DEFactor: Differentiable Edge Factorization-based Probabilistic Graph Generation

Anonymous authors
Paper under double-blind review

Abstract

Generating novel molecules with optimal properties is a crucial step in many industries such as drug discovery. Recently, deep generative models have shown a promising way of performing de-novo molecular design. Although graph generative models are currently available they are either computationally expensive, limiting their use to only small graphs or are formulated as a sequence of discrete actions needed to construct a graph, making the output graph non-differentiable w.r.t the model parameters, therefore preventing them to be used in scenarios such as conditional graph generation. In this work we propose a model for conditional graph generation that is computationally cheap, scalable, directly optimises properties of the graph, and generates a probabilistic graph, making the process differentiable, thus enabling end-to-end training with stochastic gradient descent. We demonstrate favourable performance of our model on prototype-based molecular graph conditional generation tasks.

1 Introduction

In this paper we address the problem of learning probabilistic and differentiable generative graph models for tasks such as prototype-based conditional sampling of molecules with optimal properties. More precisely we focus on generating realistic molecular graphs, similar to a target molecule (the prototype) and whose attributes can be controlled or optimised.

The main challenge stems from the discrete nature of molecules. This particularly prevents us from using global discriminators that assess generated samples and back-propagate their gradients to guide the optimisation of the generator. This becomes very important in cases where we want to either optimise the property of a graph or explore the vicinity of an input graph (prototype) for conditional optimal generations, an approach that has proven successful in controlled image generation (Odena et al., 2016; Chen et al., 2016).

A number of recent approaches (You et al., 2018a; Li et al., 2018a) have tried to overcome this limitation by performing indirect optimisation. You et al. (2018a) include molecular graph optimisation task in a reinforcement learning framework. However, this method tends to suffer from high variance during training. Kang & Cho (2018) suggest to resort to a reconstruction-based formulation which is directly applicable to discrete structure and does not require gradient estimation. However, it is limited by the number of samples available. Moreover there is always a risk that the generator simply ignores the part of the latent code containing the property that we want to optimise. Finally, Jin et al. (2018) performs Bayesian optimisation which consists of optimising a proxy (the latent code) of the molecular graph rather than the graph itself.

An orthogonal approach is to focus on a graph decoding scheme that outputs the graph all-at-once so that we are able to perform direct optimisation on its probabilistic continuous approximation. Simonovsky & Komodakis (2018) and Cao & Kipf (2018) took that direction by having their generator directly output the adjacency and node features tensors of the graph. However, both decoding schemes make use of fixed size MLP layers which restricts their use to very small graphs of a predefined maximum size.

In this work, we address these issues directly by:

- designing a novel differentiable probabilistic graph decoding scheme that is computationally scalable and is able to generate arbitrary sized graphs, and
- evaluating the generator's ability to modify molecular graphs attributes in the context of prototype-based molecular graph generation.

2 Related work

Lead-based Molecule Optimisation. In the majority of molecule design tasks, the aim is to obtain molecules that optimize certain criteria, for examply activity against a biological target, or physico-chemical properties. Currently the most popular solution is to fine-tune an existing generative model so that it is likely to produce molecules that satisfy a desired set of properties Segler et al. (2017). However finding the good training strategy for multiple objectives is challenging for these models. To that regard, Jin et al. (2018) proposed a VAE framework which performs Bayesian optimisation of the molecular graphs in the learned latent space. **?** and Li et al. (2018a) proposed a sequential graph decoding schemes in which conditioning properties can be added as input but here again these methods are unable to perform direct optimisation. In contrast, You et al. (2018a) recently suggested to perform optimisation while keeping the efficient sequential-like You et al. (2018b) generative scheme by formulating it as a reinforcement learning problem. Our approach in this paper is to design a scalable graph decoding scheme that enables us to perform direct optimisation for conditional generation tasks.

Graph Generation Models. Current work on graph generation can be divided into two type of graph decoding schemes: the models that generate graphs **sequentially** You et al. (2018b); Li et al. (2018a); You et al. (2018a); **?**. These methods aim at constructing a graph by predicting a sequence of addition/edition actions. The process starts from an empty graph and at each time step a discrete transition is predicted and the graph is updated. While this makes the model scalable as the number of parameters does not depend on the size of the graph, the final graph is still non-differentiable w.r.t. to the decoder's parameters which hinders their use for a large range of algorithms to perform direct molecular graphs optimisation.

