PRELIMINARY VERSION: DO NOT CITE
The AAAI Digital Library will contain the published
version some time after the conference

Towered Actor Critic for Handling Multiple Action Types in Reinforcement Learning For Drug Discovery

Sai Krishna Gottipati,[1] **Yashaswi Pathak,** [2] **Boris Sattarov** [1] **Sahir,** [3] **Rohan Nuttall,** [3] **Mohammad Amini,** [1] **Matthew Taylor,** [3,4] **Sarath Chandar,** [5,6,7]

[1] 99andBeyond
[2] International Institute of Information Technology, Hyderabad
[3] Department of Computing Science, University of Alberta
[4] Alberta Machine Intelligence institute
[5] Mila - Quebec AI Institute
[6] Canada CIFAR AI Chair
[7] Ecole Polytechnique Montréal
saikrishnagv1996@gmail.com

Abstract

Reinforcement learning (RL) has made significant progress in both abstract and real-world domains, but the majority of state-of-the-art algorithms deal only with monotonic actions. However, some applications require agents to reason over different types of actions. Our application simulates reaction-based molecule generation, used as part of the drug discovery pipeline, and includes both uni-molecular and bi-molecular reactions. This paper introduces a novel framework, *towered actor critic* (TAC), to handle multiple action types. The TAC framework is general in that it is designed to be combined with any existing RL algorithms for continuous action space. We combine it with TD3 to empirically obtain significantly better results than existing methods in the drug discovery setting. TAC is also applied to RL benchmarks in OpenAI Gym and results show that our framework can improve, or at least does not hurt, performance relative to standard TD3.

Introduction

Reinforcement Learning (RL) has pushed the frontiers of various domains such as robotics (OpenAI et al. 2019; Kahn, Abbeel, and Levine 2020), game playing agents (Silver et al. 2017; Anthony, Tian, and Barber 2017), and economics (Zheng et al. 2020). RL is a core problem of artificial intelligence where an agent is situated in an environment and must learn to adapt its policy to maximize a reward signal. At each time step t, the agent interacts with an MDP described by the tuple $M = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P}$ is the state transition function, and r is the reward function. The agent observes a state $s \in \mathcal{S}$, chooses an action $a \in \mathcal{A}$ according to its policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$, and then receives a reward and next state $s' \in \mathcal{S}$ from the environment. Most of the existing RL algorithms are focused on choosing "monotonic" actions i.e, the action space is restricted to $\mathcal{A} \in \mathcal{R}^m$, where m is the dimension of action space. However, several complex and crucial applications in the real world, like reaction-based molecule generation (as part of the drug discovery pipeline) (Gottipati et al. 2020b), symbolic regression (Udrescu and Tegmark 2020) need to accommodate multiple action types (uni-molecular templates and bi-molecular templates in the case of molecule generation). For bi-molecular templates, one needs to further choose a molecule (i.e, reactant) selection action corresponding to the template selection action. Existing RL algorithms do not implicitly exploit such multiple action types and their hierarchical structure. While hierarchical actor critic (Levy et al. 2017; Röder et al. 2020) or option critic (Bacon, Harb, and Precup 2016) deal with breaking action sequences over long time horizons into clear spatial or temporal sub-goals, they do not necessarily deal with having to choose between multiple action types at every time step. To achieve this, we remodel the critic to process multiple action types and combine them at an intermediate level to predict a hypothetical next state. It is then followed by another neural network to compute the value function $V(s')$ of that hypothetical next state. To the best of our knowledge, the proposed approach of a "towered" actor-critic (TAC) is the only algorithm that handles multiple action types in this manner.

Many successful and exciting applications of deep neural networks and RL have been reported over the last decade in the field of *de novo* molecule generation. These generative systems were found to be extremely efficient in optimizing structures of molecules to fit a desired properties profile such as drug-likeness, predicted activity against biological targets, lipophilicity, etc. (Gómez-Bombarelli et al. 2018; You et al. 2018b; Olivecrona et al. 2017; Segler et al. 2017; Popova, Isayev, and Tropsha 2018). However, only a few recent publications have identified and proposed solutions to the long standing challenge of embedding synthetic accessibility into the molecule generation framework. Some of them reported use of RL to tackle this problem (Gottipati et al. 2020b). These systems use reaction-based transformations of molecules as a set of actions available to the agent. In this context, there are two types of reactions / transformations (actions): uni-molecular and bi-molecular. These approaches are limited because they are built on existing RL algorithms and

do not leverage these multiple action types. Later sections elaborate on this issue and propose a novel architecture and training paradigm to demonstrate superior performance over the existing state of the art *de novo* drug design approaches.

While this work is focused on applying the TAC formulation to reaction based drug discovery, there are several other real world use cases where TAC is applicable and can boost performance. For example, in a simplistic case of symbolic regression, the mathematical operators can be classified into "unitary operators" (like abs, negation etc.) that operate on a single number and "binary operators" (like addition, subtraction etc.) that operate on two numbers and thus, another number to be operated upon has to be chosen by the policy network. Symbolic regression along with other use cases like learning-based active localization (Chaplot, Parisotto, and Salakhutdinov 2018) are described in detail in the Appendix.

We further show that this approach can be applied to any Markov Decision Process (MDP), including the ones without any inherent requirement for towered structure in the policy network, and obtain superior performance on a wide range of tasks. This paper's contributions are:

1. The introduction of a novel mechanism, towered actor critic, to train agents on MDPs with multiple action types.
2. TAC is applied to the task of reaction-based *de novo* drug design (where there are different action types at every time step) and show significant improvement over the existing state-of-the-art results.
3. TAC is also applied to standard RL tasks that do not have an inherent pyramidal/hierarchical structure of actions. Performance is comparable, or improved, relative to TD3 on several continuous action OpenAI Gym environments.

Related Work

This section summarizes the most relevant background on RL and drug discovery.

Reinforcement Learning

While most of the RL research is focused on choosing monotonic actions, there are few approaches that attempted to deal with multiple action types. Hierarchical actor critic (Levy et al. 2017; Röder et al. 2020; Masson and Konidaris 2015) learns by creating low-level and high-level sub goals for more sample efficient training. Option critic (Bacon, Harb, and Precup 2016), (Kumar and Precup 2017) learns to choose temporally extended actions. Some approaches (Lakshminarayanan et al. 2016) use a combination of both spatially and temporally extended actions, but none of these approaches deal with having to choose actions from multiple action types at the same time step. The specific architecture used for the critic in this work is inspired by the after state MDP (Sutton and Barto 2018) and the successor representation (Dayan 1993) (that attempts to solve the reward revaluation problem by providing a way to represent the relationship between different states via storing future state occupancies) and is detailed further in the later section. In the domain of continuous action space RL, several improvements have been proposed over the original deterministic policy gradient algorithm (Silver et al. 2014). Deep deterministic policy gradient (DDPG) (Lillicrap et al. 2015) extended the work to include deeper networks and demonstrated superior results over a wide range of RL environments. Twin delayed deep deterministic networks (TD3) (Fujimoto, van Hoof, and Meger 2018) tackled the problem of overestimation bias by employing multiple tricks including: using the minimum value of two critics while computing the temporal difference target and making delayed updates to the policy network.

