BIASES IN EVALUATION OF MOLECULAR OPTIMIZATION METHODS AND BIAS REDUCTION STRATEGIES

Anonymous authors
Paper under double-blind review

ABSTRACT

We are interested in *in silico* evaluation methodology for molecular optimization methods. Given a sample of molecules and their properties of our interest, we wish not only to train a generator of molecules that can find those optimized with respect to a target property but also to evaluate its performance accurately. A common practice is to train a predictor of the target property on the sample and use it for both training and evaluating the generator. We theoretically investigate this evaluation methodology and show that it potentially suffers from two biases; one is due to misspecification of the predictor and the other to reusing the same sample for training and evaluation. We discuss bias reduction methods for each of the biases, and empirically investigate their effectiveness.

1 INTRODUCTION

Molecular optimization aims to discover novel molecules with improved properties, which is often formulated as reinforcement learning by modeling the construction of a molecule using a Markov decision process. The performance of such agents is measured by the quality of generated molecules. In the community of machine learning, most of the molecular optimization methods have been verified by computer simulation. Since most of the generated molecules are novel, their properties are unknown and we have to resort to a predictor to estimate the properties. However, little attention has been paid to how reliable such estimates are, except for a few empirical studies (Renz et al., 2019; Langevin et al., 2022), making the existing performance estimates less reliable. In this paper, we study the statistical properties of such performance estimators to enhance our understanding of the evaluation protocol and we discuss several directions to improve it.

Let us first introduce a common practice to estimate the performance. Let $\mathcal{S}^\star$ be a set of molecules, $f^\star \colon \mathcal{S}^\star \to \mathbb{R}$ be a *property function* evaluating the target property of the input molecule, and $\mathcal{D} = \{(m_n, f^\star(m_n)) \in \mathcal{S}^\star \times \mathbb{R}\}_{n=1}^N$ be a sample. We typically train a predictor $f(m; \mathcal{D})$ using $\mathcal{D}$, regard it as the true property function, and follow the standard evaluation protocol of online reinforcement learning. That is, an agent is trained so as to optimize the properties of discovered molecules computed by $f(m; \mathcal{D})$, and its performance is estimated by letting it generate novel molecules and estimating their properties by $f(m; \mathcal{D})$. We call this a *plug-in performance estimator* (section 2.1).

Our research question is *how accurate the plug-in performance estimator is as compared to the true performance computed by* $f^\star$. We first point out that the plug-in performance estimator is biased in two ways, indicating that it is not reliable in general (section 2.2). The first bias called a *model misspecification bias* comes from the deviation between the predictor and the true property function evaluated over the molecules discovered by the learned agent. This bias is closely related to the one encountered in covariate shift (Shimodaira, 2000). It grows if molecules discovered by the agent become dissimilar to those used to train the predictor. The second bias called a *reusing bias* is caused by reusing the same dataset for training and testing the agent. Due to these biases, the plug-in performance estimator is not necessarily a good estimator of the true performance.

We then discuss strategies to reduce these two biases. Section 3.1 introduces three approaches to reducing the misspecification bias. Since it is caused by covariate shift, it can be reduced by training the predictor taking it into account (section 3.1.1) and/or by constraining the agent so that the generated molecules become similar to those in the sample (section 3.1.2). Yet another approach is to use a more sophisticated estimator called a doubly-robust performance estimator (section 3.1.3).

Our idea to correct the reusing bias comes from the analogy to model selection (Konishi & Kitagawa, 2007), whose objective is to estimate the test performance by correcting the bias of the training performance, *i.e.*, the performance computed by reusing the same dataset for training and testing. Given the analogy, one may consider train-test split could be the first choice. We however argue that it is not as effective as that applied to model selection due to the key difference between our setting and model selection; the test set in model selection is used to take expectation, while that in our setting is used to train a predictor, which is much more complex than expectation. This complexity introduces a non-negligible bias to the train-test split estimator, resulting in a less accurate bias estimation (section 3.2.1). We instead propose to use a bootstrap method in section 3.2.2, which is proven to estimate the reusing bias more accurately than the train-test split method does.

We empirically validate our theory in section 4. First, we quantify the two biases, and confirm that both of them are non-negligible, and the reusing bias increases as the sample size decreases, as predicted by our theory. Second, we assess the effectiveness of the bias reduction methods, and confirm that the reusing bias can be corrected, while the misspecification bias can be reduced but at the cost of performance degradation of the agent.

Notation. For any distribution G, let $\hat{G} \sim G^N$ denote the empirical distribution of a sample of N items independently drawn from G. For a set $\mathcal{X}$, let δ_x be Dirac's delta distribution at $x \in \mathcal{X}$. For any integer $M \in \mathbb{N}$, let $[M] \coloneqq \{0, \dots, M-1\}$. For any set A, $\mathcal{P}(A)$ denotes the set of probability distributions defined over A.

Problem setting. We define a molecular optimization problem using a Markov decision process (MDP) of length $H+1$ ($H \in \mathbb{N}$). See appendix A for concrete examples. Let $\mathcal{S}$ be a set of states, and $s_\perp \in \mathcal{S}$ be the terminal state. Let $\mathcal{S}^\star \subseteq \mathcal{S}$ be a subset of states that correspond to valid molecules and the rest of the states correspond to possibly incomplete representations of molecules (*invalid molecules*). Let $\mathcal{A}$ be a set of actions that transform a valid or invalid molecule into another one. There exists the terminal action $a_\perp \in \mathcal{A}$ that evaluates the property of the molecule at step H, after which the state transits to the terminal state $s_\perp$. For each step $h \in [H+1]$, let $T_h \colon \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$ be a state transition distribution, $r_h \colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ be a reward function, and $\rho_0 \in \mathcal{P}(\mathcal{S})$ be the initial state distribution. We assume that the set of states at step H is limited to $\mathcal{S}^\star$, and the reward function is defined as $r_h(s,a) = 0$ for $h \in [H]$ and $r_H(s, a_\perp) = f^\star(s)$ for $s \in \mathcal{S}^\star$. Let $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \{T_h\}_{h=0}^H, \rho_0, H\}$ be the dynamical model of the MDP. Throughout the paper, we assume we know $\mathcal{M}$ and omit the dependency on it in expressions.

Let Π be the set of policies and $\pi = \{\pi_h(\cdot \mid s)\}_{h=0}^H \in \Pi$ be a policy modeled by a probability distribution over $\mathcal{A}$ conditioned on $s \in \mathcal{S}$. At each step $h \in [H+1]$, the agent takes action a_h sampled from $\pi_h(\cdot \mid s_h)$. The performance of a policy is measured by the expected cumulative reward, $J^\star(\pi) \coloneqq \mathbb{E}^\pi[\sum_{h=0}^H r_h(S_h, A_h)] = \mathbb{E}^\pi[f^\star(S_H)]$, where $\mathbb{E}^\pi[\cdot]$ is the expectation with respect to the Markov process induced by applying policy π on $\mathcal{M}$. Letting $p_h^\pi \in \mathcal{P}(\mathcal{S}^\star)$ be the distribution of states visited by policy π at step $h \in [H+1]$, the expected cumulative reward is alternatively expressed as, $J^\star(\pi) = \mathbb{E}_{S \sim p_H^\pi} f^\star(S)$.

In practice, the property function is not available and instead a sample from it, $\mathcal{D} = \{(m_n, f^\star(m_n)) \in \mathcal{S}^\star \times \mathbb{R}\}_{n=1}^N$, is available. Let us assume that each tuple is independently distributed according to $G \in \mathcal{P}(\mathcal{S}^\star \times \mathbb{R})$. Let $G_S \in \mathcal{P}(\mathcal{S}^\star)$ be the marginalized distribution over $\mathcal{S}^\star$ induced from G. For a theoretical reason clarified in appendix B, we use the empirical distribution of the sample, $\hat{G} \in \mathcal{P}(\mathcal{S}^\star \times \mathbb{R})$, rather than the sample itself (assumption 10) and we call $\hat{G}$ an empirical distribution and a sample interchangeably.

Let us define a *policy learner* $\alpha_\pi \colon \mathcal{P}(\mathcal{S}^\star \times \mathbb{R}) \to \Pi$, an algorithm to learn a policy from a distribution over $\mathcal{S}^\star \times \mathbb{R}$. It typically receives a sample $\hat{G}$ and outputs a policy, which we denote $\hat{\pi} \coloneqq \alpha_\pi(\hat{G})$. Our objective is to evaluate its performance $J^\star(\hat{\pi})$ only given access to α_π, $\hat{G}$, and $\mathcal{M}$.

2 Biases of Plug-in Performance Estimator

A widely used approach to estimating $J^\star(\hat{\pi})$ is a *plug-in performance estimator* (section 2.1). We point out that it is biased in two ways (section 2.2) and theoretically characterize these biases in sections 2.3 and 2.4.

2.1 PLUG-IN PERFORMANCE ESTIMATOR

For any function $f\colon \mathcal{S}^\star \to \mathbb{R}$ and policy π, let us define a *plug-in performance function*,

$$J_{\mathrm{PI}}(\pi, f) \coloneqq \mathbb{E}^\pi[f(S_H)]. \tag{1}$$

Let $\alpha_f\colon \mathcal{P}(\mathcal{S}^\star \times \mathbb{R}) \to (\mathcal{S}^\star \to \mathbb{R})$ be an algorithm to learn a predictor, typically by minimizing the loss function averaged over the input distribution. Let $\hat{\pi} = \alpha_\pi(\hat{G})$ be a policy trained using $\hat{G}$ and $\hat{f} \coloneqq \alpha_f(\hat{G})$ be a predictor trained using the same $\hat{G}$. Then, the *plug-in performance estimator* is defined as $J_{\mathrm{PI}}(\hat{\pi}, \hat{f})$, which is often used as a proxy for the true performance, $J^\star(\hat{\pi})$.

2.2 BIAS DECOMPOSITION

The plug-in performance estimator is biased in two ways; the first bias comes from model misspecification of the predictor, and the second one is due to reusing the same sample for learning a policy and a predictor. Let us define $\tilde{J}_{\mathrm{PI}}(G_1, G_2) \coloneqq J_{\mathrm{PI}}(\alpha_\pi(G_1), \alpha_f(G_2))$ and $\Delta_{\mathrm{PI}}(G_1, G_2) \coloneqq \tilde{J}_{\mathrm{PI}}(G_1, G_2) - J^\star(\alpha_\pi(G_1))$. The quantity $\tilde{J}_{\mathrm{PI}}(G_1, G_2)$ denotes the estimated performance of a policy trained with distribution G_1 evaluated by a predictor trained with G_2, and $\Delta_{\mathrm{PI}}(G_1, G_2)$ denotes the deviation of the estimated performance from the ground truth. Then, the bias we care is denoted by $\mathbb{E}_{\hat{G}\sim G^N}\Delta_{\mathrm{PI}}(\hat{G}, \hat{G})$, which is decomposed as shown in theorem 1.

Theorem 1. *The bias is decomposed into a* reusing bias *and a* misspecification bias *as follows:*

$$\begin{aligned}\mathbb{E}_{\hat{G}\sim G^N}\Delta_{\mathrm{PI}}(\hat{G}, \hat{G}) =& \mathbb{E}_{\hat{G}\sim G^N}[\tilde{J}_{\mathrm{PI}}(\hat{G}, \hat{G}) - \tilde{J}_{\mathrm{PI}}(\hat{G}, G) + \tilde{J}_{\mathrm{PI}}(\hat{G}, G) - J^\star(\hat{\pi})] \\ =& \underbrace{\mathbb{E}_{\hat{G}\sim G^N}[\tilde{J}_{\mathrm{PI}}(\hat{G}, \hat{G}) - \tilde{J}_{\mathrm{PI}}(\hat{G}, G)]}_{\textit{Reusing bias}} + \underbrace{\mathbb{E}_{\hat{G}\sim G^N}\Delta_{\mathrm{PI}}(\hat{G}, G)}_{\textit{Misspecification bias}}.\end{aligned} \tag{2}$$

2.3 MISSPECIFICATION BIAS

Letting $f^\infty \coloneqq \alpha_f(G)$, the squared misspecification $\Delta_{\mathrm{PI}}(\hat{G}, G)^2$ is upperbounded by Jensen's inequality as,

$$\Delta_{\mathrm{PI}}(\hat{G}, G)^2 = \left(\mathbb{E}^{\hat{\pi}}(f^\infty(S_H) - f^\star(S_H))\right)^2 \leq \mathbb{E}_{S\sim p^{\hat{\pi}}_H}(f^\infty(S) - f^\star(S))^2. \tag{3}$$

Assuming that $f^\infty = \operatorname{argmin}_f \mathbb{E}_{S\sim G_S}(f(S) - f^\star(S))^2$ holds, the bias increases if f^∞ fails to predict the properties of molecules generated by policy $\hat{\pi}$, which occurs when the predictor is misspecified (*i.e.*, $f^\infty \neq f^\star$) and $p^{\hat{\pi}}_H$ and G_S are largely deviated (*i.e.*, the discovered molecules are not similar to those in the sample).

2.4 REUSING BIAS

The former term of equation 2,

$$b^N_{\mathrm{PI}}(G) \coloneqq \mathbb{E}_{\hat{G}\sim G^N}[\tilde{J}_{\mathrm{PI}}(\hat{G}, \hat{G}) - \tilde{J}_{\mathrm{PI}}(\hat{G}, G)], \tag{4}$$

quantifies the bias caused by reusing the same finite sample for training and testing a policy, which we call a reusing bias[1].

Let us theoretically analyze the reusing bias, assuming the sample size N is moderately large such that the asymptotic expansions are valid but $O(1/N)$ term cannot be ignored. We show in proposition 2 that the reusing bias is $O(1/N)$. See appendix B for the assumptions and appendix D.2 for its proof.

Proposition 2. *Under assumptions 10 and 12,*

$$b^N_{\mathrm{PI}}(G) = \frac{1}{2N}\mathbb{E}_{X\sim G}\left[2\tilde{J}^{(1,1)}_{G,G}(\delta_X - G, \delta_X - G) + \tilde{J}^{(0,2)}_{G,G}(\delta_X - G, \delta_X - G)\right] + O(1/N^2),$$

holds, indicating that $b^N_{\mathrm{PI}}(G) = O(1/N)$ where $\tilde{J}^{(1,1)}_{G,G}$ and $\tilde{J}^{(0,2)}_{G,G}$ are the $(1,1)$-st and $(0,2)$-nd Fréchet derivative of $\tilde{J}_{\mathrm{PI}}(G_1, G_2)$ at $(G_1, G_2) = (G, G)$.

[1]The reusing bias is caused by sample reuse as well as the finiteness of the sample, which is clear when the policy is independent from $\hat{G}$; the reusing bias still exists in such a case if $\hat{f} \neq f^\infty$.

In particular, if the policy is optimal and the estimated property function is unbiased, *i.e.*, $\mathbb{E}_{\hat{G}\sim G^N}\hat{f} = f^\infty$ (which is true at least for a linear model), we can prove that the bias is optimistic (proposition 3). See appendix E for its proof.

Proposition 3. *Assume* $\mathbb{E}_{\hat{G}\sim G^N}\hat{f} = f^\infty$ *and* $\hat{\pi} = \mathrm{argmax}_{\pi\in\Pi} J_{\mathrm{PI}}(\pi, \hat{f})$ *hold. Then,* $b^N_{\mathrm{PI}}(G) \geq 0$.

