A Chance-Constrained Generative Framework for Sequence Optimization

Xianggen Liu [1 2] **Qiang Liu** [3] **Sen Song** [1] **Jian Peng** [2]

Abstract

Deep generative modeling has achieved many successes for continuous data generation, such as producing realistic images and controlling their properties (e.g., styles). However, the development of generative modeling techniques for optimizing discrete data, such as sequences or strings, still lags behind largely due to the challenges in modeling complex and long-range constraints, including both syntax and semantics, in discrete structures. In this paper, we formulate the sequence optimization task as a chance-constrained optimization problem. The key idea is to enforce a high probability of generating valid sequences and also optimize the property of interest. We propose a novel minimax algorithm to simultaneously tighten a bound of the valid chance and optimize the expected property. Extensive experimental results in three domains demonstrate the superiority of our approach over the existing sequence optimization methods.

1. Introduction

The sequence optimization task aims to generate valid sequences with desirable properties, which is a fundamental problem in a variety of applications such as drug discovery (Zhou et al., 2019), code generation (Yin & Neubig, 2017), and gene analysis (Gupta & Zou, 2019). However, this problem is very challenging. On the one hand, the space is discrete by nature. The non-differentiability hinders back-propagate gradients to guide the learning of generators, which is effective in the continuous representation modeling (Arjovsky et al., 2017); On the other hand, the generated sequences should be subject to some complex and long-range constraints (e.g., valency conditions in molecular generation), pushing the discrete searching more difficult. For example, to generate a string representing a molecule or a mathematical expression with a desired quantitative property, we need to both ensure the validity of the generated string subject to a grammar and model the string representation so that it is predictive of the property.

Conventional approaches learn a sequence generator using the schemes of variational autoencoders (VAE) or generative adversarial networks (GAN). Gómez-Bombarelli et al. (2018) map the molecular SMILES strings (Weininger, 1988) to a latent space via VAE and employs Bayesian optimization (Jones et al., 1998) to search the sequences with desirable properties. Objective-reinforced generative adversarial networks (ORGAN, Guimaraes et al. 2017) optimizes the property of the sequence directly by making it as part of the reward function on the basis of SeqGAN (Yu et al., 2017). But the lack of syntactic restriction for the structured data makes them tend to generate invalid sequences.

To improve the validity of the generated sequences, Kusner et al. (2017) and Dai et al. (2018) impose sophisticated and task-specific grammars into the decoder of VAE. However, when coming to another type of sequences (e.g., SMART, InChI, and python code), these works require users to design new hand-crafted grammatical rules for the decoder. Furthermore, these syntax-direct generative mechanisms restrict the output space of the decoder, leading to limited learning capacity.

As a result, the current sequence optimization methods in the molecule domain achieve considerably lower performance than the methods that work on graphs (Kajino, 2019). Therefore, sequence optimization task would be largely benefited from a method that can effectively combine the objective of optimizing a property and the requirement of validity.

To this end, we present a novel *Chance-Constrained Generative Framework* (CCGF) for sequence optimization, that not only optimizes a desirable property but could also flexibly control the validity level of the generated sequences. In particular, we first formulate sequence optimization as a chance-constrained optimization problem (CCO, Charnes et al. 1958), which involves optimizing an objective (e.g., a desirable property) on the condition of meeting certain probabilistic constraints (e.g., validity requirement). Next,

[1]Laboratory for Brain and Intelligence and Department of Biomedical Engineering, Tsinghua University, Beijing, China. [2]Department of Computer Science, University of Illinois at Urbana Champaign, IL, USA. [3]Department of Computer Science, University of Texas at Austin, TX, USA. Correspondence to: Xianggen Liu <liuxg16@mails.tsinghua.edu.cn>.

Proceedings of the 37^{th} International Conference on Machine Learning, Online, PMLR 119, 2020.

we use Chernoff bound (Hagerup & Rüb, 1990) to transform the probabilistic constraint into a deterministic one, making our CCO problem tractable. Finally, we train a deep neural network to jointly tighten a bound of the valid chance and optimize the expected property.

The advantage of modeling sequence optimization as a chance-constrained sampling problem is multi-fold: (1) The CCGF framework model itself is agnostic to the specific grammars and can optimize the desirable property while guaranteeing the validity level of the sequences. (2) CCGF allows multiple forms of constraints, not limited to the validity. (3) It is capable of automatically improving the validity requirement during the training process. The initial low validity requirement enables the agent search across a large space (escaping from the realm of locality); Afterwards, the stricter validity requirement would reduce the search space and accelerate the optimization.

We first evaluate the effectiveness of our framework on various domains, including arithmetic expressions, python programs, and molecules. Experimental results show that CCGF outperforms all the sequence optimization methods among all these domains. Besides, it also achieves better performance than the graph-based optimization methods in the molecule domain. Then we design a more difficult task where the constraint requires the generated molecules to meet the Lipinski rule (Paul et al., 2010). Superior results of CCGF further confirm the advantages of our framework.

2. Related Work

In early years, sequence optimization was typically accomplished by heuristic algorithms, such as hill climbing (Edelkamp & Schroedl, 2011) and dynamic programming (Bellman, 1966; Hu et al., 2018). In natural language processing (NLP), beam search (Tillmann et al., 1997) and the Markov chain Monte Carlo method are also widely applied to text generation and optimization (Oberlander & Brew, 2000; Anderson et al., 2017; Liu et al., 2020).

Recently, more advanced methodologies have been proposed in the fields of *de novo* molecular generation and protein optimization. For example, Segler et al. (2018) utilize recurrent neural networks to generate drug-like molecules. Gómez-Bombarelli et al. (2018) employ a VAE to encode the molecule SMILES into a continuous latent space, based on which Bayesian optimization can be applied. Gupta & Zou (2019) propose a feedback-loop mechanism to iteratively improve the properties of DNA sequences produced by generative adversarial networks. However, all of these works did not consider the validity of sequences.

