https://www.scirp.org/journal/jbise

J. Biomedical Science and Engineering, 2021, Vol. 14, (No. 7), pp: 305-315

Inverse Molecule Design with Invertible Neural Networks as Generative Models

Wei Hu

Department of Computer Science, Houghton College, Houghton, USA

Correspondence to: Wei Hu, wei.hu@houghton.edu

Keywords: Inverse Molecule Design, Invertible Neural Networks, Normalizing Flows

Received: July 1, 2021 **Accepted:** July 27, 2021 **Published:** July 30, 2021

ABSTRACT

Using neural networks for supervised learning means learning a function that maps input *x* to output *y*. However, in many applications, the inverse learning is also wanted, *i.e.*, inferring *y* from *x*, which requires invertibility of the learning. Since the dimension of input is usually much higher than that of the output, there is information loss in the forward learning from input to output. Thus, creating invertible neural networks is a difficult task. However, recent development of invertible learning techniques such as normalizing flows has made invertible neural networks a reality. In this work, we applied flow-based invertible neural networks as generative models to inverse molecule design. In this context, the forward learning is to predict chemical properties given a molecule, and the inverse learning is to infer the molecules given the chemical properties. Trained on 100 and 1000 molecules, respectively, from a benchmark dataset QM9, our model identified novel molecules that had chemical property values well exceeding the limits of the training molecules as well as the limits of the whole QM9 of 133,885 molecules, moreover our generative model could easily sample many molecules (*x* values) from any one chemical property value (*y* value). Compared with the previous method in the literature that could only optimize one molecule for one chemical property value at a time, our model could be trained once and then be sampled any multiple times and for any chemical property values without the need of retraining. This advantage comes from treating inverse molecule design as an inverse regression problem. In summary, our main contributions were two: 1) our model could generalize well from the training data and was very data efficient, 2) our model could learn bidirectional correspondence between molecules and their chemical properties, thereby offering the ability to sample any number of molecules from any *y* values. In conclusion, our findings revealed the efficiency and effectiveness of using invertible neural networks as generative models in inverse molecule design.

1. INTRODUCTION

Machine learning can be divided into three major categories: supervised learning, unsupervised learning, and reinforcement learning. All these three classes of learning have found successful applications in molecule design. In the setting of supervised learning, the model learns a function that maps input x to output y, where x represents the features of a molecule and y the chemical properties of the molecule. The learning from input to output is forward learning and from output to input is inverse learning, which usually is much harder than the forward learning. Because the dimension of input is usually much higher than that of the output, there is a possibility of information loss when the information flows from a higher dimensional space to a lower one. Therefore, inverse learning problems are often both intractable and ill-posed. Nonetheless, inverse learning is much needed in cheminformatics as it is very important to infer molecules from their chemical properties [1].

Recent advances in machine learning such as normalizing flows has made it possible to create invertible neural networks by imposing extra constraints such as restrictive to the architecture of the networks or adding a normalization step during training [2]. Since their recent introduction, invertible neural networks have shown convincing performance on discriminative and generative learning tasks, which opens up many exciting new avenues for research. A normalizing flow is a sequence of bijective transformations for a random variable such that its distribution evolves from a simple distribution to a more complex one. Invertible neural networks fulfill a long-time dream of being interpretable from deep learning community. It could be expected that with the introduction of invertible networks, many of the current problems could be revisited from a completely different perspective on the same problem. This was the reason and motivation for us to start the work reported in this paper.

Creating molecules of desired chemical properties is of great importance in science as it can guide and speed up the development of new active compounds. In this paper, we proposed to apply invertible neural networks to the task of inverse molecule design, which was inspired by the work in [1] where the main idea was to tune the input (parametrized molecules) for a target value of a chemical property while keeping the weights of the neural network frozen after the forwarding learning is finished. The method in [1] could only optimize one molecule for one target value at a time because their network was not invertible. However, our work tried to investigate the bidirectional correspondence between molecules and their chemical properties by treating inverse molecule design as an inverse regression problem. With completely different approaches to the same problem, our expectation was to uncover an array of benefits of using invertible neural networks as generative models in inverse molecule design.

Our current work could be viewed as a continuation of our earlier work [3], in which we employed multi-agent reinforcement learning to molecule design. During one episode of its training, a single agent can only learn how to move forward based on the experience gained thus far. In the case of molecule design, this means the single agent can only improve the future sites of a molecule, but cannot correct or improve the sites it already has passed during one episode of training. However, in a multi-agent setting, a group of agents can improve their work on all sites of a molecule simultaneously, so the learning can be bidirectional: both forward and backward learning, and furthermore, concurrent learning of all sites. Our findings highlighted that multi-agent approach increases learning significantly compared to single-agent, which could be attributed to the synergy of team work of a group of agents and a more focused individual responsibility of each agent in the group [3].