3 Bias Reduction Strategies

We have witnessed that the plug-in performance estimator is biased in two ways. In this section, we discuss how to reduce these biases to obtain reliable performance estimates.

3.1 Reducing Misspecification Bias

There are mainly three approaches to reducing the misspecification bias, $\Delta_{\mathrm{PI}}(\hat{G}, G)$. The first one is to train the predictor considering the *covariate shift*, a mismatch between training and testing distributions (section 3.1.1). The second approach is to constrain a policy such that the molecules discovered by the policy become similar to those in the sample $\hat{G}$ (section 3.1.2). These are mainly motivated by minimizing the right-hand side of equation 3. The third one is motivated by a standard technique in contextual bandit, the *doubly-robust performance estimator* instead of the plug-in performance estimator (section 3.1.3).

Before going into details, let us introduce the notion of importance weight, which is used extensively to reduce the misspecification bias. Let $F \in \mathcal{P}(\mathcal{S}^\star)$ be any probability distribution over molecules whose support is larger than that of p^π_H. Let $(p^\pi_H/F)(s) \coloneqq p^\pi_H(s)/F(s)$ $(s \in \mathcal{S}^\star)$ denote the importance weight between them, and let $\alpha_w \colon \Pi \times \mathcal{P}(\mathcal{S}^\star) \to (\mathcal{S}^\star \to \mathbb{R}_{\geq 0})$ denote an algorithm that receives a policy and a distribution over molecules and outputs the importance weight between the state distribution induced by the policy and the distribution. We typically use the algorithm by substituting sample $\hat{G}$ from G for the distribution, expecting that $\alpha_w(\pi, \hat{G}) \approx p^\pi_H/G$.

3.1.1 Covariate Shift

The misspecification bias can be reduced by minimizing the right-hand side of equation 3, which is the mean squared error over $S \sim p^{\hat{\pi}}_H$. The predictor f^∞ is usually trained by minimizing $\mathbb{E}_{S\sim G_S}(f(S) - f^\star(S))^2$ with respect to f and does not necessarily minimize the right-hand side of equation 3 due to covariate shift (Shimodaira, 2000), *i.e.*, the mismatch between the training and testing distributions. One approach suggested by the author to alleviating it is to train the predictor by weighted maximum-likelihood estimation. Let us define the algorithm as,

$$\alpha^\lambda_f(w, G) = \operatorname*{argmin}_{f\in\mathcal{F}} \mathbb{E}_{S\sim G} w(S)^\lambda (f(S) - f^\star(S))^2, \tag{5}$$

where w is any importance weight and $\lambda \in [0, 1]$ controls the bias and variance of the estimated predictor[2]. By substituting $\alpha^\lambda_f(w, G)$ for $\alpha_f(G)$, the misspecification bias will be reduced.

3.1.2 Constrain a Policy

The first approach does not always work. If $p^{\hat{\pi}}_H$ and G are not close enough, the effective sample size of the weighted maximum-likelihood estimation becomes small, leading to poor estimation. This suggests that not all policy learners can be accurately evaluated; those whose state distribution p^π_H is deviated from G are difficult to be evaluated.

Let us assume that the policy is obtained by solving the following optimization problem: $\alpha_\pi(G) = \mathrm{argmin}_{\pi\in\Pi}\, \ell(\pi; G)$. While a natural approach is to add a divergence between the generator and the data distribution P to the objective function as a regularization term, it is computationally expensive, especially when the length of MDP, H, is large. We instead propose to regularize the policy, inspired by *behavior cloning* (Fujimoto & Gu, 2021). Let us first introduce behavior cloning, and then, discuss how to apply its idea to our problem setting.

[2]While $\lambda = 1$ is optimal for $N \to \infty$, it will increase the variance for a finite sample size N, and a smaller λ is favored.

Behavior cloning regularizes the policy so that the policy imitates a *behavior policy* that generates the data. Let us assume that there exists a behavior policy π_{b} that induces the data distribution, *i.e.*, $p_H^{\pi_{\mathrm{b}}}(s) = G(s)$ for $s \in \mathcal{S}^\star$, which may not be available in our setting. Behavior cloning employs the following regularized objective function: $\ell(\pi; G) - \frac{\nu}{H+1} \sum_{h=0}^{H} \mathbb{E}_{S_h \sim p_h^{\pi_{\mathrm{b}}}, A_h \sim \pi_{\mathrm{b}}(S_h)} [\log \pi(A_h \mid S_h)]$, where $\nu \geq 0$ is a hyperparameter controlling behavior cloning. The larger ν is, the more the learned policy resembles the behavior policy, which in turn will make p_H^π close to the data distribution, and thus, we expect to reduce the misspecification bias.

A technical challenge in applying behavior cloning to our setting is that π_{b} is not available. Our key observation to this challenge is that while π_{b} is not available, it is often the case that a trajectory towards each molecule in the dataset can be reconstructed. For example, in an MDP that constructs a molecule atom-wisely (You et al., 2018), such a trajectory is easily obtained by removing atoms one by one from the molecule; in another MDP that constructs a molecule by chemical reactions (Gottipati et al., 2020), since each molecule in the dataset is assumed to be synthesizable (because the molecules in the dataset do exist in reality and thus are synthesizable), such a trajectory is easily obtained at least for those molecules in the dataset. Letting $\pi_{\mathrm{b}}^{-1}(m) = (s_0, a_0, s_1, a_1, \ldots, s_H = m)$ be a (potentially random) function to reconstruct a trajectory from a molecule, we propose to train a policy with regularization to the data distribution by the following optimization problem:

$$\alpha_\pi^\nu(G) \coloneqq \operatorname*{argmin}_{\pi \in \Pi} \ell(\pi; G) - \frac{\nu}{H+1} \sum_{h=0}^{H} \mathbb{E}_{M \sim G} \mathbb{E}_{S_0, A_0, \ldots, S_H \sim \pi_{\mathrm{b}}^{-1}(M)} [\log \pi(A_h \mid S_h)]. \quad (6)$$

Given the discussion above, at least $\alpha_\pi^\nu(\hat{G})$ can be computed. Although this regularization is not sufficient to constrain the divergence between $p_H^{\hat{\pi}}$ and G (which has been discussed in the literature of imitation learning), we consider the idea of behavior cloning is a simple yet effective heuristic, which will be investigated in the experiment.

3.1.3 Doubly-Robust Performance Estimator

The third approach to reducing the misspecification bias is a *doubly-robust performance estimator*, which has been applied in contextual bandit (Dudík et al., 2014) and offline reinforcement learning (Tang et al., 2020) as an alternative to the plug-in performance estimator. Noticing that the performance can also be estimated via importance sampling, which we call an *importance-sampling performance estimator*, the doubly-robust performance estimator combines these two estimators so as to inherit their benefits.

Importance-Sampling Performance Estimator. Given that $J^\star(\pi) = \mathbb{E}^\pi f^\star(S_H) = \mathbb{E}_{S \sim G_S}(p_H^\pi / G_S)(S) f^\star(S)$ holds, we obtain the importance-sampling performance estimator by substituting an importance weight model for the true importance weight. For any importance weight $w \colon \mathcal{S}^\star \to \mathbb{R}_{\geq 0}$ and distribution $F \in \mathcal{P}(\mathcal{S}^\star \times \mathbb{R})$, let us define an *importance-sampling performance function* as, $J_{\mathrm{IS}}(w, F) \coloneqq \mathbb{E}_{S \sim F_S} w(S) f^\star(S)$. Then, we obtain the *importance-sampling performance estimator* as $J_{\mathrm{IS}}(\hat{w}, \hat{G})$, where $\hat{w} \coloneqq \alpha_w(\hat{\pi}, \hat{G})$.

Doubly-Robust Performance Estimator. The *doubly-robust performance function* combines the plug-in and importance-sampling performance functions as follows:

$$J_{\mathrm{DR}}(\pi, w, f, F) \coloneqq \mathbb{E}_{S \sim F_S} [w(S)(f^\star(S) - f(S))] + \mathbb{E}^\pi f(S_H). \quad (7)$$

This performance function is a combination of the two performance functions in that it is related to them as, $J_{\mathrm{DR}}(\pi, 0, f, F) = J_{\mathrm{PI}}(\pi, f)$ and $J_{\mathrm{DR}}(\pi, w, 0, F) = J_{\mathrm{IS}}(w, F)$. By substituting $\hat{\pi}$, $\hat{w}$, $\hat{f}$, and $\hat{G}$ for the arguments, we obtain the doubly-robust performance estimator as $J_{\mathrm{DR}}(\hat{\pi}, \hat{w}, \hat{f}, \hat{G})$. Let us define, $\tilde{J}_{\mathrm{DR}}(G_1, G_2) \coloneqq J_{\mathrm{DR}}(\alpha_\pi(G_1), \alpha_w(\alpha_\pi(G_1), G_2), \alpha_f(G_2), G_2)$. Then, the misspecification bias is expressed as, $\Delta_{\mathrm{DR}}(\hat{G}, G) \coloneqq \tilde{J}_{\mathrm{DR}}(\hat{G}, G) - J^\star(\hat{\pi}) = \mathbb{E}_{S \sim G_S}(w^\infty(S) - (p_H^{\hat{\pi}}/G)(S))(f^\star(S) - f^\infty(S))$, where $w^\infty \coloneqq \alpha_w(\hat{\pi}, G)$. This suggests that the misspecification bias disappears if the predictor or the importance weight is well-specified.

Discussion. Notice that the misspecification biases of J_{PI} and J_{IS} are given by the followings:

$$\Delta_{\mathrm{PI}}(\hat{G}, G) = J_{\mathrm{PI}}(\hat{\pi}, f) - J^\star(\hat{\pi}) = \mathbb{E}_{S \sim G_S} \left[(p_H^{\hat{\pi}}/G)(s)(f^\infty(S) - f^\star(S)) \right],$$
$$\Delta_{\mathrm{IS}}(\hat{G}, G) \coloneqq J_{\mathrm{IS}}(\hat{w}, G) - J^\star(\hat{\pi}) = \mathbb{E}_{S \sim G_S} \left[(w^\infty(S) - (p_H^{\hat{\pi}}/G)(S)) f^\star(S) \right].$$

We can deduce that for $S \sim G_S$ (i) if $|f^\star(S) - f^\infty(S)| \ll |f^\star(S)|$ holds, the misspecification bias of J_{DR} will be smaller than that of J_{IS}, and (ii) if $|w^\infty(S) - (p_H^{\tilde{\pi}}/G)(S)| \ll |(p_H^{\tilde{\pi}}/G)(S)|$ holds, the misspecification bias of J_{DR} will be smaller than that of J_{PI}. Therefore, if we can learn both of the predictor and the importance weight well, the doubly-robust performance estimator is preferred to the others. Otherwise, the doubly-robust one can be worse than the others.

3.1.4 SUMMARY

We have introduced three approaches to reducing misspecification bias. The first one trains the predictor by weighted maximum likelihood estimation (equation 5). The second one constrains the policy by behavior cloning (equation 6). The third one is the doubly-robust performance estimator (equation 7). Taking these into consideration, let the combined performance function be, $\tilde{J}_{\mathrm{DR}}^{\lambda,\nu}(G_1, G_2) \coloneqq J_{\mathrm{DR}}(\alpha_\pi^\nu(G_1), \alpha_w(\alpha_\pi^\nu(G_1), G_2), \alpha_f^\lambda(\alpha_w(\alpha_\pi^\nu(G_1), G_2), G_2), G_2)$, and the combined performance estimator be $\tilde{J}_{\mathrm{DR}}^{\lambda,\nu}(\hat{G}, \hat{G})$. We call the importance weight and the predictor an *evaluator*. Note that proposition 2 holds for the combined performance estimator by further assuming that w is normalized and entire. Proposition 3 holds for the importance sampling performance estimator by further assuming the unbiasedness of the importance weight, but we have not found natural assumptions for the doubly-robust one. See appendix E for details.

3.2 REDUCING REUSING BIAS

Given the discussion in the previous section, let us define the reusing bias for any $\tilde{J} \in \{\tilde{J}_{\mathrm{PI}}, \tilde{J}_{\mathrm{IS}}, \tilde{J}_{\mathrm{DR}}, \tilde{J}_{\mathrm{DR}}^{\lambda,\nu}\}$ as, $b^N(G) \coloneqq \mathbb{E}_{\hat{G}\sim G^N}\left[\tilde{J}(\hat{G}, \hat{G}) - \tilde{J}(\hat{G}, G)\right]$, and let us discuss how to reduce the reusing bias. Our approach is to estimate the reusing bias and substract it from the performance estimator. Such a bias reduction has been extensively discussed in the literature of information criteria (Konishi & Kitagawa, 2007), which aim to estimate the test performance of a predictor in a supervised learning setting by correcting the bias of its training performance. There are mainly two approaches: train-test split method and bootstrap method.

3.2.1 BIAS ESTIMATION BY TRAIN-TEST SPLIT

The first approach estimates the bias via train-test split of the sample. The sample $\mathcal{D}$ is randomly split into $\mathcal{D}_{\mathrm{train}}$ and $\mathcal{D}_{\mathrm{test}}$ such that $\mathcal{D}_{\mathrm{train}} \cap \mathcal{D}_{\mathrm{test}} = \emptyset$ and $\mathcal{D}_{\mathrm{train}} \cup \mathcal{D}_{\mathrm{test}} = \mathcal{D}$. Let $\hat{G}_{\mathrm{train}}$ and $\hat{G}_{\mathrm{test}}$ denote the corresponding empirical distributions. The reusing bias is estimated by $b_{\mathrm{split}}(\hat{G}) = \mathbb{E}[\tilde{J}(\hat{G}_{\mathrm{train}}, \hat{G}_{\mathrm{train}}) - J(\hat{G}_{\mathrm{train}}, \hat{G}_{\mathrm{test}})]$, where the expectation is with respect to the random split.

While this estimator seems to be reasonable, it is not recommended for our problem setting due to the bias of the bias estimator. As demonstrated in proposition 4, the train-test split estimator has $O(1/N)$ bias, the same order as the bias $b^N(G)$ itself, and therefore, it is not reliable. Such a bias is due to the non-linearlity of $\tilde{J}(G_1, G_2)$ with respect to G_2, the distribution used for testing[3]. See appendix D.2 for its proof and appendix G for the comparison with supervised learning.

Proposition 4. *Suppose we randomly divide the sample such that* $|\mathcal{D}_{\mathrm{train}}| : |\mathcal{D}_{\mathrm{test}}| = \lambda : (1-\lambda)$ *for some* $\lambda \in (0, 1)$. *Under assumptions 10 and 12,* $\mathbb{E}_{\hat{G}\sim G^N}[b_{\mathrm{split}}(\hat{G})] = b^N(G) + O(1/N)$ *holds.*

Note that direct estimation of test performance by $\tilde{J}(\hat{G}_{\mathrm{train}}, \hat{G}_{\mathrm{test}})$ is not recommended similarly, unless the size of the test sample is sufficiently large. See appendix G for detailed discussion.

