Copyright
by
Yuguang Yue
2021

The Dissertation Committee for Yuguang Yue
certifies that this is the approved version of the following dissertation:

Boosting Deep Reinforcement Learning Algorithms with Deep Probabilistic Models

Committee:

Mingyuan Zhou, Supervisor

Peter Müller

Abhra Sarkar

Xiaoning Qian

Boosting Deep Reinforcement Learning Algorithms with Deep Probabilistic Models

by

Yuguang Yue

DISSERTATION
Presented to the Faculty of the Graduate School of
The University of Texas at Austin
in Partial Fulfillment
of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN
May 2021

Dedicated to my loved ones.

Acknowledgments

A PhD journey is always mixed with toil and sweat, but the joy and satisfaction that come along this pursuit of knowledge are unparalleled. I am fortunate enough to receive the endless support from my advisor, colleagues and friends during the study at the University of Texas at Austin, who make this experience more cherishable and unforgettable.

It is not possible to express my gratitude to my advisor, Dr. Mingyuan Zhou, in a couple of words; his patience, enthusiasm and wisdom make me reach further than I thought I could go. I still remember the guidance and encourages he gave me when I feel disappointed at the moment I fist stepped into the reinforcement learning realm, and will never forget the sleepless nights we worked together to polish papers for conferences. He makes me realize the incomparable satisfaction of solving puzzles, and his honest and rigorous attitude toward science will affect me everlasting. In addition to the academic life, Mingyuan is also a mentor who cares about my daily life. I will not forget the kind helps he offered during the unprecedented pandemic and the snow storm, and I am really grateful for his supports toward my personal choices.

I would like to thank Dr. Peter Müller and Dr. Stephen Walker for their insightful classes that triggered my interest toward Bayesian statistics in the first place; I am also grateful to have many excellent collaborators: Dr.

Mingzhang Yin, Yunhao Tang, Xinjie Fan, Dr. Purnamrita Sarkar, Dr. Rachel Wang, Dr. Wenyuan Li, Zhendong Wang, Dr. Mingyuan Zhou; it was a great pleasure working and learning with them. Many thanks to Dr. Abhra Sarkar and Dr. Xiaoning Qian for accepting to be my committee members. I would also like to thank my mentor Dr. Jonathan Hunt and manager Dr. Wenzhe Shi from Twitter, who have been super supportive during the stressful job searching.

I want to take this chance to thank all of my friends who have been so caring for me during my PhD life and I could not possibly make it without these emotional supports. I would like to thank my cohort: Xinjie Fan, Mauricio Garcia Tec, Yanxin Li, Vera Liu and Rimli Sengupta for helping each other in class and preparing for the exams. I also want to thank Matteo Vestrucci, Giorgio Paulon, Ciara Nugent, Qiaohui Lin, Spencer Woody, Xizewen Han, Ruijiang Gao, Jingjing Fan, Preston Biro, Huangjie Zheng and Steve Zhang, who make me feel the belonging to the SDS department. I cannot express enough thanks to Ji Xue who helped me on my career and personal planning, and a special thanks to Lukas Zheng who makes me realize what I want to pursue.

Last but not least, I would like to thank my mother Wenjuan Ying and my father Han Yue for their unconditional love and caring for me all the time. I am really grateful that they are being nothing but supportive for all of my choices and decisions. I am blessed to have them behind everything I am today, and ever will be.

Boosting Deep Reinforcement Learning Algorithms with Deep Probabilistic Models

Publication No. ____________________

Yuguang Yue, Ph.D.
The University of Texas at Austin, 2021

Supervisor: Mingyuan Zhou

This thesis develops new methodologies that boost deep reinforcement learning algorithms from a probabilistic point of view. More specifically, three angles are studied to make improvements in terms of sample efficiency of the deep reinforcement learning algorithms: 1). We apply a hierarchical structure on policy construction to obtain a flexible policy so that it has the capability of capturing complex distribution and make more appropriate decisions. 2). We manage to reduce the variance of the policy gradient estimation calculated via a Monte Carlo estimation by designing a "self-critic" baseline function, the new gradient estimator has a smaller variance and leads to a better empirical performance. 3). We apply the distributional reinforcement learning framework on the continuous-action setting with a stochastic policy, and stabilize the training process with double generative networks. All the methods bring clear gains, which demonstrate the benefits of applying deep probabilistic models to improve deep reinforcement learning algorithms.

Table of Contents

Acknowledgments **v**

Abstract **vii**

List of Tables **xii**

List of Figures **xiii**

Chapter 1. Introduction **1**

1.1 Value based algorithm 3

1.2 Policy gradient based algorithm 5

1.3 Distributional Reinforcement Learning 6

1.4 Boosting deep reinforcement learning algorithms with deep probabilistic models 7

Chapter 2. ARSM: Augment-REINFORCE-Swap-Merge Gradient for Categorical Variables and Policy Optimization **10**

2.1 Introduction 11

2.1.1 Related Work 13

2.2 ARSM Gradient For Categorical Variables 15

2.2.1 AR: Augment-REINFORCE 16

2.2.2 ARS: Augment-REINFORCE-Swap 17

2.2.3 ARSM: Augment-REINFORCE-Swap-Merge 19

2.3 ARSM Estimator for Multivariate, Hierarchical, and Sequential Settings 20

2.3.1 ARSM for Multivariate Categorical Variables and Stochastic Categorical Network 20

2.3.2 ARSM for Discrete-Action Policy Optimization 21

2.4 Experimental Results 24

2.4.1 Example Results on Toy Data 25
2.4.2 Categorical Variational Auto-Encoders 26
2.4.3 Maximum Likelihood Estimation for a Stochastic Categorical Network 30
2.4.4 Discrete-Action Policy Optimization 32
2.5 Conclusion 35