2. METHODS

2.1 Normalizing Flows

Normalizing flows use invertible network structures to create bijective flows between a simple probability distribution and a very complex distribution, offering a new class of exact likelihood based generative models.

The whole idea of this new concept begins with change of variables: let Z and X be random variables

which are related by a bijective mapping $f: R^n \to R^n$ such that $x = f(z)$ and $z = f^{-1}(x)$. Then

$$p_X(x) = p_Z\left(f^{-1}(x)\right)\left|\det\left(\frac{\partial f^{-1}(x)}{\partial x}\right)\right| = p_Z(z)\left|\det\left(\frac{\partial f}{\partial z}\right)\right|^{-1} \tag{1}$$

$$\log p_X(x) = \log p_Z(z) - \log\left|\det\left(\frac{\partial f}{\partial z}\right)\right| \tag{2}$$

where $\det\left(\frac{\partial f}{\partial z}\right)$ is the determinant of Jacobian of f at x, which measures the local linear expansion or shrinkage around x. Therefore, the transform is volume preserving if the determinant equals unity. In general, the cost of computing a determinant is $O(n^3)$, which can be computationally expensive for high-dimensional distributions. Therefore, efficient calculation of the determinant is critical for practical use of this method. One common technique to achieve this goal is to design the transformation so that the Jacobian is an upper or a lower triangular matrix. Equation (2) offers a means to maximize the likelihood via change of variables. In contrast to directly parameterizing a probability distribution, the change of variables approach allows for the creation of a complex distribution implicitly by transforming a simple base distribution.

Hence, we can evaluate the exact log-likelihood as long as the prior distribution $\log p_Z(z)$ is tractable and the determinant of the Jacobian can be efficiently computed. It is clear that the design of these transformations has to balance the expressiveness and the computational cost of the determinant of the Jacobian. Also, notice that x and z have to be of the same dimension to make the bijection between them possible.

If we have K such bijection f, a flow is created through a sequential composition of these functions: $x = f_K \circ \cdots \circ f_2 \circ f_1(z)$ which is able to transform $p(z)$ from a simple distribution to a more complex one $p(x)$. $\log p_X(x)$ represents the exact log-likelihood of input data x, and the learning of these bijections is through minimizing the negative log-likelihood loss function. After learning, new samples x can be generated from the latent variables z according to the prior distribution $p(z)$ through the flow.

We introduce two popular flow models, nonlinear independent components estimation (NICE) [4] and real non-volume preserving (RealNVP) [5] as they will be utilized in our work. The core techniques of these methods are dimension partitioning and coupling layers, which transform one part of the input at a time and allow for both analytic forward and inverse mappings.

The NICE is a transformation that uses additive coupling layers. The coupling layer in NICE partitions z into two disjoints subsets, z_1 and z_2. Then it applies the following transformation:

Forward mapping $z \to x$:

$$\begin{cases} x_1 = z_1 \\ x_2 = z_2 + m(z_1) \end{cases} \tag{3}$$

Inverse mapping $x \to z$:

$$\begin{cases} z_1 = x_1 \\ z_2 = x_2 - m(x_1) \end{cases} \tag{4}$$

where m is a neural network. This defines a volume preserving transformation since the determinant of the forward mapping is 1.

RealNVP adds scaling factors to the transformation:

$$x_2 = \exp\left(s(z_1)\right) \odot z_2 + m(z_1) \tag{5}$$

where $\odot$ denotes elementwise product and s is a neural network. This results in a non-volume preserving transformation. The Jacobian of RealNVP is triangular and Jacobian diagonals are strictly positive, and the

Jacobian of NICE is triangular and Jacobian diagonals are all 1s. The major challenges of create normalizing flows are flexible invertible structures and efficient computation of log-determinant.

In addition to the flows based on dimension partitioning, there are other kinds of flows such as those based on identities of determinants, which make use of some identities to speed up the computation of determinants if the Jacobians have special structures. Some flows are based on autoregressive transformations so the Jacobian can be made triangular. Plus, flows can also be free-form, *i.e.*, whose Jacobians are not limited by special structures [6].

2.2. Invertible Neural Networks

In a typical neural network structure for supervised learning, there are two major sequential stages in the mapping of input x to output y. The first is made of layers that process the input data, *i.e.*, extracting useful features, and the second is to change the features into a decision on classification or regression. The convolutional layers and recurrent layers are key players in the first stage and the softmax layers are commonly used in the second stage when a probability distribution is needed as output. Usually the input data dimension is much higher than the output dimension. To make the information flow bijective in a network, it is easy to replace the first stage with different normalizing flows or other invertible structures. But to make the second stage invertible, it is necessary to introduce extra technique to capture the information about x that is not contained in y. For example, adding an extra variable z, so x could be inferred given [y, z] [7]. Good design of invertible neural networks entails that both inverting the network and computing the Jacobian determinant be efficient. The invertible neural network used in this work consisted of three layers. The first is RealNVP, the second is NICE, and the third is for regression.