On the other hand, methods in Cao & Kipf (2018); Simonovsky & Komodakis (2018) aim at constructing the graph **all-at-once**. These models however make use of fixed size MLP layers to predict the graph adjacency and node tensors which limit their use to graphs of a pre-chosen maximum size and restricts their study to molecular graphs with a maximum number of 9 heavy atoms, compared to ~ 40 in sequential models. Moreover, given the graph latent representation z, MLP layers model nodes and edges in the graph conditionally independent. This enforces the latent code to contain all the details about the node interdependencies within the graph. Such constraint does not leave space to learn higher level features if the size of the latent code is not chosen carefully.

We propose to tackle these drawbacks by designing a graph decoding scheme that is:

- **Scalable**: this means that the number of parameters of the decoder should not depend on a fixed pre-defined maximum graph size.
- **All-at-once**: we seek to construct the probabilistic graph all at once to have a differentiable probabilistic final output w.r.t. the decoder's parameters. This point is particularly important to be able to perform a direct discriminator-based graph optimisation.
- **Simple**: the autoregressive fashion of modeling continuous node embeddings makes our decoder powerful enough to obtain good reconstruction results on *large* graphs.
- **Interpretable**: the way we construct the adjacency tensor of the graph by learning edge-specific pairwise similarity metrics allows interpretability for our models.

3 DEFactor

Molecules can be represented as graphs $G = (V, E)$ where atoms and bonds correspond to the nodes and edges respectively. Each node in V is labeled with its atom type which can be considered as part of its features. Equally, a molecular graph can be defined by its adjacency tensor $E \in \{0,1\}^{n \times n \times e}$ where n is the number of nodes (atoms) in the graph and e is the number of edge (bond) types that we can have between two atoms and the node types are represented by a node feature tensor $N \in \{0,1\}^{n \times d}$ which is composed of several one-hot-encoded properties. In the following sections we will describe our proposed DEFactor model applied to molecular graphs.

3.1 Graph Construction Process

Given a molecular graph defined as $G = (N, E)$ we propose to leverage the edge-specific information propagation framework described in Simonovsky & Komodakis (2017) to learn a set of informative embeddings from which we can directly infer a graph. More specifically, our graph construction process is composed of two parts:

- An **Encoder** that in
  - **step 1** performs several spatial graph convolutions on the input graph, and in
  - **step 2** aggregates those embeddings into a single graph latent representation.
- A **Decoder** that in
  - **step 3** autoregressively generates a set of continuous node embeddings conditioned on the learnt latent representation, and in
  - **step 4** reconstruct the whole graph in an edge-factorization fashion.

Steps 1 and 2: Graph Representation Learning. We propose an encoder that makes an efficient use of the information contained in the bonds by having a separate information propagation channel for each bond type. The information is propagated in the graph using a Graph Convolutional Network (GCN) update rule (Kipf & Welling, 2016b) so that each node embedding can be written as a weighted sum of the edge-conditioned information of its neighbors in the graph. Namely for each l-th layer of the encoder, the representation is given by:

$$H^l = \sigma(\sum_e [D_e^{-\frac{1}{2}} E_e D_e^{-\frac{1}{2}} H^{l-1} W_e^l] + D^{-1} H^{l-1} W_s^l), \tag{1}$$

where E_e is the e-th frontal slice of the adjacency tensor, D_e the corresponding degree tensor and W_e^l and W_s^l are learned parameters of the layer.

Once we have those node embeddings we further aggregate them to obtain a fixed-length latent representation of the graph. We propose to parametrize this aggregation step by an *LSTM* and we compute the graph latent representation by a simple linear transformation of the last hidden state of this **Aggregator**:

$$z = g_{agg}(f_{LSTM}^e(\{H^K\})). \tag{2}$$

Even though the use of an LSTM makes the aggregation non permutation-invariant, similar to GraphSAGE Hamilton et al. (2017), we adapt the LSTM aggregator to work on a randomly permuted set of embeddings and noticed that it did not affect the performance of the model.

In the subsequent steps we are interested in designing a graph decoding scheme from the latent code that is both scalable and powerful enough to model the interdependencies between the nodes and edges in the graph.

Step 3: Autoregressive Embeddings Generation. As specified above, we are interested in building a graph decoding scheme that models the nodes and their connectivity (represented by continuous embeddings S) in an autoregressive fashion so that the latent code on which is conditioned the decoding has enough *dimensions* to encode more high-level features. Like stated previously, methods that suggest to compute the graph all at once (Simonovsky & Komodakis, 2018; Cao & Kipf, 2018) model each node and edge as conditionally independent given the latent code z: this means that every detail of their dependencies within the graph has to be encoded in this latent variable: that does not leave much room for high-level feature learning if the size of the latent code is not chosen carefully. We propose to tackle this drawback by autoregressive generation of the continuous embeddings $s = [s_0, s_1, ..., s_n]$ for n nodes, with the embedding containing enough information about the node itself and its neighbourhood. More precisely we model the generation of node embeddings such that:

$$p(s|z) = \prod_{i=1}^{n} p(s_i|s_{<i}, z). \tag{3}$$

In our model, the autoregressive generation of embeddings is parametrized by a simple *LSTM* and is completely deterministic such that at each time step t the *LSTM* decoder takes as input the previously generated embeddings and the latent code z which captured node-invariant features of the graph. Each embedding is computed as a function of the concatenation of the current hidden state and the latent code z such that:

$$h_{t+1} = f^d_{LSTM}(g_{in}([z, s_t]), h_t) \tag{4}$$

$$s_{t+1} = f_{embed}([h_{t+1}, z]), \tag{5}$$

where f^d_{LSTM} corresponds to the LSTM recurrence operation and g_{in} and f_{embed} are parametrized as simple MLP layers to perform nonlinear feature extraction.

Step 4: Graph Decoding from Node Embeddings. As stated previously, we want to drive the generation of the continuous embeddings s towards latent factors that contains enough information about the node they represent (i.e. we can easily retrieve the one-hot atom type performing a linear transformation of the continuous embedding) and its neighbourhood (i.e. the adjacency tensor can be easily retrieved by comparing those embeddings in a pair-wise manner). For those reasons, we suggest to factorize each bond type that we can have between two atoms in a relational inference fashion (Zitnik et al., 2018; Kipf et al., 2018).

Let $S \in \mathbb{R}^{n \times p}$ be the concatenated continuous node embeddings generated in the previous step. We reconstruct the adjacency tensor E by learning edge-specific similarity measure as follows for k-th edge type:

$$p(E_{:,:,k}|S) = \prod_{i=1}^{n} \prod_{j=1}^{n} p(E_{i,j,k}|s_i, s_j). \tag{6}$$

We model this by a set of edge-specific factors $U = (u_1, \cdots, u_e) \in \mathbb{R}^{e \times p}$ such that we can reconstruct the adjacency tensor as :

$$\tilde{E}_{i,j,k} = \sigma(s_i^T D_k s_j) = p(E_{i,j,k}|s_i, s_j), \tag{7}$$

where σ is the logistic sigmoid function, D_k the corresponding diagonal matrix of the vector u_k and the factors $(u_i) \in \mathbb{R}^{e \times p}$ are learned parameters.

We reconstruct the node features (i.e. the atom type) with a simple affine transformation such that:

$$\tilde{N}_{i,:} = p(N_i|s_i) = \text{softmax}(W s_i), \tag{8}$$

where $W \in \mathbb{R}^{p \times d}$ is a learned parameter.

These four steps define our proposed graph autoencoder.

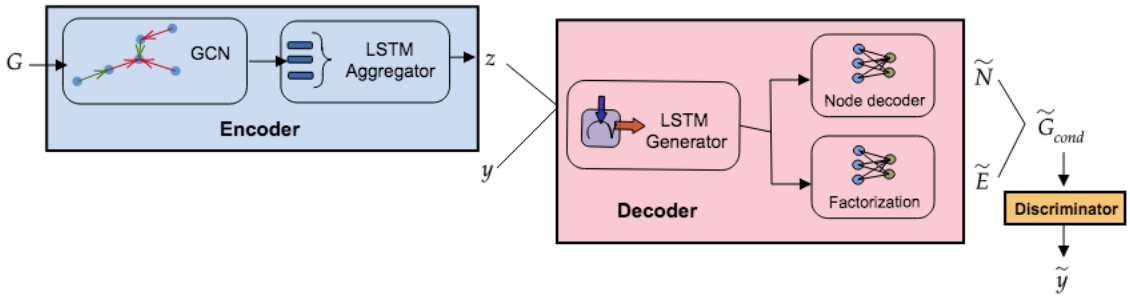

Figure 1: Full conditional autoencoder.

Generating Graphs of Arbitrary Sizes. In order to generate graphs of different sizes we need to add what we call here an **Existence** module that retrieves a probability of belonging to the final graph from each continuous embedding generated by the embeddings generator (in step 3). This module is parametrized as a simple MLP layer followed by a sigmoid activation and stops the unrolling of the embedding LSTM generator whenever we encounter a *non-informative* embedding. This module can be interpreted as an $< eos >$ translator.

