DE NOVO DRUG DESIGN WITH DEEP GENERATIVE MODELS : AN EMPIRICAL STUDY

Mehdi Cherti, Balázs Kégl, Akin Osman Kazakçi

DE NOVO DRUG DESIGN WITH DEEP GENERATIVE MODELS : AN EMPIRICAL STUDY

Mehdi Cherti & Balázs Kégl
LAL/LRI
CNRS/Université Paris-Saclay
{mehdi.cherti, balazs.kegl}@gmail.com

Akın Kazakçı
MINES ParisTech,
PSL Research University, CGS-I3 UMR 9217
akin.kazakci@mines-paristech.fr

ABSTRACT

We present an empirical study about the usage of RNN generative models for stochastic optimization in the context of *de novo* drug design. We study different kinds of architectures and we find models that can generate molecules with higher values than ones seen in the training set. Our results suggest that we can improve traditional stochastic optimizers, that rely on random perturbations or random sampling by using generative models trained on unlabeled data, to perform knowledge-driven optimization.

1 INTRODUCTION

The goal of computer-based *de novo* drug design is to build new molecules from scratch that can be used as drugs. The molecules are designed in a way that it will bind to a target (for instance to a human protein or to a virus) to change its behavior. When a molecule binds to the desired target, it is called a *ligand*.

Most available molecule generation techniques rely on combination of a library of 'fragments'. A fragment is subgraph of a molecule structure, it can be an atom or a group of atoms (e.g., a ring). The fragments are combined in a chemically meaningful way to obtain new molecules (Schneider & Schneider, 2016). One advantage of using fragments is to reduce the search space, which would be huge if molecule structure generation was done one atom at a time, like it was done in atom-based structure generation techniques which are now less popular(Hartenfeller & Schneider, 2011; Schneider, 2013). Another advantage is that the fragments are extracted manually from known drug molecules and from chemistry knowledge. On the other hand as the library of fragments is fixed, it can constrain the search space too much and thus it might be possible to overlook some interesting molecules.

There are two main techniques for molecule generation depending on how much information we have about the target: receptor-based and ligand-based techniques. *Receptor-based* techniques are used when the 3D structure of the target is available, while *ligand-based* techniques are used when a a set of molecules that bind to a given target are already known and the goal is to find new molecules that can bind to the target even better or that can satisfy other constraints like ease of synthesis.The ligand-based algorithms take as input a set of ligands that are known to bind to a target, and return a new set of ligands that bind to the target.

Most approaches in the literature involve discrete optimization techniques like genetic algorithms operating on the graph representation of the molecules.We point to the reader some works about techniques used traditionally to design new molecules : (Gillet et al., 1993; Wang et al., 2000; Pierce et al., 2004; Douguet et al., 2005; Fechner & Schneider, 2006; Dey & Caflisch, 2008; Kutchukian & Shakhnovich, 2010; White & Wilson, 2010; Li et al., 2011; Hartenfeller et al., 2012; Li et al., 2016; Masek et al., 2016).

As pointed out by Gómez-Bombarelli et al. (2016b), current approaches rely on handcrafted rules for perturbing or hybridizing molecules or inserting fragments to obtain new molecules. While these rules rely on chemistry knowledge, they may bias the search space towards a specific portion of the space and some interesting molecules might be overlooked. One attempt to solve this issue would be to replace the handcrafted rules by rules learned from data, using generative models.

2 EXPERIMENTS

2.1 METHODOLOGY AND OBJECTIVE

As in (Gómez-Bombarelli et al., 2016a; Segler et al., 2017), we used the SMILES (Weininger, 1988) representation of molecules, which is a textual representation of the molecular graph using a formal language. Like in (Gómez-Bombarelli et al., 2016a; Segler et al., 2017), we used recurrent neural networks (RNNs) to learn a generative model of sequences for SMILES strings from a dataset of known molecules.

We used the same dataset than (Gómez-Bombarelli et al., 2016a), which is a subset of the ZINC dataset (Irwin et al., 2012; Sterling & Irwin, 2015).We split the dataset used in (Gómez-Bombarelli et al., 2016a) into a training and a validation set. The size of the training set was 200000, while the size of the validation set was about 50000. We sampled the hyperparameters of the models randomly from a prior (see the appendix for more details) then we trained the models on the training set and used the validation set for early stopping. Finally, for each molecule we sampled from the model we computed the score used in (Gómez-Bombarelli et al., 2016a):

$$J(m) = LogP(m) - SA(m) - ring\text{-}penalty(m) \tag{1}$$

A high $J(m)$ selects drug-like molecules, as measured by $LogP$(the partition coeficient, Wildman & Crippen (1999)) and penalizes molecules which are difficult to synthesize, measured by SA (synthetic accessibility) and $ring\text{-}penalty$. See the appendix for more details.