3. RESULTS

The chemical property studied in this work was the logarithm of partition coefficient (log$P$) of a molecule, a measurement of its lipophilic efficiency, which is one of the indicators of drug-likeness. In [1], a neural network was trained to predict log$P$ values, then the weights of this network were frozen, and the inverse training optimized the parametrized molecule for a log$P$ value. This method had the limitation of training the model for one molecule and one log$P$ value at a time. If a different log$P$ value is given to infer the corresponding molecule, the model had to be retrained. In contrast, we took a different approach to the same problem: we treated it as an inverse regression problem. Our method could train a generative model on a dataset of molecules and all their log$P$ values at one time. Once our model was trained, it could be used to generate any number of molecules for any log$P$ values without the need of retraining. This section demonstrated the simplicity and effectiveness of using invertible neural networks as generative models in inverse molecule design.

3.1. Training Invertible Neural Networks on 100 Molecules

It is well-known that training deep neural networks requires considerably large training datasets as seen in [1], which used 10,000 molecules from QM9 dataset [8] to study the shift in distribution of log$P$ values. We first highlighted the data efficiency of our approach by generating new molecules with a very tiny dataset of 100 molecules randomly selected from QM9 dataset. Our goal was to show that trained on such a small dataset, our generative model was able to discover molecules to have log$P$ values that exceeded the limits of our dataset of size 100 and even the limits of the whole QM9 dataset of 133,885 molecules.

Our dataset of 100 molecules had an average log$P$ of 0.259, maximum of log$P$ of 3.14, and minimum of −2.12 (**Figure 1**). For reference, QM9 dataset has average log$P$ = 0.36, maximum log$P$ = 3.22, and minimum log$P$ = −2.12. log$P$ was calculated using software RDKit.

Prior to training our model, each molecule was stored in a SMILES string in QM9 [9], and then was converted into a SELFIES string [9], then to a one hot encoding binary string. This one hot encoding string was used as input x (molecule) to train our invertible neural network to predict output y (log$P$) values.

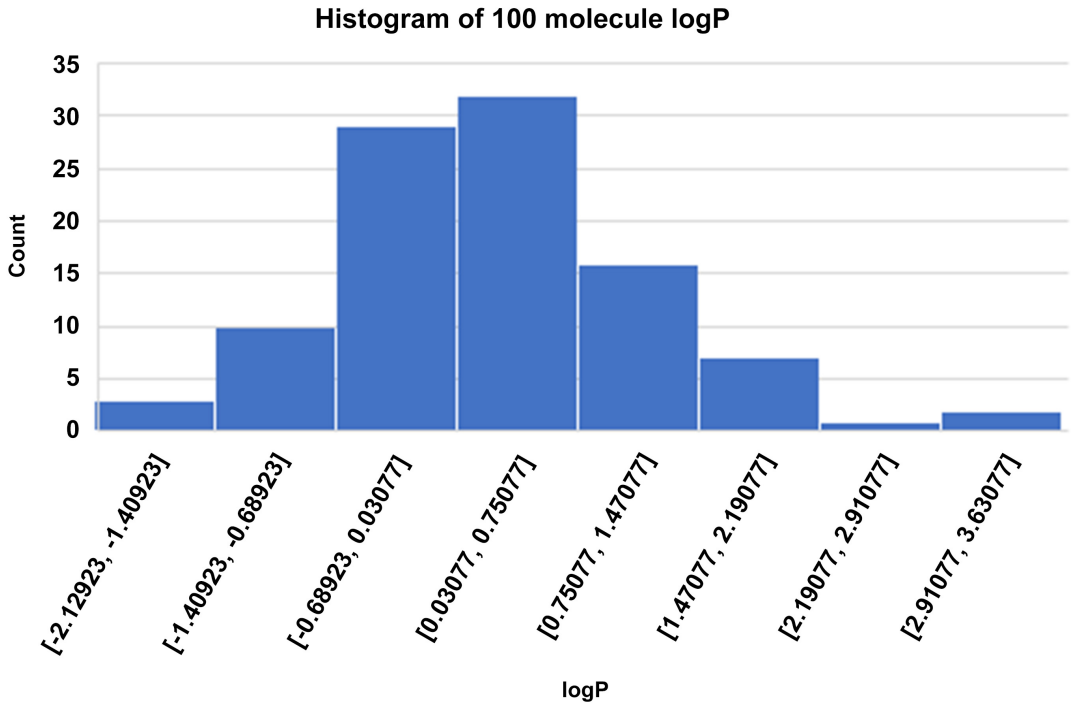

Figure 1. Actual log$P$ value distribution of the 100 molecules used in the training of our model, with average log$P$ = 0.259, maximum of log$P$ = 3.14, and minimum = −2.12.

