Discrete Action On-Policy Learning with Action-Value Critic: Supplementary Material

A Proof of Theorem 1

We first show the sparse ARSM for multidimensional action space case at one specific time point, then generalize it to stochastic setting. Since a_k are conditionally independent given $\boldsymbol{\phi}_k$, the gradient of $\boldsymbol{\phi}_{kc}$ at one time point would be (we omit the subscript t for simplicity here)

$$\nabla_{\phi_{kc}} J(\boldsymbol{\phi}) = \mathbb{E}_{\boldsymbol{a}_{\setminus k}\sim\prod_{k'\neq k}\text{Discrete}(a_{k'};\sigma(\boldsymbol{\phi}_{k'}))}\left[\nabla_{\phi_{kc}}\mathbb{E}_{a_k\sim\text{Cat}(\sigma(\boldsymbol{\phi}_k))}[Q(\boldsymbol{a},\boldsymbol{s})]\right],$$

and we apply the ARSM gradient estimator on the inner expectation part, which gives us

$$\begin{aligned}\nabla_{\phi_{kc}} J(\boldsymbol{\phi}) &= \mathbb{E}_{\boldsymbol{a}_{\setminus k}\sim\prod_{k'\neq k}\text{Discrete}(a_{k'};\sigma(\boldsymbol{\phi}_{k'}))}\left\{\mathbb{E}_{\boldsymbol{\varpi}_k\sim\text{Dir}(\mathbf{1}_C)}\left[(Q([\boldsymbol{a}_{\setminus k},a_k^{c=j}],\boldsymbol{s})-\frac{1}{C}\sum_{m=1}^{C}Q([\boldsymbol{a}_{\setminus k},a_k^{m=j}],\boldsymbol{s}))(1-C\varpi_{kj})\right]\right\}\\ &= \mathbb{E}_{\boldsymbol{\varpi}_k\sim\text{Dir}(\mathbf{1}_C)}\left\{\mathbb{E}_{\boldsymbol{a}_{\setminus k}\sim\prod_{k'\neq k}\text{Discrete}(a_{k'};\sigma(\boldsymbol{\phi}_{k'}))}\left[(Q([\boldsymbol{a}_{\setminus k},a_k^{c=j}],\boldsymbol{s})-\frac{1}{C}\sum_{m=1}^{C}Q([\boldsymbol{a}_{\setminus k},a_k^{m=j}],\boldsymbol{s}))(1-C\varpi_{kj})\right]\right\} \quad (6)\\ &= \mathbb{E}_{\boldsymbol{\varpi}_k\sim\text{Dir}(\mathbf{1}_C)}\left\{\mathbb{E}_{\prod_{k'\neq k}\text{Dir}(\boldsymbol{\varpi}_{k'};\mathbf{1}_C)}\left[(Q(\boldsymbol{a}^{c=j},\boldsymbol{s})-\frac{1}{C}\sum_{m=1}^{C}Q(\boldsymbol{a}^{m=j},\boldsymbol{s}))(1-C\varpi_{kj})\right]\right\}, \quad (7)\end{aligned}$$

where (6) is derived by changing the order of two expectations and (7) can be derived by following the proof of Proposition 5 in Yin et al. [2019]. Therefore, if given $\boldsymbol{\varpi}_k\sim\text{Dir}(\mathbf{1}_C)$, it is true that $a_k^{c=j}=a_k$ for all (c,j) pairs, then the inner expectation term in (6) will be zero and consequently we have

$$g_{kc}=0$$

as an unbiased single sample estimate of $\nabla_{\phi_{kc}}J(\boldsymbol{\phi})$; If given $\boldsymbol{\varpi}_k\sim\text{Dir}(\mathbf{1}_C)$, there exist (c,j) that $a_k^{c=j}\neq a_k$, we can use (7) to provide

$$g_{kc}=\sum_{j=1}^{C}\left[Q(\boldsymbol{s},\boldsymbol{a}^{c=j})-\frac{1}{C}\sum_{m=1}^{C}Q(\boldsymbol{s},\boldsymbol{a}^{m=j})\right]\left(\frac{1}{C}-\varpi_{kj}\right) \quad (8)$$

as an unbiased single sample estimate of $\nabla_{\phi_{kc}}J(\boldsymbol{\phi})$.