This score assesses each molecule, rather than a set of molecules generated by a model. To evaluate the models, we use the expected improvement criterion (EI; Močkus (1975)), used routinely in Bayesian optimization (Jones et al., 1998), to assess how much the score $J(m)$ improved over the training set. Given a set of M generated molecules $M = \{m_1, m_2, ..., m_M\}$ and the maximum score $J(m)$ on the training set $D = \{x_1, x_2, ..., x_N\}$, $J^*(D) = max_{i=1...N} J(x_i)$, the expected improvement(EI) is defined as:

$$EI(M) = \frac{1}{M} \sum_{i=1}^{M} \mathbb{I}\{J(m_i) > J^*(D)\}(J(m_i) - J^*(D)), m_i \in M^{1} \tag{2}$$

2.2 MODELS AND RESULTS

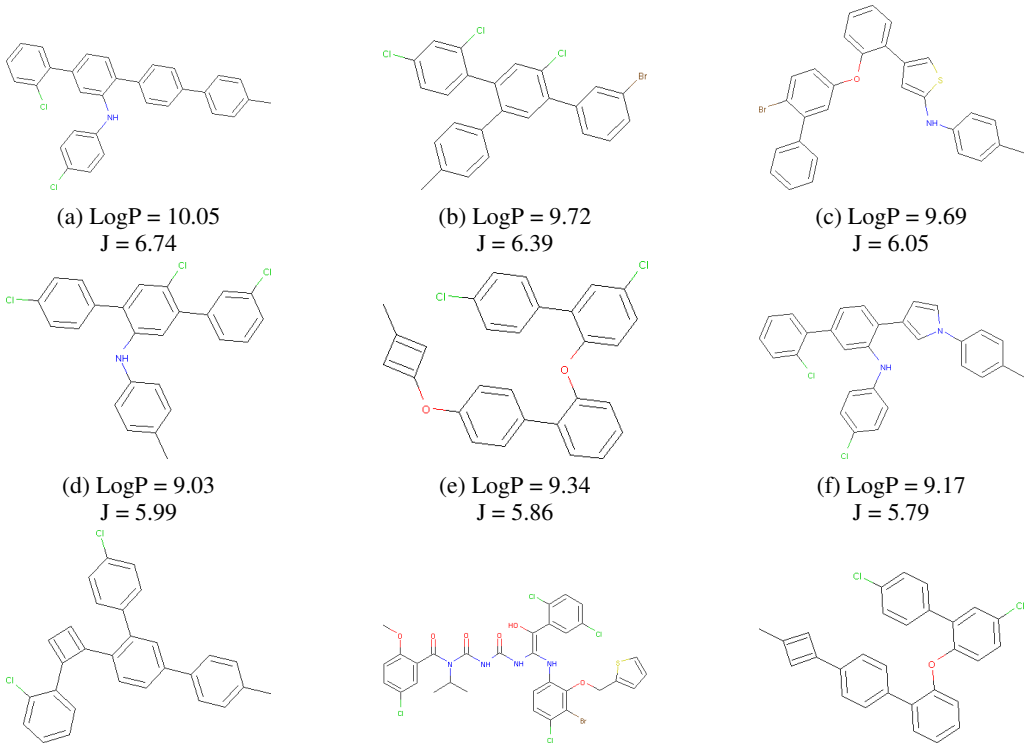
We used two kinds of RNN architectures. The first corresponds to char-RNNs (Karpathy et al., 2015) where the RNN is trained to predict the next character based on the full history of previous characters. The second type is a sequence-to-sequence autoencoder, analog to (Sutskever et al., 2014), but where the input and the output are the same and we do not use teacher forcing (Williams & Zipser, 1989).

We trained a total of 480 models. Among the top 8 of our models according to EI(eq. 2), 4 were autoencoders with a convolutional encoder and an RNN decoder, the remaining 4 models were autoencoders with an RNN encoder and decoder. Noise turned out to be helpful: all of these models were using a denoising criterion (Bengio et al., 2013) (during training and generation) which we adapted for sequential data (see the noise procedure we use in the appendix). None of the autoencoders were using LSTMs, they were either using GRUs (Cho et al., 2014) or vanilla RNNs (Karpathy et al., 2015).