On the other hand, considering sequential generators often produce invalid sequences, Janz et al. (2017) learn a deep recurrent validator model, which estimates whether a partial sequence can function as the beginning of a valid sequence. But they mainly focus on facilitating the validity of other generative models instead of optimizing a property of sequences, and is not directly comparable with ours. Kusner et al. (2017) present a VAE which encodes and decodes directly to and from syntactic parse trees, ensuring the generated outputs are always valid. Dai et al. (2018) further incorporate semantic checking into the decoder of VAE. Different from these existing approaches, CCGF does not need the grammar information and instead makes the validity requirement as a constraint of the CCO problem.

CCGF is also related to nonlinear chance-constrained methods. A number of theoretical results for the nonlinear chance-constrained problems, such as the smooth approximation (Geletu et al., 2017) and the conditions of convergence (Adam & Branda, 2016), has been obtained. But most of these studies either consider a specific continuous distribution or are only suitable for a small searching space (Adam & Branda, 2019). Therefore, they are not capable of searching discrete sequences under some complex constraints. We leverage the power of deep neural networks to optimize a minimax objective derived based on the CCO formulation, serving as the first attempt to study the sequence optimization using CCO methods.

3. Main Method

In this section, we present our CCGF framework that models the sequence optimization as a chance-constrained optimization (CCO) problem. In particular, we first introduce the CCO problem and approximate its chance constraint. Next, we describe our generative model and the technique of incorporating the diversity of the generation. Finally, we summarize the training process of our framework.

3.1. Sequence Optimization by Generative Modeling

Sequence optimization involves generating sequences with desirable properties. Formally, let $G_{\boldsymbol{\theta}}(\boldsymbol{\xi})$ be a generator parametrized by $\boldsymbol{\theta} \in \mathbb{R}^d$ and $\boldsymbol{\xi} \in \mathbb{R}^e$ is a random vector. d and e are the sizes of vector dimensions. Given a function f that measures the property of sequences, $G_{\boldsymbol{\theta}}$ is trained to produce sequences X that make $f(X)$ as high as possible. The objective can be expressed by

$$\max_{\boldsymbol{\theta}} \ \mathbb{E}_{\boldsymbol{\xi} \sim N(0,1)}[f(G_{\boldsymbol{\theta}}(\boldsymbol{\xi}))]. \tag{1}$$

Generative adversarial network (GAN) is a popular training framework for the sequence optimization. As in Guimaraes et al. (2017), $G_{\boldsymbol{\theta}}$ is trained to jointly maximize the property expectation and the discriminative accuracy. However, the sequence data are usually subject to some complex syntactic and semantic constraints, which are hard to be encoded in such models. Therefore, we introduce a novel chance-constrained optimization framework to handle them.

3.2. Chance-Constrained Optimization

The chance-constrained Optimization (CCO) problem was first introduced by Charnes et al. (1958) as an optimization task under uncertainty. Its solution is usually defined as

$$\boldsymbol{\theta}^* := \text{argmax}_{\boldsymbol{\theta}\in S}\ \hat{f}(\boldsymbol{\theta}), \quad s.t.\ P(\hat{g}(\boldsymbol{\theta},\boldsymbol{\xi}) \leq T) \leq \epsilon, \tag{2}$$

where $\hat{f}: \mathbb{R}^d \to \mathbb{R}$ and $\hat{g}: \mathbb{R}^d \times \mathbb{R}^e \to \mathbb{R}$ are two functions. For each $\boldsymbol{\theta} \in \mathbb{R}^d$, $P(\hat{g}(\boldsymbol{\theta},\boldsymbol{\xi}) \leq T)$ stands for the probability of the event $\hat{g}(\boldsymbol{\theta},\boldsymbol{\xi}) \leq T$. Such kind of constraints arises naturally in various applications and is called chance (or probabilistic) constraints. $\epsilon \in (0,1)$ is a user-defined acceptable level and $T \in \mathbb{R}$ is a user-defined threshold.

The above CCO problem explicitly allows small random violations of the constraints and thus is a relatively robust approach. It provides a compromise between good optimization performance and satisfaction of the random constraints.

Here, we formulate the sequence optimization task as a CCO problem to take its advantages of considering uncertainty. Thus the goal of sequence optimization can be rephrased as follows: generating some sequences with a desirable property and in the meantime ensuring the confidence of the validity meeting a certain level. To this end, the two functions (i.e., $\hat{f}$ and $\hat{g}$) in Eqn. (2) are implemented by

$$\hat{f}(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\xi}\sim N(0,1)}[f(G_{\boldsymbol{\theta}}(\boldsymbol{\xi}))], \tag{3}$$

$$\hat{g}(\boldsymbol{\theta},\boldsymbol{\xi}) = g(G_{\boldsymbol{\theta}}(\boldsymbol{\xi})), \tag{4}$$

where $g(x) \in \{0,1\}$ is a binary function indicating whether the sequence x is valid[1] or not.

Therefore, the CCO problem for sequence optimization can be expressed by

$$\max_{\boldsymbol{\theta}}\ \mathbb{E}_{\boldsymbol{\xi}\sim N(0,1)}[f(G_{\boldsymbol{\theta}}(\boldsymbol{\xi}))], \tag{5}$$

$$s.t.\ P(g(G_{\boldsymbol{\theta}}(\xi)) \leq T) \leq \epsilon. \tag{6}$$

3.3. Approximation of the Chance Constraint

As the validity function g is probably neither continuous nor convex and its analytic expression may be also unknown, the lower or upper bounds of the constraint (i.e., $P(g(G_{\boldsymbol{\theta}}(\boldsymbol{\xi})) \leq T)$) are helpful for finding tractable approximate solutions (Nemirovski & Shapiro, 2006) if we can transform the chance constraint into an expectation constraint. Here, we apply a concentration inequality to transform the chance constraint into a deterministic one. In particular, according to Chernoff bounds (Hagerup & Rüb, 1990)

$$P(\zeta \leq T) \leq \frac{\mathbb{E}[e^{-\alpha\zeta}]}{e^{-\alpha T}}, \tag{7}$$

the constraint (i.e., Eqn. (6)) can be transformed to

$$g_c(\boldsymbol{\theta},\alpha) = \epsilon - \mathbb{E}_{\boldsymbol{\xi}\sim N(0,1)}[e^{-\alpha g(G_{\boldsymbol{\theta}}(\boldsymbol{\xi}))}]/e^{-\alpha T} \geq 0. \tag{8}$$

where ζ is a random variable that is non-negative and α is a non-negative constant. If this new constraint holds, the chance constraint holds as well. To ensure that the bound is tight enough, we consider the following constraint.