3.2.2 BOOTSTRAP BIAS ESTIMATION

An alternative approach to estimating the reusing bias (equation 4) is bootstrap (Efron & Tibshirani, 1994). A bootstrap estimator of the reusing bias $b^N(G)$ is obtained by plugging $\hat{G}$ into G: $b^N(\hat{G}) = \mathbb{E}_{\hat{G}^\star\sim\hat{G}^N}[\tilde{J}(\hat{G}^\star, \hat{G}^\star) - \tilde{J}(\hat{G}^\star, \hat{G})]$. Let $\hat{G}^{(m)}$ ($m \in [M]$) be a bootstrap sample obtained by uniform-randomly sampling data points N times from the original sample $\hat{G}$ with replacement. Then, its Monte-Carlo approximation is, $\hat{b}^N(\hat{G}) = \frac{1}{M}\sum_{m=1}^M [\tilde{J}(\hat{G}^{(m)}, \hat{G}^{(m)}) - \tilde{J}(\hat{G}^{(m)}, \hat{G})]$. In contrast to

[3]The standard supervised learning scenario does not suffer from this bias because the performance estimator is linear with respect to the testing distribution.

the train-test split method, the bootstrap bias estimation can estimate the bias as stated in proposition 5. See appendix D.2 for its proof.

Proposition 5. *Under assumptions 10 and 12,* $\mathbb{E}_{\hat{G}\sim G^N}[b^N(\hat{G})] = b^N(G) + O(1/N^2)$ *holds.*

3.2.3 SUMMARY

We have introduced two reusing-bias estimators, referring to the literature of information criteria. We have found that the train-test split estimator, one of the most popular estimators, cannot reliably estimate the bias in our problem setting, although it works in supervised learning. In contrast, the bootstrap bias estimator is shown to be less biased than the train-test split estimator and can estimate the reusing bias more reliably. Therefore, we conclude that the bootstrap bias estimator is preferable to the train-test split estimator.

From computational point of view, the bootstrap bias estimator requires us to train M agents and $M + 1$ evaluators. We set $M = 20$ in the experiments given the result of a preliminary experiment. Since the bootstrap procedure can be easily parallelized with low overhead, its wall-clock time can be reduced in proportion to the computational resource.

4 EMPIRICAL STUDIES

Let us empirically quantify the two biases as well as the effectiveness of the bias reduction methods. We first describe our experimental setup. See appendix H for full details to ensure reproduciability.

Molecular representation. All of the functions defined over molecules use the 1024-bit Morgan fingerprint (Morgan, 1965; Rogers & Hahn, 2010) with radius 2 as a feature extractor.

Environment and Agent. We employ the environment and the agent by Gottipati et al. (2020) with minor modifications. The agent receives a molecule as the current state, and outputs an action consisting of a reaction template and a reactant. The environment, receiving the action, applies the chemical reaction defined by the action to the current molecule to generate a product, which is then set as the next state. This procedure is repeated for H times, and lastly the agent takes action $a_\perp$ to be rewarded by the property of the final product. We set $H = 1$ to reduce the variance in the estimated performance and better highlight the biases and their reduction. The agent is implemented by actor-critic using fully-connected neural networks.

We use the reaction templates curated by Button et al. (2019) and prepare the reactants from the set of commercially available substances in the same way as the original environment. The number of reaction templates is 64, 15 of which require one reactant, and 49 of which require two reactants. The number of reactants is 150,560.

Evaluators. As a predictor, we use a fully-connected neural network with one hidden layer of 96 units with softplus activations except for the last layer. It is trained by minimizing the risk defined over $S \sim G_S$. As the importance weight, we use the kernel unconstrained least-squares importance fitting (KuLSI) (Kanamori et al., 2012). In particular, we use the trained predictor except for the last linear transformation as a feature extractor and compute the linear kernel using it.

Evaluation framework. To evaluate the biases, we need the true property function $f^\star$, which however is not available in general. We thus design a semi-synthetic experiment using a real-world dataset $\mathcal{D}_0 = \{(m_n, f^\star(m_n)) \in \mathcal{S}^\star \times \mathbb{R}\}_{n=1}^{N_0}$. While any function $\mathcal{S}^\star \to \mathbb{R}$ can be used as the true property function $f^\star$, we substituted the predictor provided by Gottipati et al. (2020) for $f^\star$, which was trained with the ChEMBL database (Gaulton et al., 2017) to predict pIC_{50} value associated with C-C chemokine receptor type 5 (CCR5). With this property function, we have full access to the environment, and we can construct an offline dataset $\mathcal{D}$ of an arbitrary sample size by running a random policy on $\mathcal{M}$, which is available in our setting.

To decompose the bias into the misspecification bias and the reusing bias, we need f^∞, the predictor obtained with full access to the data-generating distribution G. We approximate it by $\alpha_f(\hat{G}_{\mathrm{test}})$, where $\hat{G}_{\mathrm{test}}$ is the empirical distribution induced by a large sample $\mathcal{D}_{\mathrm{test}}$ of size 10^5 constructed independently of $\mathcal{D}$. This approximation is valid if $|\mathcal{D}_{\mathrm{test}}|$ is sufficiently large (see proposition 23). Then, the misspecification bias can be estimated by $\tilde{J}(\hat{G}, \hat{G}_{\mathrm{test}}) - J^\star(\hat{\pi})$ and the reusing bias by

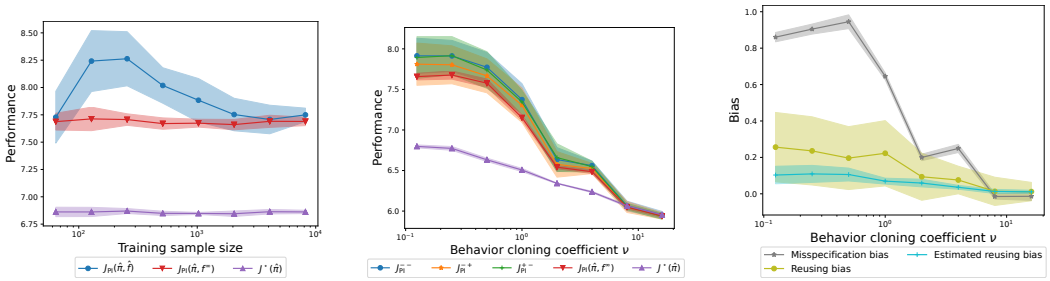

Figure 1: Lines show means and shaded areas show standard deviations. **(Left)** Biases vs. the sample size. $J_{\mathrm{PI}}(\hat{\pi}, \hat{f}) - J_{\mathrm{PI}}(\hat{\pi}, f^{\infty})$ corresponds to the reusing bias and $J_{\mathrm{PI}}(\hat{\pi}, f^{\infty}) - J^{\star}(\hat{\pi})$ to the misspecification bias. **(Middle)** Comparison between bias reduction methods. **(Right)** Comparison between the misspecification bias, reusing bias, and the estimated reusing bias.

$\tilde{J}(\hat{G}, \hat{G}) - \tilde{J}(\hat{G}, \hat{G}_{\mathrm{test}})$. The performance estimators are defined by the expectation with respect to a trajectory of a policy, and we estimate them by Monte-Carlo approximation with 1,000 trajectories.

Quantifying the two biases. First, we quantify the misspecification and reusing biases. In specific, we aim to study the relationship between these biases and the sample size. We vary the training sample size N in $\{2^6, 2^7, \ldots, 2^{13}\}$. For each N, we generate five pairs of train and test sets, and evaluate the biases as indicated above. We report the means and standard deviations.

Figure 1 (left) illustrates the result. We have three observations. First, when $N = 2^7$, the misspecification bias, $J_{\mathrm{PI}}(\hat{\pi}, f^{\infty}) - J^{\star}(\hat{\pi})$, was roughly twice as large as the reusing bias, $J_{\mathrm{PI}}(\hat{\pi}, \hat{f}) - J_{\mathrm{PI}}(\hat{\pi}, f^{\infty})$, demonstrating that both are non-negligible. Second, for $N \geq 2^7$, the reusing bias increased as the size of the training sample decreased, which coincides with proposition 2. The results for $N < 2^7$ did not coincide with it because the sample size is not large enough for asymptotic expansion to be justified. Third, the ground-truth performance of the policies was rather stable across different training sample sizes. We found that the policies were similar to each other, suggesting that this environment has a local optimum with a reasonably good performance (*cf.*, the performance of a random policy is around 5.8). This also suggests that the policy learner in our experiment was insensitive to the particular sample, and the reusing bias in this case is mainly caused by the finiteness of the sample to train the predictor, not by reusing the same sample.

Quantifying Bias Reduction Methods. We then study the effectiveness of the bias reduction methods presented in section 3. Since the behavior cloning coefficient ν will control the trade-off between the misspecification bias and the performance of the learned policy, it should be determined according to the user's requirement, *i.e.*, whether the accuracy of performance estimation or the actual performance is prioritized. Therefore, we design an experiment to evaluate the effectiveness of the bias reduction methods, varying ν in the range of $\{2^{-4}, \ldots, 2^4\}$.

Let $J_{\mathrm{PI}}^{b_1 b_2}$ $(b_1, b_2 \in \{+, -\})$ be the plug-in performance estimator with covariate shift ($b_1 = +$) or without it ($b_1 = -$) and with bootstrap bias reduction ($b_2 = +$) or without it ($b_2 = -$). Let us define $J_{\mathrm{DR}}^{b_1 b_2}$ accordingly for the doubly-robust performance estimator. We compare the performance estimates by J_{PI}^{--}, J_{PI}^{+-}, J_{PI}^{-+}, and J_{DR}^{--} to see the effectiveness of each bias reduction strategy.

Figure 1 (middle) illustrates the performance estimates for $N = 10^3$. Since J_{DR}^{--} performs significantly worse than the baseline J_{PI}^{--}, we omit it from the figure. See appendix I for the full result. We observe that the bootstrap bias reduction worked well, while the benefit of the covariate shift strategy is marginal. This indicates that the importance weight estimation did not work well in this setting.

Figure 1 (right) illustrates the biases in J_{PI}^{--} and the reusing bias estimated by the bootstrap method. As we expected, the misspecification bias tends to decrease as we increase ν. The reusing bias is under-estimated, but the estimated reusing bias contributes to bias correction.

In summary, we confirm that (i) behavior cloning can reduce the misspecification bias at the expense of performance degradation, (ii) the reusing bias can be estimated and corrected by bootstrap, and (iii) the methods using importance weights did not perform well in our setting.

5 Related Work

Our primary contribution is the comprehensive study of theoretically-sound evaluation methodology for *in silico* molecular optimization algorithms using real-world data. Since the pioneering work by Gómez-Bombarelli et al. (2018), a number of studies on this topic have been published in the communities of machine learning and cheminformatics to advance the state-of-the-art. While some of them (Gómez-Bombarelli et al., 2016; Takeda et al., 2020; Das et al., 2021) have been validated *in vitro*, many others have been evaluated *in silico*.

Early studies (Kusner et al., 2017) adopted the octanol-water partition coefficient, $\log P$, penalized by the synthetic accessibility score (Ertl & Schuffenhauer, 2009) and the number of long rings as the target property to be maximized. The score can be easily computed by RDKit, and is often implicitly regarded as a reliable score computed by an accurate simulator. Some recently consider that the $\log P$ optimization is not appropriate as a benchmark task because it is easy to optimize (Brown et al., 2019) or its prediction can be inaccurate (Yang et al., 2021), and alternative benchmark tasks have been investigated; some of them propose a suite of benchmark tasks (Brown et al., 2019; Polykovskiy et al., 2020) and the others use other property functions trained by real-world data (Olivecrona et al., 2017; Li et al., 2018a; Jin et al., 2020; Gottipati et al., 2020; Xie et al., 2021). However, most of the current evaluation protocols rely on the naive plug-in performance estimator.

As far as we are aware of, there are at least two empirical studies concerning about potential biases in the plug-in performance estimator. Renz et al. (2019) pointed out that the plug-in performance estimator is biased due to data reuse and random initialization of the predictor, while a follow-up study by Langevin et al. (2022) attributed the bias to the train-test split used by Renz et al. (2019); the train and test sets were far from being identically distributed. While these two pioneering studies shed light on the potential flaw in the plug-in performance estimator, we have not fully understand it partially because these studies are empirical.

Our contribution to this line of studies is that we not only empirically but also theoretically demonstrate potential biases in the current evaluation methodology and present bias reduction methods. This also unveils why the $\log P$ optimization task has been hacked and suggests that the alternative benchmark tasks will be hacked as long as no bias reduction method is applied. The $\log P$ function implemented in RDKit (Wildman & Crippen, 1999) is obtained by fitting a linear model to a dataset of experimental $\log P$ values, and is in fact a predictor. Our theory suggests that unless the bias reduction methods are applied, the learned agent generates unrealistic molecules that are far from those in the dataset (which has been often reported in $\log P$ optimization), and the resultant performance estimate is biased. This mechanism is also valid for the alternative benchmark tasks, and we conjecture they will also be hacked sooner or later. It also suggests that by incorporating bias reduction methods, we can reliably estimate the performance and therefore can safely compare different methods even when using the $\log P$ optimization task.

Our work shares a similar objective with a seminal work by Ito et al. (2018), which aims to reduce the reusing bias that appears when solving an optimization problem whose parameters are estimated from data. A major contribution to this literature is to relax their assummption that the predictor is well-specified. This introduces the concept of misspecification bias, which was confirmed to be non-negligible in our application. Another minor contribution is to formalize their reusing-bias correction method by bootstrap and investigate the theoretical properties.

6 Conclusion and Future Work

We have discussed that the plug-in performance estimator is biased in two ways; one is due to model misspecification and the other is due to reusing the same dataset for training and testing. In order to reduce these biases to obtain more accurate estimates, we recommend to (i) add a constraint to the policy such that the state distribution stays close to the data distribution and (ii) correct the bias by bootstrapping if it is non-negligible and we can afford to do it.

A future research direction is to improve the importance weight estimation so that the other bias reduction methods work. Another is to constrain a policy with less performance degradation. Since the methods using variational autoencoders (Gómez-Bombarelli et al., 2018; Jin et al., 2018; Kajino, 2019) can naturally generate molecules similar to those in the data, such methods could be reevaluated.

REFERENCES

Nathan Brown, Marco Fiscato, Marwin H S Segler, and Alain C Vaucher. GuacaMol: Benchmarking Models for de Novo Molecular Design. *Journal of Chemical Information and Modeling*, 59(3): 1096–1108, 2019.

Alexander Button, Daniel Merk, Jan A Hiss, and Gisbert Schneider. Automated de novo molecular design by hybrid machine intelligence and rule-driven chemical synthesis. *Nature Machine Intelligence*, 1(7):307–315, 2019. ISSN 2522-5839. doi: 10.1038/s42256-019-0067-7. URL https://doi.org/10.1038/s42256-019-0067-7.

Payel Das, Tom Sercu, Kahini Wadhawan, Inkit Padhi, Sebastian Gehrmann, Flaviu Cipcigan, Vijil Chenthamarakshan, Hendrik Strobelt, Cicero dos Santos, Pin-Yu Chen, Yi Yan Yang, Jeremy P K Tan, James Hedrick, Jason Crain, and Aleksandra Mojsilovic. Accelerated antimicrobial discovery via deep generative models and molecular dynamics simulations. *Nature Biomedical Engineering*, 5(6):613–623, 2021. ISSN 2157-846X. doi: 10.1038/s41551-021-00689-x. URL https://doi.org/10.1038/s41551-021-00689-x.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011. ISSN 15324435. URL http://jmlr.org/papers/v12/duchi11a.html.

Miroslav Dudík, Dumitru Erhan, John Langford, and Lihong Li. Doubly Robust Policy Evaluation and Optimization. *Statistical Science*, 29(4):485–511, dec 2014. ISSN 08834237, 21688745. URL http://www.jstor.org/stable/43288496.

Bradley Efron and Robert J Tibshirani. *An introduction to the bootstrap*. CRC press, 1994.