To compare the models, we generated 100K molecules from each model (among the top 8 models), including the baseline variational autoencoder of Gómez-Bombarelli et al. (2016a)[2]. We discarded illegal and duplicate molecules. The distribution of the score J (eq.3) is visualized by violin plots in Figure 2. Interestingly, our models do not necessarily generate better molecules *on average*, but the thicker upper tail of the distributions ensures that there is significant mass above the best molecules

[1] $\mathbb{I}\{x > y\}$ is the indicator function, 1 if $x > y$, 0 if not.

[2] We sampled from their variational autoencoder by first sampling from the gaussian prior on the latent space then decoded back to the string space stochastically using the decoder.We used the code provided here.



(a) LogP = 10.05
J = 6.74

(b) LogP = 9.72
J = 6.39

(c) LogP = 9.69
J = 6.05

(d) LogP = 9.03
J = 5.99

(e) LogP = 9.34
J = 5.86

(f) LogP = 9.17
J = 5.79

(g) LogP = 9.17
J = 5.79

(h) LogP = 9.98
J = 4.91

(i) LogP = 9.46
J = 6.05

Figure 1: Generated molecules which have $LogP$ and J(eq.3) better than training data D, for which $LogP^*(D) = 8.25$ and $J^*(D) = 4.52$. We note that except for 1h, all the other molecules have a better LogP and J than the ones in (Gómez-Bombarelli et al., 2016a), for which the best reported molecule had $LogP = 8.51$ and $J = 5.02$.

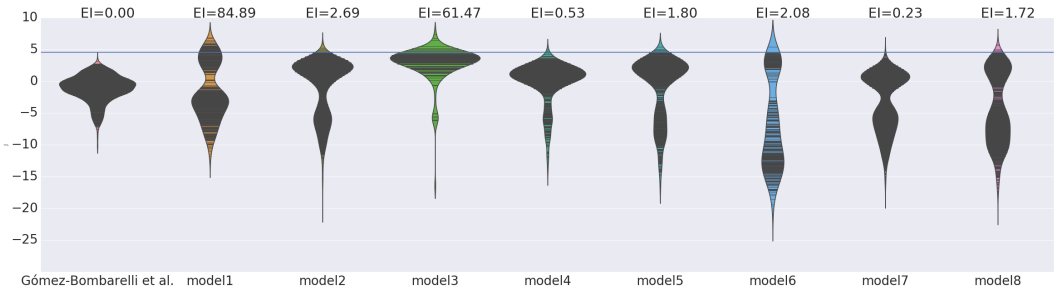


Figure 2: Violin plots showing the distributions of J(eq.3) of the molecules generated by the top 8 of our models and the ones generated by the model proposed in (Gómez-Bombarelli et al., 2016a). The blue horizontal line corresponds to the highest value of J in the training data, $J^*(D) = 4.52$. We report the expected improvement $EI * 10^3$(eq.2) of each model. We can see that our models exhibit more diversity than the baseline(Gómez-Bombarelli et al., 2016a), this allows them to find molecules with better scores.

in the training set. In a stochastic optimization loop generating a lot of suboptimal candidates is not necessarily a problem since the selection operator can get rid of them. On the other hand, generating a significant number of high-value candidates (above the current best) accelerates the optimization.

REFERENCES

Guillaume Alain and Yoshua Bengio. What regularized auto-encoders learn from the data-generating distribution. *Journal of Machine Learning Research*, 15(1):3563–3593, 2014.

Yoshua Bengio, Li Yao, Guillaume Alain, and Pascal Vincent. Generalized denoising auto-encoders as generative models. In *Advances in Neural Information Processing Systems*, pp. 899–907, 2013.

Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*, 2015.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

François Chollet. Keras. https://github.com/fchollet/keras, 2015.

Fabian Dey and Amedeo Caflisch. Fragment-based de novo ligand design by multiobjective evolutionary optimization. *Journal of chemical information and modeling*, 48(3):679–690, 2008.

Dominique Douguet, Hélène Munier-Lehmann, Gilles Labesse, and Sylvie Pochet. Lea3d: a computer-aided ligand design for structure-based drug design. *Journal of medicinal chemistry*, 48(7):2457–2468, 2005.

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

Uli Fechner and Gisbert Schneider. Flux (1): a virtual synthesis scheme for fragment-based de novo design. *Journal of chemical information and modeling*, 46(2):699–707, 2006.

Valerie Gillet, A Peter Johnson, Pauline Mata, Sandor Sike, and Philip Williams. Sprout: a program for structure generation. *Journal of computer-aided molecular design*, 7(2):127–153, 1993.