3.2 Training

Teacher forcing. In order to make the model converge in reasonable time we used a trick similar to the teacher-forcing based training of language models (Williams & Zipser, 1989). The training is thus done in 3 phases:

- We first pre-train the GCN part along with the embedding decoder (factorization, nodes and existence modules) to reconstruct the graphs. This corresponds to the training of a simple Graph AE as in Kipf & Welling (2016a) except that we also want to reconstruct the nodes' one-hot features (and not just the *relations*).
- We then append those two units to the embedding aggregator and generator while keeping them fixed. In this second phase, the embedding generator is trained in a teacher forcing fashion where at each time step t the *LSTM* decoder does not take as input the previously generated embedding but the *true* one that is the direct output of the pretrained GCN embedding encoder.
- Finally in order to transition from a teacher-forcing to a fully autoregressive state we increasingly (Bengio et al., 2015) feed the LSTM generator more of its own predictions. When that fully autoregressive state is reached the pre-trained units are not frozen anymore and the whole model continues training end-to-end.

Log-Likelihood Estimates We train this first autoencoder framework in a MLE fashion with the following negative log-likelihood estimate $\mathcal{L}_{rec} = \mathcal{L}_X + \mathcal{L}_{\bar{X}} + \mathcal{L}_N$ corresponding to the existing edges (X), the non-existing edges ($\bar{X}$) and the node features (N) reconstruction terms:

$$\mathcal{L}_X = -\frac{1}{|X|} \sum_{(i,j)\in X} E^T_{i,j,:} \log(\tilde{E}_{i,j,:}) + (1 - E_{i,j,:})^T \log(1 - \tilde{E}_{i,j,:}) \tag{9}$$

$$\mathcal{L}_{\bar{X}} = -\frac{1}{|\bar{X}|} \sum_{(i,j)\in \bar{X}} \sum_k \log(1 - \tilde{E}_{i,j,k}) \tag{10}$$

$$\mathcal{L}_N = -\frac{1}{n} \sum N^T \log(\tilde{N}), \tag{11}$$

where X is the set of existing edges, $\bar{X}$ the set of non existing edges, E the adjacency tensor, N the node features tensor, n the number of nodes in the graph. As molecular graphs are sparse we found that such separate normalisations were crucial for the training.

3.3 Conditional Generation and Optimisation

Model overview. In this part we discuss our approach to build a conditional framework starting from the previous autoencoder architecture where the construction of a probabilistic graph $\tilde{G}$ is conditioned on some unregularized latent code z derived from a given input graph. We then augment this unstructured z with a set of structured attributes y that represent some physico-chemical properties of interest so that the decoder is conditioned on the joint (z, y). At the end of a successful training we expect this decoder to generate samples that have the properties specified in y and to be *similar* (in terms of information contained in z) to the original query molecular graph (encoded as z). To do so we choose a mutual information maximization approach (detailed in the appendix) that involves the use of discriminators that assess the properties $\tilde{y}$ of the generated samples.

Discriminator Pre-Training In this phase we pre-train a discriminator to assess the property y of a generated sample so that we can backpropagate its feedback to the generator (the discriminator can be trained on another dataset and we can have several discriminators for several attributes of interest). In order to have informative gradients in the early stages of the training we have trained the discriminator on continuous approximations of the discrete training graphs (details in the appendix A.1) so that our objective becomes:

$$\mathcal{L}_{dis} = \mathbb{E}_{(x,y)\sim\tilde{p}_{data}(x,y)}[-\log Q(y|x)], \tag{12}$$

where the graphs sampled from $\tilde{p}_{data}(x)$ are the probabilistic approximations of the discrete ones.

The next step is to incorporate the feedback signal of the trained discriminator in order to formulate the property attribute constraint. The training is decomposed in two phases in which we learn to reconstruct graphs of the dataset (**MLE** phase) and to modify chemical attributes (**Variational MI maximization** phase).

Encoder Learning. The encoder is updated only during the reconstruction phase where we sample attributes y from the true posterior. The encoder loss is a linear combination of the molecular graph reconstruction ($\mathcal{L}_{rec}$) and the property reconstruction($\mathcal{L}_{prop}$) s.t. $\mathcal{L}_{rec} = \mathbb{E}_{(x,y)\sim p_{data}(x,y), z\sim E(z|x)}[-\log p_{gen}(x|z, y)]$ (using the log-likelihood estimates in **(7)**) and $\mathcal{L}_{prop} = \mathbb{E}_{(x,y)\sim p_{data}(x,y), z\sim E(z|x), x'\sim p_{gen}(x|z,y)}[-\log Q(y|x')]$. The total encoder loss is:

$$\mathcal{L}_{enc} = \mathcal{L}_{rec} + \beta\mathcal{L}_{prop}. \tag{13}$$

Generator Learning. The generator is updated in both reconstruction and conditional phases. In the **MLE** phase the generator is trained with same loss $\mathcal{L}_{enc}$ as the encoder so that it is pushed towards generating realistic molecular graphs. In the **MI maximization** phase we sample the attributes from a prior $p(y)$ s.t. we minimize the following objective: $\mathcal{L}_{cond} = \mathbb{E}_{x\sim p_{data}(x), y\sim p(y) z\sim E(z|x), x'\sim p_{gen}(x|z,y)}[-\log Q(y|x')]$,

$$\mathcal{L}_{gen} = \mathcal{L}_{rec} + \alpha\mathcal{L}_{cond} + \beta\mathcal{L}_{prop}. \tag{14}$$