Chapter 3. Discrete Action On-Policy Learning with Action-Value Critic 37

3.1 Introduction 38
3.2 Preliminaries 41
3.2.1 On-Policy Optimization 41
3.2.2 Variance Control Techniques 43
3.3 CARSM Policy Gradient 45
3.4 Experiments 50
3.4.1 Motivation and Illustration 52
3.4.2 Comparing CARSM and ARSM-MC 55
3.4.3 Large Discrete Action Space 56
3.4.4 OpenAI Gym Benchmark Tasks 58
3.4.5 Combining CARSM with TRPO 59
3.5 Conclusion 60

Chapter 4. Implicit Distributional Reinforcement Learning 61

4.1 Introduction 62
4.2 Implicit distributional actor critic 66
4.2.1 Implicit distributional RL with deep generator network (DGN) 67
4.2.2 Learning of DGN 68
4.2.3 Twin delayed DGNs 70
4.2.4 Semi-implicit actor (SIA) 72
4.2.5 Learning of SIA 74
4.2.6 Off policy learning with IDAC 76
4.3 Experiments 76
4.3.1 Evaluation study to answer questions (a)-(c) 79

4.3.2 Ablation study to answer questions (d)-(g) 82
4.4 Conclusion 84

Chapter 5. Conclusions and future directions 85
5.1 Conclusions 85
5.2 Future directions 85
5.2.1 Diverse proposal policy for DQN based algorithm 85
5.2.2 Offline reinforcement learning 86

Appendices 87

Appendix A. Appendix for ARSM: Augment-REINFORCE-Swap-Merge Gradient for Categorical Variables and Policy Optimization 88
A.1 Derivation of AR, ARS, and ARSM 88
A.1.1 Augmentation of a Categorical Variable 88
A.1.2 REINFORCE Estimator in the Augmented Space 90
A.1.3 Merge of Augment-REINFORCE Gradients 90
A.2 Fast Computation for the Swap Step 92
A.3 ARSM for Multivariate, Hierarchical, and Sequential Categorical Variables 93
A.3.1 ARSM for Multivariate Categorical Variables 93
A.3.2 ARSM for Categorical Stochastic Networks 94
A.3.3 Proofs 96
A.4 Additional Figures and Tables 100
A.5 Algorithm 103

Appendix B. Appendix for Discrete Action On-Policy Learning with Action-Value Critic 107
B.1 Proof of Theorem 5 107
B.2 Experiment setup 110
B.2.1 Toy example setup 110
B.2.2 Baselines and CARSM setup 112
B.2.3 Comparison between CARSM and ARSM-MC for fixed timestep 115
B.3 Pseudo Code 116

Appendix C. Appendix for Implicit Distributional Reinforcement Learning **118**
C.1 Proof of Lemma 6 118
C.2 Detailed pseudo code 119
C.3 Hyperparameters of IDAC 121

Vita **139**

List of Tables

2.1 Comparison of training and testing negative ELBOs (nats) on binarized MNIST between ARSM and various gradient estimators. . . . 28

2.2 Comparison of the test negative log-likelihoods between ARSM and various gradient estimators in Jang et al. [2017], for the MNIST conditional distribution estimation benchmark task. 31

4.1 Comparison of average maximal returns ± 1 std over 4 different random seeds. . . . 79

4.2 Variants for ablation study 83

A.1 The constructions of variational auto-encoders. The following symbols “→”, “]”,)”, and “⇝” represent deterministic linear transform, leaky rectified linear units (LeakyReLU) [Maas et al., 2013] nonlinear activation, softmax nonlinear activation, and discrete stochastic activation, respectively, in the encoder; their reversed versions are used in the decoder. 100

C.1 IDAC hyperparameters 121

List of Figures

2.1 Comparison of a variety of gradient estimators in maximizing (2.13). The optimal solution is $\sigma(\boldsymbol{\phi}) = (0, \ldots, 1)$, which means $z = C$ with probability one. The reward is computed analytically by $\mathbb{E}_{z\sim\mathrm{Cat}(\sigma(\boldsymbol{\phi}))}[f(z)]$ with maximum as 0.533. Rows 1, 2, and 3 show the trace plots of reward $\mathbb{E}[f(z)]$, the gradients with respect to ϕ_1 and ϕ_C, and the probabilities $\sigma(\boldsymbol{\phi})_1$ and $\sigma(\boldsymbol{\phi})_C$, respectively. Row 4 shows the gradient variance estimation with 100 Monte Carlo samples at each iteration, averaged over categories 1 to C. 24

2.2 Plots of negative ELBOs (nats) on binarized MNIST against training iterations (analogous ones against times are shown in Figure A.5). The solid and dash lines correspond to the training and testing respectively (best viewed in color). 28

2.3 *Top row:* Moving average reward curves. *Bottom row:* Log-variance of gradient estimator. In each plot, the solid lines are the median value of ten independent runs (ten different random seeds for random initializations). The opaque bars are 10th and 90th percentiles. Dashed straight lines in Cart Pole and Lunar Lander represent task-completion criteria. 32

3.1 **left panel**: Change of policy over iterations in a single random trial between Gaussian policy (left) and discrete policy (right) on a bimodal-reward toy example. **right panel**: Average density on each action along with the training iteration between Gaussian and discrete policies for 100 random trials. Under this setting, the Gaussian policy fails to converge to the global optimum while discrete policy always finds the global optimum. 51