Though the TAC framework is designed to be combined with any existing RL algorithm for continuous action spaces, this paper combines and benchmarks TAC with TD3.

De novo drug design

There has been significant progress in the application of machine learning methods for *de novo* drug design. Molecule generation is a well-studied problem and has been handled by several methods like genetic or evolutionary algorithms (Brown et al. (2004); Jensen (2019); Ahn et al. (2020)), generative models (Simonovsky and Komodakis (2018); Gómez-Bombarelli et al. (2018); Winter et al. (2019a); Jin, Barzilay, and Jaakkola (2018); Popova, Isayev, and Tropsha (2018); Olivecrona et al. (2017); Griffiths and Hernández-Lobato (2020)) and RL based approaches (You et al. (2018a); Zhou et al. (2018)). Although these methods perform very well on popular benchmarks, such as Guacamol (Brown et al. (2019)), the molecules proposed can be infeasible to synthesize in the real world. This issue has also been systematically highlighted by Gao and Coley (2020), where a synthesis planning program is used to quantify how often molecules proposed by some of the popular generative models cannot be readily synthesized.

Recent research has tried to ameliorate this problem, such as when Bradshaw et al. (2019) used a method based on variational auto-encoders to optimize for a target property using single-step reactions. A similar solution was proposed by Korovina et al. (2019), where the authors employed random selection of reactants and reaction conditions in a multi-step process to generate molecules. These generated molecules were then subjected to property evaluation. More recently, Gottipati et al. (2020b) introduced PGFS (policy gradient for forward synthesis) which attempted to generate molecules via multi-step chemical synthesis and at the same time optimized the generation towards maximising an objective function. As an extension of PGFS, Gottipati et al. (2020a) optimized for the maximum reward objective instead of the usual cumulative return objective that helped in slightly improving the performance.

PGFS operates in the realm of off-policy continuous action space. The actor module Π that consists of f and π networks predicts a reaction template and a continuous action (which is in the space defined by the feature representations of all second reactants). Specifically, the f network takes in the current state s (reactant-1 $R^{(1)}$) as input and outputs the best reaction template T. The π network then takes in both $R^{(1)}$ and T as inputs and outputs the continuous action a. The environment then takes in a and computes k closest valid second reactants ($R^{(2)}$). For each of these $R^{(2)}$s, it computes

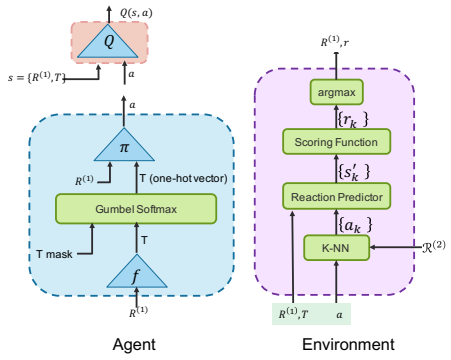

Figure 1: PGFS agent (comprising actor and critic modules) and environment.

the corresponding product of the chemical reaction between $R^{(1)}$ and $R^{(2)}$, computes the reward for the obtained product and chooses the product (next state, s_{t+1}) that corresponds to the highest reward. All these quantities are stored in the replay buffer. Thus, by formulating forward synthesis as a continuous action space RL problem, PGFS is able to leverage TD3 (Fujimoto, van Hoof, and Meger 2018) to update the actor (f, π) and critic (Q) networks.

Specifically, the authors used the following optimizations:

$$\min L(\theta^Q) = \frac{1}{N}\sum_i |y_i - Q(R_i^{(1)}, \{T_i, a_i\})|^2 \quad (1)$$

where, the parameters θ^Q of the critic (Q network) are updated by minimizing mean squared error between the critic value of the current state ($R^{(1)}$) action ($\{T_i, a_i\}$) pair and the 1-step TD target y_i

$$\min L(\theta^{f,\pi}) = -\sum_i \text{Critic}(R_i^{(1)}, \text{Actor}(R_i^{(1)})) \quad (2)$$

where, the parameters θ^f and θ^π of the actor networks f and π respectively are updated by minimizing the negative (equivalent to maximizing) of the critic's value of current state-action pair, where the actions T_i and a_i are computed by the actor networks.

$$\min L(\theta^f) = -\sum_i (T_i^{(1)}, log(f(R_i^{(1)}))) \quad (3)$$

where, the parameters θ^f of the f-network were also updated by minimizing the cross entropy loss between the output of the f-network $f(R_i^{(1)})$ and the actual valid template $T_i^{(1)}$ chosen. This is done to encourage the f-network to predict valid templates during the initial phases of training.

PGFS used two types of reactions (reaction templates): uni-molecular reactions (also called transformation reactions) and bi-molecular reactions. If we consider Equation 1, we notice that for both types of reactions, the same update rule is being used i.e, even for uni-molecular reactions that do not require an action a (because, uni-molecular reactions do not need a second reactant $R^{(2)}$), the critic is still evaluating such actions and using it to update its parameters. Similarly, in Equation 2, the parameters of the π network are getting updated even though its output is not used by the environment in case of uni-molecular reactions. Thus, every time a uni-molecular template is chosen by the f-network, the parameters of the critic and π networks are getting updated by arbitrary gradients. Since the percent of uni-molecular templates with respect to all available templates is sufficiently small (about 15 percent), there might not be any noticeable damage if the task under consideration doesn't need to utilize lot of uni-molecular templates. Nonetheless, the overall training becomes slower and we recognize this as a fundamental issue that couldn't be addressed with any of the existing RL algorithms.