In this phase the only optimisation signal comes from the trained discriminator. Consequently there is a risk of falling off the manifold of the molecular graphs as no *realism* constraint is specified. A good way to make sure that this does not happen is to add a similar discriminator trained to distinguish between the real probabilistic graph and the generated ones so that when trying to satisfy the attribute constraint the generator is forced to produce valid molecular graphs. We leave that additional feature for future work.

4 Experiments

4.1 Dataset

Training is performed on a subset of the ZINC Irwin et al. (2012) dataset, keeping the molecules with less than 33 heavy atoms ($\sim$ 200k examples).

Table 1: Reconstruction accuracy. Baseline results are taken from Jin et al. (2018). Other models use a latent code of size 56.

Model	**Reconstruction**
CVAE	44.6
GVAE	53.7
SD-VAE	76.2
JT-VAE	76.7
DEFactor - 56	89.2
DEFactor - 64	89.4
DEFactor - 100	**89.8**

4.2 Results

Reconstruction. We evaluate the ability of our probabilistic factorization-based model to generate a graph through the full autoencoder framework. We want to check whether given a molecular graph we are able to encode it and decode it back with a multi-relational tensor factorization approach, correctly reconstructing it. We report here the ratio of exact reconstructions for the full graph autoencoder on the ZINC test set compared to other graph generation methods.

Our results are reported in Table 1. It shows that DEFactor outperforms previous models in terms of molecular graph reconstruction with fewer parameters and no validity check added during training. We believe that is explainable by a powerful decoder that models the interdependencies between the nodes in the graph through the autoregressive generation of continuous embeddings, thus the size of the latent space does not seem to have much effect on the results (we report the reconstruction results as a function of the number of heavy atoms in the molecule in the appendix).

LogP Optimisation and Conditional Generation. In this section, we evaluate the ability of the decoder to control its output attributes in the context of conditional *LogP* optimisation. First we directly evaluate the optimisation ability in the conditional context by plotting the target property (i.e. input of the decoder) as a function of the real attribute of the output for molecules in the test set. The results are shown in Fig. 2, where we see that in a majority of cases the model is able to generate a molecule with the desired conditioning property using the input molecule as a prototype. More visual examples of the prototype and generated molecules with increasing LogP can be seen in the Appendix.

For the second experiment, we quantify the the ability of the model to increase LogP in an implicit constrained scenario for a set of molecules in the test set. Similar to You et al. (2018a); Jin et al. (2018) we took the 800 molecules with lowest penalized LogP (as defined in GÃşmez-Bombarelli et al. (2016)) score. However, instead of training the model to condition on the penalized LogP we trained it to condition on the LogP score only and evaluated the improvements made in terms of penalized LogP. We are interested to know if the model learns by itself to target the given LogP while producing synthetically accessible molecules. In Table 2, we report the success rate for how often a modification succeeds (Suc.) and among those the average improvement (Imp.) and molecular similarity (Sim.) between the original molecules and the highest improvement for different thresholds δ on the similarity constraint (measured based on the Tanimoto distance). As can be seen, DEFactor while being slightly behind GCPN w.r.t. success rates, significantly outperforms other models in terms of improvements achieved (between 130$\times$ and 195$\times$ better optimised molecules compared to GCPN, the next best model). We believe the main reason for lower success rate for DEFactor is that we do not formulate any explicit validity/realism constraint in the training process.