$$\min_{\alpha} g_c(\boldsymbol{\theta},\alpha) \geq 0. \tag{9}$$

Applying the Lagrange multiplier method (Bertsekas, 2014), our original CCO problem can be formulated to the following optimization problem, that is,

$$\max_{\boldsymbol{\theta}}\ \mathbb{E}_{\boldsymbol{\xi}\sim N(0,1)}[f(G_{\boldsymbol{\theta}}(\boldsymbol{\xi}))] + \lambda[\min_{\alpha} g_c(\boldsymbol{\theta},\alpha)]_- \tag{10}$$

where $[x]_- = \min(0,x)$ is the function that takes the negative value of the input and λ is a nonnegative multiplier that coordinates the importance of the constraint.

Applying the Monte Carlo (MC) technique, we can approximate the objective function by averaging stochastic samples, given by

$$\max_{\boldsymbol{\theta}}\min_{\alpha} \mathcal{J}(\boldsymbol{\theta},\alpha) = \max_{\boldsymbol{\theta}} \{\frac{1}{N}\sum_{i}^{N}[f(G_{\boldsymbol{\theta}}(\boldsymbol{\xi}_i))] + \lambda[\min_{\alpha\in\mathcal{A}}(\epsilon - \frac{1}{N}\sum_{i}^{N} e^{-\alpha(g(G_{\boldsymbol{\theta}}(\boldsymbol{\xi}_i))-T)})]_-\}, \tag{11}$$

where $\epsilon \in (0,1), T \in (0,1)$ and $\lambda \in \mathbb{R}$ are constants and N is the number of samples. α is a learnable variable and $\mathcal{A} = [0,+\infty)$ is its searching range.

We train $G_{\boldsymbol{\theta}}$ to optimize the above objective $\mathcal{J}$ by the REINFORCE algorithm (Williams, 1992). α is trained directly through back-propagation since $\mathcal{J}$ is differentiable to it.

Alternative Approximation. We would like to note that there are other approaches to deal with the chance constraint. For example, there is a weaker approximate optimization by replacing the chance constraint to an expectation inequality. By removing the constraint, we obtain a linear combination of the property of interest and the validity score, which can be regarded as a weaker approximate optimization objective, given by

$$\max_{\theta}\ \frac{1}{N}\sum_{i}^{N}[f(G_{\boldsymbol{\theta}}(\boldsymbol{\xi}_i)) + \lambda_c g(G_{\boldsymbol{\theta}}(\boldsymbol{\xi}_i))],$$

where λ_c is a weight similar to λ. Since g may not be differentiable, we can apply the REINFORCE algorithm to optimize it. We refer to this approach as RL-Linear and find that our approximation outperforms it in terms of both property optimization and generation validity in the experimental section. Another possible way is the simply treat chance constraint as the expectation of the binary indicator function of the validity constraint. However, this approach is

[1] In this paper, the term "valid" means the grammar of the sequence is correct.

very hard to initialize, largely because the constraint-derived penalty terms are discrete and very few sequences satisfy the constraint at the beginning of the optimization.

3.4. Deep Generative Model for Sequence Data

There are many ways to specify generative models for sequence generation. In this work, we consider a widely used recurrent architecture described as following. The generative model $G_{\boldsymbol{\theta}}$ takes a random vector $\boldsymbol{\xi} \in \mathbb{R}^e$ as input and recurrently yields a sequence of characters $x = (c_1, \ldots, c_L)$. L is the length of the sequence. We use a recurrent neural network (RNN) as the generative model $G_{\boldsymbol{\theta}}$, which maps the character embedding representations of the sequence into a sequence of hidden states recursively. Concretely, the hidden state $\boldsymbol{h}_t$ is initialized by a nonlinear transformation of the random vector $\boldsymbol{\xi}$, computed by

$$\boldsymbol{h}_0 = \text{Tanh}(\boldsymbol{\xi} W + \boldsymbol{b}), \tag{12}$$

where $W \in \mathbb{R}^{e \times m}$ and $\boldsymbol{b} \in \mathbb{R}^m$ are learnable parameters and m is the hidden size of the RNN. We adopt GRU (Cho et al., 2014) as the update unit of the RNN, given by

$$\boldsymbol{h}_l = \text{GRU}(\boldsymbol{h}_{l-1}, \boldsymbol{c}_{l-1}),\ l = 1, 2, \ldots, L, \tag{13}$$

where $\boldsymbol{c}_l$ is the embedding representations of character c_l and c_0 is set to a special token. It should be noticed that more complex update units like LSTM (Hochreiter & Schmidhuber, 1997), LSTM with attention mechanism (Bahdanau et al., 2015), and Transformer (Vaswani et al., 2017) can also be used here. At each time step, a softmax output layer maps hidden states into the output token distribution, i.e.,

$$p(\boldsymbol{c}_l | \boldsymbol{c}_0, \ldots, \boldsymbol{c}_{l-1}) = \text{softmax}(\boldsymbol{h}_l W_s + \boldsymbol{b}_s), \tag{14}$$

where $W_s \in \mathbb{R}^{m \times k}$ and $\boldsymbol{b}_s \in \mathbb{R}^k$ are learnable parameters. k is the vocabulary size. Therefore, the entire sequence x is generated by

$$x = G_{\boldsymbol{\theta}}(\boldsymbol{\xi}) = \prod_{l=1}^{L} p(\boldsymbol{c}_l | \boldsymbol{c}_0, \ldots, \boldsymbol{c}_{l-1}) \tag{15}$$

3.5. Incorporating Diversity

Based on the objective function defined by Eqn. (11), the generated sequences by $G_{\boldsymbol{\theta}}$ will converge into a few extremely high-scored sequences but lack diversities. To encourage more divergent sequences, we extend f by taking the adversarial loss into consideration, that is,

$$f(G_{\boldsymbol{\theta}}(\xi)) = \gamma f_p(G_{\boldsymbol{\theta}}(\xi)) + D_\tau(G_{\boldsymbol{\theta}}(\xi)), \tag{16}$$

where f_p is the function that measures the property of interest. D_τ is a discriminator that estimates the probability that a sample is from the training data rather than generated. In this way, the augmented loss function will encourage generated sequences to be similar to a given set of data. We implement it using bidirectional gated recurrent units (BiGRU). τ is the set of its parameters. γ is a weight coordinating the importance of the property score.