Our invertible neural network had two methods, one called forward for forward learning and the other inverse for inverse learning. For input x, the output y = model.forward(x), then $\tilde{x}$ = model.inverse(y). Our model was trained with the loss function defined as the squared difference of log$P$ and predicted log$P$, and of x and its inverse image $\tilde{x}$, simultaneously optimizing both the input and output domains.

$$\text{loss} = \left\|\log P - \text{pred\_}\log P\right\|^2 + \left\|x - \tilde{x}\right\|^2 \tag{6}$$

This loss function implied the training of our model was bidirectional: forward and backward (inverse) (Figure 2).

Since $\tilde{x}$ was a vector of rea numbers, we had to convert it to a one-hot encoding vector, denoted by $\tilde{x}_{hot}$, by setting value to 1 at the position of maximum value in $\tilde{x}$, and all other positions to have value of 0. The logP, SMILES and SELFIES representations in Table 1 and Table 2 were based on $\tilde{x}_{hot}$.

As observed in [1], our model recognized the presence of carbon atom in a molecule as an important indicator of a high log$P$ value and the nitrogen atom as an indicator of a low log$P$ value. Next, we displayed the molecular graphs of the two novel molecules uncovered by our model, with one of extreme positive log$P$ value and one of extreme negative log$P$ value (Figure 3 and Figure 4).

3.2. Training Invertible Neural Networks on 1000 Molecules

To further elucidate the advantage of using invertible neural networks as generative models, we chose a dataset of 1000 molecules from QM9 to train our model using the same loss function as in Section 3.1 (Figure 5). Section 3.1 reported the molecules generated for one time by our model from one y value. However, in this section our model generated 1000 molecules from one y value, so we could calculate the average and standard deviation of their log$P$ values, thus enabling us to show the distribution of these predicted log$P$ values from an array of y values (hypothetical log$P$) (Table 3 and Figure 6). The average log$P$

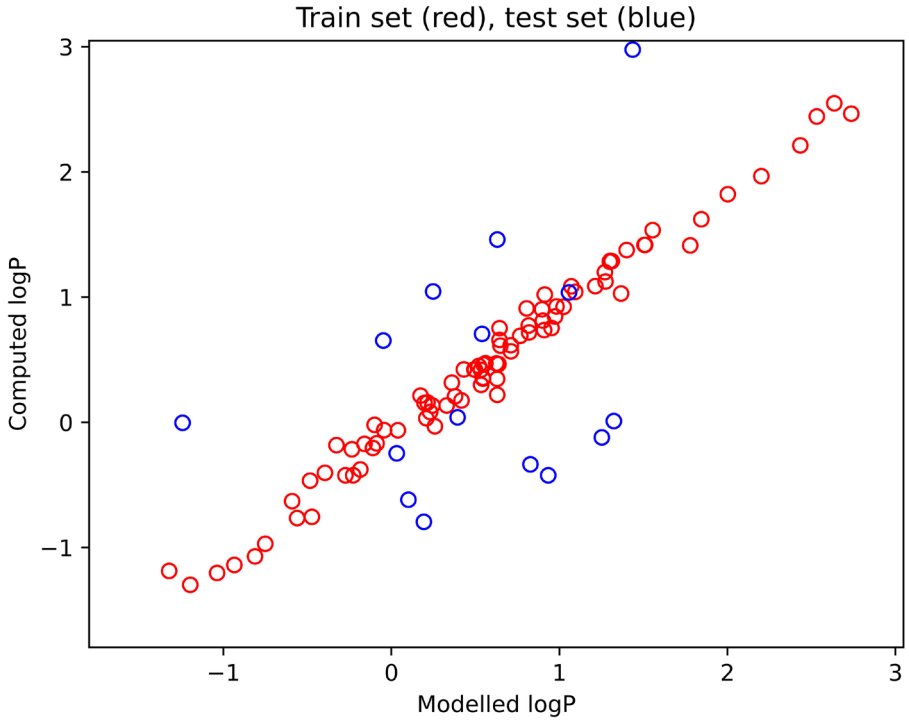

Figure 2. X-axis represents the predicted log$P$ of the 100 molecules by our model and y-axis represents their true log$P$. We split the 100 molecules into two subsets, 85 molecules for training and 15 for test. This plot shows the quality of forward learning of our model.