For a specific time point t, the objective function can be decomposed as

$$\begin{aligned}J(\boldsymbol{\phi}_{0:\infty}) &= \mathbb{E}_{\mathcal{P}(\boldsymbol{s}_0)\left[\prod_{t'=0}^{t-1}\mathcal{P}(\boldsymbol{s}_{t'+1}\mid\boldsymbol{s}_{t'},\boldsymbol{a}_{t'})\text{Cat}(\boldsymbol{a}_{t'};\sigma(\boldsymbol{\phi}_{t'}))\right]}\left\{\mathbb{E}_{\boldsymbol{a}_t\sim\text{Cat}(\sigma(\boldsymbol{\phi}_t))}\left[\sum_{t'=0}^{t-1}\gamma^{t'}r(\boldsymbol{s}_{t'},\boldsymbol{a}_{t'})+\gamma^t Q(\boldsymbol{s}_t,\boldsymbol{a}_t)\right]\right\}\\ &= \mathbb{E}_{\mathcal{P}(\boldsymbol{s}_0)\left[\prod_{t'=0}^{t-1}\mathcal{P}(\boldsymbol{s}_{t'+1}\mid\boldsymbol{s}_{t'},\boldsymbol{a}_{t'})\text{Cat}(\boldsymbol{a}_{t'};\sigma(\boldsymbol{\phi}_{t'}))\right]}\left\{\mathbb{E}_{\boldsymbol{a}_t\sim\text{Cat}(\sigma(\boldsymbol{\phi}_t))}\left[\sum_{t'=0}^{t-1}\gamma^{t'}r(\boldsymbol{s}_{t'},\boldsymbol{a}_{t'})\right]\right\}\\ &\quad+\mathbb{E}_{\mathcal{P}(\boldsymbol{s}_0)\left[\prod_{t'=0}^{t-1}\mathcal{P}(\boldsymbol{s}_{t'+1}\mid\boldsymbol{s}_{t'},\boldsymbol{a}_{t'})\text{Cat}(\boldsymbol{a}_{t'};\sigma(\boldsymbol{\phi}_{t'}))\right]}\left\{\mathbb{E}_{\boldsymbol{a}_t\sim\text{Cat}(\sigma(\boldsymbol{\phi}_t))}\left[\gamma^t Q(\boldsymbol{s}_t,\boldsymbol{a}_t)\right]\right\},\end{aligned}$$

where the first part has nothing to do with $\boldsymbol{\phi}_t$, we therefore have

$$\nabla_{\boldsymbol{\phi}_{tkc}}J(\boldsymbol{\phi}_{0:\infty})=\mathbb{E}_{\mathcal{P}(\boldsymbol{s}_t\mid\boldsymbol{s}_0,\pi_{\boldsymbol{\theta}})\mathcal{P}(\boldsymbol{s}_0)}\left\{\gamma^t\nabla_{\boldsymbol{\phi}_{tkc}}\mathbb{E}_{\boldsymbol{a}_t\sim\text{Cat}(\sigma(\boldsymbol{\phi}_t))}\left[Q(\boldsymbol{s}_t,\boldsymbol{a}_t)\right]\right\}$$

.

With the result from (8), the statements in Theorem1 follow.