TAC for *de novo* drug design

The environment (underlying MDP) in reaction based molecule generation assumes that the transition function is deterministic (i.e., the environment computes only the most probable product for a given state and action). Under this assumption, the value function of the next state $V(s_{i+1})$ is exactly equal to $Q(s_i, a_i)$ value of the current state s_i, action a_i pair. Also, we are interested in the quality (reward) of the product P (s_{i+1}) and not necessarily on the multiple ways in which the same product can be obtained using different chemical reactions (state-action pairs). Thus, drawing further inspiration from after state MDP (Sutton and Barto 2018) enables us to (hypothetically) break the "critic" down into a two step process by introducing two modules in the critic: product predictor module (that predicts the product P of the chemical reaction) and value function predictor module (that predicts the value function $V(s_{i+1})$ of the next state s_{i+1} i.e, of the product P). While the after state MDP learns the value functions of next states by computing all of the next states in the environment, we incorporate a (hypothetical) next state prediction module inside the critic and enable gradient propagation through this module to enable it to learn potentially more robust representations of the next state that are useful for the task under consideration. The critic architecture is also inspired from work in successor representations (Dayan 1993). Instead of representing the value function as a product of a reward vector and a successor representation matrix (or their corresponding learned variants (Barreto et al. 2017; Borsa et al. 2018; Hansen et al. 2019; Machado, Bellemare, and Bowling 2018)), we represent the action-value function $Q(s, a)$ as a composite function of the product prediction module (i.e, next state prediction module) and the value network. Thus, when the task is changed, (i.e, when the reward to be optimized is changed) the product prediction modules need not be retrained (analogous to how the successor representations need not be retrained always).

The product predictor module has two different networks, U-net (for processing uni-molecular reactions) and B-net (for processing bi-molecular reactions). U-net takes in $R^{(1)}$ and template T as inputs and computes the (hypothetical) product

$$P_u = U_{\theta_U}(R^{(1)}, T)$$

B-net takes $R^{(1)}$, and action a as inputs and computes the

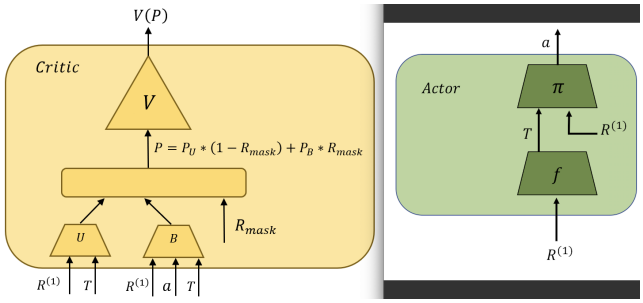

Figure 2: Illustration of TAC-FS in the context of reaction-based molecule generation for handling multiple action types (uni-molecular and bi-molecular templates)

(hypothetical) product:

$$P_b = B_{\theta_B}(R^{(1)}, a)$$

These are then combined by:

$$P = P_u \times (1 - R_{mask}) + P_b \times R_{mask},$$

where P represents the final (hypothetical) product of the chemical reaction and R_{mask} is a reaction mask that is equal to 1 when its a bi-molecular reaction and 0 if uni-molecular. The final predicted product obtained from these product predictor modules is passed through value network V to get the $Q(s, a)$.

Like in PGFS, the actor module takes in reactant at the current time step $R^{(1)}$ and outputs the best template T and action a. Additionally, the actor module in the proposed approach - TAC-FS (towered actor critic for forward synthesis) also identifies the type of reaction template (whether uni-molecular or bi-molecular) and outputs the corresponding binary mask R_{mask}. The f network computes T which is a tensor that contains the probability of each template. Invalid templates are masked off using a binary template mask T_{mask} (as, only a few reaction templates are valid for a given $R^{(1)}$). It is then passed through a gumbel softmax layer to obtain the template in one-hot tensor format, which indicates the final chosen template.

$$\begin{aligned} T &= f(R^{(1)}) \\ T &= T \odot T_{\text{mask}} \\ T &= \text{GumbelSoftmax}(T, \tau) \end{aligned} \tag{4}$$

Based on the chosen template, the environment identifies the type of template (either uni-molecular or bi-molecular). We then compute the action a by passing the reactant $R^{(1)}$ and chosen template T as inputs to the π network. Thus, the actor module finally returns T, a and R_{mask}.

During the "backward" phase, after sampling a random minibatch from the buffer, the networks are updated as follows: First, we compute the action a_{i+1}, template T_{i+1} and $R_{\text{mask}_{i+1}}$ for the next time step given state ($R^{(1)}_{i+1}$) using the actor module.

$$T_{i+1}, a_{i+1}, R_{\text{mask}_{i+1}} = \text{Actor-target}(R^{(1)}_{i+1}) \tag{5}$$

We then add clipped noise to the action output a_{i+1}

$$a_{i+1} = a_{i+1} + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c) \tag{6}$$

The one-step TD target can then be computed as:

$$y_i = r_i + \min_{j=1,2} \text{Critic-target}_j(R^{(1)}_{i+1}, \{T_{i+1}, R_{mask_{i+1}}, a_{i+1}\}) \tag{7}$$

The value loss computes the mean square error between the one-step TD target computed above and the $Q(s, a)$ value of current state $R^{(1)}_i$, action (T_i, a_i) pair. Using the critic, we compute the predicted product P_i and the $Q(s, a)$ value which are then used to compute the value loss and an auxiliary loss.

$$P_i, Q_i = \textsc{Critic}(R^{(1)}_i, \{T_i, R_{mask_i}, a_i\}) \tag{8}$$

$$\mathcal{L}_{\text{value}} = \sum_i |y_i - Q_i|^2 \tag{9}$$

We also ensure that the predicted product representation is almost the same as the actual product representation by minimizing an auxiliary loss which is mean squared error between these two representations:

$$\mathcal{L}_{\text{auxil}} = |P_i - R^{(1)}_{i+1}|^2, \tag{10}$$

where P_i is product predicted by the critic, and $R^{(1)}_{i+1}$ is the actual product. Thus, the overall critic loss is a weighted sum of the value loss and auxiliary loss

$$\mathcal{L}_{\text{critic}} = \mathcal{L}_{\text{value}} + \alpha \mathcal{L}_{\text{auxil}}, \tag{11}$$

where α is a hyper parameter.

The parameters of the critic networks (θ_U, θ_B, θ_V) are updated by minimizing the overall critic loss $\mathcal{L}_{\text{critic}}$. Note that we actually use two critics (to prevent overestimation bias, as elaborated by Fujimoto, van Hoof, and Meger (2018)), and follow the same steps in Equations 6-10 for both the critics to update their parameters.

