representation of two Kronecker graphs, using a linear interpolation. Each column corresponds to a graph, the top row shows the latent representation of all nodes for the graphs in the bottom row, and the middle row shows the (inferred) initiator matrix for the Kronecker graph model.h

both initiator matrix. Here, we infer the initiator matrices of the graphs generated by our trained decoder using the method by Leskovec et al. (Leskovec et al., 2010). Table 6 provides a quantitative evaluation of the quality of the generated graphs, *i.e.*, it shows that the graphs our model generates are *indistinguishable* from true Kronecker graphs.

Finally, we create a set of 100 graphs with up to $n = 1000$ number of nodes sampled from the Barabási-Albert graph model with generation parameter $m = 1$. For both Barabási-Albert and Kronecker graphs, we evaluate the quality of the generated graphs using two quantitative evaluation metrics:

(i) Rank correlation: we use this metric to test to which extent the models we trained using Barabási-Albert and Kronecker graphs do generate *plausible* Barabási-Albert and Kronecker graphs, respectively. Intuitively, if the trained models generate plausible graphs, we expect that a graph $\mathcal{G}$ with a very high value of likelihood under the true model, $p(\mathcal{G}|\mathcal{P})$, should also have a high value of likelihood, $\mathbb{E}_{\mathcal{Z} \sim p(\mathcal{Z})} \log p_\theta(\mathcal{G}|\mathcal{Z})$, and ELBO under our trained model. For a set of graphs, we verify this expectation by computing the rank correlation between lists of graphs as follows. First, for each set of generated graphs $\mathbb{G}$, we order them in decreasing

order of $p(\mathcal{G}|\mathcal{P})$ and keep the top 10% in a ranked list[10], which we denote as $\mathcal{T}_p$. Then, we take the graphs in $\mathcal{T}_p$ and create two ranked lists, one in decreasing order of $\mathbb{E}_{\mathcal{Z}\sim p(\mathcal{Z})} \log p_\theta(\mathcal{G}|\mathcal{Z})$, which we denote as $\mathcal{T}_{p_\theta}$, and another one in decreasing order of ELBO, which we denote as $\mathcal{T}_{\text{ELBO}}$. Finally, we compute two Spearman's rank correlation coefficients between these lists:

$$\rho_{p_\theta} := \frac{\text{Cov}(\mathcal{T}_p, \mathcal{T}_{p_\theta})}{\sigma_{\mathcal{T}_p}\sigma_{\mathcal{T}_{p_\theta}}} \qquad \rho_{\text{ELBO}} := \frac{\text{Cov}(\mathcal{T}_p, \mathcal{T}_{p_{\text{ELBO}}})}{\sigma_{\mathcal{T}_p}\sigma_{\mathcal{T}_{\text{ELBO}}}}$$

where $\rho_{p_\theta}, \rho_{\text{ELBO}} \in [-1, 1]$.

(ii) Precision: we use this metric, which we compute as follows, as an alternative to the rank correlation above for Barabási-Albert and Kronecker graphs. For each set of generated graphs $\mathbb{G}$, we also order them in decreasing order of $p(\mathcal{G}|\mathcal{P})$ and create an ordered list $\mathcal{T}_p$, and select $\mathcal{T}_p^\uparrow$ as the top 10% and $\mathcal{T}_p^\downarrow$ as the bottom 10% of $\mathcal{T}_p$. Then, we re-rank this list in decreasing order of $\mathbb{E}_{\mathcal{Z}\sim p(\mathcal{Z})} \log p_\theta(\mathcal{G}|\mathcal{Z})$ and ELBO to create two new ordered lists, $\mathcal{T}_{p_\theta}$ and $\mathcal{T}_{\text{ELBO}}$. Here, if the trained models generate plausible graphs, we expect that each of the top and bottom halves of $\mathcal{T}_{p_\theta}$ and $\mathcal{T}_{\text{ELBO}}$ should have a high overlap with $\mathcal{T}_p^\uparrow$ and $\mathcal{T}_p^\downarrow$, respectively. Then, we define top and bottom precision as:

$$\gamma^\uparrow = \frac{|\mathcal{T}_x^\uparrow \cap \mathcal{T}_p^\uparrow|}{|\mathcal{T}_p^\uparrow|} \qquad \gamma^\downarrow = \frac{|\mathcal{T}_x^\downarrow \cap \mathcal{T}_p^\downarrow|}{|\mathcal{T}_p^\downarrow|}$$

where $\gamma^\uparrow, \gamma^\downarrow \in [0, 1]$ and $\mathcal{T}_x^\uparrow$ ($\mathcal{T}_x^\downarrow$) is the top (bottom) half of either $x = \mathcal{T}_{p_\theta}$ or $x = \mathcal{T}_{\text{ELBO}}$.

Table 6 summarizes the results, which show that our model is able to learn the generative process of Barabási-Albert more accurately than Kronecker graphs. This may be due to the higher complexity of the generative process Kronecker graph use. That being said, it is remarkable that our model is able to achieve correlation and precision values over 0.4 in both cases.