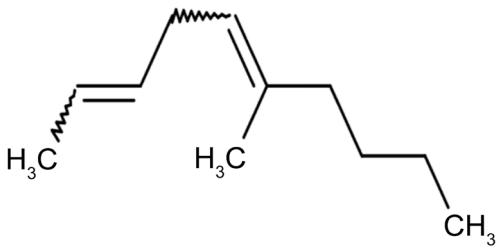

Figure 3. Graph of a novel molecule in table 1 with y value = 5.0 and log$P$ = 4.08, SMILES = CCCCC(C)=CCC=CC, and SELFIES = [#C][C][C][C][C][Branch2_1][C][=O][C][#C][C][C][=C][C][Branch2_3][=O][Ring1][Branch1_3][F].

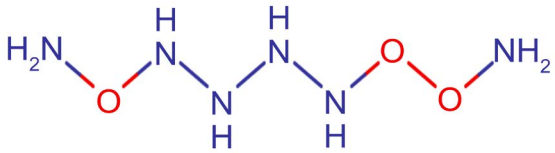

Figure 4. Graph of a novel molecule in table 2 with y value = −3.3 and log$P$ = −3.36, SMILES = O(NNNNON)ON, SELFIES = [Branch1_2][Branch1_1][=O][Branch2_1][C][O][=N][Branch1_3][Ring1][Branch1_1][Branch2_2][#N][N][N][O][=N][O][=N].

Table 1. This table presents the novel molecules sampled individually from one positive y value by our model. The table contains log$P$ of each molecule, along with its SMILES and SELFIES strings.

y value	3.3	3.3	3.3
log$P$ of $\tilde{x}_{hot}$	3.48	3.59	4.00
SMILES	CCCC(=C)CCCC(F)	CCCCCCC=C=CF	CCCCC(C)CCCC
SELFIES	[#C][C][C][C][Branch1_3][Ring1][=C][Branch1_2][=C][C][C][C][Branch2_1][O][O][F][#N][Branch1_2]	[#C][C][C][C][Branch1_3][#N][Branch1_1][C][Branch1_2][C][Branch2_1][#C][C][C][Ring1][F][=C][F][N][Branch1_2]	[#C][C][C][C][C][Branch1_2][=O][C][Branch1_2][C][C][C][C][Branch2_3][N][Ring2]
y value	4.0	4.0	4.0
log$P$ of $\tilde{x}_{hot}$	3.67	3.77	4.00
SMILES	CCCCCCC(OCCC)F	CCCC=C(C)C=CC=CF	CCCCCC(C)CC#CC(C)
SELFIES	[C][C][C][C][C][C][C][Branch1_2][Branch1_1][O][C][C][C][F][O][N][N]	[C][C][C][C][=C][Branch1_2][=O][C][Branch2_1][C][=C][C][=C][F][Ring1][Ring2]	[#C][C][C][C][C][C][Branch1_1][=O][C][C][C][#C][C][Branch1_2][F][C][Ring2]
y value	5.0	5.0	5.0
log$P$ of $\tilde{x}_{hot}$	3.75	3.99	4.08
SMILES	C1CCCC(=C)CCC=C1F	CCC=CCCCCC=CF	CCCCC(C)=CCC=CC
SELFIES	[#C][C][C][C][C][Branch1_2][C][#C][Branch2_3][C][C][C][=C][Ring2][F][N][F]	[#C][C][C][C][Ring1][#N][C][C][C][C][C][Branch2_1][C][#N][Branch1_2][#C][F][C]	[#C][C][C][C][C][Branch2_1][C][=O][C][#C][C][C][=C][C][Branch2_3][=O][Ring1][Branch1_3][F]
y value	6.0	6.0	6.0
log$P$ of $\tilde{x}_{hot}$	3.60	3.85	3.87
SMILES	CCCC(=C)CCC=CF	CCC(CCC=C(CC=C=C))	FCCCC#CCCCC=CCC
SELFIES	[#C][C][C][C][Branch1_3][#N][#C][Ring1][C][C][C][C][=C][F][Branch1_3][Branch1_2][F]	[#C][C][C][Branch2_3][Ring1][Ring1][C][C][C][=C][Branch2_2][Branch2_1][Branch2_1][=C][C][=C][#C][Branch1_2][C]	[F][C][C][C][C][#C][=C][Ring1][O][C][C][C][Branch1_1][#N][Branch2_1][#C][C][C][Branch2_1][Ring2]

Table 2. This table presents the novel molecules sampled individually from one negative y value by our model. The table contains log$P$ of each molecule, along with its SMILES and SELFIES strings.

y value	−3.3	−3.3	−3.3
log$P$ of $\tilde{x}_{hot}$	−3.13	−3.16	−3.36
SMILES	N(ON(O))NNNC(O)	N(N(NN(N)))N	O(NNNNON)ON