3.2 **top row**: Performance curves for discrete domains. Comparison between: A2C, RELAX, ARSM-MC, and CARSM. We show the cumulative rewards during training, moving averaged across 100 epochs; the curves show the mean ± std performance across 5 random seeds. **bottom row**: Performance curves on CartPole with very large discrete action space. Comparison between: A2C and CARSM over a range of different discretization scale $C \in \{101, 501, 1001\}$. We show the cumulative rewards during training, moving averaged across 100 epochs; the curves show the mean ± std performance across 5 random seeds. 52

3.3 Performance curves on six benchmark tasks (all except the last are MuJoCo tasks). Comparison between: continuous A2C (Gaussian policy), discrete A2C, RELAX, and CARSM policy gradient. We show the cumulative rewards during training, moving averaged across 100 epochs; The curves show the mean ± std performance across 5 random seeds. 54

3.4 Performance curves on six benchmark tasks (all except the last are MuJoCo tasks). Comparison between: continuous TRPO (Gaussian policy), discrete TRPO, and CARSM policy gradient combined with TRPO. We show the cumulative rewards during training, moving averaged across 100 epochs; the curves show the mean ± std performance across 5 random seeds. 56

4.1 Training curves on continuous control benchmarks. The solid line is the average performance over seeds with ± 1 std shaded, and with a smoothing window of length 100. 80

4.2 Visualization of Gaussian policy, SIA, and distributional matching for critic generators under SIA. Panels (a) and (b) show the density contour of 1000 random sampled actions at an early training stage, where x-axis and y-axis correspond to dimensions 1 and 4, respectively; Panel (c) shows the empirical density of 10000 DGN samples at an early training stage and the final one, where (target) G samples are in (red) blue. 80

4.3 Training Curves of Ablation study. 82

A.1 Analogous plots to these in Figure 2.1, obtained with $C = 1,000$. 100

A.2 Analogous plots to these in Figure 2.1, obtained with $C = 10,000$. 101

A.3 Trace plots of the log variance of the gradient estimators for categorical VAE on MNIST. The variance is estimated by exponential moving averages of the first and second moments with a decay factor of 0.999. The variance is averaged over all elements of the gradient vector. 101

A.4 The entropy of latent categorical distributions and the number of distinct pseudo actions, which differ from their corresponding true actions, both decrease as the training progresses. We plot the average entropy for $\{z_{tk}\}$ for all $t = 1 : T$ and $k = 1 : K$. The pseudo action proportion for the k-th categorical random variable at the t-th stochastic layer is calculated as the number of unique values in $\{z_{tk}^{c\leftrightharpoons j}\}_{c=1:C,j=1:C} \backslash z_{tk}$ divided by $C - 1$, the maximum number of distinct pseudo actions that differ from the true action z_{tk}. We plot the average pseudo action proportion for $\{z_{tk}\}$ for all $t = 1 : T$ and $k = 1 : K$. Subplots (a), (b), and (c) correspond to the Toy data ($T = K = 1$, $C = 30$), VAE with a single stochastic layer ($T = 1$, $K = 20$, $C = 10$), and Acrobot RL task ($0 \leq T \leq 500$, $K = 1$, $C = 3$); other settings yield similar trace plots. 102

A.5 Plots of −ELBOs (nats) on binarized MNIST against wall clock times on NVIDIA Tesla V100 GPU (analogous ones against training iterations are shown in Figure 2.2). The solid and dash lines correspond to the training and testing respectively (best viewed in color). 103

B.1 **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). 110

B.2 Performance curves for comparison between ARSM-MC and CARSM given fix timesteps 115

Chapter 1

Introduction

Reinforcement Learning (RL) is a broad topic that has been studied for decades. Though originated from a control perspective, it has substantively and fruitfully interacted with other engineering and scientific disciplines. More recently, with the exciting achievements of Deep Learning (DL), the combination of deep neural networks and RL has made unprecedented successes in sophisticated real-world challenges in many areas such as biology, robotics and natural language processing. Many state-of-the-art algorithms are proposed under this framework such as Deep-Q-network [Mnih et al., 2015], policy-gradient based method [Mnih et al., 2016], guided policy search [Levine and Koltun, 2013], adversarial training based imitation learning [Ho and Ermon, 2016].

Though the integration of RL and DL has a long history, there is still significant space for improvement on existing literature, especially from a probabilistic point of view. We will briefly go through the background knowledge of RL tasks, and then look into how to boost deep RL algorithms with the help of the deep probabilistic models.

The goal of a RL task is to learn how to map situations to actions so as to maximize a numerical reward. A RL task is usually proposed under a Markov

Decision Process (MDP) framework defined by tuple $(\mathcal{S}, \mathcal{A}, p, r)$, where the state space $\mathcal{S}$ and action space $\mathcal{A}$ can be either continuous or discrete. The state transition probability $p : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, \infty)$ represents the probability density of the next state $\boldsymbol{s}_{t+1} \in \mathcal{S}$ given the current state $\boldsymbol{s}_t \in \mathcal{S}$ and action $\boldsymbol{a}_t \in \mathcal{A}$. The environment emits a bounded reward $r : \mathcal{S} \times \mathcal{A} \to [r_{min}, r_{max}]$ on each transition. Denoting the policy as a map from state to action $\pi(\boldsymbol{a}|\boldsymbol{s}) : \mathcal{S} \to \mathcal{A}$, the objective function we want to maximize can be expressed as

$$J(\pi) = \sum_{t=0}^{\infty} \gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t), \tag{1.1}$$

where γ is a discounting factor, $\boldsymbol{s}_0$ is the initial state, $\boldsymbol{a}_t$ is generated with respect to policy $\pi(\boldsymbol{a}|\boldsymbol{s})$ and $p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$ follows the transition probability p. Since the trajectory is sampled based on the policy $\pi(\boldsymbol{a}|\boldsymbol{s})$, the cumulative discounted return $J(\pi)$ is noted as a function of π. Similarly, we can define the value functions starting from any state or state-action pairs as follows:

$$V^{\pi}(\boldsymbol{s}) = \mathbb{E}_{\pi}\big[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | \boldsymbol{s}_t = \boldsymbol{s}\big], \text{ for all } s \in \mathcal{S}$$

and

$$Q^{\pi}(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}_{\pi}\big[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | \boldsymbol{s}_t = \boldsymbol{s}, \boldsymbol{a}_t = \boldsymbol{a}\big]$$