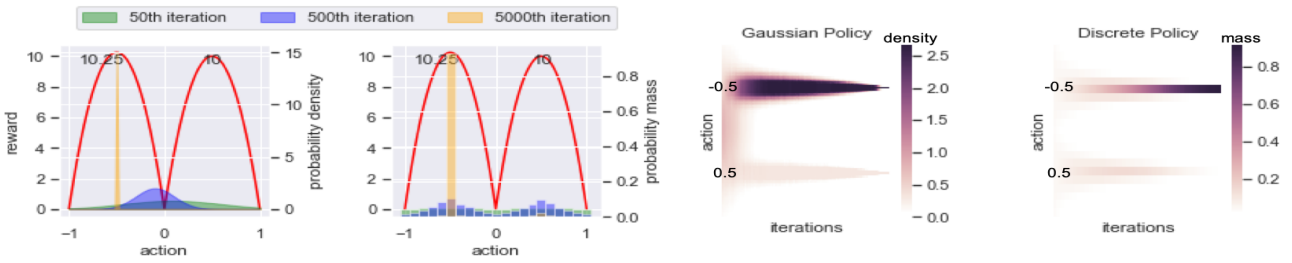

Figure 6: **left panel**: Change of policy over iterations between Gaussian policy (left) and discrete policy (right) on toy example setting. **right panel**: Average density on each action along with the training iterations between Gaussian policy and discrete policy for 100 experiments.(The Gaussian policy converges to the inferior optimal solution 12 times out of 100 times, and discrete policy converges to the global optimum all the time).

B Experiment setup

B.1 Toy example setup

Assume the true reward is a bi-modal distribution (as shown in Figure 6 left panel red curves) with a difference between its two peaks:

$$r(a) = \begin{cases} -c_1(a-1)(a-m) + \epsilon_1 & \text{for } a \in [m, 1] \\ -c_2(a+1)(a-m) + \epsilon_2 & \text{for } a \in [-1, m], \end{cases}$$

where the values of c_1, c_2, and m determine the heights and widths of these two peaks, and $\epsilon_1 \sim \mathrm{N}(0, 2)$ and $\epsilon_2 \sim \mathrm{N}(0, 1)$ are noise terms. It is clear that $a^*_{\text{left}} = (m-1)/2$ and $a^*_{\text{right}} = (1+m)/2$ are two local-optimal solutions and corresponding to $r_{\text{left}} := \mathbb{E}[r((a^*_{\text{left}})] = c_2(1+m)^2/4$ and $r_{\text{right}} := \mathbb{E}[r(a^*_{\text{right}})] = c_1(1-m)^2/4$. Here we always choose c_1 and c_2 such that r_{left} is slightly bigger than r_{right} which makes a^*_{left} a better local-optimal solution. It is clear that the more closer a^*_{left} to -1, the more explorations a policy will need to converge to a^*_{left}. Moreover, the noise terms can give wrong signals and may lead to bad update directions, and exploration will play an essential role in preventing the algorithm from acting too greedily. The results shown on Section 4.1 has $m = -0.8$, $c_1 = 40/(1.8^2)$ and $c_2 = 41/(0.2^2)$, which makes $r_{\text{left}} = 10.25$ and $r_{\text{right}} = 10$. We also show a simple example at Figure 6 with $m = 0$, $c_1 = 40/(0.5^2)$ and $c_2 = 41/(0.5^2)$, which maintains the same peak values.

The experiment setting is as follows: for each episode, we collect 100 samples and update the corresponding parameters ($[\mu, \sigma]$ for Gaussian policy and $\boldsymbol{\phi} \in \mathbb{R}^{21}$ for discrete policy where the action space is discretized to 21 actions), and iterate until N samples are collected. We add a quadratic decaying coefficient for the entropy term for both policies to encourage explorations on an early stage. The Gaussian policy is updated using reparametrization trick [Kingma and Welling, 2013], which can be applied to this example since we know the derivative of the reward function (note this is often not the case for RL tasks). The discrete policy is updated using ARSM gradient estimator described in Section 2.

On the heatmap, the horizontal axis is the iterations, and vertical axis denotes the actions. For each entry corresponding to a at iteration i, its value is calculated by $v(i, a) = \frac{1}{U}\sum_{u=1}^{U} p_u(a \,|\, i)$, where $p_u(a \,|\, i)$ is the probability of taking action a at iteration i for that policy in uth trial.

We run the same setting with different seeds for Gaussian policy and discrete policy for 100 times, where the initial parameters for Gaussian Policy is $\mu_0 = m, \sigma = 1$ and for discrete policy is $\phi_i = 0$ for any i to eliminate the effects of initialization.