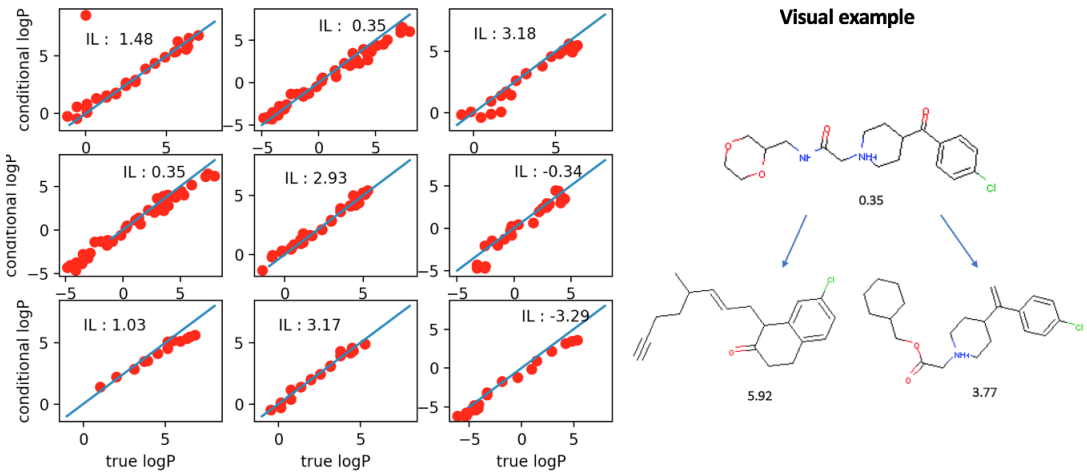

Figure 2: **Conditional generation**: The initial LogP value of the query molecule is specified as *IL*. We report on the y-axis the conditional value given as input and on the x-axis the true LogP of the generated graph when translated back into molecule (we have of course only reported the values when the decoded graphs correspond to valid molecules, which explains the difference in the total number of points for each query molecule).

δ	JT-VAE			GCPN			DEFactor		
	Imp.	Sim.	Suc.	Imp.	Sim.	Suc.	Imp.	Sim.	Suc.
0.0	1.91± 2.04	0.28±0.15	97.5%	4.20±1.28	**0.32±0.12**	**100%**	**6.62±2.50**	0.20±0.16	91.5%
0.2	1.68± 1.85	0.33±0.13	97.1%	4.12±1.19	**0.34±0.11**	**100%**	**5.55±2.31**	0.31±0.12	90.8%
0.4	0.84± 1.45	**0.51±0.10**	83.6%	2.49±1.30	0.47±0.08	**100%**	**3.41±1.8**	0.49±0.09	85.9%
0.6	0.21± 0.71	0.69±0.06	46.4%	0.79±0.63	0.68±0.08	**100%**	**1.55±1.19**	**0.69±0.06**	72.6%

Table 2: **Constrained penalized LogP maximisation task**: each row shows a different level of similarity constraint δ and columns are for improvements (Imp.), similarity to the original query (Sim.), and the success rate (Suc.). Values from other models are taken from You et al. (2018a)

5 Future work

In this paper, we designed a new way of modelling graphs and generating new ones in a conditional optimisation setup s.t. the final graph being fully differentiable w.r.t to the model parameters. We believe that our *DEFactor* model is a significant step forward to build ML-driven applications for de-novo drug design or generation of molecules with optimal properties without resorting to methods that do not directly optimise the desired properties.

Note that a drawback of our model is that it uses an MLE training process which forces us to either fix the ordering of nodes or to perform a computationally expensive graph matching operation to compute the loss.Moreover in our fully deterministic conditional formulation we assume that chemical properties optimisation is a one-to-one mapping but in reality there may exist many suitable way of optimizing a molecule to satisfy one property condition while staying similar to the query molecule. To that extent it could be interesting to augment our model to include the possibility of a one-to-many mapping. Another way of improving the model could also be to include a validity constraint formulated as training a discriminator that discriminates between valid and generated graphs.

References

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks, 2015.

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

Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets, 2016.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

Rafael GÃşmez-Bombarelli, Jennifer N. Wei, David Duvenaud, JosÃľ Miguel HernÃąndez-Lobato, BenjamÃŋn SÃąnchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D. Hirzel, Ryan P. Adams, and AlÃąn Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules, 2016.

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

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9 (8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL `http://dx.doi.org/10.1162/neco.1997.9.8.1735`.

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. doi: 10.1021/ci3001277. URL `https://doi.org/10.1021/ci3001277`. PMID: 22587354.

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation, 2018.

Seokho Kang and Kyunghyun Cho. Conditional molecular design with deep generative models. 2018. doi: 10.1021/acs.jcim.8b00263.

Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems, 2018.

Thomas N. Kipf and Max Welling. Variational graph auto-encoders, 2016a.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2016b.

Yibo Li, Liangren Zhang, and Zhenming Liu. Multi-objective de novo drug design with conditional graph generative model, 2018a.

Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning deep generative models of graphs, 2018b.

Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets, 2014.

Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier gans, 2016.

Marcus Olivecrona, Thomas Blaschke, Ola Engkvist, and Hongming Chen. Molecular de novo design through deep reinforcement learning, 2017a.