By definition, the relationship between $V^{\pi}(\boldsymbol{s})$ and $Q^{\pi}(\boldsymbol{s}, \boldsymbol{a})$ is $V^{\pi}(\boldsymbol{s}) = \mathbb{E}_{\boldsymbol{a} \sim \pi(\boldsymbol{a}|\boldsymbol{s})} Q^{\pi}(\boldsymbol{s}, \boldsymbol{a})$. We further use $\rho_{\pi}(\boldsymbol{s})$ and $\rho_{\pi}(\boldsymbol{s}, \boldsymbol{a})$ to denote the state and state-action marginals of the trajectory distribution induced by policy $\pi(\boldsymbol{a}|\boldsymbol{s})$.

After introducing the notations and preliminary of classical RL setup, we present two big categories of deep RL algorithms: value based algorithms and policy gradient based algorithms. Note that the boundary between them can be pretty blurred. During the introduction, we will mark the highlights where probabilistic models can be applied to make improvements, and we will summarize those points at the end of the this Chapter.

1.1 Value based algorithm

Value based algorithms are always composed of two iterative steps: policy evaluation and policy improvement.

Policy evaluation: For the policy evaluation part, we want to evaluate $V^\pi(\boldsymbol{s})$ and $Q^\pi(\boldsymbol{s}, \boldsymbol{a})$ (here we mainly discuss the state-action value function Q since that is more widely used), which can be obtained by either a *Monte Carlo* (MC) estimation method or a *Temporal Difference* (TD) method. The estimation via MC is straightforward as follows

$$Q^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t) = \sum_{t=0}^{\infty} \gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t),$$

where $(\boldsymbol{s}_t, \boldsymbol{a}_t)$ are *on-policy* samples, which means they are collected by running the current policy π. Otherwise, one can use *bootstrap* to avoid sampling the whole trajectory for policy evaluation with the help of a **Bellman Equation**:

$$Q^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t) = r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma \mathbb{E}_{\boldsymbol{a}_{t+1} \sim \pi(\boldsymbol{a}_{t+1}|\boldsymbol{s}_{t+1})} Q^\pi(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}).$$

By TD method, the policy evaluation can be achieved only with the next state action tuple rather than the whole trajectory as needed by the MC method; the trade-off between the MC and TD methods is a bias-variance trade-off.

The MC method is an unbiased estimator while more samples are needed for a single estimation. By contrast, the TD method is a biased estimator (the bias comes from the function approximation of Q^π) but has a smaller variance.

Policy improvement: After evaluating the action-value function Q^π, the policy can be naturally updated by $\pi^+(\boldsymbol{a}|\boldsymbol{s}) = \arg\max_{\boldsymbol{a}} Q^\pi(\boldsymbol{s}, \boldsymbol{a})$ where π^+ denotes the updated policy; otherwise, it can also be updated with respect to the Boltzmann distribution of Q^π as in Haarnoja et al. [2018a], which is updated by minimizing $\mathrm{KL}(\pi(\boldsymbol{a}|\boldsymbol{s})||\frac{e^{Q^\pi(\boldsymbol{s},\boldsymbol{a})}}{Z(\boldsymbol{s})})$, where Z is a normalizing constant. However, in the deep RL setting, the Q^π function is always modeled by a neural network where the state-action pair is an input to the network. As a result, when the state and action space are continuous, the Boltzmann distribution induced by Q^π is no longer analytic. One naive way to overcome this is to approximate the Boltzmann distribution by a multivariate Gaussian distribution with a diagonal covariance matrix [Haarnoja et al., 2018a], which may lead to unsatisfactory results. Now we highlight the first place where a probabilistic model can be used to make improvements: *1. how to construct a model to approximate a flexible distribution.*

Deep-Q-learning (DQN) algorithm [Mnih et al., 2013] is proposed following these two steps, where the policy evaluation and policy improvement are replaced by an action-value function iteration for efficiency; more specifically, the action-value iteration follows the **Bellman Optimality Equation**:

$$Q^*(\boldsymbol{s}_t, \boldsymbol{a}_t) = r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma \mathbb{E}\big[\arg\max_{\boldsymbol{a}' \in \mathcal{A}} Q^*(\boldsymbol{s}_{t+1}, \boldsymbol{a}')\big].$$

Note that the *argmax* operator is the bottleneck of the DQN, and makes DQN only applicable to the discrete-action space or low-dimensional continuous-action space tasks. To overcome this challenge, Deep Deterministic Policy Gradient algorithm (DDPG) [Lillicrap et al., 2015] proposes using a deterministic map $\boldsymbol{a} = \pi(\boldsymbol{s})$ to approximate the action that maximize $Q^*(\boldsymbol{s}, \boldsymbol{a})$, since the mapping π and optimal action-value function Q^* are deterministic, the gradient can be propagated. On the other hand, another feasible way is to approximately solve the *argmax* problem by proposing a number of candidate actions and take the *argmax* from them; one naive proposal is the uniform distribution, which can be inefficient when action dimension is high [Sun et al., 2020]. We highlight the second place where a probabilistic model can be applied: *2. how to construct a diverse proposal policy for DQN based algorithm.*