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. ISSN 1758-2946. doi: 10.1186/1758-2946-1-8. URL https://doi.org/10.1186/1758-2946-1-8.

Scott Fujimoto and Shixiang Gu. A Minimalist Approach to Offline Reinforcement Learning. In A Beygelzimer, Y Dauphin, P Liang, and J Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=Q32U7dzWXpc.

Anna Gaulton, Anne Hersey, Michał Nowotka, A Patrícia Bento, Jon Chambers, David Mendez, Prudence Mutowo, Francis Atkinson, Louisa J Bellis, Elena Cibrián-Uhalte, Mark Davies, Nathan Dedman, Anneli Karlsson, María Paula Magariños, John P Overington, George Papadatos, Ines Smit, and Andrew R Leach. The ChEMBL database in 2017. *Nucleic acids research*, 45(D1): D945–D954, jan 2017. ISSN 1362-4962 (Electronic). doi: 10.1093/nar/gkw1074.

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, Georgios Markopoulos, Soonok Jeon, Hosuk Kang, Hiroshi Miyazaki, Masaki Numata, Sunghan Kim, Wenliang Huang, Seong Ik Hong, Marc Baldo, Ryan P Adams, and Alan Aspuru-Guzik. Design of efficient molecular organic light-emitting diodes by a high-throughput virtual screening and experimental approach. *Nature Materials*, 15(10):1120–1127, 2016.

Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 2018.

Sai Krishna Gottipati, Boris Sattarov, Sufeng Niu, Yashaswi Pathak, Haoran Wei, Shengchao Liu, Shengchao Liu, Simon Blackburn, Karam Thomas, Connor Coley, Jian Tang, Sarath Chandar, and Yoshua Bengio. Learning to Navigate The Synthetically Accessible Chemical Space Using Reinforcement Learning. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 3668–3679. PMLR, 2020. URL http://proceedings.mlr.press/v119/gottipati20a.html.

Shinji Ito, Akihiro Yabe, and Ryohei Fujimaki. Unbiased Objective Estimation in Predictive Optimization. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 2176–2185. PMLR, 2018. URL http://proceedings.mlr.press/v80/ito18a.html.

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction Tree Variational Autoencoder for Molecular Graph Generation. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 2323–2332. PMLR, 2018. URL https://proceedings.mlr.press/v80/jin18a.html.

Wengong Jin, Dr.Regina Barzilay, and Tommi Jaakkola. Multi-Objective Molecule Generation using Interpretable Substructures. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 4849–4859. PMLR, 2020. URL https://proceedings.mlr.press/v119/jin20b.html.

Hiroshi Kajino. Molecular Hypergraph Grammar with Its Application to Molecular Optimization. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 3183–3191. PMLR, 2019. URL http://proceedings.mlr.press/v97/kajino19a.html.

Takafumi Kanamori, Taiji Suzuki, and Masashi Sugiyama. Statistical analysis of kernel-based least-squares density-ratio estimation. *Machine Learning*, 86(3):335–367, 2012. ISSN 1573-0565. doi: 10.1007/s10994-011-5266-3. URL https://doi.org/10.1007/s10994-011-5266-3.

Sadanori Konishi and Genshiro Kitagawa. *Information Criteria and Statistical Modeling*. Springer Publishing Company, Incorporated, 1st edition, 2007. ISBN 0387718869.

Mario Krenn, Florian Häse, AkshatKumar Nigam, Pascal Friederich, and Alan Aspuru-Guzik. Self-referencing embedded strings (SELFIES): A 100% robust molecular string representation. *Machine Learning: Science and Technology*, 1(4):45024, oct 2020. doi: 10.1088/2632-2153/aba947. URL https://doi.org/10.1088/2632-2153/aba947.

Matt J Kusner, Brooks Paige, and José Miguel Hernández-Lobato. Grammar Variational Autoencoder. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1945–1954. PMLR, 2017. URL https://proceedings.mlr.press/v70/kusner17a.html.

Maxime Langevin, Rodolphe Vuilleumier, and Marc Bianciotto. Explaining and avoiding failure modes in goal-directed generation of small molecules. *Journal of Cheminformatics*, 14(1):20, 2022. ISSN 1758-2946. doi: 10.1186/s13321-022-00601-y. URL https://doi.org/10.1186/s13321-022-00601-y.

Yibo Li, Liangren Zhang, and Zhenming Liu. Multi-objective de novo drug design with conditional graph generative model. *Journal of Cheminformatics*, 10(1):33, 2018a. ISSN 1758-2946. doi: 10.1186/s13321-018-0287-6. URL https://doi.org/10.1186/s13321-018-0287-6.

Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning Deep Generative Models of Graphs, 2018b.

H L Morgan. The Generation of a Unique Machine Description for Chemical Structures-A Technique Developed at Chemical Abstracts Service. *Journal of Chemical Documentation*, 5(2):107–113, may 1965. ISSN 0021-9576. doi: 10.1021/c160017a018. URL https://doi.org/10.1021/c160017a018.

Marcus Olivecrona, Thomas Blaschke, Ola Engkvist, and Hongming Chen. Molecular de-novo design through deep reinforcement learning. *Journal of Cheminformatics*, 9(1):48, 2017. ISSN 1758-2946. doi: 10.1186/s13321-017-0235-x. URL https://doi.org/10.1186/s13321-017-0235-x.

Abhishek Panigrahi, Raghav Somani, Navin Goyal, and Praneeth Netrapalli. Non-Gaussianity of Stochastic Gradient Noise, 2019.

Daniil Polykovskiy, Alexander Zhebrak, Benjamin Sanchez-Lengeling, Sergey Golovanov, Oktai Tatanov, Stanislav Belyaev, Rauf Kurbanov, Aleksey Artamonov, Vladimir Aladinskiy, Mark Veselov, Artur Kadurin, Simon Johansson, Hongming Chen, Sergey Nikolenko, Alán Aspuru-Guzik, and Alex Zhavoronkov. Molecular Sets (MOSES): A Benchmarking Platform for Molecular Generation Models. *Frontiers in Pharmacology*, 11:1931, 2020. ISSN 1663-9812. doi: 10.3389/fphar.2020.565644. URL https://www.frontiersin.org/article/10.3389/fphar.2020.565644.

Philipp Renz, Dries Van Rompaey, Jörg Kurt Wegner, Sepp Hochreiter, and Günter Klambauer. On failure modes in molecule generation and optimization. *Drug Discovery Today: Technologies*, 32-33:55–63, 2019. ISSN 1740-6749. doi: https://doi.org/10.1016/j.ddtec.2020.09.003. URL https://www.sciencedirect.com/science/article/pii/S1740674920300159.

David Rogers and Mathew Hahn. Extended-Connectivity Fingerprints. *Journal of Chemical Information and Modeling*, 50(5):742–754, may 2010. ISSN 1549-9596. doi: 10.1021/ci100050t. URL https://doi.org/10.1021/ci100050t.

Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of Statistical Planning and Inference*, 90(2):227–244, 2000. ISSN 0378-3758. doi: https://doi.org/10.1016/S0378-3758(00)00115-4. URL https://www.sciencedirect.com/science/article/pii/S0378375800001154.

Seiji Takeda, Toshiyuki Hama, Hsiang-Han Hsu, Victoria A Piunova, Dmitry Zubarev, Daniel P Sanders, Jed W Pitera, Makoto Kogoh, Takumi Hongo, Yenwei Cheng, Wolf Bocanett, Hideaki Nakashika, Akihiro Fujita, Yuta Tsuchiya, Katsuhiko Hino, Kentaro Yano, Shuichi Hirose, Hiroki Toda, Yasumitsu Orii, and Daiju Nakano. *Molecular Inverse-Design Platform for Material Industries*, pp. 2961–2969. Association for Computing Machinery, New York, NY, USA, 2020. ISBN 9781450379984. URL https://doi.org/10.1145/3394486.3403346.

Ziyang Tang, Yihao Feng, Lihong Li, Dengyong Zhou, and Qiang Liu. Doubly Robust Bias Reduction in Infinite Horizon Off-Policy Estimation. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1glGANtDr.

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.

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, sep 1999. ISSN 0095-2338. doi: 10.1021/ci9903071. URL https://doi.org/10.1021/ci9903071.

Jingfeng Wu, Wenqing Hu, Haoyi Xiong, Jun Huan, Vladimir Braverman, and Zhanxing Zhu. On the Noisy Gradient Descent that Generalizes as SGD. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 10367–10376. PMLR, 2020. URL https://proceedings.mlr.press/v119/wu20c.html.

Yutong Xie, Chence Shi, Hao Zhou, Yuwei Yang, Weinan Zhang, Yong Yu, and Lei Li. {MARS}: Markov Molecular Sampling for Multi-objective Drug Discovery. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=kHSu4ebxFXY.

Xiufeng Yang, Tanuj Aasawat, and Kazuki Yoshizoe. Practical Massively Parallel Monte-Carlo Tree Search Applied to Molecular Design. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=6k7VdojAIK.

Jiaxuan You, Bowen Liu, Zhitao Ying, Vijay Pande, and Jure Leskovec. Graph Convolutional Policy Network for Goal-Directed Molecular Graph Generation. In *Advances in Neural Information Processing Systems 31*, pp. 6412–6422, 2018.

A EXAMPLES OF GENERATIVE MODELS

We provide several examples that can or cannot be handled by our formulation. Intuitively, a generative process that is guaranteed to generate a valid molecule within H steps can be handled in our formulation.

Example 6 (String-based generation)**.** *Early attempts to generate molecules (Gómez-Bombarelli et al., 2018) represent a molecule by a string called SMILES (Weininger, 1988). Since the generated SMILES string is not necessarily valid, it cannot be handled within our framework. Another string representation called SELFIES (Krenn et al., 2020) can be always decoded into a valid molecule; hence, a generative process using it can be handled by our formulation.*

Example 7 (Atom-wise generation)**.** *Another approach to generating a molecule is to start from void and add atoms one by one (Li et al., 2018b; You et al., 2018). By prohibiting actions that lead to invalid molecules, we can guarantee that the generated molecules are always valid.*

Example 8 (Chemical-synthetic generation)**.** *A sequential application of chemical reactions to a molecule can be modeled by our MDP, where each state corresponds to a molecule and each action corresponds to a possible reaction to it. In particular, Gottipati et al. (2020) used a template-based chemical reaction, where an action consists of selecting a reaction template and selecting reactants.*

B TECHNICAL ASSUMPTIONS

This section introduces a set of technical assumptions for proposition 2. Informally, we assume that (i) algorithms α_π and α_f depend on $\mathcal{D}$ only through the empirical distribution $\hat{G}$ induced by sample $\mathcal{D}$ and (ii) algorithms α_π and α_f are "smooth" with respect to the input distribution. These are sufficient conditions to justify the expansion.

The first assumption is formally stated by assumption 10 with the notion of a normalized algorithm (definition 9).

Definition 9 (Normalized data-dependent algorithm)**.** *A data-dependent algorithm α, receiving data $\mathcal{D}$ as input, is* normalized *if its output $\alpha(\mathcal{D})$ depends on $\mathcal{D}$ through its empirical distribution $\hat{G}$. If algorithm α is normalized, we denote its output by $\alpha(\hat{G})$.*

Assumption 10. *Algorithms α_f and α_π are normalized.*

The second assumption about the smoothness of the algorithms is formalized by *entirety*. An entire function[4] allows Taylor-series expansion everywhere.

Definition 11 (Entire function)**.** *Let V and W be Banach spaces. A function $\alpha\colon V \to W$ is* entire *if there exist symmetric bounded multilinear maps $\{a_k : V^k \to W\}_{k=0}^{\infty}$ such that $\alpha(v) = \sum_{k=0}^{\infty} a_k(v^{\otimes k})$ $(v \in V)$ and $\lim_{k\to\infty} \|a_k\|^{1/k} = 0$. Here, $\|a_k\| \coloneqq \sup_{v_\ell \in V\setminus\{0\}, 1\le\ell\le k} \frac{a_k(v_1,\ldots,v_k)}{\prod_{\ell=1}^{k}\|v_\ell\|}$ and $v^{\otimes k}$ denotes the k-repetition of v.*

Assumption 12. *Algorithms α_f and α_π are entire.*

Finally, let us discuss that SGD-like algorithms satisfy the assumptions. Informally, a normalized data-dependent algorithm that is defined by a sequence of noisy parameter updates is entire. Since a family of SGD-based algorithms can be regarded as such algorithms, algorithms that train neural networks by SGD-like algorithms are regarded as entire. See appendix C.3 for full discussion.

C TECHNICAL BACKGROUND

For completeness, let us first define the Fréchet derivative and Taylor expansion using it in appendix C.1, and let us discuss several basic properties of entire functions in appendix C.2.

C.1 TAYLOR EXPANSION USING FRÉCHET DERIVATIVE

In this paper, we mainly analyze a function between Banach spaces by its Taylor expansion. To do so, it is necessary to introduce the Fréchet derivative, which is a generalization of the total derivative on

[4]We regard any algorithm as a function.

the space of real numbers to that on Banach spaces. In this section, we provide a brief introduction to the Fréchet derivative and Taylor expansion.

Let V and W be Banach spaces, $U \subset V$, and $f\colon U \to W$ be a function. If a bounded linear mapping $A\colon V \to W$ such that

$$\lim_{\|h\|\to 0} \frac{\|f(x+h) - f(x) - A_x(h)\|}{\|h\|} = 0 \tag{8}$$

exists, $f_x^{(1)} \coloneqq A_x$ is called the Fréchet derivative of f at $x \in U$. Let D be the Fréchet differential operator and we express $Df_x \coloneqq f_x^{(1)}$ when emphasizing the operator. Equation (8) implies that,

$$f(x+h) = f(x) + f_x^{(1)}(h) + o(\|h\|), \tag{9}$$

holds. Similarly, we can define a higher-order Fréchet derivative $f_x^{(k)}$ for $k \geq 0$, and it is a symmetric multilinear map from V^k to W when fixing x. The Taylor expansion of f is obtained as,

$$f(x+h) = \sum_{k=0}^{\infty} \frac{1}{k!} f_x^{(k)}(h^{\otimes k}), \tag{10}$$

where $h^{\otimes k}$ represents k repetitions of h.

For a bivariate function $f(x,y)\colon U^2 \to W$, let us introduce partial Fréchet derivatives. If a bounded linear mapping $A\colon V \to W$ such that,

$$\lim_{\|h_x\|\to 0} \frac{\|f(x+h_x, y) - f(x,y) - A_{x,y}(h_x)\|}{\|h_x\|} = 0 \tag{11}$$

exists, $f_{x,y}^{(1,0)} \coloneqq A_{x,y}$ is called the $(1,0)$-th Fréchet derivative of f at $(x,y) \in U$ with respect to x. Similarly, we can define the (k,l)-th Fréchet derivative $f_{x,y}^{(k,l)}$ as a multilinear map from $V^k \times V^l$ to W, when fixing $(x,y) \in U^2$. Let $D^{(k,l)}$ be the Fréchet differential operator and we express $D^{(k,l)} f_{x,y} \coloneqq f_{x,y}^{(k,l)}$ when putting emphasis on the operator. Then, the Taylor expansion of f is obtained as,

$$f(x+h_x, y+h_y) = \sum_{k=0}^{\infty} \frac{1}{k!} \sum_{l=0}^{k} \binom{k}{l} f_{x,y}^{(l,k-l)}(h_x^{\otimes l}, h_y^{\otimes (l-k)}). \tag{12}$$

C.2 Properties of Entire Functions

We present several properties of entire functions, which will be used to prove our main theoretical results.