The policy loss is defined as negative of the critic's value for the current state-action pair, where the actions are computed by the current version of the actor networks.

$$\mathcal{L}_{\text{policy}} = -\sum_i \textsc{Critic}(R^{(1)}_i, \textsc{Actor}(R^{(1)}_i)) \tag{12}$$

Similar to (Gottipati et al. 2020b), to enable faster training, we also aim to minimize an auxilary loss, which is the cross entropy loss between the template tensor predicted by the f-network and the corresponding template T obtained for the reactant $R^{(1)}$.

$$\mathcal{L}_{\text{auxil-actor}} = -\sum_i (T^{(1)}_i, log(f(R^{(1)}_i))) \tag{13}$$

The overall actor loss is a weighted sum of the policy loss $\mathcal{L}_{\text{policy}}$ and the actor's auxilary loss $\mathcal{L}_{\text{auxil-actor}}$:

$$\mathcal{L}_{\text{actor}} = \mathcal{L}_{\text{policy}} + \beta \mathcal{L}_{\text{auxil-actor}}, \tag{14}$$

where β is a hyper parameter. The parameters of the f and π networks are updated by minimizing the actor loss $\mathcal{L}_{\text{actor}}$. The pseudo-code for the entire algorithm is provided in Algorithm 1.

Algorithm 1 TAC-FS

procedure ACTOR($R^{(1)}$)
  $T \leftarrow f(R^{(1)})$
  $T \leftarrow T \odot T_{mask}$
  $T \leftarrow \text{GumbelSoftmax}(T, \tau)$
  $R_{mask} \leftarrow \text{Env.reaction\_type}(T)$
  $a \leftarrow \pi(R^{(1)}, T)$
  return T, a, R_{mask}
procedure CRITIC($R^{(1)}$, T,R_{mask} a)
  $P_u \leftarrow Unet(R^{(1)}, T)$
  $P_b \leftarrow Bnet(R^{(1)}, a, T)$
  $P \leftarrow P_u \times (1 - R_{mask}) + P_b \times R_{mask}$
  return $P, V(P)$
procedure ENV.STEP($R^{(1)}, T, a$)
  $\mathcal{R}^{(2)} \leftarrow \text{GetValidReactants}(T)$
  $\mathcal{A} \leftarrow \text{kNN}(a, \mathcal{R}^{(2)})$
  $\mathcal{R}^{(1)}_{t+1} \leftarrow \text{ForwardReaction}(R^{(1)}, T, \mathcal{A})$
  $\mathcal{R}ewards \leftarrow \text{ScoringFunction}(\mathcal{R}^{(1)}_{t+1})$
  $r_t, R^{(1)}_{t+1}, \textit{done} \leftarrow \arg\max \mathcal{R}ewards$
  return $R^{(1)}_{t+1}, r_t, \textit{done}$
procedure BACKWARD(buffer minibatch)
  $T_{i+1}, a_{i+1}, R_{\text{mask}_{i+1}} \leftarrow \text{Actor-target}(R^{(1)}_{i+1})$
  $a_{i+1} \leftarrow a_{i+1} + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
  $A_{i+1} \leftarrow \{T_{i+1}, R_{mask_{i+1}}, a_{i+1}\}$
  $P_{i+1}, Q_{i+1} \leftarrow \min_{j=1,2} \text{Critic-target}_j(R^{(1)}_{i+1}, A_{i+1})$
  $y_i \leftarrow r_i + \gamma Q_{i+1}$
  $P_i, Q_i \leftarrow \text{CRITIC}(R^{(1)}_i, \{T_i, R_{mask_i}, a_i\})$
  $\mathcal{L}_{\text{value}} \leftarrow \sum_i |y_i - Q_i)|^2$
  $\mathcal{L}_{\text{auxil}} \leftarrow |P_i - R^{(1)}_{i+1}|^2$
  $\mathcal{L}_{\text{critic}} \leftarrow \mathcal{L}_{\text{value}} + \alpha \mathcal{L}_{\text{auxil}}$
  $\mathcal{L}_{\text{policy}} \leftarrow -\sum_i \text{CRITIC}(R^{(1)}_i, \text{ACTOR}(R^{(1)}_i))$
  $\mathcal{L}_{\text{auxil-actor}} \leftarrow -\sum_i (T^{(1)}_i, log(f(R^{(1)}_i)))$
  $\mathcal{L}_{\text{actor}} \leftarrow \mathcal{L}_{\text{policy}} + \beta \mathcal{L}_{\text{auxil-actor}}$
  $\min \mathcal{L}_{\text{actor}}, \mathcal{L}_{\text{critic}}$
procedure MAIN(f, π, U, B, V)
  for episode = 1, M **do**
    sample $R^{(1)}_0$
    for t = 0, N **do**
      $T_t, a_t, R_{\text{mask}_t} \leftarrow \text{Actor}(R^{(1)}_t)$
      $R^{(1)}_{t+1}, r_t, \textit{done} \leftarrow \text{ENV.STEP}(R^{(1)}_t, T_t, a_t)$
      store $(R^{(1)}_t, T_t, a_t, R^{(1)}_{t+1}, r_t, R_{mask}, \textit{done})$ in buffer
      minibatch $\leftarrow$ random sample from buffer
      BACKWARD(*minibatch*)

TAC-generic

There are a few challenges/differences to directly applying the TAC formulation to environments that do not have/need an inherent hierarchical structure in action space. In TAC-generic, the MDP remains unchanged (i.e., the environment still takes in the same state and action as inputs and returns the same reward and next state as the original MDP). Only the actor and critic networks as well as the algorithm for updating these networks are being changed. Let us introduce a hypothetical action $a^{(1)}$ (analogous to T in TAC-FS) of dimensions the same as that of the actual action a. In the actor network, unlike the drug discovery environment, we do not need this other action type (hypothetical action) as inputs to the higher levels of network. Thus, we can re-structure the actor network as having some base layers followed by a head-network f_1, and one main network f_2 that outputs the real action a spawning from the output of a base network f (which can be interpreted as a robust representation of the state for the task under consideration). Thus, the actor module consists of the base network f that takes in state s as input and computes some representation $r = f(s)$. The first head-network f_1 takes in the representation r as input and computes the hypothetical action $a^{(1)}$. The other head network f_2 also takes in r as input and computes the action $a = f_2(r)$. Note that this hypothetical action $a^{(1)}$ is not used by the environment.