In those 100 trials, when $m = -0.8, N = 1e^6$, Gaussian policy fails to find the true global optimal solution (0/100) while discrete policy can always find that optimal one (100/100). When $m = 0, N = 5e^5$, the setting is easier and Gaussian policy performs better in this case with only 12/100 percentage converging to the inferior sub-optimal point 0.5, and the rest 88/100 chances getting to global optimal solution. On the other hand, discrete policy always converges to the global optimum (100/100). The similar plots are shown on Figure 6. The *p-value* for this proportion test is 0.001056, which shows strong evidence that discrete policy outperforms Gaussian policy on this example.

B.2 Baselines and CARSM setup

Our experiments aim to answer the following questions: **(a)** How does the proposed CARSM algorithm perform when compared with ARSM-MC (when ARSM-MC is not too expensive to run). **(b)** Is CARSM able to efficiently solve tasks with large discrete action spaces (i.e., C is large). **(c)** Does CARSM have better sample efficiency than the algorithms, such as A2C and RELAX, that have the same idea of using baselines for variance reduction. **(d)** Can CARSM combined with other standard algorithms such as TRPO to achieve a better performance.

Baselines and Benchmark Tasks. We evaluate our algorithm on benchmark tasks on OpenAI Gym classic-control and MuJoCo tasks [Todorov et al., 2012]. We compare the proposed CARSM with ARSM-MC [Yin et al., 2019], A2C [Mnih et al., 2016], and RELAX [Grathwohl et al., 2017]; all of them rely on introducing baseline functions to reduce gradient variance, making it fair to compare them against each other. We then integrate CARSM into TRPO by replacing the A2C gradient estimator for $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$, and evaluate the performances on MuJoCo tasks to show that a simple plug-in of the CARSM estimator can bring the improvement.

Hyper-parameters: Here we detail the hyper-parameter settings for all algorithms. Denote β_{policy} and β_{critic} as the learning rates for policy parameters and Q critic parameters, respectively, n_{critic} as the number of training time for Q critic, and α as the coefficient for entropy term. For CARSM, we select the best learning rates $\beta_{\text{policy}},\ \beta_{\text{critic}} \in \{1, 3\} \times 10^{-2}$, and $n_{\text{critic}} \in \{50, 150\}$; For A2C and RELAX, we select the best learning rates $\beta_{\text{policy}} \in \{3, 30\} \times 10^{-5}$. In practice, the loss function consists of a policy loss L_{policy} and value function loss L_{value}. The policy/value function are optimized jointly by optimizing the aggregate objective at the same time $L = L_{\text{policy}} + cL_{\text{value}}$, where $c = 0.5$. Such joint optimization is popular in practice and might be helpful in cases where policy/value function share parameters. For A2C, we apply a batched optimization procedure: at iteration t, we collect data using a previous policy iterate π_{t-1}. The data is used for the construction of a differentiable loss function L. We then take v_{iter} gradient updates over the loss function objective to update the parameters, arriving at π_t. In practice, we set $v_{\text{iter}} = 10$. For TRPO and TRPO combined with CARSM, we use max KL-divergence of 0.01 all the time without tuning. All algorithms use a initial α of 0.01 and decrease α exponentially, and target network parameter τ is 0.01. To guarantee fair comparison, we only apply the tricks that are related to each algorithm and didn't use any general ones such as normalizing observation. More specifically, we replace Advantage function with normalized Generalized Advantage Estimation (GAE) [Schulman et al., 2015b] on A2C, apply normalized Advantage on RELAX.

Structure of Q critic networks: There are two common ways to construct a Q network. The first one is to model the network as $Q : \mathbb{R}^{n_S} \to \mathbb{R}^{|\mathcal{A}|}$, where n_S is the state dimension and $|\mathcal{A}| = C^K$ is the number of unique actions. The other structure is $Q : \mathbb{R}^{n_S+K} \to \mathbb{R}$, which means we need to concatenate the state vector $\boldsymbol{s}$ with action vector $\boldsymbol{a}$ and feed that into the network. The advantage of first structure is that it doesn't involve the issue that action vector and state vector are different in terms of scale, which may slow down the learning process or make it unstable. However, the first option is not feasible under most multidimensional discrete action situations because the number of actions grow exponentially along with the number of dimension K. Therefore, we apply the second kind of structure for Q network, and update Q network multiple times before using it to obtain the CARSM estimator to stabilize the learning process.