The following proposition says that entire functions are closed under function composition.

Proposition 13. *Let V, W, and U be Banach spaces. Let $f : V \to W$ and $g : W \to U$ be entire functions. Then, the composition $g \circ f$ is also entire.*

Proof. Since f and g are entire, f and g can be expanded with coefficients $\{a_k\}_{k=0}^{\infty}$ and $\{b_k\}_{k=0}^{\infty}$ that decay super-exponentially. Define $h : V \to U$ as the formal expansion of $g \circ f$,

$$\begin{aligned} h(v) &\coloneqq \sum_{k=0}^{\infty} b_k \left(\left(\sum_{\ell=0}^{\infty} a_\ell (v^{\otimes \ell}) \right)^{\otimes k} \right) \\ &= \sum_{k=0}^{\infty} \sum_{\ell_1,\ldots,\ell_k=0}^{\infty} b_k \left(\bigotimes_{i=1}^{k} a_{\ell_i}(v^{\otimes \ell_i}) \right) \\ &= \sum_{m=0}^{\infty} c_m (v^{\otimes m}), \end{aligned}$$

where $c_m : V^m \to U$ is the symmetric bounded multilinear map given by

$$c_m(v_1, ..., v_m) \coloneqq \sum_{k=0}^{\infty} \sum_{\ell_1+...+\ell_k=m} b_k \left(\bigotimes_{i=1}^{k} a_{\ell_i}(v_{\ell^{k-1}+1}, ..., v_{\ell^k}) \right), \quad m \geq 0,$$

and $\ell^k \coloneqq \sum_{s=1}^{k} \ell_s$. Then, it suffices to show h is entire, i.e., $\|c_m\|^{1/m} \to 0$ as $m \to \infty$. By the assumption of entirety, there exist $C_a, C_b < \infty$ for all $r_a, r_b > 0$ such that

$$\|a_k\| \leq C_a r_a^k, \qquad \|b_k\| \leq C_b r_b^k$$

for all $k \geq 0$. Thus, by the triangle inequality,

$$\begin{aligned}
\|c_m\| &\leq \sum_{k=0}^{\infty} \|b_k\| \sum_{\ell_1+...+\ell_k=m} \prod_{i=1}^{k} \|a_{\ell_i}\| \\
&\leq \sum_{k=0}^{\infty} C_b r_b^k \sum_{\ell_1+...+\ell_k=m} C_a^k r_a^m \\
&= C_b r_a^m \sum_{k=0}^{\infty} \sum_{\ell_1+...+\ell_k=m} (C_a r_b)^k \\
&= C_b r_a^m \left\{ 1 + \sum_{k=0}^{\infty} \binom{k+m}{m} (C_a r_b)^{k+1} \right\}. \qquad (k \leftarrow k-1)
\end{aligned}$$

Let $S_m(r) \coloneqq \sum_{k=0}^{\infty} \binom{k+m}{m} r^k$ and observe $S_m(r) = \frac{1}{(1-r)^{m+1}}$ for $0 < r < 1$ according to the recursion starting with $S_0(r) = \frac{1}{1-r}$ and then $(1-r)S_m(r) = \sum_{k=0}^{\infty} \binom{k+m-1}{m-1} r^k = S_{m-1}(r)$. Therefore, taking r_b such that $C_a r_b < 1$, we have

$$\begin{aligned}
\|c_m\| &\leq C_b \, r_a^m \, [1 + C_a \, r_b \, S_m(C_a r_b)] \\
&= C_b r_a^m \left[1 + \frac{C_a r_b}{(1 - C_a r_b)^{m+1}} \right], \quad m \geq 0.
\end{aligned}$$

Since $r_a > 0$ can be taken arbitrarily small, we have $\|c_m\|^{1/m} \to 0$. □

Proposition 14. *Let $f : V \to W$ be an infinitely Fréchet-differentiable map from a Banach space to another. Then f is entire if and only if*

$$\lim_{k \to \infty} \sup_{v \in K} \left(\frac{\|D^k f(v)\|}{k!} \right)^{1/k} = 0$$

for all compact subset $K \subset V$. Here, D denotes the Fréchet differential operator.

Proof. Let $a_k(v) \coloneqq D^k f(v)/k!$ be the k-th order coefficient of the Taylor expansion of f at $v \in V$, $k \geq 0$. Also let $\sigma_k(v) \coloneqq \|a_k(v)\|^{1/k}$. The goal of the proof is to show the equivalence between the entirety of f and the uniform decay of $\sigma_k(v)$ on any compact $K \subset V$.

First, we show the uniform decay implies the entirety. Take an arbitrary $v \in V$ with $v \neq 0$ and let $K_v \coloneqq \{\lambda v : \lambda \in [0, 1]\}$. Since K_v is compact, we have $\lim_{k\to\infty} \sup_{u \in K_v} \sigma_k(u) = 0$, which implies there exists $C < \infty$ such that $\|a_k(u)\| \leq C(2\|v\|)^{-k}$ for all $u \in K_v$ and $k \geq 0$. Now let f_k be the k-th order finite partial sum of the Taylor expansion of f at the origin,

$$f_k(u) \coloneqq \sum_{\ell=0}^{k} a_\ell(0)(u^{\otimes \ell}), \quad k \geq 0, \quad u \in V.$$

Then, there exists $u_0 \in K_v$ for all $k \geq 0$ such that

$$\begin{aligned}
\|f(v) - f_k(v)\| &= \|a_{k+1}(u_0)(v^{\otimes k})\| \qquad (\because \text{Taylor's theorem}) \\
&\leq \|a_{k+1}(u_0)\| \|v\|^k \\
&\leq \frac{C\|v\|^k}{2^{k+1}\|v\|^{k+1}} \overset{k\to\infty}{\longrightarrow} 0.
\end{aligned}$$

Since the above argument holds for any $v \in V$ with $v \neq 0$ and trivially $f_k(0) = f(0)$, f_k converges to f everywhere. This implies the entirety of f on V.

Second, we show the entirety implies the uniform decay. For all $v, u \in V$, we have

$$\begin{aligned} f(v+u) &= \sum_{k=0}^{\infty} a_k(0)((v+u)^{\otimes k}) \\ &= \sum_{k=0}^{\infty} \sum_{\ell=0}^{k} \binom{k}{\ell} a_k(0)(v^{\otimes k-\ell}, u^{\otimes \ell}) \\ &= \sum_{\ell=0}^{\infty} \sum_{k=\ell}^{\infty} \binom{k}{\ell} a_k(0)(v^{\otimes k-\ell}, u^{\otimes \ell}) \\ &= \sum_{\ell=0}^{\infty} a_\ell(v)(u^{\otimes \ell}). \end{aligned}$$

This implies, for all $v \in K$,

$$\|a_\ell(v)\| = \left\| \sum_{k=\ell}^{\infty} \binom{k}{\ell} a_k(0)(v^{\otimes k-\ell}, \cdot) \right\| \leq \sum_{k=\ell}^{\infty} \binom{k}{\ell} \|a_k(0)\| R^{k-\ell},$$

where $R \coloneqq \sup_{v \in K} \|v\|$, which is finite since K is compact. Note that the entirety of f ensures the existence of $C < \infty$ for all $r > 0$ such that $a_k(0) \leq Cr^k$, $k \geq 0$. Take such a pair (C, r) with $r < 1/R$ and we have $\|a_\ell(v)\| \leq Cr^\ell \sum_{k=0}^{\infty} \binom{\ell+k}{\ell} (rR)^k = \frac{Cr^\ell}{(1-rR)^{\ell+1}}$, which implies $\limsup_{k \to \infty} \sup_{v \in K} \sigma_k(v) \leq r/(1-rR)$. Since $r > 0$ can be taken arbitrarily small, the uniform convergence of $\sigma_k(v)$ to zero follows. □

C.3 Entirety of Algorithms

This section discusses entirety of algorithms. First, let us prove that the plug-in performance function is entire with respect to input policy π and evaluator f.

Lemma 15. *$J_{\mathrm{PI}} \colon \Pi \times \mathcal{F} \to \mathbb{R}$ is entire, where Π is a set of policies and $\mathcal{F} \subseteq (\mathcal{S}^\star \to \mathbb{R})$ is a set of predictors.*

Proof. For any $\pi \in \Pi$ and $f \in \mathcal{F}$, we have,

$$J_{\mathrm{PI}}(\pi, f) = \mathbb{E}^\pi[f(S_H)] = \mathbf{p} P_0^\pi P_1^\pi \cdots P_H^\pi \mathbf{f},$$

where $\mathbf{p} \coloneqq (\rho_0(s))_{s \in \mathcal{S}} \in \mathbb{R}^{|\mathcal{S}|}$ is the row vector representing the initial state distribution, $P_h^\pi \coloneqq (\sum_{a \in \mathcal{A}} T_h(s' \mid s, a)\pi(a|s))_{s,s' \in \mathcal{S}} \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{S}|}$ is the transition matrix, and $\mathbf{f} \coloneqq (f(s))_{s \in \mathcal{S}} \in \mathbb{R}^{|\mathcal{S}|}$ is the column vector representing the reward at the final step. Then, it is obvious that $J_{\mathrm{PI}}(\pi, f)$ is a polynomial function, which is entire. □

Then, the following proposition shows that a sequential noisy linear computation defined by equation 13 is entire.

Proposition 16 (Entirety of sequential noisy linear computation)**.** *Let T and d be positive integers. Let $\mathbf{v} = (v_1, ..., v_T) \in V^T$ be a sequence of input variables in a Banach space V, and $\mathbf{E} = (\epsilon_1, ..., \epsilon_T) \in \mathbb{R}^{d \times T}$ be a sequence of d-dimensional noise vectors whose elements are drawn independently from the standard normal distribution. Let $\{f_t : V \times \mathbb{R}^d \to \mathbb{R}^d\}_{t=1}^T$ be a sequence of maps that are linear in the first argument and satisfy $\|f_t\| \coloneqq \sup_{\|v\|=1, \theta \in \mathbb{R}^d} |f_t(v, \theta)| < \infty$, which is used to define the sequential computation*

$$\begin{aligned} &\theta_0 \coloneqq 0, \\ &\theta_t \coloneqq f_t(v_t, \theta_{t-1}) + \epsilon_t, \quad 1 \leq t \leq T. \end{aligned} \tag{13}$$

Then, letting $g : \mathbb{R}^d \to W$ be an arbitrary bounded map to a Banach space W, the following function $h : V^T \to W$ is entire,

$$h(\mathbf{v}) \coloneqq \mathbb{E}_{\mathbf{E}}[g(\theta_T)].$$

Proposition 16 implies that SGD-based algorithms can be considered to be entire. For example, let us consider an SGD-based algorithm to minimize $\ell(\theta; \hat{G}) = \mathbb{E}_{Z\sim\hat{G}}\ell(\theta, Z)$, which operates as follows for $t = 1, \ldots, T$, starting from $\theta_0 = 0$:

$$\begin{aligned} &\hat{G}_t \sim \hat{G}^B, \\ &\theta_t = \theta_{t-1} - \alpha_t \frac{\partial}{\partial\theta}\mathbb{E}_{Z\sim\hat{G}_t}\ell(\theta_{t-1}, Z), \end{aligned} \tag{14}$$

where $\hat{G}_t$ denotes a mini-batch of size B. Theoretical studies of SGD algorithms (Wu et al., 2020) often regard the noisy mini-batch gradient as the true gradient corrupted by a Gaussian noise, which is also supported by empirical studies (for example, Panigrahi et al. (2019) concluded that the gradient noise follows a Gaussian at least in the early stage of learning when the batch size is large). Given this approximation, a mini-batch SGD algorithm is desribed as follows:

$$\begin{aligned} &\theta_0 = 0, \\ &\theta_t = \theta_{t-1} - \alpha_t \frac{\partial}{\partial\theta}\mathbb{E}_{Z\sim\hat{G}}\ell(\theta_{t-1}, Z) + \epsilon_t. \end{aligned} \tag{15}$$

Equation (15) can be handled by proposition 16 by setting $v_t = \hat{G}$ and $f_t(\hat{G}, \theta_{t-1}) = \theta_{t-1} - \alpha_t \frac{\partial}{\partial\theta}\mathbb{E}_{Z\sim\hat{G}}\ell(\theta_{t-1}, Z)$, which is linear in the first argument and is bounded. By setting g as a function that maps from a parameter of the policy to a collection of $|\mathcal{S}|$ probability measures in $\mathcal{P}(\mathcal{A})$ (which is bounded for any neural network architecture), the whole algorithm $\pi(G)$ is described by the sequential noisy linear computation.

Proof of proposition 16. Let D_t, $1 \le t \le T$, denote the Fréchet differential operator on v_t and $D^{\mathbf{m}} \coloneqq \prod_{t=1}^{T} D_t^{m_t}$ denote the multivariate higher-order counterpart, where $\mathbf{m} \coloneqq (m_1, ..., m_T) \in \mathbb{Z}_{\ge 0}^T$ is multi-index. Define $c_m(\mathbf{v})$ as the m-th order coefficient of the Taylor expansion of h around $\mathbf{v}$,

$$c_m(\mathbf{v})(\mathbf{u}) \coloneqq \sum_{\mathbf{m}:|\mathbf{m}|=m} \frac{D^{\mathbf{m}}h(\mathbf{v})(u_1^{\otimes m_1}, ..., u_T^{\otimes m_T})}{\mathbf{m}!}, \quad \mathbf{u} \in V^T, \tag{16}$$

where $|\mathbf{m}| \coloneqq \sum_{t=1}^{T} m_t$ and $\mathbf{m}! \coloneqq \prod_{t=1}^{T} m_t!$. Below, we show the entirety of h through the decay rate of $c_m(\mathbf{v})$.