Marcus Olivecrona, Thomas Blaschke, Ola Engkvist, and Hongming Chen. Molecular de novo design through deep reinforcement learning, 2017b.

RDKit, online. RDKit: Open-source cheminformatics. `http://www.rdkit.org`. [Online; accessed 11-April-2013].

Marwin H. S. Segler, Thierry Kogej, Christian Tyrchan, and Mark P. Waller. Generating focussed molecule libraries for drug discovery with recurrent neural networks, 2017.

Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs, 2017.

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

Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks, 1989.

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

Jiaxuan You, Rex Ying, Xiang Ren, William L. Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models, 2018b.

Marinka Zitnik, Monica Agrawal, and Jure Leskovec. Modeling polypharmacy side effects with graph convolutional networks. 2018. doi: 10.1093/bioinformatics/bty294.

Appendix A Conditionnal setting

A.1 Graphs continuous approximation

For the pre-training of the discriminators we suggested to train them on continuous approximation of the discrete graphs that *ressembles* the output of the decoder. To that extent we used the trained partial graph autoencoder (used for the teacher forcing at the beginning of the training of the full autoencoder)

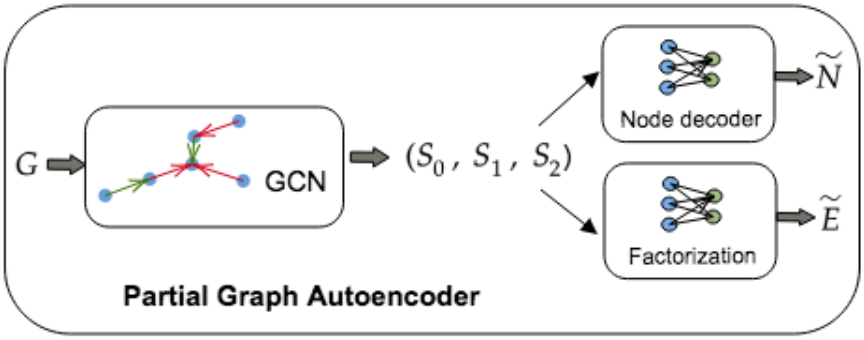

Figure 3: Partial graph Autoencoder used for the pre-training part

A.2 Mutual information maximization

For the conditional setting we choose a simple mutual information maximization formulation. The objective is to maximize the MI $I(X;Y)$ between the target property Y and the decoder's output $X = G_\theta(Y)$ under the joint $p_\theta(X,Y)$ defined by the decoder G_θ. In the conditional setting G_θ is also conditioned on the encoded molecule z but for simplicity we treat it as a parameter of the decoder (and thus reason with one target molecule from which we want to modify attributes). We define the MI as:

$$
\begin{aligned}
I(y; G_\theta(y)) &= \mathbb{E}_{x\sim G_\theta(y)}[\mathbb{E}_{y\prime\sim p_\theta(y|x)}[\log p_\theta(y\prime|x)]] + H(y) \\
&= \mathbb{E}_{x\sim G_\theta(y)}[D_{KL}(p_\theta(.|x)||Q(.|x)) \\
&+ \mathbb{E}_{y\prime\sim p_\theta(y|x)}[\log Q(y\prime|x)]] + H(y) \\
&\geq \mathbb{E}_{x\sim G_\theta(y)}[\mathbb{E}_{y\prime\sim p_\theta(y|x)}[\log Q(y\prime|x)]] + H(y)
\end{aligned}
$$

In our conditional setting we pre-trained the discriminators (parametrized by Q in the lower bound derivation) to approximate $p_{data}(y|x)$ which makes the bound tight only when $p_\theta(y_{paired}|x)$ is close to $p_{data}(y|x)$ and this corresponds to a stage where the decoder has maximized the log-likelihood of the data well enough (i.e. when it is able to reconstruct input graphs properly when z and y are paired). Thus, in the conditional setting we are maximizing the following objective:

$$
\mathcal{L}_{cond} = \mathbb{E}_{x,y\sim p_{data}(x,y),z\sim E(x),y'\sim p(y)}[\log G_\theta(y,z) + I(y\prime; G_\theta(y\prime, z))]
$$

A.3 Reconstruction as a function of number of atoms

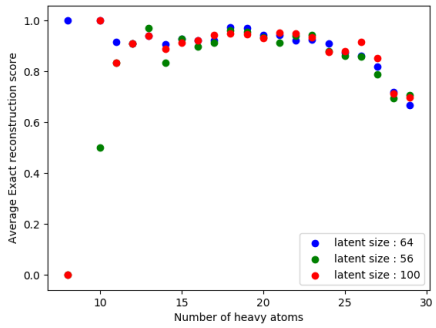

Figure 4: Accuracy score as a function of the number of heavy atoms in the molecule(x axis) for different size of the latent code

Notice that as we make use of a simple LSTM to encode a graph representation, there is a risk that for the largest molecules the long term dependencies of the embeddings are not captured well resulting in a bad reconstruction error. We capture this observation in figure 4. One possible amelioration could be to add other at each step other non-sequential aggregation of the embeddings (average pooling of the emebeddings for example) or to make the encoder more powerful by adding some attention mechanisms. We leave those for future work.