Structure of policy network: The policy network will be a function of $\mathcal{T}_{\boldsymbol{\theta}} : \mathbb{R}^{n_S} \to \mathbb{R}^{K \times C}$, which feed in state vector $\boldsymbol{s}$ and generate $K \times C$ logits ϕ_{kc}. Then the action is obtained for each dimension k by $\pi(a_k \mid \boldsymbol{s}, \boldsymbol{\theta}) = \sigma(\boldsymbol{\phi}_k)$, where $\boldsymbol{\phi}_k = (\phi_{k1}, \ldots, \phi_{kC})'$. For both the policy and Q critic networks, we use a two-hidden-layer multilayer perceptron with 64 nodes per layer and tanh activation.

Environment setup

- *HalfCheetah* ($\mathcal{S} \subset \mathbb{R}^{17}, \mathcal{A} \subset \mathbb{R}^6$)
- *Hopper* ($\mathcal{S} \subset \mathbb{R}^{11}, \mathcal{A} \subset \mathbb{R}^3$)
- *Reacher* ($\mathcal{S} \subset \mathbb{R}^{11}, \mathcal{A} \subset \mathbb{R}^2$)
- *Swimmer* ($\mathcal{S} \subset \mathbb{R}^8, \mathcal{A} \subset \mathbb{R}^2$)
- *Walker2D* ($\mathcal{S} \subset \mathbb{R}^{17}, \mathcal{A} \subset \mathbb{R}^6$)
- *LunarLander Continuous* ($\mathcal{S} \subset \mathbb{R}^8, \mathcal{A} \subset \mathbb{R}^2$)

B.3 Comparison between CARSM and ARSM-MC for fixed timestep

We compare ARSM-MC and CARSM for fixed timestep setting, with their performances shown in Figure 7

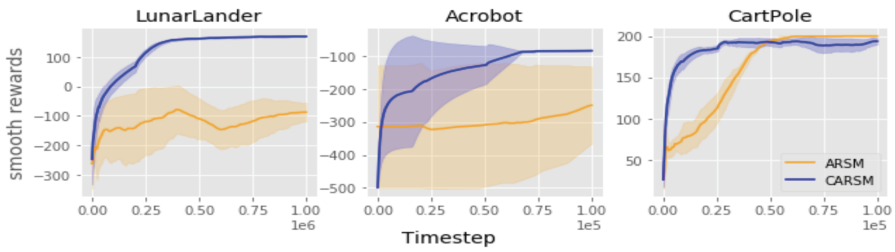

Figure 7: Performance curves for comparison between ARSM-MC and CARSM given fix timesteps

C Pseudo Code

We provide detailed pseudo code to help understand the implementation of CARSM policy gradient. There are four major steps for each update iteration: (1) Collecting samples using augmented Dirichlet variables $\boldsymbol{\varpi}_t$; (2) Update the Q critic network using both on-policy samples and off-policy samples; (3) Calculating the CARSM gradient estimator; (4) soft updating the target networks for both the policy and critic. The (1) and (3) steps are different from other existing algorithms and we show their pseudo codes in Algorithms 1 and 2, respectively.

Algorithm 1: Collecting samples from environment

Input: Policy network $\pi(\boldsymbol{a} \mid \boldsymbol{s}, \boldsymbol{\theta})$, initial state $\boldsymbol{s}_0$, sampled step T, replay buffer $\mathcal{R}$

Output: Intermediate variable matrix $\boldsymbol{\varpi}_{1:T}$, logit variables $\boldsymbol{\phi}_{1:T}$, rewards vector $r_{1:T}$, state vectors $\boldsymbol{s}_{1:T}$, action vectors $\boldsymbol{a}_{1:T}$, replay buffer $\mathcal{R}$

for $t = 1 \cdots T$ **do**

- Generate Dirichlet random variable $\boldsymbol{\varpi}_{tk} \sim \mathrm{Dir}(\mathbf{1}_C)$ for each dimension k;
- Calculate logits $\boldsymbol{\phi}_t = \mathcal{T}_{\boldsymbol{\theta}}(\boldsymbol{s}_t)$ which is a $K \times C$ length vector
- Select action $a_{tk} = \mathrm{argmin}_{i \in \{1, \cdots, C\}}(\ln \varpi_{tki} - \phi_{tki})$ for each dimension k;
- Obtain next state values $\boldsymbol{s}_{t+1}$ and reward r_t based actions $\boldsymbol{a}_t = (a_{t1}, \ldots, a_{tK})'$ and current state $\boldsymbol{s}_t$.
- Store the transition $\{\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1}\}$ to replay buffer $\mathcal{R}$
- Assign $\boldsymbol{s}_t \leftarrow \boldsymbol{s}_{t+1}$.