Algorithm 1 Training of CCGF

Input: Dataset $\hat{X} = \{\hat{x}_i\}_{i=1}^N$, constants in the constraint ϵ and T, the weights λ and γ, batch size B, classification threshold T_{acc}, task-specific functions f_p, g.
Initialize θ, τ and α.
Pre-train $G_{\boldsymbol{\theta}}$ using MLE on $\hat{X}$.
repeat
  Sample random vectors $\Xi = \{\xi_j\}_{j=1}^B \sim N(0, 1)$.
  Generate $X = G_\theta(\Xi)$.
  Compute property scores $f_p(X)$ and validity $g(X)$.
  Compute the probabilities of being true samples $D_\tau(X)$.
  Update $\boldsymbol{\theta}$ using REINFORCE algorithm via Eqn. (11).
  Update α using gradient descent via Eqn. (11).
  Use current $G_{\boldsymbol{\theta}}$ to generate negative examples.
  Combine negative examples with $\hat{X}$ as the training data of D_τ.
  Update τ using gradient descent via Eqn. (17).
until mean$(D_\tau(X)) < T_{\text{acc}}$.

D_τ is trained simultaneously to maximize the accuracy of the classification between the training examples and generated samples. The objective of D_τ is given by

$$\max_\tau \sum_i^N \log D_\tau(\hat{x}_i) + \sum_i^N \log(1 - D_\tau(G_{\boldsymbol{\theta}}(\xi_i))), \tag{17}$$

where $\hat{x}_i$ is a sequence sample from training data.

3.6. CCGF Sequence Optimization

We summarize our chance-constrained framework for sequence optimization, also shown in Algorithm 1.

Given a dataset $\hat{X}$ and a corresponding task that is defined by the functions f_p and g, CCGF learns to generate different sequences with desired properties and high validity. First of all, the generative model $G_{\boldsymbol{\theta}}$ is initialized by a pre-trained model through maximum likelihood estimation (MLE) on the dataset. For each updating step, CCGF randomly samples a batch of random vectors Ξ, based on which the generator $G_{\boldsymbol{\theta}}$ generates a batch of sequences. The rewards of the generation are derived by Eqn. (11) and are used to update the parameter $\boldsymbol{\theta}$ and α. We also use the generator to produce negative examples and combine them with the positive examples in $\hat{X}$ to train the discriminative model D_τ. We repeat the above operations iteratively. It is certain that optimizing a property of sequences hurts their diversities. Therefore, we leverage the discriminative model $D\tau$ as our diversity indicator. When the classification performance of $D\tau$ is lower than a threshold T_{acc}, we stop the training

process.

Theorem 1. *For any $T \in (0,1), \epsilon \in (0,1)$ and the average validity $v \in [0,1]$ of the generation, there exists $\hat{\alpha} \in [0,+\infty)$ and once the variable $\alpha \geq \hat{\alpha}$, we have $\frac{\partial \mathcal{J}(\boldsymbol{\theta},\alpha)}{\partial \alpha} \leq 0$.* (The proof is in Appendix B.)

The above theorem shows that, once α is larger than a constant $\hat{\alpha}$, the validity requirement of CCGF (also termed as the lower bound of validity that meets the constraint, Appendix A) will monotonically increase as the optimization process proceeds. This coincides with simulated annealing (Kirkpatrick et al., 1983): the initial low validity requirement enables the searching agent to escape from the realm of locality in the beginning of learning. Afterward, the stricter validity requirement would reduce the search space and accelerate the optimization.

Initialization of α. To make Theorem 1 work throughout the optimization process, we initialize α with $\hat{\alpha}$, whose computation is provided in Appendix C.

4. Experimental Results

In this section, we evaluate the proposed CCGF framework with applications in three domains, including arithmetic expressions, python programs and molecules.

4.1. Implementation Details

CCGF is implemented based on the PyTorch library (Paszke et al., 2017). Among the CCGF framework, the generative model $G_{\boldsymbol{\theta}}$ is trained by policy gradient using the REINFORCE algorithm; the discriminative model D_{τ} is trained via gradient descent. The Adam algorithm with a learning rate of r is used to update their parameters. The SGD algorithm with a learning rate of r_{α} is used to update α.

As both the scales of properties to be optimized and the learning difficulties of individual tasks are different, the optimal hyperparameters of CCGF vary across tasks. To reduce the searching range of the hyperparameters, we first empirically determine the global values of some robust hyperparameters for all the tasks and then apply grid search procedures for the other hyperparameters in each task. Specifically, the hidden states of the discriminative model, the hidden states of the generative model and the random vector $\boldsymbol{\xi}$ are set to share the same dimension. We fix the validity threshold T to 0.5 and adjust ϵ for each task since they act the same role in the constraint (defining the validity requirement). The batch size B is set to $1,000$ because MC sampling works in a relatively large distribution but the memory of the machine is limited. We investigate influences of classification threshold T_{acc} to the performance of CCGF (Appendix D) and finally set T_{acc} to 0.1. As for the determination of the other hyperparameters (including the constant ϵ in the constraint, the two weights γ and λ in the objective function, two learning rates r and r_{α}, hidden size m of RNN), we perform grid search procedures for individual tasks. Their selected values for each task are listed in Appendix E.

Table 1. Optimization performance of individual methods on the dataset of arithmetic expressions. Following Kusner et al. (2017), the desirable property in the arithmetic expression is the similarity of the results yielded by the sampled expression and the target expression. A higher score is preferred.