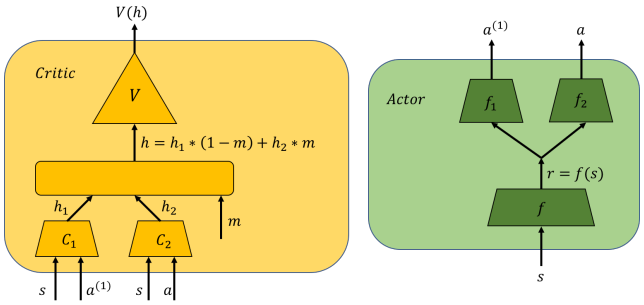

Figure 3: TAC-generic Architecture

We construct the critic network in such a way that it takes in state s, multiple action types ($a^{(1)}$ and a), and action mask M_a as inputs, and then computes the $Q(s, a)$ value. Let us consider a neural network C_1 that takes in the current state s and the hypothetical action $a^{(1)}$ as inputs and predicts the hypothetical next state (technically, the output of C_1 need not be in the space of next states. It could be of any dimensions and in any representation space):

$$h_1 = C_1(s, a^{(1)}).$$

One such "hypothetical next state prediction modules" (HyNeSP) can be assigned to process the action a as well:

$$h_2 = C_2(s, a).$$

The hypothetical next state h to be finally chosen from the two hypothetical next states h_1 and h_2 is determined by the action mask M_a which is a binary tensor. Similar to the drug discovery setting, we introduce a new network V that takes in a hypothetical next state h as input and predicts it's value function.

To summarize, the critic module takes in state s_t, hypothetical action $a^{(1)}$, action a and action mask M_a and computes the action value $Q(s, a)$. The C_i networks first predict the hypothetical next state, These are then combined using the

action mask as follows:

$$h = h_1 \times M_a + h_2 \times (1 - M_a).$$

This hypothetical next state h is then passed through the V-network to obtain the the corresponding value function $V(h)$.

After sampling a random minibatch from the buffer, the actor and critic networks are updated as follows: The actions for the next time step $a^{(1)}_{i+1}$ and a_{i+1} are computed by passing the next state s' through the target actor network:

$$a^{(1)}_{i+1}, a_{i+1} = \text{Actor-target}(s_{i+1}).$$

Unlike the drug discovery setting, the action mask M_a cannot be determined by the environment. Thus, we first set the action mask $M_a = 0$ and compute the actual one-step TD targets and then compute the corresponding value loss:

$$y_i = r_i + \gamma \min_{i=1,2} \text{critic-target}(s_{i+1}, a^{(1)}_{i+1}, a_{i+1}, [0])$$

$$\mathcal{L}_{\text{value}} = MSE(y_i, \textsc{Critic}(s_i, a^{(1)}_i, a_i, [0])),$$

where MSE is the mean squared error. We then set the action mask $M_a = 1$ and perform the same computations to get auxiliary loss $\mathcal{L}_{\text{critic-auxil}}$:

$$y_{i_{\text{auxil}}} = r_i + \gamma \min_{i=1,2} \text{critic-target}(s_{i+1}, a^{(1)}_{i+1}, a_{i+1}, [1])$$

$$\mathcal{L}_{\text{critic-auxil}} = MSE(y_{i_{\text{auxil}}}, \textsc{Critic}(s_i, a^{(1)}_i, a_i, [1])).$$

The overall critic loss is a weighted sum of value loss and critic's auxiliary loss:

$$\mathcal{L}_{\text{critic}} = \mathcal{L}_{\text{value}} + \beta \times \mathcal{L}_{\text{critic-auxil}},$$

where β is a hyper parameter that could also be learned (for example, in a meta learning setup). The parameters of the critic networks (C_1, C_2, V) are then updated by minimizing this critic loss $\mathcal{L}_{\text{critic}}$.

Similarly, the overall actor loss is computed as follows:

$$\mathcal{L}_{\text{policy}} = -\textsc{Critic}(s_i, \textsc{Actor}(s_i), [0])$$

$$\mathcal{L}_{\text{actor-auxil}} = -\textsc{critic}(s_i, \textsc{Actor}(s_i), [1])$$

$$\mathcal{L}_{\text{actor}} = \mathcal{L}_{\text{policy}} + \beta \times \mathcal{L}_{\text{actor-auxil}}.$$

All the actor networks (f, f_1, f_2) are updated by minimizing the actor loss $\mathcal{L}_{\text{actor}}$. The complete algorithm is summarized in Algorithm 2. While the entire discussion so far has been only focused on having two action types, the algorithm can be used for any number of layers and is detailed in the Appendix.

Experiments

We first elaborate on the results in drug discovery setting and then go over results on several OpenAI Gym environments.

Algorithm 2 TAC-generic

procedure ACTOR(s)
  $r \leftarrow base(s)$
  $a^{(1)} \leftarrow f_1(r)$
  $a \leftarrow f_2(r)$
  return $a^{(1)}, a$
procedure CRITIC(s, $a^{(1)}$, a M_a)
  $h^{(1)} \leftarrow C_1(a^{(1)}, s)$
  $h^{(2)} \leftarrow C_2(a, s)$
  $h = h^{(1)} \times M_a + h^{(2)} \times (1 - M_a)$
  return $V(h')$
procedure ENV.STEP(s_t, a_t)
  return s_{t+1}, r_t, *done*
procedure BACKWARD(buffer minibatch)
  $a^{(1)}_{i+1}, a_{i+1} \leftarrow$ Actor-target(s_{i+1})
  $y_i \leftarrow r_i + \gamma \min_{i=1,2}$ critic-target$(s_{i+1}, a^{(1)}_{i+1}, a_{i+1}, [0])$
  $\mathcal{L}_{\text{value}} \leftarrow MSE(y_i, \textsc{Critic}(s_i, a^{(1)}_i, a_i, [0]))$
  $y_{i_{\text{auxil}}} \leftarrow r_i + \gamma \min_{i=1,2}$ critic-target$(s_{i+1}, a^{(1)}_{i+1}, a_{i+1}, [1])$
  $\mathcal{L}_{\text{critic-auxil}} \leftarrow MSE(y_{i_{\text{auxil}}}, \textsc{Critic}(s_i, a^{(1)}_i, a_i, [1]))$
  $\mathcal{L}_{\text{critic}} = \mathcal{L}_{\text{value}} + \beta \times \mathcal{L}_{\text{critic-auxil}}$
  $\mathcal{L}_{\text{policy}} \leftarrow -\textsc{Critic}(s_i, \textsc{Actor}(s_i), [0])$
  $\mathcal{L}_{\text{actor-auxil}} \leftarrow -\textsc{critic}(s_i, \textsc{Actor}(s_i), [1])$
  $\mathcal{L}_{\text{actor}} \leftarrow \mathcal{L}_{\text{policy}} + \beta \times \mathcal{L}_{\text{actor-auxil}}$
  min $\mathcal{L}_{\text{actor}}, \mathcal{L}_{\text{critic}}$
procedure MAIN(f, f_1 , f_2, C_1, c_2, V)
  for episode = 1, M **do**
    for t = 0, N **do**
      $a^{(1)}_t, a_t \leftarrow \textsc{Actor}(s_t)$
      s_{t+1}, r_t, *done* $\leftarrow$ ENV.STEP(s_t, a_t)
      store $(s_t, a^{(1)}_t, a_t, s_{t+1}, r_t$, *done*$)$ in buffer
      minibatch $\leftarrow$ random sample from buffer
      BACKWARD(*minibatch*)