1.2 Policy gradient based algorithm

Policy gradient (PG) based algorithm aims to maximize the discounted cumulative return (1.1) directly. Under the PG framework, a policy is always modeled as a neural network $\pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s})$, and the parameters $\boldsymbol{\theta}$ are updated via a stochastic gradient descent based method [Bottou, 2012]. Based on the *policy gradient theorem* of Sutton et al. [2000], we have

$$\nabla_{\boldsymbol{\theta}} J_{\boldsymbol{\theta}}(\pi) = \sum_{\boldsymbol{s}} \rho_\pi(\boldsymbol{s}) Q^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}, \boldsymbol{a}) \nabla_{\boldsymbol{\theta}} \pi(\boldsymbol{a}|\boldsymbol{s}),$$

where $\rho_\pi(\boldsymbol{s})$ is the discounted state marginal distribution. Though a PG based algorithm optimize (1.1) directly and has a straightforward implementation, it is notorious that the gradient estimator $\nabla_{\boldsymbol{\theta}} J_{\boldsymbol{\theta}}(\pi)$ has a large variance due to the nature of MC estimation. Our third highlight place is: *3. how can we reduce the variance of a MC estimation with Bayesian methods.*

1.3 Distributional Reinforcement Learning

Under the classic RL framework, the value functions V and Q are modeled as scalars, which may not be able to model the uncertainty from the environment entirely. For example, in the mushroom dataset from Dua and Graff [2017], there are poisonous and edible mushrooms; the reward of eating an edible mushroom is always $+1$, however, eating a poisonous one will have a probability of p getting sick and receive a -1 reward, but also with probability $1-p$ of feeling good and receive a reward $+1$. In this case, assume $\boldsymbol{s}_p$ as the state of a poisonous mushroom, we will have

$$Q(\boldsymbol{s}_p, eat) = \begin{cases} +1 & \text{with probability } 1-p \\ -1 & \text{with probability } p \end{cases} \tag{1.2}$$

where the action-value function Q should be modeled as a random variable rather than a scalar. Dabney et al. [2018b] proposes the distributional RL framework that models the value function as a random variable. Under this framework, the classical Bellman Equation is modified to a **Distributional Bellman Equation**:

$$Z^{\pi}(\boldsymbol{s}, \boldsymbol{a}) \overset{D}{=} R(\boldsymbol{s}, \boldsymbol{a}) + \gamma Z^{\pi}(\boldsymbol{s}', \boldsymbol{a}'). \tag{1.3}$$

where $Z^{\pi}(\boldsymbol{s}, \boldsymbol{a})$ is a random variable, the distributional version of $Q^{\pi}(\boldsymbol{s}, \boldsymbol{a})$. Several papers apply this distributional RL framework on discrete-action space tasks [Dabney et al., 2018a,b], and Singh et al. [2020] combines distributional RL with a deterministic policy. Our fourth place to make contribution is, *4. how to apply the distributional RL framework with a stochastic policy on the continuous-action space tasks.*

1.4 Boosting deep reinforcement learning algorithms with deep probabilistic models

In this section, we will summarize the ideas we mentioned in the previous sections and explain in detail how to achieve those goals.

1. How to construct a model to approximate a flexible distribution.

Since we want to minimize the KL-divergence between our target policy and the Boltzmann distribution of an action-value function, the target policy should have the following two properties: 1). The policy should be flexible to characterize multi-modality, skewness, correlations, etc. 2). The policy should have a tractable or an approximate entropy expression so that the KL-divergence can be optimized. With a hierarchical Bayesian construction, Yin and Zhou [2018b] introduces a complex marginal distribution with a neural network transformation, and proves an asymptotic lower bound for the entropy of the complex distribution. We implement this idea under a RL framework, and empirically demonstrate its effectiveness. We will elaborate this in Chapter 4.

2. How to construct a diverse proposal policy for DQN based algorithm.

One feasible way of applying a DQN based algorithm on the continuous-action space tasks is to propose reasonable candidate actions that potentially have large action-values and choose the argmax over the candidate set rather than solving a complex optimization problem. We approach this by proposing candidate actions from a proposal policy that tries to balance between maximizing Q greedily and being diverse. We defer this part to Chapter 5 as a potential future work.

3. How can we reduce the variance of a MC estimation with Bayesian methods.
When evaluating MC estimation in Bayesian literature, it is common to control the variance with a baseline whose expectation is easier to calculate or even is a certain number. There are a number of papers working from various aspects to make the policy gradient based algorithm more stable and efficient [Maddison et al., 2017, Jang et al., 2017], but most of them are working on the continuous-action cases. In Chapter 2, we propose a "try-and-see self-critic" method to produce an unbiased and low-variance policy gradient estimator, referred to as the ARSM gradient estimator. Further in Chapter 3, we design a sample-efficient on-policy RL algorithm with the help of the ARSM gradient estimator and demonstrate its efficacy on a set of benchmark tasks.

4. How to apply the distributional RL framework with a stochastic policy on continuous-space tasks.

The difficulty of applying a distributional RL framework is fitting a distributional Bellman equation rather than the classical Bellman equation, which requires distributional matching instead of scalar matching. Dabney et al. [2018b], Bellemare et al. [2017a], Dabney et al. [2018a] propose methods to discretize the value function by its value range or quantiles for the discrete-action tasks; on the other hand, Singh et al. [2020] approaches this by applying a generative network that does not require any prefixed discretization. Inspired by Singh et al. [2020], we propose a distributional RL framework for stochastic policy on the continuous-space tasks; combined with a flexible policy, the proposed method achieves the state-of-art performance across a number of benchmarks. We present this part in Chapter 4.