Methods	1st	2nd	3nd	Average	Validity
CVAE	-0.39	-0.40	-0.40	-4.75	86%
GVAE	-0.04	-0.10	-0.37	-3.47	99%
CCGF	**-0.04**	**-0.04**	**-0.04**	**-0.52**	**99%**

4.2. Arithmetic Expressions

The task of arithmetic expressions aims to search for the expression sequences that fit a target one. The target arithmetic expression is "1 / 3 + x + s i n (x * x)." We follow Kusner et al. (2017) to generate expression sequences by the following rules of grammars:

```
S → S'+'T | S'*'T | S'/'T | T
T → '('S')' | 'sin ('S')' | 'exp ('S')'
T → 'x' | '1' | '2' | '3'
```

where `S` and `T` are non-terminals and the symbol `|` is a separator between the possible production rules. We randomly collect 100,000 univariate arithmetic expressions that have at most 15 production rules. We use all of them as the training data of CCGF, and 90% of them are used for the training of MLE and 10% of them for validating of MLE (similar protocols are also applied to the following experiments).

The property scores of arithmetic expressions are calculated based on their computed outputs. Concretely, given a sequence (arithmetic expression), the score of the property is computed by $-\log(1 + \text{MSE})$, where MSE indicates the mean square error between the outputs of the expression and the outputs of the target expression. The outputs of an expression comprise of 1,000 results that are obtained by passing the linearly-spaced input values between -10 and 10. Based on this metric, a higher property score corresponds to a more similar expression to the target one. Since the invalid sequences can not be computed, we train a BiGRU network to fit the property based on the collected dataset and take it as the function f_p. As for the function g, it is implemented by the grammar rules described previously.

We compare our method with CVAE (Gómez-Bombarelli et al., 2018) and GVAE (Kusner et al., 2017). Both of them learn a latent space and leverage Bayesian optimization

Table 2. Best expressions produced by individual methods.

Methods	Rank	Expression
Target expression	-	1 / 3 + x + sin(x * x)
CVAE	1	x * 1 + sin(3) + sin(3 / 1)
	2	x * 1 + sin(1) + sin(2 * 3)
	3	x + 1 + sin(3) + sin(3 + 1)
GVAE	1	x / 1 + sin(3) + sin(x * x)
	2	1 / 2 + (x) + sin(x * x)
	3	x / x + (x) + sin(x * x)
CCGF	1	x * 1 + sin(3) + sin(x * x)
	2	sin(3) + x / 1 + sin(x * x)
	3	1 * x + sin(3) + sin(x * x)

algorithm (Jones et al., 1998) to search the sequences with high property scores. CVAE only takes character sequence information; GVAE also utilizes the context-free grammar.

Table 1 shows the top 3 property scores, average scores and the average validity among the 2, 500 sequences generated by each method. Compared with CVAE, GVAE largely improves the properties of generated sequences and validity performance by making use of the grammar information. CCGF does not need the grammar information and achieves the best results in terms of all the metrics, showing the advantages of modeling the sequence optimization to the CCO problem. Table 2 shows the top 3 expressions found by each method. All the top 3 expressions produced by CCGF are the closest to the target expressions, with different surfaces.

Table 3. Optimization performance of individual methods on the task of python programs.

Methods	1st	2nd	3nd	Validity
CVAE	-0.174	-0.289	-0.304	0.02%
GVAE	-0.545	-0.550	-0.575	2.96%
SD-VAE	-0.121	-0.144	-0.146	**100.00%**
CCGF	**-0.048**	**-0.048**	**-0.064**	99.5%

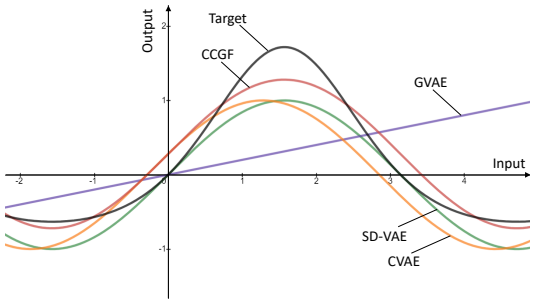

Figure 1. Curves of the best programs found by different methods where programs are regarded as functions of the input (i.e., `v0`).

4.3. Short Python Programs

In this task, we aim at finding some python programs that fit the target one. We follow Kusner et al. (2017) to collect 130, 000 univariate programs as the training data. All the program sequences comply with the grammar rules of the python programming language. The input of the program is `v0` and its output variable is specified by `return`. In this experiment, the function g is implemented by the python compiler to check the grammar of the generated programs. The function f_p, also implemented by a well trained BiGRU, yields the similarity (i.e., $-\log(1 + \text{MSE})$ of the execution outputs between the given program and the target program. The execution outputs of a python program comprise 1,000 results obtained by assigned `v0` to the linearly-spaced values between -5 to 5. The target program is listed below.

```
v1=sin(v0);v2=exp(v1);v3=v2-1;return v3
```

Besides CVAE and GVAE, we also compare CCGF with SD-VAE. SD-VAE is another VAE-based sequential generator that makes use of both syntactic and semantic checking in the decoder of VAE. Table 3 shows the performance of individual methods in the optimization of python programs. We notice that the average validity scores of CVAE and GVAE is quite low, which is largely due to the more complex grammar of python programs. SD-VAE improves both the optimization performance and the validity of generation by leveraging syntax and semantic checking in the decoding process. CCGF surpasses all these methods in the optimization scores and also obtains similar validity performance with SD-VAE. Moreover, Figure 1 illustrates the curves of the best programs found by each method, which further confirms the superiority of the best program found by CCGF.

4.4. Generating SMILES Strings for Molecule Optimization

The goal of sequence optimization in the molecule domain involves generating molecule sequences with desired properties. Here we consider generating molecules in the format of SMILES strings. We use the ZINC molecular dataset, which contains 250, 000 drug-like molecules (Irwin et al., 2012). We consider the following two properties.

- **Solubility**. A property that measures how likely a molecule can mix with water. As in Shi et al. (2020), we measure it via the logP score penalized by the ring size and synthetic accessibility (Ertl & Schuffenhauer, 2009).
- **Druglikeness**. A property that describes how likely a molecule to be a drug. We use the quantitative estimation of drug-likeness (QED) to measure it.