Continued

SELFIES	[=N][Branch1_3][Branch1_2][O][=N][Branch1_1][=N][O][Branch2_3][=N][N][N][C][Branch2_3][Branch2_3][=N][O][Branch1_2]	[=N][Branch1_3][N][Branch1_1][#C][=N][Branch1_1][Branch1_3][N][N][Branch1_3][Branch2_2][=N][Branch2_3][N][Branch2_2][Branch1_1][Branch1_3][Branch1_3][Branch2_1]	[Branch1_2][Branch1_1][=O][Branch2_1][C][O][=N][Branch1_3][Ring1][Branch1_1][Branch2_2][#N][N][N][O][=N][O][=N]
y value	−4.0	−4.0	−4.0
log$P$ of $\tilde{x}_{hot}$	−2.41	−2.91	−3.28
SMILES	NC(C#CN(N))ON	NC(OOC(O))=NNNN	NN(OOON)NN(N)
SELFIES	[=N][C][Branch1_2][Branch2_3][C][#C][Branch2_2][Branch2_3][#N][Branch1_3][O][Branch1_1][=N][Branch2_1][=O][Branch2_2][Branch1_3][=N][Branch1_2][Branch2_3]	[N][C][Branch1_1][Branch2_1][O][O][=C][Branch2_2][Branch2_1][#C][O][#N][Branch1_2][=N][N][N][Branch1_2]	[#N][Ring1][Branch1_2][N][Branch1_1][Branch2_1][O][=O][Ring2][Branch1_1][Branch2_2][=O][N][N][N][Branch1_2][Branch2_1][Branch2_2][N][Branch2_1]
y value	−5.0	−5.0	−5.0
log$P$ of $\tilde{x}_{hot}$	−2.60	−2.80	−3.08
SMILES	NOON(CN(N))C(O)	NON(NON(N))	N(N(N)NN(N))
SELFIES	[Branch2_3][Branch2_2][#N][O][O][Branch1_1][Branch1_1][N][Branch1_2][Branch1_2][=C][N][Branch1_1][Ring2][N][=C][Branch1_3][Branch2_3][O]	[=N][Ring2][Branch1_2][=N][O][Branch2_3][Branch1_1][N][Branch2_3][N][=O][Branch2_3][#N][O][Branch2_2][=N][Branch2_1][Branch2_3][Ring2][=N][Branch1_1]	[Ring2][#N][Ring2][Branch2_3][Branch1_1][Branch1_1][=C][#N][Ring1][#N][Branch1_3][=O][#N][=N][N][Branch2_3][=N][Branch1_2][=N]

Table 3. Average and standard deviation of log$P$ of 1000 collectively sampled molecules from each one y value by our model, because we treated the inverse molecule design as an inverse regression problem. Notice that the y values can be any values either positive or negative, and our model could easily find the corresponding molecules regardless. For the purpose of demonstration, we intentionally chose y values with a large range.

y value	−25	−23	−20	−17	−15	−13	−10	−7	−5	−3	0
Average log$P$	−1.09	−1.26	−1.63	−1.82	−1.92	−1.60	−1.08	−0.39	−0.16	−0.02	0.19
Std of log$P$	0.91	0.94	0.96	1.07	1.13	1.21	1.21	0.92	0.72	0.60	0.61
y value	0	3	5	7	10	13	15	17	20	23	25
Average log$P$	0.19	0.73	1.11	1.50	1.71	1.86	1.91	1.97	1.97	1.93	1.90
Std of log$P$	0.61	0.81	0.89	0.90	0.83	0.82	0.83	0.82	0.81	0.79	0.78