De Novo Drug Design

We compare the performance of the proposed TAC-FS algorithm with five of the existing approaches: GCPN - graph convolutional policy network (You et al. 2018a), JT-VAE - junction tree variational auto encoder (Jin, Barzilay, and Jaakkola 2018), MSO - molecular swarm optimization (Winter et al. 2019b), PGFS, RS - random search baseline also introduced by (Gottipati et al. 2020b), and the Enamine dataset containing all the initial reactants used in this study. It is important to note that only RS, PGFS and TAC-FS incorporates the reaction based synthesis in its molecule generation framework, thus ensuring that all the generated molecules are theoretically synthesizable based on the synthesis path that is simultaneously generated by these methods. While QED (measures drug-likeness) and clogP (measures lipophilicity) are the standard metrics that are used to measure the performance of any molecule generation algorithm, Gottipati et al. (2020b) introduced three new performance metrics (based on HIV targets) that attempt to mimic a real world use case of drug discovery. We thus compare the performance on all

these five reward metrics.

To ensure a fair comparison, TAC-FS is trained for only 400,000 time steps, same as that of PGFS and the results included in the Tables 1 and 2 were obtained by fixing $\alpha = 0$ and $\beta = 1$ for all the TAC-FS experiments. We noted much better performance of TAC-FS when the parameters α and β are tuned and are reported only in the Appendix. Finally, while in PGFS, the authors performed hyper-parameter tuning over the state feature representations, action feature representations, policy noise and noise clip, we tuned only for policy noise and noise clip parameters. In Table 1, we computed the highest reward achieved over the entire training process of 400,000 time steps. We can observe that TAC-FS improved over the existing state-of-the-art on all the tasks. In Table 2 we computed the mean and standard deviation of the top-100 rewards achieved over the entire training process. The values reported in Table 1 and Table 2 for ENAMINEBB, RS, GCPN(You et al. 2018a), JT-VAE(Jin, Barzilay, and Jaakkola 2018), MSO (Winter et al. 2019a), PGFS (Gottipati et al. 2020b) are taken from Gottipati et al. (2020b). While for HIV-RT, both TAC-FS and PGFS achieved similar results, we see that for HIV-INT and HIV-CCR5, TAC-FS performed significantly better than existing state-of-the-art. To further prove the effectiveness of the proposed approach, we provide the statistics of molecules obtained under the applicability domain of the models for HIV-targets, inference reward results, plots of actor loss, critic loss, auxiliary losses, rewards, inference rewards in the supplementary material.

Table 1: Comparison of maximum reward achieved over the entire course of training.

Method	QED	clogP	RT	INT	CCR5
ENAMINEBB	**0.948**	5.51	7.49	6.71	8.63
RS	**0.948**	8.86	7.65	7.25	8.86
GCPN	**0.948**	7.98	7.45	6.45	8.62
JT-VAE	0.925	5.30	7.58	7.25	8.23
MSO	**0.948**	26.10	7.76	7.28	8.77
PGFS	**0.948**	27.22	7.89	7.55	9.05
TAC-FS	**0.948**	**28.97**	**7.92**	**7.75**	**9.17**

Table 2: Comparison of mean (and standard deviation) of top-100 rewards achieved over the entire course of training

Method	RT	INT	CCR5
ENAMINEBB	6.87 ± 0.11	6.32 ± 0.12	7.10 ± 0.27
RS	7.39 ± 0.10	6.87 ± 0.13	8.65 ± 0.06
GCPN	7.07 ± 0.10	6.18 ± 0.09	7.99 ± 0.12
JT-VAE	7.20 ± 0.12	6.75 ± 0.14	7.60 ± 0.16
MSO	7.46 ± 0.12	6.85 ± 0.10	8.23 ± 0.24
PGFS	**7.81** ± 0.03	7.16 ± 0.09	8.96 ± 0.04
TAC-FS	7.79 ± 0.018	**7.54** ± 0.022	**9.07** ± 0.05

TAC-generic

We tested on the following environments: BipedalWalker-v3, LunarLanderContinous-v2, MountainCarContinous-v0, Pendulum-v0, Hopper (PyBullet), Ant (PyBullet), Reacher (PyBullet), Inverted Pendulum Swingup, Inverted Pendulum and Inverted Double Pendulum. For each environment, we ran the experiment for 1 million time steps and 5 random seeds. Evaluation reward is computed after every 5000 steps by taking the average reward achieved over 10 episodes, initialized with a fixed random seed that is not used during training. For a fair comparison, we did not perform any hyper-parameter tuning other than for the β parameter. The baseline comparison we used is the TD3 algorithm with exactly same network architecture as that of TAC, and exactly same parameters (discount factor, policy noise, noise clip etc..). Thus, the only difference is the β parameter. For TD3, it is fixed at zero, where as for TAC, it is tuned to compute an appropriate value for the task under consideration. As we can observe from Figure 4, TAC performed same or better than the baseline on all the 10 environments we tested.