Rafael Gómez-Bombarelli, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, David Duvenaud, Dougal Maclaurin, Martin A Blood-Forsythe, Hyun Sik Chae, Markus Einzinger, Dong-Gwang Ha, Tony Wu, et al. Design of efficient molecular organic light-emitting diodes by a high-throughput virtual screening and experimental approach. *Nature Materials*, 15(10):1120–1127, 2016a.

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*, 2016b.

Markus Hartenfeller and Gisbert Schneider. Enabling future drug discovery by de novo design. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 1(5):742–759, 2011.

Markus Hartenfeller, Heiko Zettl, Miriam Walter, Matthias Rupp, Felix Reisen, Ewgenij Proschak, Sascha Weggen, Holger Stark, and Gisbert Schneider. Dogs: reaction-driven de novo design of bioactive compounds. *PLoS Comput Biol*, 8(2):e1002380, 2012.

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

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.

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.

Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

Peter S Kutchukian and Eugene I Shakhnovich. De novo design: balancing novelty and confined chemical space. *Expert opinion on drug discovery*, 5(8):789–812, 2010.

Alex M Lamb, Anirudh Goyal ALIAS PARTH GOYAL, Ying Zhang, Saizheng Zhang, Aaron C Courville, and Yoshua Bengio. Professor forcing: A new algorithm for training recurrent networks. In *Advances In Neural Information Processing Systems*, pp. 4601–4609, 2016.

Greg Landrum. Rdkit: Open-source cheminformatics. *Online). http://www. rdkit. org. Accessed*, 3 (04):2012, 2006.

Yan Li, Yuan Zhao, Zhihai Liu, and Renxiao Wang. Automatic tailoring and transplanting: a practical method that makes virtual screening more useful, 2011.

Yan Li, Zhixiong Zhao, Zhihai Liu, Minyi Su, and Renxiao Wang. Autot&t v. 2: An efficient and versatile tool for lead structure generation and optimization. *Journal of chemical information and modeling*, 56(2):435–453, 2016.

Brian B Masek, David S Baker, Roman J Dorfman, Karen DuBrucq, Victoria C Francis, Stephan Nagy, Bree L Richey, and Farhad Soltanshahi. Multistep reaction based de novo drug design: Generating synthetically feasible design ideas. *Journal of chemical information and modeling*, 56 (4):605–620, 2016.

J Močkus. On bayesian methods for seeking the extremum. In *Optimization Techniques IFIP Technical Conference*, pp. 400–404. Springer, 1975.

Albert C Pierce, Govinda Rao, and Guy W Bemis. Breed: Generating novel inhibitors through hybridization of known ligands. application to cdk2, p38, and hiv protease. *Journal of medicinal chemistry*, 47(11):2768–2775, 2004.

Gisbert Schneider. De novo design–hop (p) ing against hope. *Drug Discovery Today: Technologies*, 10(4):e453–e460, 2013.

Petra Schneider and Gisbert Schneider. De novo design at the edge of chaos: Miniperspective. *Journal of medicinal chemistry*, 59(9):4077–4086, 2016.

Marwin HS Segler, Thierry Kogej, Christian Tyrchan, and Mark P Waller. Generating focussed molecule libraries for drug discovery with recurrent neural networks. *arXiv preprint arXiv:1701.01329*, 2017.

Teague Sterling and John J Irwin. Zinc 15-ligand discovery for everyone. 2015.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pp. 3104–3112, 2014.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pp. 1096–1103. ACM, 2008.

Renxiao Wang, Ying Gao, and Luhua Lai. Ligbuilder: a multi-purpose program for structure-based drug design. *Molecular modeling annual*, 6(7-8):498–516, 2000.

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

David White and Richard C Wilson. Generative models for chemical structures. *Journal of chemical information and modeling*, 50(7):1257–1274, 2010.

Scott A Wildman and Gordon M Crippen. Prediction of physicochemical parameters by atomic contributions. *Journal of chemical information and computer sciences*, 39(5):868–873, 1999.

Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

A DETAILS ABOUT THE EXPERIMENTS

A.1 PREPROCESSING

We preprocessed the SMILES strings as the following. A special begin character and end character are added respectively in the beginning and the end of the strings. As in (Gómez-Bombarelli et al., 2016b), the maximum length of the strings were 120, we thus padded all the strings after the end character with a special zero character so that each string had a length of 120. We converted the strings into onehot representation which was the input to the models.