Chapter 2

ARSM: Augment-REINFORCE-Swap-Merge Gradient for Categorical Variables and Policy Optimization

To address the challenge of backpropagating the gradient through categorical variables, we propose the augment-REINFORCE-swap-merge (ARSM) gradient estimator that is unbiased and has low variance. ARSM first uses variable augmentation, REINFORCE, and Rao-Blackwellization to re-express the gradient as an expectation under the Dirichlet distribution, then uses variable swapping to construct differently expressed but equivalent expectations, and finally shares common random numbers between these expectations to achieve significant variance reduction. Experimental results show ARSM closely resembles the performance of the true gradient for optimization in univariate settings; outperforms existing estimators by a large margin when applied to categorical variational auto-encoders; and provides a "try-and-see self-critic" variance reduction method for discrete-action policy gradient, which removes the need of estimating baselines by generating a random number of pseudo

---

The content in this chapter was published in Yin et al. [2019]; I was mainly involved in the reinforcement learning section, where I worked on designing the ARSM policy gradient algorithm, finished the theoretical proof and conducted empirical experiments.

actions and estimating their action-value functions.

2.1 Introduction

The need to maximize an objective function, expressed as the expectation over categorical variables, arises in a wide variety of settings, such as discrete latent variable models Zhou [2014], Jang et al. [2017], Maddison et al. [2017] and policy optimization for reinforcement learning (RL) with discrete actions Sutton and Barto [1998], Weaver and Tao [2001], Schulman et al. [2015a], Mnih et al. [2016], Grathwohl et al. [2018]. More specifically, let us denote $z_k \in \{1, 2, \dots, C\}$ as a univariate C-way categorical variable, and $\boldsymbol{z} = (z_1, \dots, z_K) \in \{1, 2, \dots, C\}^K$ as a K-dimensional C-way multivariate categorical vector. In discrete latent variable models, K will be the dimension of the discrete latent space, each dimension of which can be further represented as a C-dimensional one-hot vector. In RL, C represents the size of the discrete action space and $\boldsymbol{z}$ is a sequence of discrete actions from that space. In even more challenging settings, one may have a sequence of K-dimensional C-way multivariate categorical vectors, which appear both in categorical latent variable models with multiple stochastic layers, and in RL with a high dimensional discrete action space or multiple agents, which may consist of as many as C^K unique combinations at each time step.

With $f(\boldsymbol{z})$ and $q_{\boldsymbol{\phi}}(\boldsymbol{z})$ denoted as the reward function and distribution for categorical $\boldsymbol{z}$, respectively, we need to optimize parameter $\boldsymbol{\phi}$ to maximize the expected reward as

$$\mathcal{E}(\boldsymbol{\phi}) = \int f(\boldsymbol{z}) q_{\boldsymbol{\phi}}(\boldsymbol{z}) d\boldsymbol{z} = \mathbb{E}_{\boldsymbol{z} \sim q_{\boldsymbol{\phi}}(\boldsymbol{z})}[f(\boldsymbol{z})]. \tag{2.1}$$

Here we consider both categorical latent variable models and policy optimization for discrete actions, which arise in a wide array of real-world applications. A number of unbiased estimators for backpropagating the gradient through discrete latent variables have been recently proposed Tucker et al. [2017], Grathwohl et al. [2018], Yin and Zhou [2019], Andriyash et al. [2018]. However, they all mainly, if not exclusively, focus on the binary case (*i.e.*, $C = 2$). The categorical case (*i.e.*, $C \geq 2$) is more widely applicable but generally much more challenging. In this paper, to optimize the objective in (2.1), inspired by the augment-REINFORCE-merge (ARM) gradient estimator restricted for binary variables [Yin and Zhou, 2019], we introduce the augment-REINFORCE-swap-merge (ARSM) estimator that is unbiased and well controls its variance for categorical variables.

The proposed ARSM estimator combines variable augmentation [Tanner and Wong, 1987, Van Dyk and Meng, 2001], REINFORCE [Williams, 1992b] in an augmented space, Rao-Blackwellization [Casella and Robert, 1996], and a merge step that shares common random numbers between different but equivalent gradient expectations to achieve significant variance reduction. While ARSM with $C = 2$ reduces to the ARM estimator [Yin and Zhou, 2019], whose merge step can be realized by applying antithetic sampling [Owen, 2013] in the augmented space, the merge step of ARSM with $C > 2$ cannot be realized in this manner. Instead, ARSM requires distinct variable-swapping operations to construct differently expressed but equivalent expectations under the Dirichlet distribution before performing its merge step.

Experimental results on both synthetic data and several representative tasks involving categorical variables are used to illustrate the distinct working mechanism of ARSM. In particular, our experimental results on latent variable models with one or multiple categorical stochastic hidden layers show that ARSM provides state-of-the-art training and out-of-sample prediction performance. Our experiments on RL with discrete action spaces show that ARSM provides a "try-and-see self-critic" method to produce unbiased and low-variance policy gradient estimates, removing the need of constructing baselines by generating a random number of pseudo actions at a given state and estimating their action-value functions. These results demonstrate the effectiveness and versatility of the ARSM estimator for gradient backpropagation through categorical stochastic layers. Python code for reproducible research is available at https://github.com/ARM-gradient/ARSM.