We adopt the scripts used in You et al. (2018) to compute

Table 4. Optimization performance of individual methods on the ZINC dataset.

	Methods	Penalized LogP					QED				
		1st	2nd	3nd	Top 50 Avg.	Validity	1st	2nd	3nd	Top 50 Avg.	Validity
	ZINC (Dataset)	4.52	4.30	4.23	0.78	100.0%	0.948	0.948	0.948	0.948	100.0%
Graph-based	Random walk (Zhou et al., 2019)	-3.99	-4.31	-4.37	-	100.0%	0.640	0.560	0.560	-	100.0%
	JT-VAE (Jin et al., 2018)	5.30	4.93	4.49	3.93	100.0%	0.925	0.911	0.910	-	100.0%
	MHG-VAE (Kajino, 2019)	5.56	5.40	5.34	4.49	100.0%	-	-	-	-	-
	GCPN (You et al., 2018)	7.98	7.85	7.80	-	100.0%	0.948	0.947	0.946	-	100.0%
	MRNN (Popova et al., 2019)	8.63	6.08	4.73	-	100.0%	0.844	0.796	0.736	-	100.0%
	MolDQN (Zhou et al., 2019)	11.51	11.51	11.50	-	100.0%	0.934	0.931	0.930	-	100.0%
	GraphAF (Shi et al., 2020)	12.23	11.29	11.05	-	100.0%	0.948	0.948	0.947	-	100.0%
Sequence-based	CVAE (Gómez-Bombarelli et al., 2018)	1.98	1.42	1.19	-	0.7%	-	-	-	-	-
	GVAE (Kusner et al., 2017)	2.94	2.89	2.80	-	7.2%	-	-	-	-	-
	SD-VAE (Dai et al., 2018)	4.04	3.50	2.96	-	43.5%	-	-	-	-	-
	ORGAN (Guimaraes et al., 2017)	3.63	3.49	3.44	-	0.4%	0.896	0.824	0.820	-	2.2%
	CCGF	**12.32**	**11.79**	**11.61**	**10.15**	98.8%	**0.948**	**0.948**	**0.948**	**0.947**	99.0%

Table 5. Diversity and novelty of the sequences generated by individual methods. Results with † are obtained based on their published codes.

Methods	Diversity				Novelty
	MorganFp	MACCS	PairFp	TopologicalFp	
GVAE†	0.59 ± 0.13	0.43 ± 0.15	0.47 ± 0.16	0.45 ± 0.12	100.0%
SD-VAE†	$\mathbf{0.62} \pm 0.10$	0.42 ± 0.13	0.49 ± 0.13	0.46 ± 0.12	100.0%
MolDQN†	0.58 ± 0.13	0.33 ± 0.17	0.51 ± 0.13	0.44 ± 0.12	100.0%
GraphAF†	0.48 ± 0.33	0.39 ± 0.33	0.51 ± 0.30	0.49 ± 0.31	100.0%
CCGF	0.57 ± 0.29	$\mathbf{0.62} \pm 0.22$	$\mathbf{0.62} \pm 0.32$	$\mathbf{0.53} \pm 0.19$	100.0%

the above two properties for a fair comparison. We also train a BiGRU network as the function f_p for each of the optimization tasks, to obtain properties of invalid sequences during the training phase. The function g is implemented by grammar rules of SMILES via the RDKit (Landrum) library.

The property optimization in molecule domains has attracted considerable attention in recent years. The related works include ORGAN, Random walk, JT-VAE, MHG-VAE, GCPN, MRNN, MolDQN and GraphAF. All of these methods except ORGAN generate molecules based on graphs instead of sequences. We refer to them as graph-based methods. The Random walk produces graphs by generating nodes and edges randomly (Zhou et al., 2019). JT-VAE (Jin et al., 2018) generates molecules by first decoding a tree structure of chemical groups and then assembling them into molecules. GCPN (You et al., 2018) learns to optimize domain-specific rewards and adversarial loss through policy gradient. MRNN (Popova et al., 2019) uses RNN to generate atoms and bonds of molecules sequentially. MolDQN (Zhou et al., 2019) utilizes double Q-learning for molecular optimization. GraphAF (Shi et al., 2020) is a flow-based autoregressive model that generates the nodes and edges based on existing sub-graphs. We take all of these methods into comparison as shown in Table 4.

Among sequence-based approaches, CVAE and ORGAN achieve the worst performance in terms of validity, indicating that directly maximizing the penalized logP hurts the grammar of sequences. We further observe that CCGF yields significantly better results than all the other sequence-based approaches. This shows sequence optimization is better modeled as a chance-constraint optimization problem, instead of searching from a learned latent space.

It is curious to see how CCGF is compared with graph-based generative approaches. Although the graph-based methods explicitly specify the validity within their model and always generate valid molecules, they have to face a more complex searching space. We notice in Table 4 that CCGF also outperforms most graph-based approaches and achieves comparable performance with GraphAF, the best method, in terms of the two desirable properties, with a negligible loss in the average validity ($\sim$1%). We have also depicted the molecules with the highest penalized logP generated by CCGF and GraphAF (Appendix F), the latter of which is the previously state-of-the-art optimization approach.

Furthermore, we have measured the diversity of the optimized molecules. Following Dai et al. (2018), we use the pair-wise distance to indicate the diversity, which is calculated by $1 - s$, where s is the similarity score based on one of the fingerprint types of molecules. Four fingerprint types are adopted in this experiment, namely MorganFp, MACCS,

Table 6. Performance of individual methods on the task of optimizing solubility (penalized logP) based on the constraint of Lipinski rule. The results of each method are obtained based on the 2, 500 generated sequences. The "Validity" stands for the fraction of generated sequences that meet the Lipinski rule.