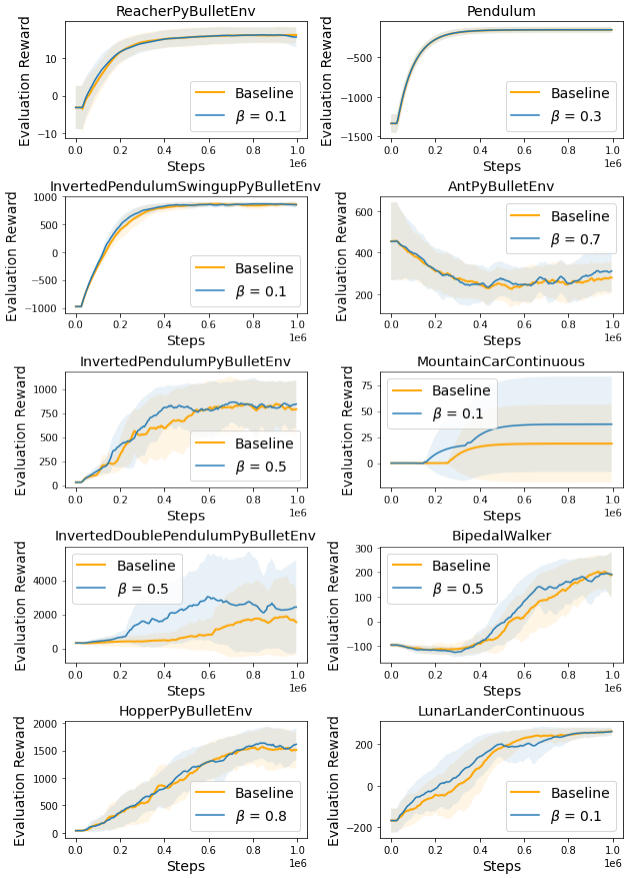

Figure 4: Performance comparison of generic TAC vs. TD3 averaged over 5 random seeds on 10 gym environments. We can observe that generic TAC performed better than, or same as TD3 on all the environments.

Conclusion and Future work

In this work, we introduced a novel RL algorithm that solved the severe limitation of the existing reaction based molecule generation framework and demonstrated better results than existing state-of-the-art on all the five tasks. We further ex-

amined the algorithm on MDPs that do not have any inherent hierarchical structure and demonstrated comparable or better performance over TD3 on a wide range of Gym environments. While this work was developed using the concepts of TD3 algorithm, there is a potential for improvement when some of the latest advances from deep RL literature are incorporated such as (Haarnoja et al. 2018; Kuznetsov et al. 2020; Wang et al. 2019; Schaul et al. 2015; Fedus et al. 2020; Sinha et al. 2020; Fujimoto, Meger, and Precup 2020). Another interesting avenue is to experiment on procedurally generated environments (Chevalier-Boisvert, Willems, and Pal 2018).

References

Ahn, S.; Kim, J.; Lee, H.; and Shin, J. 2020. Guiding Deep Molecular Optimization with Genetic Exploration.

Anthony, T.; Tian, Z.; and Barber, D. 2017. Thinking Fast and Slow with Deep Learning and Tree Search. *CoRR* abs/1705.08439. URL http://arxiv.org/abs/1705.08439.

Bacon, P.-L.; Harb, J.; and Precup, D. 2016. The Option-Critic Architecture.

Barreto, A.; Dabney, W.; Munos, R.; Hunt, J. J.; Schaul, T.; van Hasselt, H. P.; and Silver, D. 2017. Successor Features for Transfer in Reinforcement Learning. In Guyon, I.; Luxburg, U. V.; Bengio, S.; Wallach, H.; Fergus, R.; Vishwanathan, S.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 30*, 4055–4065. Curran Associates, Inc. URL http://papers.nips.cc/paper/6994-successor-features-for-transfer-in-reinforcement-learning.pdf.

Borsa, D.; Barreto, A.; Quan, J.; Mankowitz, D. J.; Munos, R.; van Hasselt, H.; Silver, D.; and Schaul, T. 2018. Universal Successor Features Approximators. *CoRR* abs/1812.07626. URL http://arxiv.org/abs/1812.07626.

Bradshaw, J.; Paige, B.; Kusner, M. J.; Segler, M. H. S.; and Hernández-Lobato, J. M. 2019. A Model to Search for Synthesizable Molecules. *CoRR* abs/1906.05221.

Brown, N.; Fiscato, M.; Segler, M. H.; and Vaucher, A. C. 2019. Guacamol: benchmarking models for de novo molecular design. *Journal of chemical information and modeling* 59(3): 1096–1108.

Brown, N.; McKay, B.; Gilardoni, F.; and Gasteiger, J. 2004. A graph-based genetic algorithm and its application to the multiobjective evolution of median molecules. *Journal of chemical information and computer sciences* 44(3): 1079–1087.

Chaplot, D. S.; Parisotto, E.; and Salakhutdinov, R. 2018. Active Neural Localization. In *International Conference on Learning Representations*. URL https://openreview.net/forum?id=ry6-G_66b.

Chevalier-Boisvert, M.; Willems, L.; and Pal, S. 2018. Minimalistic Gridworld Environment for OpenAI Gym. https://github.com/maximecb/gym-minigrid.

Dayan, P. 1993. Improving Generalization for Temporal Difference Learning: The Successor Representation. *Neural Computation* 5(4): 613–624.

Fedus, W.; Ramachandran, P.; Agarwal, R.; Bengio, Y.; Larochelle, H.; Rowland, M.; and Dabney, W. 2020. Revisiting Fundamentals of Experience Replay.

Fujimoto, S.; Meger, D.; and Precup, D. 2020. An Equivalence between Loss Functions and Non-Uniform Sampling in Experience Replay.

Fujimoto, S.; van Hoof, H.; and Meger, D. 2018. Addressing Function Approximation Error in Actor-Critic Methods. *CoRR* abs/1802.09477.

Gao, W.; and Coley, C. W. 2020. The Synthesizability of Molecules Proposed by Generative Models. *Journal of Chemical Information and Modeling* ISSN 1549-960X. doi:10.1021/acs.jcim.0c00174. URL http://dx.doi.org/10.1021/acs.jcim.0c00174.

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

Gottipati, S. K.; Pathak, Y.; Nuttall, R.; Sahir; Chunduru, R.; Touati, A.; Subramanian, S. G.; Taylor, M. E.; and Chandar, S. 2020a. Maximum Reward Formulation In Reinforcement Learning .

Gottipati, S. K.; Sattarov, B.; Niu, S.; Pathak, Y.; Wei, H.; Liu, S.; Thomas, K. M. J.; Blackburn, S.; Coley, C. W.; Tang, J.; Chandar, S.; and Bengio, Y. 2020b. Learning To Navigate The Synthetically Accessible Chemical Space Using Reinforcement Learning.

Griffiths, R.-R.; and Hernández-Lobato, J. M. 2020. Constrained Bayesian optimization for automatic chemical design using variational autoencoders. *Chem. Sci.* 11: 577–586. doi:10.1039/C9SC04026A. URL http://dx.doi.org/10.1039/C9SC04026A.

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. *CoRR* abs/1801.01290.

Hansen, S.; Dabney, W.; Barreto, A.; de Wiele, T. V.; Warde-Farley, D.; and Mnih, V. 2019. Fast Task Inference with Variational Intrinsic Successor Features. *CoRR* abs/1906.05030. URL http://arxiv.org/abs/1906.05030.