2.1.1 Related Work

For optimizing (2.1) for categorical $\boldsymbol{z}$, the difficulty lies in developing a low-variance and preferably unbiased estimator for its gradient with respect to $\boldsymbol{\phi}$, expressed as $\nabla_{\boldsymbol{\phi}}\mathcal{E}(\boldsymbol{\phi})$. An unbiased but high-variance gradient estimator that is universally applicable to (2.1) is REINFORCE [Williams, 1992b]. Using the score function $\nabla_{\boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}(\boldsymbol{z}) = \nabla_{\boldsymbol{\phi}} q_{\boldsymbol{\phi}}(\boldsymbol{z})/q_{\boldsymbol{\phi}}(\boldsymbol{z})$, REINFORCE expresses the gradient as an expectation as

$$\nabla_{\boldsymbol{\phi}} \mathcal{E}(\boldsymbol{\phi}) = \mathbb{E}_{\boldsymbol{z} \sim q_{\boldsymbol{\phi}}(\boldsymbol{z})}[f(\boldsymbol{z}) \nabla_{\boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}(\boldsymbol{z})], \tag{2.2}$$

and approximates it with Monte Carlo integration [Owen, 2013]. However, the estimation variance with a limited number of Monte Carlo samples is often too high to make vanilla REINFORCE a sound choice for categorical $\boldsymbol{z}$.

To address the high-estimation-variance issue for categorical $\boldsymbol{z}$, one often resorts to a biased gradient estimator. For example, Maddison et al. [2017] and Jang et al. [2017] relax the categorical variables with continuous ones and then apply the reparameterization trick to estimate the gradients, reducing variance but introducing bias. Other biased estimators for backpropagating through binary variables include the straight-through estimator Hinton [2012], Bengio et al. [2013] and the ones of Gregor et al. [2014], Raiko et al. [2014], Cheng et al. [2018]. With biased gradient estimates, however, a gradient ascent algorithm may not be guaranteed to work, or may converge to unintended solutions.

To keep REINFORCE unbiased while sufficiently reducing its variance, a usual strategy is to introduce appropriate control variates, also known as baselines [Williams, 1992b], into the expectation in (2.2) before performing Monte Carlo integration [Paisley et al., 2012, Ranganath et al., 2014, Mnih and Gregor, 2014, Gu et al., 2016a, Mnih and Rezende, 2016, Ruiz et al., 2016, Kucukelbir et al., 2017, Naesseth et al., 2017]. For discrete $\boldsymbol{z}$, Tucker et al. [2017] and Grathwohl et al. [2018] improve REINFORCE by introducing continuous relaxation based baselines, whose parameters are optimized by minimizing the sample variance of gradient estimates.

2.2 ARSM Gradient For Categorical Variables

Let us denote $z \sim \text{Cat}(\sigma(\boldsymbol{\phi}))$ as a categorical variable such that $P(z = c \,|\, \boldsymbol{\phi}) = \sigma(\boldsymbol{\phi})_c = e^{\phi_c} \big/ \sum_{i=1}^{C} e^{\phi_i}$, where $\boldsymbol{\phi} := (\phi_1, \ldots, \phi_C)$ and $\sigma(\boldsymbol{\phi}) := (e^{\phi_1}, \ldots, e^{\phi_C}) / \sum_{i=1}^{C} e^{\phi_i}$ is the softmax function. For the expectated reward defined as

$$\mathcal{E}(\boldsymbol{\phi}) := \mathbb{E}_{z \sim \text{Cat}(\sigma(\boldsymbol{\phi}))}[f(z)] = \textstyle\sum_{i=1}^{C} f(i)\sigma(\boldsymbol{\phi})_i,$$

the gradient can be expressed analytically as

$$\nabla_{\phi_c} \mathcal{E}(\boldsymbol{\phi}) = \sigma(\boldsymbol{\phi})_c f(c) - \sigma(\boldsymbol{\phi})_c \mathcal{E}(\boldsymbol{\phi}) \tag{2.3}$$

or expressed with REINFORCE as

$$\nabla_{\phi_c} \mathcal{E}(\boldsymbol{\phi}) = \mathbb{E}_{z \sim \text{Cat}(\sigma(\boldsymbol{\phi}))} \left[f(z)(\mathbf{1}_{[z=c]} - \sigma(\boldsymbol{\phi})_c) \right], \tag{2.4}$$

where $\mathbf{1}_{[\cdot]}$ is an indicator function that is equal to one if the argument is true and zero otherwise. However, the analytic expression quickly becomes intractable for a multivariate setting, and the REINFORCE estimator often comes with significant estimation variance. While the ARM estimator of Yin and Zhou [2019] is unbiased and provides significant variance reduction for binary variables, it is restricted to $C = 2$ and hence has limited applicability.

Below we introduce the augment-REINFORCE (AR), AR-swap (ARS), and ARS-merge (ARSM) estimators for a univariate C-way categorical variable, and later generalize them to multivariate, hierarchical, and sequential settings.

2.2.1 AR: Augment-REINFORCE

Let us denote $\boldsymbol{\pi} := (\pi_1, \ldots, \pi_C) \sim \text{Dir}(\mathbf{1}_C)$ as a Dirichlet distribution whose C parameters are all ones. We first state three statistical properties that can directly lead to the proposed AR estimator. We describe in detail in Appendix A.1 how we actually arrive at the AR estimator, with these properties obtained as by-products, by performing variable augmentation, REINFORCE, and Rao-Blackwellization. Thus we are in fact reverse-engineering our original derivation of the AR estimator to help concisely present our findings.