	ρ_{p_θ}	ρ_{ELBO}	$\gamma_{p_\theta}^\uparrow$	$\gamma_{p_\theta}^\downarrow$	$\gamma_{\text{ELBO}}^\uparrow$	$\gamma_{\text{ELBO}}^\downarrow$
Barabási-Albert	0.69	0.72	0.98	0.98	1.00	1.00
Kronecker	0.50	0.21	0.47	0.47	0.70	0.70

Table 6: Rank correlation (ρ), top precision ($\gamma^\uparrow$) and bottom precision ($\gamma^\downarrow$) achieved by our variational autoencoder trained with either Barabási-Albert or Kronecker graphs. In both cases, $\dim(z_i) = 7$ and $K = 3$. Here, the higher the value of rank correlation and (top and bottom) precision, the more accurately the trained models mimic the generative processes for Barabási-Albert and Kronecker graphs.

D.3. Effect of permuting node labels on decoder parameter estimation

In this section, we evaluate the permutation invariant property of our decoder over two networks—a Barabási-Albert graph with 1000 nodes and a Kronecker graph with 1024 nodes and an initiator matrix $\Theta = [0.6, 0.6; 0.6, 0.6]$. For each of these graphs $\mathcal{G}$, we first generate K isomorphic networks $\{\mathcal{G}_\pi\}$ with different node labels and then train our decoder (with encoder) for three different source node (s) sampling protocols ζ: (i) degree- distribution based sampling i.e. $\zeta(s) = d_s/\sum_{i\in\mathcal{V}} d_i$; (ii) maximum degree based sampling i.e. $\zeta(s) = \mathcal{U}\{s|d_s = \max_{i\in\mathcal{V}} d_i\}$; and uniform distribution

---

10. We discard the remaining graphs since their likelihood is very similar.
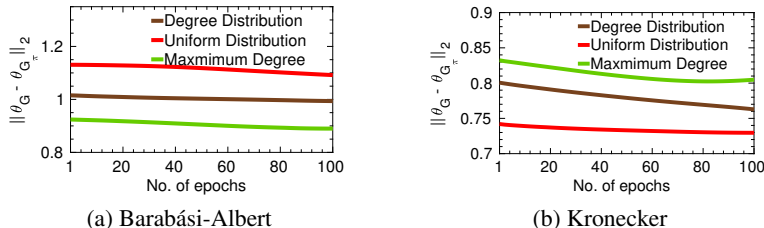
(a) Barabási-Albert   (b) Kronecker

Figure 15: Effect of node label permutations on the decoder parameter estimation, for different source node distribution ζ. Each plot shows the variation of mean difference in learned decoder weights $\mathbb{E}\big(||\theta_{\mathcal{G}_1}-\theta_{\mathcal{G}'}||\big)$ with the number of training iterations, where $\mathcal{G}$ and $\mathcal{G}'$ are the representation of same graphs different node labels. Panel (a) shows this variation for Barabási-Albert graphs with X nodes. Panel (b) shows this variation for a Kronecker graph with Y nodes and initiator matrix $\Theta = [0.6\,0.6; 0.6\,0.6]$.

i.e. $\zeta(s) = \mathcal{U}\{s \in \mathcal{V}\}$. Then, we investigate the variation of $\mathbb{E}(||\theta_{\mathcal{G}_\pi-\mathcal{G}_{\pi'}}||)$, the mean difference between the estimated decoder parameters over the pairs of $\{\mathcal{G}_\pi\}$, with the number of training iterations.

Figure 15 summarizes the results which show that degree based methods perform best in case of Barabási-Albert graph and uniform distribution performs best in Kronecker graph. This is because, the degree distribution of Barabási-Albert graph is skewed and as a result, a very few source nodes are sampled again and again, thereby giving similar parameter values. On the other hand, for homogeneous Kronecker graph, the degree distribution is more or less uniform. Consequently, the degree based methods perform worse in that case.

D.4. Effect of K (search-depth in encoder) on model performance

In this section, we investigate the behavior of our model with respect to the search depths K used in the decoder. Figure 16 summarizes the results, which show that, for Barabási-Albert graphs, our model performs consistently well for low values of K, however, for Kronecker graphs, the performance is better for high values of K. A plausible explanation for this is that Barabási-Albert networks are generated sequentially using only local topological features (only node-degrees), whereas the generation process of Kronecker graphs incorporates global topological features.
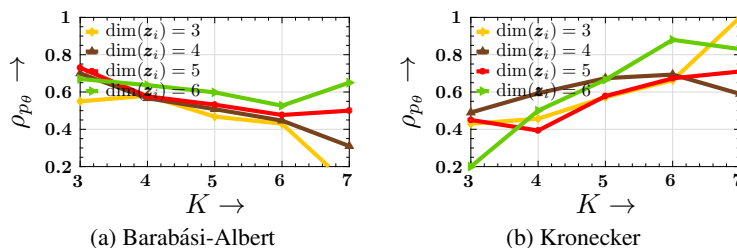

(a) Barabási-Albert   (b) Kronecker

Figure 16: Rank correlation (ρ_{p_θ}) with respect to the search depths K used in the decoder for Barabási-Albert graphs, small values of K achieve better performance, whereas for Kronecker graphs, a larger K provides better performance.