A.4 Visual similarity samples

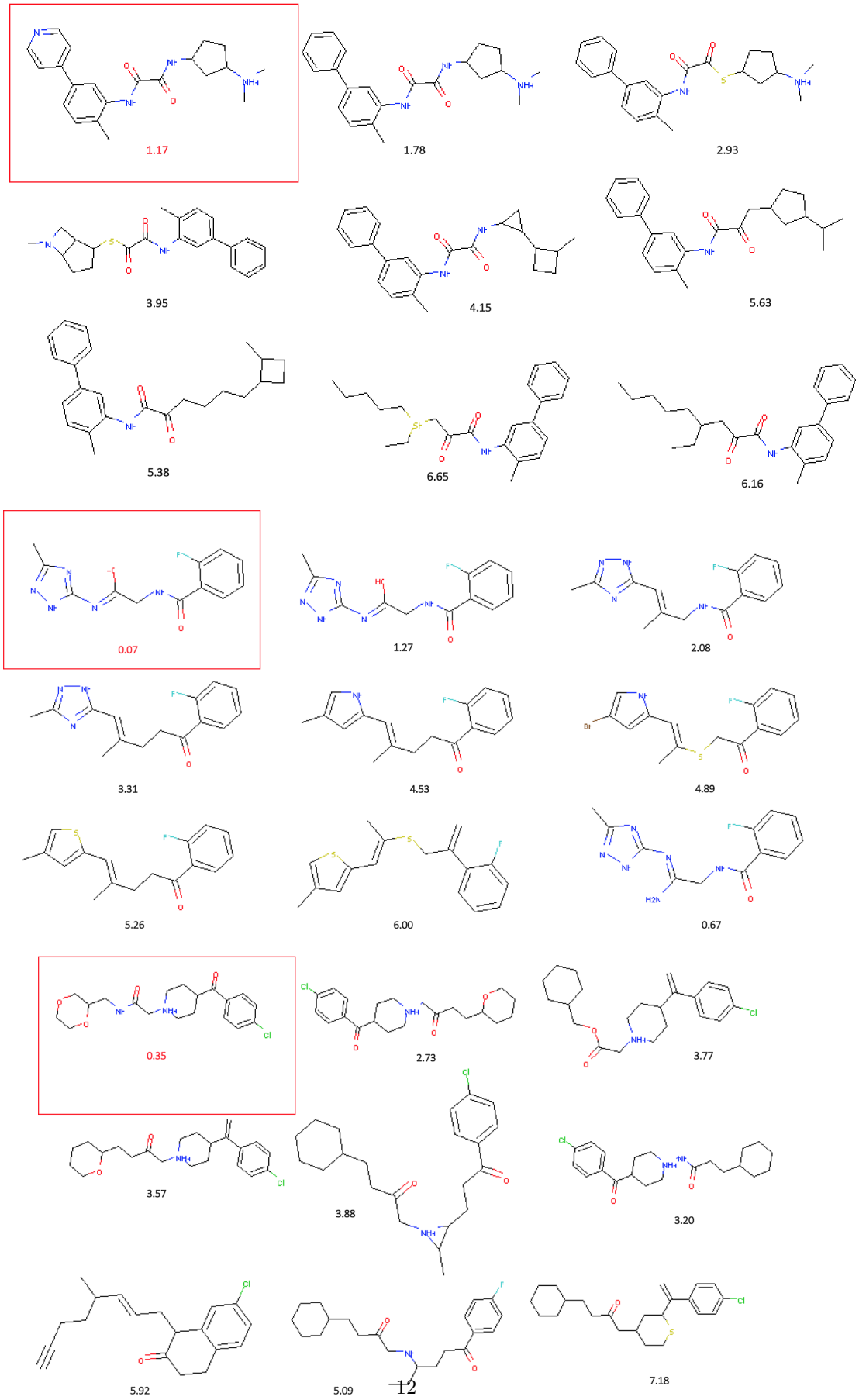

Figure 5: LogP increasing task visual example. The original molecule is circled in red.