end for

Algorithm 2: CARSM policy gradient for a K-dimensional C-way categorical action space.

Input: Critic network $Q_{\boldsymbol{\omega}}$, policy network $\pi_{\boldsymbol{\theta}}$, on-policy samples including states $\boldsymbol{s}_{1:T}$, actions $\boldsymbol{a}_{1:T}$, intermediate Dirichlet random variables $\boldsymbol{\varpi}_{1:T}$, logits vectors $\boldsymbol{\phi}_{1:T}$, discounted cumulative rewards $y_{1:T}$.

Output: an updated policy network

Initialize $g \in \mathbb{R}^{T\times K\times C}$;

for $t = 1\cdots\ T$ *(in parallel)* **do**

  for $k = 1\cdots\ K$ *(in parallel)* **do**

    Let $A_{tk} = \{(c,j)\}_{c=1:C,\ j<c}$, and initialize $P^{tk} \in \mathbb{R}^{C\times C}$ with all element equals to a_{tk} (true action).

    for $(c,j) \in A_{tk}$ *(in parallel)* **do**

      Let $a_{tk}^{c\rightleftharpoons j} = \arg\min_{i\in\{1,\ldots,C_k\}}(\ln \varpi_{tki}^{c\rightleftharpoons j} - \phi_{tki})$

      if $a_{tk}^{c\rightleftharpoons j}$ *not equals to* a_{tk} **then**

        Assign $a_{tk}^{c\rightleftharpoons j}$ to $P^{tk}(c,j)$

      end if

    end for

  end for

  Let $S_t = \text{unique}(P^{t1}\otimes P^{t2}\cdots\otimes P^{tK})\backslash\{a_{t1}\otimes a_{t2}\cdots\otimes a_{tK}\}$, which means S_t is the set of all unique values across K dimensions except for true action $\boldsymbol{a}_t = \{a_{t1}\otimes a_{t2}\cdots\otimes a_{tK}\}$; denote pseudo action of swapping between coordinate c and j as $S_t(c,j) = (P^{t1}(c,j)\otimes P^{t1}(c,j)\cdots\otimes P^{tK}(c,j))$, and define $\mathcal{I}_t$ as unique pairs contained in S_t.

  Initialize matrix $F^t \in \mathbb{R}^{C\times C}$ with all elements equal to y_t;

  for $(\tilde{c},\tilde{j}) \in \mathcal{I}_t$ *(in parallel)* **do**

    $F^t(\tilde{c},\tilde{j}) = Q_{\boldsymbol{\omega}}(\boldsymbol{s}_t, S_t(\tilde{c},\tilde{j}))$

  end for

  Plug in number for matrix $g_{tkc} = \sum_{j=1}^{C}(F_c^t - \bar{F}_c^t)(\frac{1}{C} - \varpi_{tkj})$, where F_c^{tk} denotes the cth row of matrix F^t and $\bar{F}_c^t$ is the mean of that row;

  for $k = 1\cdots K$ **do**

    if *every element in* P^{tk} *is* a_{tk} **then**

      $g_{tkc} = 0$

    end if

  end for

end for

Update the parameter for $\boldsymbol{\theta}$ for policy network by maximize the function

$$J = \frac{1}{TKC}\sum_{t=1}^{T}\sum_{k=1}^{K}\sum_{c=1}^{C} g_{tkc}\phi_{tkc}$$

where ϕ_{tkc} are logits and g_{tkc} are placeholders that stop any gradients, and use auto-differentiation on ϕ_{tkc} to obtain gradient with respect to $\boldsymbol{\theta}$.