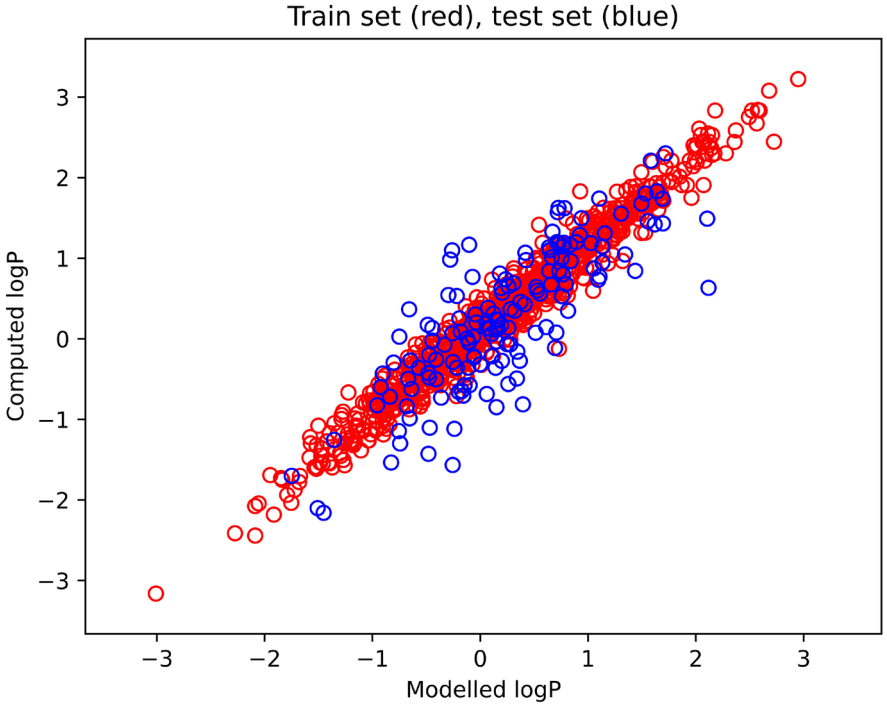

Figure 5. X-axis represents the predicted log$P$ of the 1000 molecules by our model and y-axis represents their true log$P$. We split the 1000 molecules into two subsets, 850 molecules for training and 150 for test. This plot shows the quality of forward learning of our model.

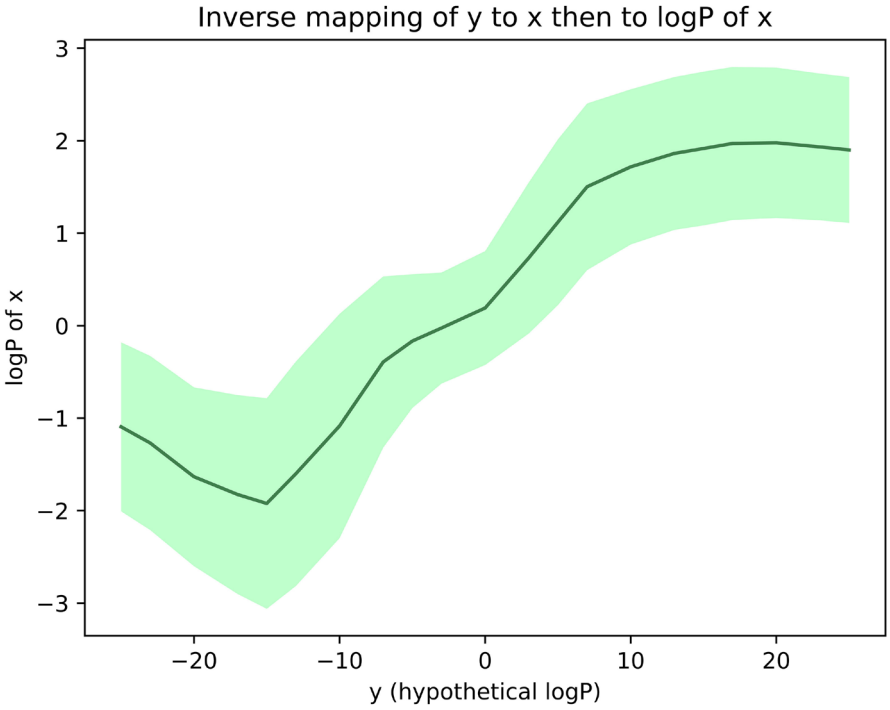

Figure 6. Plot one curve using the results in Table 3. As we did in Section 3.1, here log$P$ of x means log$P$ of $\tilde{x}_{hot}$. The mean of log$P$ of x is the curve and the standard deviation of log$P$ of x is the shaded area around the curve.

of these 1000 sampled molecules from each one y value is the curve in Figure 6 and the shaded area around the curve is their standard deviation.

The results in Table 3 and Figure 6 implied that the minimum log$P$ value occurred when y value was near −15, while the maximum log$P$ value occurred when y value was around 17 and 19, not symmetric. The inverse learning of our model provided the whole landscape of viewing molecules through any y values (hypothetical log$P$), which allowed for easy sampling of molecules given a y value, a clear advantage of our model. Figure 6 also suggested that within the range of −15 and 17, 19, the curve was monotonically increasing. At the same time, it is of particular interest to observe that our model did not produce molecules of log$P$ values that correlated with the movement of y when it went to the extreme values either in the positive or negative direction. In other words, by stretching y values too far, our model did not respond in the same manner. The curve in Figure 6 visualized the main contribution of our work as no previous models in the literature could have this ability.

In summary, the results in Sections 3.1 showed the molecules individually sampled from one y value by our model, and Section 3.2 presented the molecules collectively sampled from one y value by our model. Combined together, they illustrated the complete capability of our generative model by learning bidirectional association between molecules and their log$P$ values in the task of inverse molecule design.

4. CONCLUSIONS

Although many challenges remain, machine learning has shown its potential in discovering novel drug-likeness molecules from a virtually infinite search space, which suggests that human intelligence and artificial intelligence are both needed in the smart search of new drugs.