Define the functions $\{g_t : \mathbb{R}^d \to W\}_{t=1}^{T}$ representing the computation from step t to T, starting with $g_T(\theta) \coloneqq g(\theta)$ and then backward-recursively $g_{t-1}(\theta) \coloneqq g_t(f_{t-1}(v_{t-1}, \theta) + \epsilon_t)$ for $1 \le t \le T$ and $\theta \in \mathbb{R}^d$. Then, we have the following decomposition formulae

$$g(\theta_T) \equiv g_t(\phi_t + \epsilon_t), \quad 1 \le t \le T,$$

where $\phi_t \coloneqq f_t(v_t, \theta_{t-1})$. Note that $g_t(\cdot)$, ϕ_t and ϵ_t are mutually statistically independent. Thus, letting $p(\epsilon) \coloneqq \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}\|\epsilon\|_2^2}$, $\epsilon \in \mathbb{R}^d$ be the density function of ϵ_t, we have

$$\begin{aligned} D_t h(\mathbf{v}) &= D_t \mathbb{E}\left[g(\theta_T)\right] \\ &= D_t \mathbb{E}\left[g_t(\phi_t + \epsilon_t)\right] \\ &= D_t \mathbb{E}\left[\int g_t(\phi_t + \epsilon) p(\epsilon) \mathrm{d}\epsilon\right] && (\because \text{independence among } g_t(\cdot), \epsilon_t, \phi_t) \\ &= D_t \mathbb{E}\left[\int g_t(\epsilon) p(\epsilon - \phi_t) \mathrm{d}\epsilon\right] \\ &= \mathbb{E}\left[D_t f_t(v_t, \theta_{t-1})^\top \cdot \frac{\partial \int g_t(\epsilon) p(\epsilon - \phi_t) \mathrm{d}\epsilon}{\partial \phi_t}\right] && (\because \text{chain rule}) \\ &= \mathbb{E}\left[\xi_t(\theta_{t-1})^\top \int (\epsilon - \phi_t) g_t(\epsilon) p(\epsilon - \phi_t) \mathrm{d}\epsilon\right] \\ &= \mathbb{E}\left[\xi_t(\theta_{t-1})^\top \int \epsilon g_t(\phi_t + \epsilon) p(\epsilon) \mathrm{d}\epsilon\right] \\ &= \mathbb{E}\left[g(\theta_T)\epsilon_t^\top \xi_t(\theta_{t-1})\right], \end{aligned}$$

where $\xi_t(\theta) \coloneqq D_t f_t(v_t, \theta)$, which is constant with respect to v_t since f_t is linear on its first argument. Iterating the same procedure, we have

$$D_t^m h(\mathbf{v}) = \mathbb{E}\left[g(\theta_T)\, \hat{H}_m(\epsilon_t; \xi_t(\theta_{t-1}))\right], \quad m \geq 0,$$

where $\hat{H}_m(\epsilon; a) : V^m \to \mathbb{R}$ is given by the recursion, starting with $\hat{H}_0(\epsilon; a) \coloneqq 1$,

$$\hat{H}_m(\epsilon; a)(u_1, ..., u_m) \coloneqq a(u_m)^\top (\epsilon - \nabla_\epsilon)\, \hat{H}_{m-1}(\epsilon; a)(u_1, ..., u_{m-1}), \quad m \geq 1,$$

for all $\epsilon \in \mathbb{R}^d$, $a : V \to \mathbb{R}^d$ and $u_1, ..., u_m \in V$. Here, ∇_ϵ denotes the gradient operator with respect to ϵ. Further applying this procedure to multiple ts, we have

$$D^{\mathbf{m}} h(\mathbf{v}) = \mathbb{E}\left[g(\theta_T) \prod_{t=1}^{T} \hat{H}_{m_t}(\epsilon_t; \xi_t(\theta_{t-1}))\right], \quad m \geq 0.$$

Observe that, by induction with respect to $m \geq 0$, we have $\hat{H}_m(\epsilon; a)(u^{\otimes m}) = \|a(u)\|^m H_m(\epsilon^\top \widehat{a(u)})$, $u \in V$, where $\{H_m(x)\}_{m=1}^{\infty}$ is the (probabilist's) Hermite polynomials and $\widehat{x} = x/\|x\|_2$ denotes the normalization of a vector $x \in \mathbb{R}^d$. Thus, for all $m \geq 0$,

$$D^{\mathbf{m}} h(\mathbf{v})(u_1^{\otimes m_1}, ..., u_T^{\otimes m_T}) = \mathbb{E}\left[g(\theta_T) \prod_{t=1}^{T} \|f_t(u_t, \theta_{t-1})\|^{m_t} H_{m_t}(\epsilon_t^\top \widehat{f}_t(u_t, \theta_{t-1}))\right]. \tag{17}$$

Since the scaled Hermite polynomials $H_n(x)/\sqrt{n!}$ form an orthonormal basis of $L^2(p)$, by Hölder's inequality,

$$|D^{\mathbf{m}} h(\mathbf{v})(u_1^{\otimes m_1}, ..., u_T^{\otimes m_T})| \leq \|g\|_\infty \prod_{t=1}^{T} \|f_t\|^{m_t} \|u_t\|^{m_t} \sqrt{m_t!},$$

where $\|g\|_\infty$ denotes the supremum norm of g. Substituting this back to equation 16, we get an upper bound on $\|c_m(\mathbf{v})\|$,

$$\begin{aligned}
\|c_m(\mathbf{v})\| &= \sup_{\mathbf{u}:\forall t, \|u_t\| \leq 1} |c_m(\mathbf{v})(\mathbf{u})| \\
&\leq \|g\|_\infty \sum_{\mathbf{m}:|\mathbf{m}|=m} \prod_{t=1}^{T} \frac{\|f_t\|^{m_t}}{\sqrt{m_t!}} \\
&\leq \|g\|_\infty \binom{m+T-1}{m} \frac{\max_{1 \leq t \leq T} \|f_t\|^m}{\sqrt{\Gamma(\frac{m}{T}+1)}}, \qquad (\because \text{convexity of } \ln \Gamma(x)),
\end{aligned}$$

where $\Gamma(x)$ denotes the gamma function. This proves the super-exponential decay of $\|c_m(\mathbf{v})\|$ independent of $\mathbf{v} \in V^T$, which yields the lemma by proposition 14. □

C.4 Stochastic Expansion

Stochastic expansion is a mathematical tool to expand an estimator $\theta(G)$ with respect to its input distribution G. We are often interested in the estimator averaged over the possible sample space, $\mathbb{E}_{\hat{G} \sim G^N} \theta(\hat{G})$, and we often expand $\theta(\hat{G})$ around G to understand the averaged estimator. This section provides a useful formula to compute it.

Lemma 17 (Stochastic expansion formula)**.** *Let $\mathcal{X}$ be a set and let $\mathcal{P}(\mathcal{X})$ be the set of probability measures on $\mathcal{X}$. Let V be a Banach space, and let $f\colon \mathcal{P}(\mathcal{X}) \to V$ be a function. Let $\mathcal{D} = \{X_n\}_{n=1}^N$ be an i.i.d. sample from $G \in \mathcal{P}(\mathcal{X})$. Let $n_1, \ldots, n_k \in [N]$. If the k-th Fréchet derivative of f exists at G and there exists $i \in [k]$ such that $n_i \neq n_j$ for all $j \in [k] \backslash \{i\}$ (such an index i is called* singular*), then*

$$\mathbb{E}_{\mathcal{D}}[f_G^{(k)}(\delta_{X_{n_1}} - G, \ldots, \delta_{X_{n_k}} - G)] = 0, \tag{18}$$

holds. Moreover, the number of assignments $(n_1, \ldots, n_k) \in [N]^k$ such that there does not *exist singular indices is $O(N^{\lfloor k/2 \rfloor})$ regarding k as a constant.*

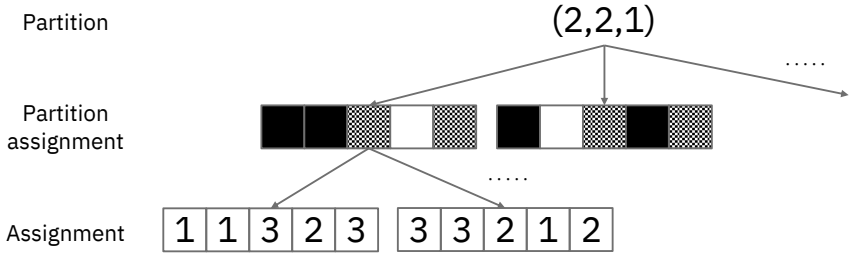

Figure 2: Assignment in $N = 3$, $k = 5$ can be abstracted into a *partition assignment*, where blocks with the same pattern fill will have the same index. A partition assignment is further abstracted into a *partition*, a sequence of non-increasing integers, each of which indicates the size of each pattern.

Proof. The expectation can be calculated as,

$$
\begin{aligned}
&\mathbb{E}_{\mathcal{D}}[f_G^{(k)}(\delta_{X_{n_1}} - G, \dots, \delta_{X_{n_k}} - G)] \\
=&\mathbb{E}_{\mathcal{D}\setminus X_i}[\mathbb{E}_{X_i}[f_G^{(k)}(\delta_{X_{n_1}} - G, \dots, \delta_{X_{n_i}} - G, \dots, \delta_{X_{n_k}} - G)]] \quad (\because \{X_n\}_{n=1}^N \text{are independent}) \\
=&\mathbb{E}_{\mathcal{D}\setminus X_i}[f_G^{(k)}(\delta_{X_{n_1}} - G, \dots, \mathbb{E}_{X_i}[\delta_{X_{n_i}} - G], \dots, \delta_{X_{n_k}} - G)] \quad (\because f_G^{(k)} \text{is multilinear}) \\
=&\mathbb{E}_{\mathcal{D}\setminus X_i}[f_G^{(k)}(\delta_{X_{n_1}} - G, \dots, 0, \dots, \delta_{X_{n_k}} - G)] \\
=&0. \quad (\because f_G^{(k)} \text{is multilinear})
\end{aligned}
$$

Then, let us count the number of non-singular assignments $(n_1, \dots, n_k) \in [N]^k$ by using the abstraction illustrated in fig. 2. Assignments can be abstracted into *partition assignments*, and they are further abstracted into *partitions*, as explained in the caption of fig. 2.

A sequence of integers $p = (p_1, \dots, p_L)$ is a partition if and only if $p_1 \geq \cdots \geq p_L \geq 1$ and $\sum_{l=1}^{L} p_l = k$. The number of partitions depends only on k, not on N. For each partition, the number of associated partition assignments depends only on k, not on N. For each partition assignment with partition $p = (p_1, \dots, p_L)$, the number of associated assignments is at most N^L. Therefore, the number of assignments associated with partition $p = (p_1, \dots, p_L)$ is at most $C(k)N^L$. Since for a partition to be non-singular, it must not contain 1, *i.e.*, $p_L \geq 2$, $L \leq \lfloor \frac{k}{2} \rfloor$ holds. Therefore, the number of non-singular assignments is at most $C(k)N^{\lfloor \frac{k}{2} \rfloor}$. □

D Asymptotic Reusing Bias

This section investigates the reusing bias in the asymptotic case. We first investigate the bias in a general case in appendix D.1 and then prove the propositions in appendix D.2.

D.1 Reusing Bias for Bivariate Entire Functions

Lemma 19 shows that the bias defined for a general bivariate entire function τ (see definition 18) is $O(1/N)$.

Definition 18 (Bivariate entire function)**.** *We refer to $\alpha \colon V_1 \times V_2 \to W$ as a bivariate entire function if V_1 and V_2 are Banach spaces and α is entire on the direct product $V_1 \times V_2$.*

Lemma 19 (Asymptotic reusing bias for bivariate entire functions)**.** *Let $\mathcal{X}$ be a set. Let $\mathcal{P}(\mathcal{X})$ denote the set of probability measures on $\mathcal{X}$. Let $\tau(G_1, G_2)$ be a bivariate entire function on $\mathcal{P}(\mathcal{X})^2$ and define the N-th bias function of τ by*

$$
b^N(G; \tau) \coloneqq \mathbb{E}_{\hat{G} \sim G^N}[\tau(\hat{G}, \hat{G}) - \tau(\hat{G}, G)], \quad N \geq 1, \quad G \in \mathcal{P}(\mathcal{X}), \tag{19}
$$

where $\hat{G} \coloneqq \frac{1}{N}\sum_{n=1}^{N} \delta_{X_n}$ denotes the empirical distribution of an i.i.d. sample $\mathcal{D} = \{X_n\}_{n=1}^N \sim G$. Then, we have

$$
b^N(G; \tau) = \frac{1}{2N}\mathbb{E}_{X \sim G}\left[2\tau_{G,G}^{(1,1)}(\delta_X - G, \delta_X - G) + \tau_{G,G}^{(0,2)}(\delta_X - G, \delta_X - G)\right] + O(1/N^2).
$$

Proof. Since it is entire, τ admits the Taylor expansion everywhere, i.e.,

$$\tau(G_1, G_2) = \sum_{k=0}^{\infty} \frac{1}{k!} \sum_{\ell=0}^{k} \binom{k}{\ell} \tau_{G,G}^{(\ell,k-\ell)}((G_1 - G)^{\otimes \ell}, (G_2 - G)^{\otimes (k-\ell)})$$

for all $G, G_1, G_2 \in \mathcal{P}(\mathcal{X})$. Thus, substituting $\hat{G}$ for G_1 and G_2 and taking the expectation with respect to $\hat{G}$, we have

$$\begin{aligned}
&\mathbb{E}[\tau(\hat{G}, \hat{G})] \\
=&\sum_{k=0}^{\infty} \frac{1}{k!} \sum_{\ell=0}^{k} \binom{k}{\ell} \mathbb{E}[\tau_{G,G}^{(\ell,k-\ell)}((\hat{G} - G)^{\otimes l}, (\hat{G} - G)^{\otimes (k-l)})] \\
=&\sum_{k=0}^{\infty} \frac{1}{k!} \sum_{\ell=0}^{k} \binom{k}{\ell} \frac{1}{N^k} \sum_{n_1,\ldots,n_k=1}^{N} \mathbb{E}\left[\tau_{G,G}^{(\ell,k-\ell)}\left(\bigotimes_{i=1}^{l}(\delta_{X_{n_i}} - G)), \bigotimes_{j=l+1}^{k}(\delta_{X_{n_j}} - G))\right)\right],
\end{aligned}$$

where the first equality is owing to the entirety of τ and the last equality follows from the multilinearity of the Fréchet derivatives.

Let us calculate each of the summands using the stochastic expansion formula (lemma 17) in the following. The summand of $k = 0$ is $\tau(G, G)$. The summand of $k = 1$ is calculated as,

$$\begin{aligned}
&\frac{1}{1!} \sum_{\ell=0}^{1} \binom{1}{\ell} \frac{1}{N} \sum_{n_1=1}^{N} \mathbb{E}\left[\tau_{G,G}^{(\ell,1-\ell)}\left(\bigotimes_{i=1}^{l}(\delta_{X_{n_i}} - G)), \bigotimes_{j=l+1}^{1}(\delta_{X_{n_j}} - G))\right)\right] \\
=&\frac{1}{N} \sum_{n_1=1}^{N} \mathbb{E}\left[\tau_{G,G}^{(0,1)}(\delta_{X_{n_1}} - G) + \tau_{G,G}^{(1,0)}(\delta_{X_{n_1}} - G)\right] \\
=&0.
\end{aligned}$$

The summand of $k = 2$ is calculated as

$$\begin{aligned}
&\frac{1}{2!} \sum_{\ell=0}^{2} \binom{2}{\ell} \frac{1}{N^2} \sum_{n_1,n_2=1}^{N} \mathbb{E}\left[\tau_{G,G}^{(\ell,2-\ell)}\left(\bigotimes_{i=1}^{l}(\delta_{X_{n_i}} - G)), \bigotimes_{j=l+1}^{2}(\delta_{X_{n_j}} - G))\right)\right] \\
=& \frac{1}{2N^2} \sum_{\ell=0}^{2} \binom{2}{\ell} \left(\sum_{n_1=n_2} + \sum_{n_1 \neq n_2}\right) \mathbb{E}\left[\tau_{G,G}^{(\ell,2-\ell)}\left(\bigotimes_{i=1}^{l}(\delta_{X_{n_i}} - G)), \bigotimes_{j=l+1}^{2}(\delta_{X_{n_j}} - G))\right)\right] \\
=& \frac{1}{2N^2} \sum_{\ell=0}^{2} \binom{2}{\ell} \sum_{n=1}^{N} \mathbb{E}\left[\tau_{G,G}^{(\ell,2-\ell)}(\delta_{X_n} - G, \delta_{X_n} - G)\right] \\
=& \frac{1}{2N} \sum_{\ell=0}^{2} \binom{2}{\ell} \mathbb{E}_{X \sim G}\left[\tau_{G,G}^{(\ell,2-\ell)}(\delta_X - G, \delta_X - G)\right].
\end{aligned}$$

It is also shown that the summands of $k \geq 3$ is $O(1/N^{\lceil \frac{k}{2} \rceil})$ by lemma 17. Summing up, we have

$$\mathbb{E}[\tau(\hat{G}, \hat{G})] = \tau(G, G) + \frac{1}{2N} \sum_{\ell=0}^{2} \binom{2}{\ell} \mathbb{E}_{X \sim G}\left[\tau_{G,G}^{(\ell,2-\ell)}(\delta_X - G, \delta_X - G)\right] + O\left(\frac{1}{N^2}\right).$$

This procedure of asymptotic expansion is called *the stochastic expansion*. The stochastic expansion is similarly applicable to the other half of the expectations in $b^N(G)$,

$$\mathbb{E}[\tau(\hat{G}, G)] = \tau(G, G) + \frac{1}{2N} \mathbb{E}_{X \sim G}[\tau_{G,G}^{(2,0)}(\delta_X - G, \delta_X - G)] + O\left(\frac{1}{N^2}\right).$$

Combining these two expansions yields the desired result. □

The bias $b^N(G;\tau)$ in lemma 19 is dependent on G, and therefore, we cannot calculate it in practice. By substituting $\hat{G}$ for G, we can construct its bootstrap estimator. The following lemma suggests that $\tau(\hat{G},\hat{G})$ with bias correction by the bootstrap estimator $b^N(\hat{G};\tau)$ is the second-order biased estimator of $\tau(\hat{G},G)$, which is better than the plug-in estimator $\tau(\hat{G},\hat{G})$.