(a) Training time vs. # of nodes
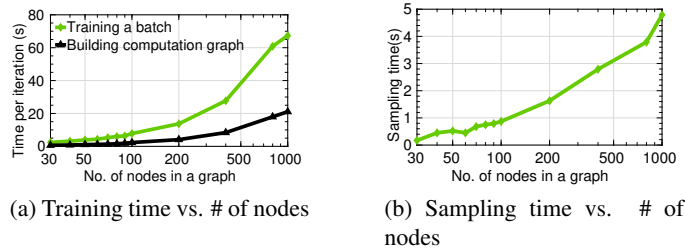
(b) Sampling time vs. # of nodes

Figure 17: Scalability of our inference procedure and probabilistic decoder. Panel (a) shows the time per iteration of our variational inference procedure against the size of the graphs in the training set using batches of 10 graphs with average degree 3. Panel (b) shows the time our probabilistic decoder takes to sample an entire graph with average degree 3 against the size of the graph.

D.5. Scalability

We first compute the running time of our variational inference procedure against the size of the graphs in the training set and then compute the running time of our probabilistic decoder against the size of the sampled (generated) graphs. Figure 17 summarizes the results, which show that both in terms of inference and sampling, our model easily scales to $\sim$1,000 nodes. For example, for graphs with 1000 nodes (average degree 3), our inference procedure takes $67 + 20$ seconds to run one iteration of SGD with a batch size of 10 graphs and, for graphs with 50 nodes, our inference procedure takes less than 10 seconds per iteration. Moreover, our probabilistic decoder can sample a graph with 1000 (50) nodes (average degree 3) in only 5 (0.5) seconds.
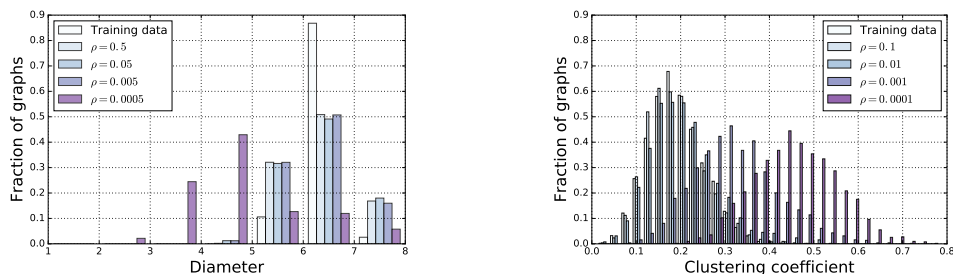

Figure 18: Diameter and clustering coefficient distribution for different values of ρ.

D.6. Property oriented graph generation

We first train NEVAE over 10,000 Barabási-Albert (Barabási and Albert, 1999) graphs with mean number of nodes $\lambda_n = 50$ and mean number of edges $\lambda_l = 95$ and then we optimize this trained model using Algorithm 1 twice so that it generates graphs with (i) low diameter and (ii) high clustering coefficients, respectively. To this aim, we set the loss functions $\ell(\mathcal{G}) = \text{Diameter}(\mathcal{G})$ and $\ell(\mathcal{G}) = 1 - \text{Clustering-coefficient}(\mathcal{G})$, respectively. Figure 18 summarizes the results by means of the distributions of diameters and clustering coefficients of the generated graphs against different

values of the parameter ρ. The results show that, the smaller the value of ρ, the lower (higher) the diameter (clustering coefficient), as one could expect.

	$D = 4$	$D = 6$
$\rho = 1$	0.23	0.40
$\rho = 10$	0.38	0.45
$\rho = 100$	0.62	0.75

Table 7: Success rate at generating graphs with diameter smaller than $\bar{D}$ for different values of ρ.

Next, we specify an upper bound for the diameter $(\bar{D})$ and then train our property oriented decoder to generate graphs whose diameters are always smaller than $\bar{D}$. To this aim, we set the loss function $\ell(\mathcal{G}) = \max\{\text{Diameter}(\mathcal{G}) - \bar{D}, 0\}$. Here $\mathcal{G}$ is the generated graph and $\bar{D}$ is the maximum allowed specified diameter value. We measure the quality of the generated graphs in terms of success rate (You et al., 2018b), *i.e.*,

$$\text{Success rate} = \frac{|\{\mathcal{G}|\text{Diameter}(\mathcal{G}) \leq \bar{D}\}|}{|\{\mathcal{G}\}|} \tag{22}$$

Table 7 summarizes the results for different ρ, which shows that, the lower the value of ρ, the higher the success rate.