In supervised machine learning, the goal is to learn a function that maps input x to output y, where x usually is a feature vector and y is a label. The forward learning process is to learn a function that maps x to y. But in the design of molecules, very often we need to solve the inverse problem: given y, what is x? Because of possible information loss during the forward learning, it is a hard problem to infer x from y, a form of inverse learning. In molecule design, the forward learning is to predict a chemical property value from molecular features, and the inverse learning is to identify molecules given a chemical property value, which is close to inverse regression problems.

Recently, invertible neural networks have been proposed for inverse learning, using additional techniques to capture the information otherwise lost. This leads to discovery of many invertible architectures. Normalizing flows are such invertible mappings with exact likelihoods, therefore, they could be used as building blocks in invertible neural networks. Furthermore, they provide a new method for generative modeling, carrying the benefits of efficient likelihood calculation and data generation. In this research, we proposed to apply flow-based invertible neural networks as generative models to inverse molecule design.

We experimented with our approach to produce novel molecules of desirable chemical properties. Notably, trained with only 100 molecules randomly selected from QM9, our model could generate new molecules that had chemical property values well exceeding the limits of the training molecules as well as the limits of the whole QM9 of 133,885 molecules. Trained with 1000 molecules randomly selected from QM9, our model could easily sample many molecules from any one log$P$ value, providing a whole picture of the bidirectional correlation between molecules and their log$P$ values. Recall that the method in [1] could only optimize one molecule and one log$P$ value at a time because their network was not invertible, and their model required retraining whenever a new log$P$ value was used. Whereas our generative model once trained could sample any number of molecules for any one log$P$ value without the need of retraining. Essentially, we treated inverse molecule design as an inverse regression problem, so our model could be trained once and be sampled any multiple times for any log$P$ values.

Our work confirmed two unique features of using invertible neural networks as generative models in inverse molecule design: 1) the ability to learn from even a very tiny dataset and to generalize well beyond the training data, 2) the bidirectional learning between molecules and their chemical properties, rendering an easy and flexible way of generating novel molecules with desired chemical properties.

ACKNOWLEDGEMENTS

We are grateful to the authors in [1] who made their code public.

CONFLICTS OF INTEREST

The author declares no conflicts of interest regarding the publication of this paper.

REFERENCES

1. Shen, C., Krenn, M., Eppel, S. and Aspuru-Guzik, A. (2021) Deep Molecular Dreaming: Inverse Machine Learning for De-Novo Molecular Design and Interpretability with Surjective Representations. *Machine Learning*: *Science and Technology*, **2**, Article ID: 03LT02. https://doi.org/10.1088/2632-2153/ac09d6
2. Behrmann, J., Grathwohl, W., Chen, R.T.Q., Duvenaud, D. and Jacobsen, J.-H. (2018) Invertible Residual Networks. https://arxiv.org/abs/1811.00995
3. Hu, W. (2021) Exploring Local Chemical Space in De Novo Molecular Generation Using Multi-Agent Deep Reinforcement Learning. *Natural Science*, **13**, 412-424. https://doi.org/10.4236/ns.2021.139034
4. Dinh, L., Krueger, D. and Bengio, Y. (2015) NICE: Non-Linear Independent Components Estimation. *ICLR* 2015 *Workshop Track*, San Diego, 7-9 May 2015.
5. Dinh, L., Sohl-Dickstein, J. and Bengio, S. (2017) Density Estimation Using Real NVP. *ICLR* 2017, Palais des Congrès Neptune, Toulon, 24-26 April, 2017. https://dblp.org/db/conf/iclr/iclr2017.html
6. Song, Y., Meng, C.L. and Ermon, S. (2019) MintNet: Building Invertible Neural Networks with Masked Convolutions. 33*rd Conference on Neural Information Processing Systems* (*NeurIPS* 2019), Vancouver.
7. Ardizzone, L., Kruse, J., Wirkert, S., Rahner, D., Pellegrini, E.W., Klessen, R.S., Maier-Hein, L., Rother, C. and Köthe, U. (2018) Analyzing Inverse Problems with Invertible Neural Networks. https://arxiv.org/abs/1808.04730
8. Ramakrishnan, R., Dral, P.O., Rupp, M. and Von Lilienfeld, O.A. (2014) Quantum Chemistry Structures and Properties of 134 Kilo Molecules. *Scientific Data*, **1**. https://doi.org/10.1038/sdata.2014.22
9. Krenn, M., Haese, F., Nigam, A.K., Friederich, P. and Aspuru-Guzik, A. (2020) Self-Referencing Embedded Strings (SELFIES): A 100% Robust Molecular String Representation. *Machine Learning*: *Science and Technology*, **1**, Article ID: 045024. https://doi.org/10.1088/2632-2153/aba947