Lemma 20 (Bias of a bootstrap estimator of the reusing bias)**.** *Take G, $\hat{G}$ and $b^N(G;\tau)$ as in lemma 19. Then, we have*

$$\mathbb{E}[b^N(\hat{G};\tau)] = b^N(G;\tau) + O(1/N^2),$$

which implies,

$$\mathbb{E}[\tau(\hat{G},\hat{G}) - b^N(\hat{G};\tau)] = \mathbb{E}[\tau(\hat{G},G)] + O(1/N^2). \tag{20}$$

Moreover,

$$\sqrt{\mathbb{E}\{b^N(\hat{G};\tau) - b^N(G;\tau)\}^2} = O(1/N^{1.5}).$$

Proof. Define the coefficient of the $O(1/N)$ term of the bias by,

$$\beta(G;\tau) \coloneqq \mathbb{E}_{X\sim G}\left[2\tau^{(1,1)}_{G,G}(\delta_X - G, \delta_X - G) + \tau^{(0,2)}_{G,G}(\delta_X - G, \delta_X - G)\right],$$

so that $b^N(G;\tau) = \frac{1}{2N}\beta(G;\tau) + O(1/N^2)$. Note that $\beta(G;\tau)$ is an entire function independent of N. Thus, the stochastic expansion of $\beta(\hat{G};\tau)$ at G gives

$$\begin{aligned}
\mathbb{E}[\beta(\hat{G};\tau)] &= \sum_{k=0}^{\infty}\frac{1}{k!}\mathbb{E}\left[\beta^{(k)}_G((\hat{G}-G)^{\otimes k};\tau)\right] \\
&= \sum_{k=0}^{\infty}\frac{1}{k!}\frac{1}{N^k}\sum_{n_1,\dots,n_k=1}^{N}\mathbb{E}\left[\beta^{(k)}_G(\delta_{X_{n_1}} - G, \dots, \delta_{X_{n_k}} - G;\tau)\right] \\
&= \beta(G;\tau) + \frac{1}{2N}\mathbb{E}_{X\sim G}\left[\beta^{(2)}_G((\delta_X - G)^{\otimes 2};\tau)\right] + O(1/N^2).
\end{aligned}$$

Substituting this back to $\mathbb{E}[b^N(\hat{G};\tau)] = \frac{1}{2N}\mathbb{E}[\beta(\hat{G};\tau)] + O(1/N^2)$, we have

$$\begin{aligned}
\mathbb{E}[b^N(\hat{G};\tau)] &= \frac{1}{2N}\left[\beta(G;\tau) + \frac{1}{2N}\mathbb{E}_{X\sim G}\left[\beta^{(2)}_G((\delta_X - G)^{\otimes 2};\tau)\right] + O(1/N^2)\right] + O(1/N^2) \\
&= \frac{1}{2N}\beta(G;\tau) + O(N^{-2}) = b^N(G;\tau) + O(1/N^2),
\end{aligned}$$

which is the first desired result.

Similarly, the stochastic expansion gives

$$\begin{aligned}
&\mathbb{E}\{\beta(\hat{G};\tau) - \beta(G;\tau)\}^2 \\
&= \mathbb{E}\left\{\sum_{k=1}^{\infty}\frac{1}{k!}\beta^{(k)}_G((\hat{G}-G)^{\otimes k};\tau)\right\}^2 \\
&= \sum_{k=1}^{\infty}\sum_{\ell=1}^{\infty}\frac{1}{k!}\frac{1}{\ell!}\mathbb{E}\left[\beta^{(k)}_G((\hat{G}-G)^{\otimes k};\tau)\beta^{(\ell)}_G((\hat{G}-G)^{\otimes \ell};\tau)\right] \\
&= \sum_{k=1}^{\infty}\sum_{\ell=1}^{\infty}\frac{1}{k!}\frac{1}{\ell!}\frac{1}{N^k}\frac{1}{N^\ell}\sum_{n_1,\dots,n_k=1}^{N}\sum_{n'_1,\dots,n'_\ell=1}^{N}\mathbb{E}\left[\beta^{(k)}_G(\delta_{X_{n_1}} - G, \dots, \delta_{X_{n_k}} - G;\tau)\beta^{(\ell)}_G(\delta_{X_{n'_1}} - G, \dots, \delta_{X_{n'_\ell}} - G;\tau)\right] \\
&= \frac{1}{N}\mathbb{E}_{X\sim G}\left[\beta^{(1)}_G(\delta_X - G;\tau)\beta^{(1)}_G(\delta_X - G;\tau)\right] + O(1/N^2) \\
&= O(1/N),
\end{aligned}$$

and thus

$$\sqrt{\mathbb{E}\{b^N(\hat{G};\tau) - b^N(G;\tau)\}^2} = \sqrt{\frac{1}{4N^2}\mathbb{E}\{\beta(\hat{G};\tau) - \beta(G;\tau)\}^2 + O\left(1/N^4\right)} = O\left(1/N^{1.5}\right).$$

□

D.2 Asymptotic Reusing Bias and Its Estimators

This section proves the main results presented in the main body.

Proof of proposition 2. By applying the composition property of entire functions (proposition 13) to lemma 15 and assumption 12, $J_{\mathrm{PI}}(\alpha_\pi(G_1), \alpha_f(G_2))\colon \mathcal{P}(\mathcal{S}\times\mathbb{R})^2 \to \mathbb{R}$ is a bivariate entire function. Therefore, by applying lemma 19, we obtain $b^N_{\mathrm{PI}}(G) = O(1/N)$. □

Proof of proposition 4. Since $\tilde{J}(G_1, G_2)$ is entire, it can be expanded around $(G_1, G_2) = (G, G)$ and we have,

$$\tilde{J}(G_1, G_2) = \sum_{k=0}^{\infty} \frac{1}{k!} \sum_{\ell=0}^{k} \binom{k}{\ell} \tilde{J}^{(\ell,k-\ell)}_{G,G}((G_1 - G)^{\otimes \ell}, (G_2 - G)^{\otimes(k-\ell)}).$$

Suppose we split the sample $\mathcal{D}$ into $\mathcal{D}_{\mathrm{train}}$ and $\mathcal{D}_{\mathrm{test}}$ whose sample sizes are N_{train} and N_{test} respectively such that $\mathcal{D}_{\mathrm{train}} \cup \mathcal{D}_{\mathrm{test}} = \mathcal{D}$, $\mathcal{D}_{\mathrm{train}} \cap \mathcal{D}_{\mathrm{test}} = \emptyset$, and $N_{\mathrm{train}}\colon N_{\mathrm{test}} = \lambda\colon (1-\lambda)$ $(0 < \lambda < 1)$. Then, we have,

$$\mathbb{E}_{\hat{G}\sim G^N}[b_{\mathrm{split}}(\hat{G})] = \mathbb{E}_{\substack{\hat{G}_{\mathrm{train}}\sim G^{N_{\mathrm{train}}} \\ \hat{G}_{\mathrm{test}}\sim G^{N_{\mathrm{test}}}}}[\tilde{J}(\hat{G}_{\mathrm{train}}, \hat{G}_{\mathrm{train}}) - \tilde{J}(\hat{G}_{\mathrm{train}}, \hat{G}_{\mathrm{test}})].$$

Each of the terms in the right-hand side can be expanded as follows:

$$\mathbb{E}_{\hat{G}_{\mathrm{train}}\sim G^{N_{\mathrm{train}}}} \tilde{J}(\hat{G}_{\mathrm{train}}, \hat{G}_{\mathrm{train}}) = \tilde{J}(G, G) + \frac{1}{2N_{\mathrm{train}}} \sum_{l=0}^{2} \binom{2}{l} \mathbb{E}_{X\sim G}\left[\tilde{J}^{(l,2-l)}_{G,G}(\delta_X - G, \delta_X - G)\right] + O(1/N^2_{\mathrm{train}}), \tag{21}$$

$$\begin{aligned}\mathbb{E}_{\substack{\hat{G}_{\mathrm{train}}\sim G^{N_{\mathrm{train}}} \\ \hat{G}_{\mathrm{test}}\sim G^{N_{\mathrm{test}}}}} \tilde{J}(\hat{G}_{\mathrm{train}}, \hat{G}_{\mathrm{test}}) =& \tilde{J}(G, G) + \frac{1}{2N_{\mathrm{train}}} \mathbb{E}_{X\sim G}\left[\tilde{J}^{(2,0)}_{G,G}(\delta_X - G, \delta_X - G)\right] \\ &+ \frac{1}{2N_{\mathrm{test}}} \mathbb{E}_{X\sim G}\left[\tilde{J}^{(0,2)}_{G,G}(\delta_X - G, \delta_X - G)\right] + O(1/N^2_{\mathrm{test}}).\end{aligned} \tag{22}$$

By combining the above expansions, we have,

$$\begin{aligned}\mathbb{E}_{\hat{G}\sim G^N}[b_{\mathrm{split}}(\hat{G})] =& \frac{1}{\lambda N} \mathbb{E}_{X\sim G}\left[\tilde{J}^{(1,1)}_{G,G}(\delta_X - G, \delta_X - G)\right] + \frac{1}{2N}\left(\frac{1}{\lambda} - \frac{1}{1-\lambda}\right) \mathbb{E}_{X\sim G}\left[\tilde{J}^{(0,2)}_{G,G}(\delta_X - G, \delta_X - G)\right] \\ &+ O(1/N^2) = b^N(G) + O(1/N).\end{aligned}$$

□

Proof of proposition 5. As shown in the proof of proposition 2, $\tilde{J}(G_1, G_2)$ is a bivariate entire function. By applying lemma 20, we obtain $\mathbb{E}_{\hat{G}\sim G^N} b^N(\hat{G}) = b^N(G) + O(1/N^2)$. □

E Reusing Bias for Optimal Policies

This section proves that the reusing bias is non-negative for optimal policies under certain assumptions (proposition 3).

Proof of proposition 3. Let $\alpha_\pi(H) \coloneqq \mathrm{argmax}_{\pi\in\Pi} J_{\mathrm{PI}}(\pi, \alpha_f(H))$ for any distribution $H \in \mathcal{P}(\mathcal{S}^\star \times \mathbb{R})$. Since $J_{\mathrm{PI}}(\alpha_\pi(\hat{G}), \alpha_f(\hat{G})) \geq J_{\mathrm{PI}}(\alpha_\pi(G), \alpha_f(\hat{G}))$ holds for any empirical distribution $\hat{G}$, taking the expectations of $\hat{G} \sim G^N$ yields,

$$\mathbb{E}_{\hat{G}\sim G^N} J_{\mathrm{PI}}(\alpha_\pi(\hat{G}), \alpha_f(\hat{G})) \geq \mathbb{E}_{\hat{G}\sim G^N} J_{\mathrm{PI}}(\alpha_\pi(G), \alpha_f(\hat{G})) = J_{\mathrm{PI}}(\alpha_\pi(G), \alpha_f(G)). \tag{23}$$

Since $J_{\mathrm{PI}}(\alpha_\pi(G), \alpha_f(G)) \geq J_{\mathrm{PI}}(\alpha_\pi(\hat{G}), \alpha_f(G))$ holds for any $\hat{G}$, taking the expectations of $\hat{G} \sim G^N$ yields,

$$J_{\mathrm{PI}}(\alpha_\pi(G), \alpha_f(G)) \geq \mathbb{E}_{\hat{G}\sim G^N} J_{\mathrm{PI}}(\alpha_\pi(\hat{G}), \alpha_f(G)). \tag{24}$$

By combining these, we obtain the inequality. □

Proposition 21 shows the same statement for the importance-sampling performance estimator. The assumpotion says that the importance weight is consistent.

$J_{\mathrm{IS}}(w, F) = \mathbb{E}_{S\sim F_S} w(S) f^\star(S)$

Proposition 21 (Reusing bias is optimistic for the importance-sampling performance estimator)**.** *Let $\hat{w} = \alpha_w(\pi, \hat{G})$ and $w^\infty = \alpha_w(\pi, G)$. Assume $\mathbb{E}_{\hat{G}\sim G^N}\mathbb{E}_{S\sim\hat{G}}\hat{w}(S)f(S) = \mathbb{E}_{S\sim G} w^\infty(S) f(S)$ holds for any π and f. Let us define $\alpha_\pi(\hat{G}) = \mathrm{argmax}_{\pi\in\Pi} J_{\mathrm{IS}}(\alpha_w(\pi, \hat{G}), \hat{G})$ and $\alpha_\pi(G) = \mathrm{argmax}_{\pi\in\Pi} J_{\mathrm{IS}}(\alpha_w(\pi, G), G)$. Then, the reusing bias is optimistic: $b^N(G) \geq 0$.*

Proof. Since $J_{\mathrm{IS}}(\alpha_w(\alpha_\pi(\hat{G}), \hat{G}); \hat{G}) \geq J_{\mathrm{IS}}(\alpha_w(\alpha_\pi(G), \hat{G}); \hat{G})$ holds for any empirical distribution $\hat{G}$, taking the expectations of $\hat{G} \sim G^N$ yields,

$$\mathbb{E}_{\hat{G}\sim G^N} J_{\mathrm{IS}}(\alpha_w(\alpha_\pi(\hat{G}), \hat{G}); \hat{G}) \geq \mathbb{E}_{\hat{G}\sim G^N} J_{\mathrm{IS}}(\alpha_w(\alpha_\pi(G), \hat{G}); \hat{G}) = J_{\mathrm{IS}}(\alpha_w(\alpha_\pi(G), G); G), \tag{25}$$

by the assumption. Since $J_{\mathrm{IS}}(\alpha_w(\alpha_\pi(G), G); G) \geq J_{\mathrm{IS}}(\alpha_w(\alpha_\pi(\hat{G}), G); G)$ holds for any sample $\hat{G}$, taking the expectations of $\hat{G} \sim G^N$ yields,

$$J_{\mathrm{IS}}(\alpha_w(\alpha_\pi(G), G); G) \geq \mathbb{E}_{\hat{G}\sim G^N} J_{\mathrm{IS}}(\alpha_w(\alpha_\pi(\hat{G}), G); G). \tag{26}$$

By combining these, we obtain the inequality. □

For the doubly-robust performance estimator, the reusing bias can be proven to be non-negative if we can assume that (i) $\mathbb{E}_{\hat{G}\sim G^N}\hat{f} = f^\infty$, (ii) $\mathbb{E}_{\hat{G}\sim G^N}\mathbb{E}_{S\sim\hat{G}}\hat{w}(S)f(S) = \mathbb{E}_{S\sim G} w^\infty(S) f(S)$ holds for any π and f, and (iii) $\mathbb{E}_{\hat{G}\sim G^N}\mathbb{E}_{S\sim\hat{G}}\hat{w}(S)\hat{f}(S) = \mathbb{E}_{S\sim G} w^\infty(S) f^\infty(S)$. However, the last assumption is less natural than the other two assumptions, and we left this for future work.