Methods	1st	2nd	3nd	Top 50 Avg.	"Validity"
GVAE	2.78	2.73	2.59	1.42	91.6%
SD-VAE	3.10	3.01	2.77	1.71	92.7%
MolDQN	3.88	3.55	3.29	2.97	98.5%
GraphAF	4.15	3.74	3.64	3.31	97.4%
CCGF	**4.30**	**4.02**	**3.97**	**3.58**	**98.9%**

PairFp and TopologicalFp (Nilakantan et al., 1987). For each method, we randomly sample 100 molecules and compute the average diversities and the standard deviations. As shown in Table 5, we find the diversity of sequences generated by CCGF is better than those generated by other methods in three of the four metrics.

Last but not least, We have additionally evaluated the novelty (the percentage of molecules not appearing in the training set) of the molecules generated by several competitive models including GVAE, SD-VAE, MolDQN, GraphAF, CCGF. We find the molecules produced by each method are mostly new. We conjecture the high novelty is mainly due to the vast sampling space and the objective of sequence optimization (instead of MLE).

4.5. Molecule Optimization under Complex Constraints

Our framework allows various formations of constraints; grammar rules are just one kind of them. Here we consider a more difficult constraint in molecule optimization—the Lipinski rule (Paul et al., 2010), which is a widely-used metric to determine whether the molecule is a likely orally active drug in humans (Zhavoronkov et al., 2019). Compared with the continuous QED, the binary value of the Lipinski rule is simpler to computed and free of manually selecting a cutoff. Based on this constraint, CCGF is required to optimize the solubility of a given molecule.

Optimizing multiple goals simultaneously is an emerging and challenging research topic. We only find MolDQN provides a solution that maximizes the linear combination of the two goals by reinforcement learning. Here, we follow MolDQN and extend the several competitive approaches (i.e., GVAE, SD-VAE and GraphAF) to this task by imposing the Lipinski rule into their original objective functions. The adaptations are all based on their published codes and more implementation details are elaborated in Appendix G.

The optimization performances of individual methods are shown in Table 6. GVAE and SD-VAE achieve the worst performance, indicating searching molecules from latent space by Bayesian optimization is less effective than RL. CCGF outperforms GraphAF in terms of all the metrics including the fraction of sequences meeting the Lipinski rule. This shows that CCGF is a better framework to simultaneously optimize two objectives than graph-based methods. The molecules with the highest scores discovered by our model and GraphAF are illustrated in Figure 2.

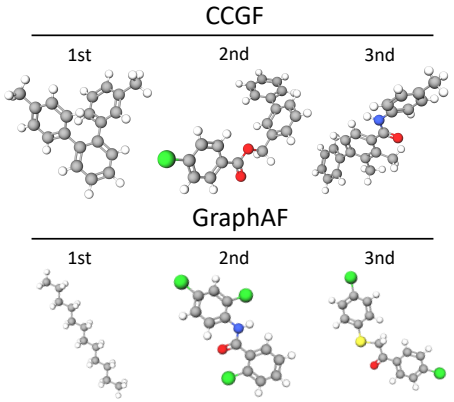

Figure 2. The top-3 molecules produced by CCGF subject to the Lipinski rule. Following the color convention in chemistry, individual sizes and colors of atoms stand for different chemical elements, where black atoms stand for carbon and red for oxygen.

Table 7. Ablation study.

Methods	Molecules (Penalized LogP)		Programs	
	Top 50 Avg.	Validity	Top 50 Avg.	Validity
MLE	3.17	80.2%	-0.453	92.2%
RL-Linear	8.81	98.5%	-0.131	99.2%
CCGF-fixed	8.53	**98.9%**	-0.139	99.1%
CCGF	**10.15**	98.8%	**-0.113**	**99.5%**

4.6. Further Analysis

Ablation Study. We first introduce two control models, denoted by CCGF-fixed and RL-Linear. CCGF-fixed follows the basic framework of CCGF but uses a fixed α during the learning. α is set to the finalized value of α in the original CCGF. As mentioned previously, RL-Linear shares the same generative model but differs in the objectives. Note that the diversity term was also incorporated in RL-Linear a fair comparison with CCGF.

We evaluate the two control models on two tasks, i.e., optimizing the penalized logP of molecules and fitting the target program. The results in Table 7 show that CCGF with fixed α obtains similar validity but significantly worse performance of logP scores than the original CCGF. We conjecture it is because the initial smaller α enables CCGF to explore more sequences with high predicted properties even if they are not valid, leading to better performance. With the same generative model and a similar training scheme, RL-Linear

still lags behind CCGF significantly, further demonstrating the advantages of our modeling strategy.

Optimization Convergence. We show the learning curves of CCGF and RL-Linear in Appendix Figure 2. In the learning curve of CCGF, we observe that both the validity and α are gradually increasing along with the number of training epochs, which validates the theoretical analysis in Theorem 1. In addition, the increase of penalized logP and the validity is more slowly than those of RL-Linear, making the training process significant longer and contributing more gains in the optimized property.

5. Conclusion

In this paper, we present a new framework that models sequence optimization as a chance-constrained optimization problem, to not only optimize a desirable property of sequences but also guarantee the validity of the generation. Our framework can automatically improve the validity requirement during the training process and facilitate the agent to escape from the locality. Intensive experiments on three domains show our framework outperforms previously state-of-the-art optimization methods to a large extent. We also outperform graph-based generators in a task of the molecule optimization under complex constraints.

Acknowledgements

We thank the anonymous reviewers for their insightful suggestions. We also thank Fangping Wan for the helpful discussion. Jian Peng acknowledges the support of the National Science Foundation (NSF) CAREER award (1652815).

References

Adam, L. and Branda, M. Nonlinear chance constrained problems: optimality conditions, regularization and solvers. *Journal of Optimization Theory and Applications*, 170(2):419–436, 2016.

Adam, L. and Branda, M. Machine learning approach to chance-constrained problems: An algorithm based on the stochastic gradient descent. *arXiv preprint arXiv:1905.10986*, 2019.

Anderson, P., Fernando, B., Johnson, M., and Gould, S. Guided open vocabulary image captioning with constrained beam search. In *EMNLP*, pp. 936–945, 2017.

Arjovsky, M., Chintala, S., and Bottou, L. Wasserstein generative adversarial networks. In *International Conference on Machine Learning*, volume 70, pp. 214–223, 2017.

Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*, 2015.

Bellman, R. Dynamic programming. *Science*, 153(3731): 34–37, 1966.

Bertsekas, D. P. *Constrained optimization and Lagrange multiplier methods*. Academic press, 2014.

Charnes, A., Cooper, W. W., and Symonds, G. H. Cost horizons and certainty equivalents: an approach to stochastic programming of heating oil. *Management science*, 4(3): 235–263, 1958.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *EMNLP*, pp. 1724–1734, 2014.

Dai, H., Tian, Y., Dai, B., Skiena, S., and Song, L. Syntax-directed variational autoencoder for structured data. In *International Conference on Learning Representations*, 2018.

Edelkamp, S. and Schroedl, S. *Heuristic Search: Theory and Applications*. Elsevier, 2011.

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

Geletu, A., Hoffmann, A., Kloppel, M., and Li, P. An inner-outer approximation approach to chance constrained optimization. *SIAM Journal on Optimization*, 27(3):1834–1857, 2017.

Gómez-Bombarelli, R., Wei, J. N., Duvenaud, D., Hernández-Lobato, J. M., Sánchez-Lengeling, B., Sheberla, D., Aguilera-Iparraguirre, J., Hirzel, T. D., Adams, R. P., and Aspuru-Guzik, A. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276, 2018.

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

Gupta, A. and Zou, J. Feedback gan for DNA optimizes protein functions. *Nature Machine Intelligence*, 1(2):105, 2019.

Hagerup, T. and Rüb, C. A guided tour of chernoff bounds. *Information processing letters*, 33(6):305–308, 1990.

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

Hu, H., Liu, X., Xiao, A., Li, Y., Zhang, C., Jiang, T., Zhao, D., Song, S., and Zeng, J. Rationalizing translation elongation by reinforcement learning. *bioRxiv:10.1101/463976*, 2018.

Irwin, J. J., Sterling, T., Mysinger, M. M., Bolstad, E. S., and Coleman, R. G. ZINC: a free tool to discover chemistry for biology. *Journal of chemical information and modeling*, 52(7):1757–1768, 2012.

Janz, D., van der Westhuizen, J., Paige, B., Kusner, M. J., and Hernández-Lobato, J. M. Learning a generative model for validity in complex discrete structures. In *International Conference on Learning Representations*, 2017.

Jin, W., Barzilay, R., and Jaakkola, T. Junction tree variational autoencoder for molecular graph generation. In *International Conference on Machine Learning (ICML)*, 2018.

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

Kajino, H. Molecular hypergraph grammar with its application to molecular optimization. In *International Conference on Machine Learning (ICML)*, pp. 3183–3191, 2019.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. Optimization by simulated annealing. *Science*, 220(4598): 671–680, 1983.

Kusner, M. J., Paige, B., and Hernández-Lobato, J. M. Grammar variational autoencoder. *Proceedings of the 34th International Conference on Machine Learning*, 70: 1945–1954, 2017.

Landrum, G. RDKit: Open-source cheminformatics.

Liu, X., Mou, L., Meng, F., Zhou, H., Zhou, J., and Song, S. Unsupervised paraphrasing by simulated annealing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 302–312, 2020.

Nemirovski, A. and Shapiro, A. Convex approximations of chance constrained programs. *SIAM Journal on Optimization*, 17(4):969–996, 2006.

Nilakantan, R., Bauman, N., Dixon, J. S., and Venkataraghavan, R. Topological torsion: a new molecular descriptor for sar applications. comparison with other descriptors. *Journal of Chemical Information and Computer Sciences*, 27(2):82–85, 1987.

Oberlander, J. and Brew, C. Stochastic text generation. *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 358(1769):1373–1387, 2000.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in pytorch. *Neural Information Processing Systems*, 2017.

Paul, S. M., Mytelka, D. S., Dunwiddie, C. T., Persinger, C. C., Munos, B. H., Lindborg, S. R., and Schacht, A. L. How to improve R&D productivity: the pharmaceutical industry's grand challenge. *Nature reviews Drug discovery*, 9(3):203, 2010.

Popova, M., Shvets, M., Oliva, J., and Isayev, O. MolecularRNN: Generating realistic molecular graphs with optimized properties. *arXiv preprint arXiv:1905.13372*, 2019.

Segler, M. H., Kogej, T., Tyrchan, C., and Waller, M. P. Generating focused molecule libraries for drug discovery with recurrent neural networks. *ACS central science*, 4 (1):120–131, 2018.

Shi, C., Xu, M., Zhu, Z., Zhang, W., Zhang, M., and Tang, J. GraphAF: a flow-based autoregressive model for molecular graph generation. In *International Conference on Learning Representations*, 2020.

Tillmann, C., Vogel, S., Ney, H., Zubiaga, A., and Sawaf, H. Accelerated dp based search for statistical translation. In *European Conference on Speech Communication and Technology*, pp. 2667–2670, 1997.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Weininger, D. 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.

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

Yin, P. and Neubig, G. A syntactic neural model for general-purpose code generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pp. 440–450, 2017.

You, J., Liu, B., Ying, Z., Pande, V., and Leskovec, J. Graph convolutional policy network for goal-directed molecular graph generation. In *Advances in Neural Information Processing Systems*, pp. 6410–6421, 2018.

Yu, L., Zhang, W., Wang, J., and Yu, Y. Seqgan: Sequence generative adversarial nets with policy gradient. In *Thirty-First AAAI Conference on Artificial Intelligence*, pp. 2852–2858, 2017.

Zhavoronkov, A., Ivanenkov, Y. A., Aliper, A., Veselov, M. S., Aladinskiy, V. A., Aladinskaya, A. V., Terentiev, V. A., Polykovskiy, D. A., Kuznetsov, M. D., Asadulaev, A., et al. Deep learning enables rapid identification of potent DDR1 kinase inhibitors. *Nature biotechnology*, 37 (9):1038–1040, 2019.

Zhou, Z., Kearnes, S., Li, L., Zare, R. N., and Riley, P. Optimization of molecules via deep reinforcement learning. *Scientific reports*, 9(1):1–10, 2019.