Jensen, J. H. 2019. A graph-based genetic algorithm and generative model/Monte Carlo tree search for the exploration of chemical space. *Chemical science* 10(12): 3567–3572.

Jin, W.; Barzilay, R.; and Jaakkola, T. 2018. Junction Tree Variational Autoencoder for Molecular Graph Generation. In Dy, J.; and Krause, A., eds., *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, 2323–2332. Stockholmsmässan, Stockholm Sweden: PMLR. URL http://proceedings.mlr.press/v80/jin18a.html.

Kahn, G.; Abbeel, P.; and Levine, S. 2020. Badgr: An autonomous self-supervised learning-based navigation system. *arXiv preprint arXiv:2002.05700* .

Korovina, K.; Xu, S.; Kandasamy, K.; Neiswanger, W.; Poczos, B.; Schneider, J.; and Xing, E. P. 2019. ChemBO: Bayesian Optimization of Small Organic Molecules with Synthesizable Recommendations. *arXiv:1908.01425 [physics, stat]* URL http://arxiv.org/abs/1908.01425. ArXiv: 1908.01425.

Kumar, P.; and Precup, D. 2017. Multi-Timescale, Gradient Descent, Temporal Difference Learning with Linear Options. *ArXiv* abs/1703.06471.

Kuznetsov, A.; Shvechikov, P.; Grishin, A.; and Vetrov, D. 2020. Controlling Overestimation Bias with Truncated Mixture of Continuous Distributional Quantile Critics.

Lakshminarayanan, A. S.; Krishnamurthy, R.; Kumar, P.; and Ravindran, B. 2016. Option Discovery in Hierarchical Reinforcement Learning using Spatio-Temporal Clustering. *arXiv: Learning* .

Levy, A.; Konidaris, G.; Platt, R.; and Saenko, K. 2017. Learning Multi-Level Hierarchies with Hindsight.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N. M. O.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2015. Continuous control with deep reinforcement learning. *CoRR* abs/1509.02971.

Machado, M. C.; Bellemare, M. G.; and Bowling, M. 2018. Count-Based Exploration with the Successor Representation. *CoRR* abs/1807.11622. URL http://arxiv.org/abs/1807.11622.

Masson, W.; and Konidaris, G. D. 2015. Reinforcement Learning with Parameterized Actions. *CoRR* abs/1509.01644. URL http://arxiv.org/abs/1509.01644.

Olivecrona, M.; Blaschke, T.; Engkvist, O.; and Chen, H. 2017. Molecular de-novo design through deep reinforcement learning. *Journal of cheminformatics* 9(1): 48.

OpenAI; Akkaya, I.; Andrychowicz, M.; Chociej, M.; Litwin, M.; McGrew, B.; Petron, A.; Paino, A.; Plappert, M.; Powell, G.; Ribas, R.; Schneider, J.; Tezak, N.; Tworek, J.; Welinder, P.; Weng, L.; Yuan, Q.; Zaremba, W.; and Zhang, L. 2019. Solving Rubik's Cube with a Robot Hand.

Popova, M.; Isayev, O.; and Tropsha, A. 2018. Deep reinforcement learning for de novo drug design. *Science advances* 4(7): eaap7885.

Röder, F.; Eppe, M.; Nguyen, P. D. H.; and Wermter, S. 2020. Curious Hierarchical Actor-Critic Reinforcement Learning.

Schaul, T.; Quan, J.; Antonoglou, I.; and Silver, D. 2015. Prioritized Experience Replay.

Segler, M. H.; Kogej, T.; Tyrchan, C.; and Waller, M. P. 2017. Generating focused molecule libraries for drug discovery with recurrent neural networks. *ACS central science* 4(1): 120–131.

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; Lillicrap, T. P.; Simonyan, K.; and Hassabis, D. 2017. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. *CoRR* abs/1712.01815. URL http://arxiv.org/abs/1712.01815.

Silver, D.; Lever, G.; Heess, N.; Degris, T.; Wierstra, D.; and Riedmiller, M. 2014. Deterministic Policy Gradient Algorithms. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, I–387–I–395. JMLR.org.

Simonovsky, M.; and Komodakis, N. 2018. Graphvae: Towards generation of small graphs using variational autoencoders. In *International Conference on Artificial Neural Networks*, 412–422. Springer.

Sinha, S.; Song, J.; Garg, A.; and Ermon, S. 2020. Experience Replay with Likelihood-free Importance Weights.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement Learning: An Introduction*. The MIT Press, second edition. URL http://incompleteideas.net/book/the-book-2nd.html.

Udrescu, S.-M.; and Tegmark, M. 2020. AI Feynman: A physics-inspired method for symbolic regression. *Science Advances* 6(16). doi:10.1126/sciadv.aay2631. URL https://advances.sciencemag.org/content/6/16/eaay2631.

Wang, C.; Wu, Y.; Vuong, Q.; and Ross, K. 2019. Striving for Simplicity and Performance in Off-Policy DRL: Output Normalization and Non-Uniform Sampling.

Winter, R.; Montanari, F.; Steffen, A.; Briem, H.; Noé, F.; and Clevert, D.-A. 2019a. Efficient multi-objective molecular optimization in a continuous latent space. *Chemical science* 10(34): 8016–8024.

Winter, R.; Montanari, F.; Steffen, A.; Briem, H.; Noé, F.; and Clevert, D.-A. 2019b. Efficient multi-objective molecular optimization in a continuous latent space. *Chem. Sci.* 10: 8016–8024. doi:10.1039/C9SC01928F. URL http://dx.doi.org/10.1039/C9SC01928F.

You, J.; Liu, B.; Ying, R.; Pande, V. S.; and Leskovec, J. 2018a. Graph Convolutional Policy Network for Goal-Directed Molecular Graph Generation. *CoRR* abs/1806.02473. URL http://arxiv.org/abs/1806.02473.

You, J.; Liu, B.; Ying, Z.; Pande, V.; and Leskovec, J. 2018b. Graph convolutional policy network for goal-directed molecular graph generation. In *Advances in neural information processing systems*, 6410–6421.

Zheng, S.; Trott, A.; Srinivasa, S.; Naik, N.; Gruesbeck, M.; Parkes, D. C.; and Socher, R. 2020. The AI Economist: Improving Equality and Productivity with AI-Driven Tax Policies. *arXiv preprint arXiv:2004.13332* .

Zhou, Z.; Kearnes, S. M.; Li, L.; Zare, R. N.; and Riley, P. 2018. Optimization of Molecules via Deep Reinforcement Learning. *CoRR* abs/1810.08678. URL http://arxiv.org/abs/1810.08678.