A.2 OBJECTIVE FUNCTION

The objective function we optimize is :

$$J(m) = LogP(m) - SA(m) - ring\text{-}penalty(m) \tag{3}$$

where $LogP$ is the partition coefficient(Wildman & Crippen, 1999)[3], SA is the synthetic accessibility(Ertl & Schuffenhauer, 2009)[4] and ring-penalty is defined as : $ring\text{-}penalty(m) = max\text{-}ring\text{-}length(m)$ if $max\text{-}ring\text{-}length(m) > 6$ else 0 [5], where $max\text{-}ring\text{-}length(m)$ of a molecule is the maximum ring length among the rings contained in the molecule. We standardize each term ($LogP$, SA and $ring\text{-}penalty$) by substracting the mean and dividing by the standard deviation (both computed on the training set).

A.3 MODELS

We used two kinds of architectures, Char-RNNs and sequential autoencoders. We provide below details about the architectures, the prior of the hyperparameters and how the generation is done. To train a new model, we sample from the prior of the hyperparameters and we train the model. After the model is trained, we generate samples from it, filter the duplicates and the non valid strings(using RDKIT (Landrum, 2006)), then compute the scores provided in eq.3 (using RDKIT (Landrum, 2006)) and eq.2. We used keras (Chollet, 2015) in our experiments.

A.3.1 CHAR-RNNS

As in (Karpathy et al., 2015), we used a stack of RNN layers that predict the character at time step t then feed back in the next time step. The number of layers were between 1 and 5. We used a dropout rate in each layer, where the dropout rate could be either 0 (disabled) or 0.1 or 0.3 or 0.5. The size of the hidden state in each layer was selected from $\{100, 200, 300, ..., 1000\}$. The type of the parametrization of the RNN could either be a Vanilla RNN (Karpathy et al., 2015), a GRU (Cho et al., 2014) or an LSTM (Hochreiter & Schmidhuber, 1997).

For generation, we initialize the string by the begin character, predict the next character probabilities, sample then feed back the sampled character as an input to the next timestep. We repeat the process until the end character is generated.

A.3.2 SEQUENTIAL AUTOENCODERS

We used sequential autoencoders as in (Bowman et al., 2015; Gómez-Bombarelli et al., 2016a) but without the variational objective and without teacher forcing (Williams & Zipser, 1989; Lamb et al., 2016). The advantage of teacher forcing is that it makes training much easier, on the other hand, it makes the model rely less on the fixed-length representation computed from the input sequence and more on the input tokens fed to the decoder at each time step. The encoder was either a stack of 1d convolutions like (Gómez-Bombarelli et al., 2016a) or a stack of RNNs.

For convolutional encoders, the number of filters and the size of the filters were respectively selected from $\{8, 16, 32, 64, 128\}$ and $\{3, 5, 7, 9, 11\}$. The number of encoder layers was between 1 and 5.

[3] We used RDKIT(Landrum, 2006) to compute the partition coefficient.

[4] We used the code here from RDKIT(Landrum, 2006) to calculate synthetic accessibility.

[5] Personal communication with Rafael Gómez-Bombarelli and José Miguel Hernández-Lobato.

We used a fully connected layer with a linear activation after the last convolution. The number of hidden units in that layer was selected from $\{100, 200, 300, ..., 1000\}$. The fully connected layer was used to condition the decoder, which was a stack of RNN layers where the number of hidden layers was between 1 and 5. The size of the hidden state of the decoder layers was selected from $\{100, 200, 300, ..., 1000\}$. We used the 'relu' activation in all convolutional layers.

For RNN encoders, the number of encoder layers were between 1 and 5. The size of the hidden state in each layer was selected from $\{100, 200, 300, ..., 1000\}$. We use the hidden state in the last time step of the last RNN of the encoder to condition the decoder. The decoder part was identical to sequential autoencoders with a convolutional encoder, defined above.

Sequential autoencoders were trained to reconstruct the input sequence from a noisified version of the input, this makes them a kind of denoising autoencoders (Vincent et al., 2008).

Noise was applied for each input character independently : with a probability p the i-th character was replaced with a random character uniformly from the vocabulary. Noise probability was selected from $\{0, 0.1, 0.2, 0.3, 0.5\}$.

For generation, inspired by (Bengio et al., 2013), we use an iterative generation procedure. We start by a completely random string. In each iteration, we apply the same noise procedure we used during training (with the same noise probability), then we reconstruct and feed the reconstructed input again to the autoencoder. We repeat this process for 100 iterations and save all the generated sequences along the chain. The advantage we see with this kind of generation is that contrary to Char-RNNs or variational autoencoders where the generation is one-shot, this generation procedure allows a form of iterative refinement of the string because the repeated application of the autoencoder reconstruction function goes towards regions with high probability density (Bengio et al., 2013; Alain & Bengio, 2014).