F Analytical Reusing-Bias Estimation

Proposition 2 suggests an analytical expression of the $O(1/N)$-term of the reusing bias:

$$b_N^{(1)}(G) = \frac{1}{2N}\mathbb{E}_{X\sim G}\left[2\tilde{J}^{(1,1)}_{G,G}(\delta_X - G, \delta_X - G) + \tilde{J}^{(0,2)}_{G,G}(\delta_X - G, \delta_X - G)\right].$$

In the literature of information criteria, such an analytical expression is further embodied by substituting a concrete algorithm and/or model so as to derive a simpler expression. Such a derivation is not possible for recent models including neural networks optimized by SGD. One feasible way is to compute it by approximating the Fréchet derivatives numerically and substituting $\hat{G}$ for G.

The Fréchet derivatives can be numerically approximated as follows:

$$\begin{aligned}
\tilde{J}^{(1,1)}_{G,G}(\delta_X - G, \delta_X - G) \approx \frac{1}{4\epsilon^2}\Big[&\tilde{J}((1-\epsilon)G + \epsilon\delta_X, (1-\epsilon)G + \epsilon\delta_X) \\
&- \tilde{J}((1-\epsilon)G + \epsilon\delta_X, (1+\epsilon)G - \epsilon\delta_X) \\
&- \tilde{J}((1+\epsilon)G - \epsilon\delta_X, (1-\epsilon)G + \epsilon\delta_X) \\
&+ \tilde{J}((1+\epsilon)G - \epsilon\delta_X, (1+\epsilon)G - \epsilon\delta_X)\Big], \\
\tilde{J}^{(0,2)}_{G,G}(\delta_X - G, \delta_X - G) \approx \frac{1}{\epsilon^2}\Big[&\tilde{J}(G, (1-\epsilon)G + \epsilon\delta_X) - 2\tilde{J}(G, G) + \tilde{J}(G, (1+\epsilon)G - \epsilon\delta_X)\Big].
\end{aligned}$$

The reusing bias can be estimated by approximating Fréchet derivatives as above and substituting $\hat{G}$ for G.

The analytical bias estimation is almost equivalent to the bootstrap method in terms of statistical performance but is worse than the bootstrap method in terms of computational complexity. In fact, while the bootstrap method requires us to train M agents and $M+1$ evaluators, the analytical bias estimation requires us to train $3M$ agents and $3M$ evaluators. Therefore, we conclude that the analytical bias estimation is not preferred to the bootstrap method in general.

G TRAIN-TEST SPLIT METHOD

This section investigates more on the train-test split method. First, proposition 22 shows that the test performance estimated by the train-test split method is biased by $O(1/N)$, which is the same order as the difference between $\mathbb{E}\tilde{J}(\hat{G}, G)$ and $\tilde{J}(G, G)$. In contrast, the test performance estimated by bootstrap (*e.g.*, equation 20) is biased by $O(1/N^2)$, which is better than the train-test split method.

Proposition 22. *Test performance estimation by train-test split also fails:*

$$\mathbb{E}\tilde{J}(\hat{G}_{\mathrm{train}}, \hat{G}_{\mathrm{test}}) = \mathbb{E}_{\hat{G}\sim G^N}\tilde{J}(\hat{G}, G) + O(1/N),$$

where $\mathbb{E}_{\hat{G}\sim G^N}\tilde{J}(\hat{G}, G) = \tilde{J}(G, G) + O(1/N)$ *holds.*

Proof. Recall that we split the sample $\mathcal{D}$ into $\mathcal{D}_{\mathrm{train}}$ and $\mathcal{D}_{\mathrm{test}}$ with sample sizes N_{train} and N_{test} such that $N_{\mathrm{train}} : N_{\mathrm{test}} = \lambda : (1-\lambda)$. Equation (22) suggests that,

$$\begin{aligned}\mathbb{E}_{\substack{\hat{G}_{\mathrm{train}}\sim G^{N_{\mathrm{train}}} \\ \hat{G}_{\mathrm{test}}\sim G^{N_{\mathrm{test}}}}}\tilde{J}(\hat{G}_{\mathrm{train}}, \hat{G}_{\mathrm{test}}) =& \tilde{J}(G, G) + \frac{1}{2\lambda N}\mathbb{E}_{X\sim G}\left[\tilde{J}^{(2,0)}_{G,G}(\delta_X - G, \delta_X - G)\right] \\ &+ \frac{1}{2(1-\lambda)N}\mathbb{E}_{X\sim G}\left[\tilde{J}^{(0,2)}_{G,G}(\delta_X - G, \delta_X - G)\right] + O(1/N^2),\end{aligned} \tag{27}$$

whereas the following holds:

$$\mathbb{E}_{\hat{G}\sim G^N}\tilde{J}(\hat{G}, G) = \tilde{J}(G, G) + \frac{1}{2N}\mathbb{E}_{X\sim G}\left[\tilde{J}^{(2,0)}_{G,G}(\delta_X - G, \delta_X - G)\right] + O(1/N^2). \tag{28}$$

By comparing the equations above, the coefficients of $O(1/N)$ terms do not coincide, and therefore, the test performance estimated by the train-test split is biased by $O(1/N)$. □

The train-test split method is less biased if the test set is sufficiently large (proposition 23). This proposition is not useful in practice, but it guarantees that we can estimate $\tilde{J}(\hat{G}_{\mathrm{train}}, G)$ by using a sufficiently large test sample.

Proposition 23. *In the limit of* $N_{\mathrm{test}} \to \infty$, $\tilde{J}(\hat{G}_{\mathrm{train}}, \hat{G}_{\mathrm{test}})$ *coincides with* $\tilde{J}(\hat{G}_{\mathrm{train}}, G)$ *in expectation up to* $O(1/N_{\mathrm{train}})$*-term.*

Proof of proposition 23. Equation (22) suggests that,

$$\lim_{N_{\mathrm{test}}\to\infty}\mathbb{E}_{\substack{\hat{G}_{\mathrm{train}}\sim G^{N_{\mathrm{train}}} \\ \hat{G}_{\mathrm{test}}\sim G^{N_{\mathrm{test}}}}}\tilde{J}(\hat{G}_{\mathrm{train}}, \hat{G}_{\mathrm{test}}) = \tilde{J}(G, G) + \frac{1}{2N_{\mathrm{train}}}\mathbb{E}_{X\sim G}\left[\tilde{J}^{(2,0)}_{G,G}(\delta_X - G, \delta_X - G)\right] + O(1/N^2_{\mathrm{train}}), \tag{29}$$

which coincides with $\mathbb{E}_{\hat{G}_{\mathrm{train}}\sim G^{N_{\mathrm{train}}}}\tilde{J}(\hat{G}_{\mathrm{train}}, G)$ up to $O(1/N_{\mathrm{train}})$ term. □

Let us finally discuss why the train-test split method is less accurate in our setting, whereas it is a common practice in supervised learning. The key difference is that $\tilde{J}(G_1, G_2)$ is non-linear with respect to G_2 in our setting, while it is linear in the setting of supervised learning. Let us re-define $\tilde{J}(G_1, G_2)$ as an abstract performance estimator of a data-dependent algorithm using G_1 evaluated by another data-dependent algorithm using G_2. The evaluation of a supervised learning algorithm α_f can be instantiated as follows:

$$\tilde{J}(G_1, G_2) = \mathbb{E}_{Z\sim G_2}\ell(\alpha_f(G_1), Z), \tag{30}$$

where $Z = (X, Y)$ is an example and ℓ denotes a loss function. The key observation is that equation 30 is linear in G_2, whereas the performance estimator in RL is generally non-linear in G_2. If $\tilde{J}(G_1, G_2)$ is linear in G_2, $\tilde{J}^{(0,2)}_{G,G}(\delta_X - G, \delta_X - G) = 0$ holds, and therefore, the train-test split estimator coincides with the true bias up to $O(1/N)$ and is biased by $O(1/N^2)$.

H EXPERIMENTAL SETTINGS

In this section, we introduce the details of our experimental settings. While we follow the environment and agent developed by Gottipati et al. (2020) as much as possible, we made some modifications for consistency with our setting. The main difference from the original environment is that we employ a finite-horizon reinforcement learning rather than an infinite-horizon RL formulation. The environment and agent are modified accordingly.

H.1 ENVIRONMENT

The state space $\mathcal{S}$ is the set of molecules. The action space $\mathcal{A}$ is the direct product of the set of reaction templates and that of reactants. In particular, let us represent the set of reactants by their feature vectors consisting of 35 molecular descriptors used by Gottipati et al. (2020), $\mathcal{V} = \{\mathbf{v}_1, \ldots, \mathbf{v}_L\} \subset \mathbb{R}^{35}$, and the set of reaction templates by $\Omega = \{\omega_1, \ldots, \omega_W\}$, and the action space is defined as $\mathcal{A} = \mathbb{R}^{35} \times \Omega$. Given an action $(\mathbf{v}, \omega_w) \in \mathcal{A}$ at the current state s_h, the environment transits to the next state as follows.

1. If the reaction template ω_w requires one reactant, the reaction template is applied to the current mol s_h.
   (a) If the reaction succeeds, one of the possible products is randomly selected as the next state.
   (b) If it fails, the current molecule is set as the next state.
2. If the reaction template ω_w requires two reactants, the next state is defined as follows.
   (a) Assume that the current molecule is used as the first reactant. The set of reactants is sorted by the distance to the query vector $\mathbf{v}$ in ascending order, and the second reactant is selected by the one with the smallest distance of those in the set which can be reacted with the first reactant using the reaction template ω_w. As a result, a set of possible products is obtained.
   (b) Assuming that the current molecules is used as the second reactant, the first reactant is selected in the same way as the previous procedure, and another set of possible products is obtained.
   (c) One of these possible products is set as the next state.

Note that all of the reaction templates require no more than two reactants, and the above two cases cover all.

H.2 AGENT

We employ an actor-critic architecture following the existing work (Gottipati et al., 2020). To adapt to the finite-horizon setting, we prepare $H + 1$ copies of actors and critics, and use each copy for each step. Let $\pi_h(a \mid s; \theta_h)$ be the actor and $Q_h(s, a; \phi_h)$ be the critic at step $h \in [H + 1]$. The learning algorithm repeatedly obtains pairs of an actor and a critic for each $h = H, H - 1, \ldots, 0$ backwardly. At each step, the parameters of the actor and critic are updated as follows for $t = 0, 1, 2, \ldots, T - 1$:

$$\theta_h^{(t+1)} \leftarrow \theta_h^{(t)} + \alpha^{(t)} \frac{\partial}{\partial \theta_h} \mathbb{E}_{(s,a)\sim\bar{\mathcal{D}}_h^{(t)}} \left[Q_h(s, \pi_h(s; \theta_h^{(t)}); \phi_h^{(t)}) \right], \tag{31}$$

$$\phi_h^{(t+1)} \leftarrow \phi_h^{(t)} - \beta^{(t)} \frac{\partial}{\partial \phi_h} \mathbb{E}_{(s,a)\sim\bar{\mathcal{D}}_h^{(t)}} \left[\left(Q_h(s, a; \phi_h^{(t)}) - r - Q_{h+1}(s', \pi_{h+1}(s'; \theta_{h+1}); \phi_{h+1}) \right)^2 \right], \tag{32}$$

where $\alpha^{(t)}$ and $\beta^{(t)}$ are learning rates and $\bar{\mathcal{D}}_h^{(t)}$ is a mini-batch of state-action pairs at step h drawn from a sample of trajectories $\bar{\mathcal{D}}$.

The actor $\pi_h(a \mid s; \theta_h)$ consists of a *template selector*, which receives the current state and outputs a reaction template to be aplied at the next step, followed by a *reactant selector*, which receives the current state and the output of the template selector and outputs a query vector of a reactant, $\mathbf{v} \in \mathbb{R}^{35}$.

The template selector consists of a Morgan fingerprint module with radius 2 and 1024 bits, followed by a fully-connected neural network with one hidden layer with 256 units, interleaved with a softplus

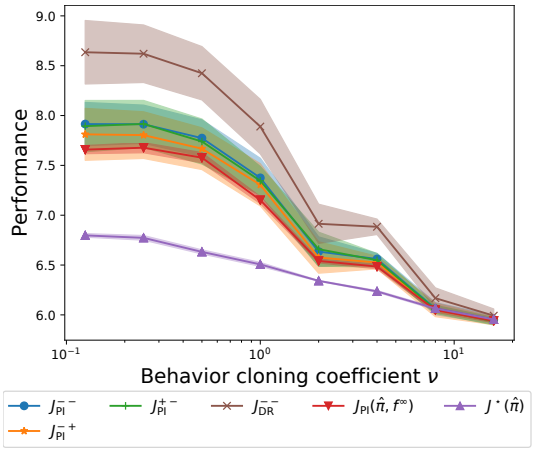

Figure 3: Experimental result with J_{DR}^{--}.

activation except for the last layer. The reactant selector consists of a Morgan fingerprint module with radius 2 and 1024 bits, which is then combined with the output of the template selector and is fed into a fully-connected neural network with one hidden layer with 256 units, interleaved with a softplus activation except for the last layer.

The critic network is a mapping from the current state and the outputs of the template selector and reactant selector to the estimated value of the current state and the action generated by the actor. The current molecule is converted into a continuous vector using the Morgan fingerprint with radius 2 and 1024 bits, which are concatenated with the outputs and fed into a fully-connected neural network with one hidden layer with 256 units, interleaved with a softplus activation except for the last layer.

For the first 500 steps, we only update the parameters of the critic, fixing those of the actor, and after that, both of them are updated for another 1,500 steps. They are optimized by AdaGrad (Duchi et al., 2011) with initial learning rate 4×10^{-4} and batch size 64.

H.3 Evaluators

The reward model $\alpha_f(\hat{G})$ is a fully-connected neural network with one hidden layer of 96 units with softplus activations except for the last layer. It is trained by minimizing the risk defined over $S \sim G$ by AdaGrad for 10^4 steps with initial learning rate 10^{-3} and batch size 128.

The importance weight, KuLSI, has a regularization hyperparameter λ_w. We chose it from $\{2^{-20}, \ldots, 2^0\}$ by leave-one-out cross validation.

H.4 Computational Environment

We implement the whole simulation in Python 3.9.0. All of the chemistry-related operations including the template-based chemical reaction is implemented by RDKit (2021.09.3). We used an IBM Cloud with 16×2.10GHz CPU cores, 128GB memory, and two NVIDIA Tesla P100 GPUs.

I Full Experimental Result

We report the performance estimates by the doubly-robust performance estimator J_{DR}^{--} in fig. 3. As evident from it, the scores are over-estimated primarily due to the over-estimation by the importance sampling performance estimator.