Property I. *The categorical variable* $z \sim \text{Cat}(\sigma(\boldsymbol{\phi}))$ *can be equivalently generated as*

$$z := \arg\min_{i \in \{1,\ldots,C\}} \pi_i e^{-\phi_i}, \ \boldsymbol{\pi} \sim \text{Dir}(\mathbf{1}_C).$$

Property II. $\mathcal{E}(\boldsymbol{\phi}) = \mathbb{E}_{\boldsymbol{\pi} \sim \text{Dir}(\mathbf{1}_C)}[f(\arg\min_i \pi_i e^{-\phi_i})]$.

Property III. $\mathbb{E}_{\boldsymbol{\pi} \sim \text{Dir}(\mathbf{1}_C)}[f(\arg\min_i \pi_i e^{-\phi_i}) C \pi_c] = \mathcal{E}(\boldsymbol{\phi}) + \sigma(\boldsymbol{\phi})_c \mathcal{E}(\boldsymbol{\phi}) - \sigma(\boldsymbol{\phi})_c f(c)$.

These three properties, Property III in particular, are previously unknown to the best of our knowledge. They are directly linked to the AR estimator shown below.

Theorem 1 (AR estimator)**.** *The gradient of* $\mathcal{E}(\boldsymbol{\phi}) = \mathbb{E}_{z \sim \text{Cat}(\sigma(\boldsymbol{\phi}))}[f(z)]$*, as shown in* (2.3)*, can be re-expressed as an expectation under a Dirichlet distribution as*

$$\begin{aligned} \nabla_{\phi_c} \mathcal{E}(\boldsymbol{\phi}) &= \mathbb{E}_{\boldsymbol{\pi} \sim \text{Dir}(\mathbf{1}_C)}[g_{\text{AR}}(\boldsymbol{\pi})_c], \\ g_{\text{AR}}(\boldsymbol{\pi})_c : &= f(z)(1 - C\pi_c), \\ z : &= \arg\min_{i \in \{1,\ldots,C\}} \pi_i e^{-\phi_i}. \end{aligned} \tag{2.5}$$

Distinct from REINFORCE in (2.4), the AR estimator in (2.5) now expresses the gradient as an expectation under a Dirichlet distributed random noise. From this point of view, it is somewhat related to the reparameterization trick [Kingma and Welling, 2013, Rezende et al., 2014], which is widely used to express the gradient of an expectation under reparameterizable random variables as an expectation under random noises. Thus one may consider AR as a special type of reparameterization gradient, which, however, requires neither z to be reparameterizable nor $f(\cdot)$ to be differentiable.

2.2.2 ARS: Augment-REINFORCE-Swap

Let us swap the mth and jth elements of $\boldsymbol{\pi}$ to define vector

$$\boldsymbol{\pi}^{m\leftrightharpoons j} := (\pi_1^{m\leftrightharpoons j}, \ldots, \pi_C^{m\leftrightharpoons j}),$$

where $\pi_m^{m\leftrightharpoons j} = \pi_j$, $\pi_j^{m\leftrightharpoons j} = \pi_m$, and $\forall\ c \notin \{m, j\}$, $\pi_c^{m\leftrightharpoons j} = \pi_c$. Another property to be repeatedly used is:

Property IV. *If* $\boldsymbol{\pi} \sim \mathrm{Dir}(\mathbf{1}_C)$, *then* $\boldsymbol{\pi}^{m\leftrightharpoons j} \sim \mathrm{Dir}(\mathbf{1}_C)$.

This leads to a key observation for the AR estimator in (2.5): swapping any two variables of the probability vector $\boldsymbol{\pi}$ inside the expectation does not change the expected value. Using the idea of sharing common random numbers between different expectations to potentially significantly reduce Monte Carlo integration variance [Owen, 2013], we propose to swap π_c and π_j in (2.5), where $j \in \{1, \ldots, C\}$ is a reference category chosen independently of $\boldsymbol{\pi}$ and $\boldsymbol{\phi}$. This variable-swapping operation changes the AR estimator to

$$\begin{aligned} \nabla_{\phi_c}\mathcal{E}(\boldsymbol{\phi}) &= \mathbb{E}_{\boldsymbol{\pi}\sim \mathrm{Dir}(\mathbf{1}_C)}[g_{\mathrm{AR}}(\boldsymbol{\pi}^{c\leftrightharpoons j})_c] \\ g_{\mathrm{AR}}(\boldsymbol{\pi}^{c\leftrightharpoons j})_c &: = f(z^{c\leftrightharpoons j})(1-C\pi_j), \\ z^{c\leftrightharpoons j} &: = \arg\min_{i\in\{1,\dots,C\}} \pi_i^{c\leftrightharpoons j} e^{-\phi_i}, \end{aligned} \tag{2.6}$$

where we have applied identity $\pi_c^{c\leftrightharpoons j} = \pi_j$ and Property IV. We refer to z defined in (2.5) as the "true action," and $z^{c\leftrightharpoons j}$ defined in (2.6) as the cth "pseudo action" given j as the reference category. Note the pseudo actions satisfy the following properties: $z^{c\leftrightharpoons j} = z^{j\leftrightharpoons c}$ and $z^{c\leftrightharpoons j} = z$ if $c = j$, and the number of unique values in $\{z^{c\leftrightharpoons j}\}_{c,j}$ that are different from the true action z is between 0 and $C-1$.

With (2.3), we have another useful property as

Property V. $\sum_{c=1}^{C} \nabla_{\phi_c}\mathcal{E}(\boldsymbol{\phi}) = 0.$

Combining it with the estimator in (2.6) leads to

$$\mathbb{E}_{\boldsymbol{\pi}\sim \mathrm{Dir}(\mathbf{1}_C)}\left[\tfrac{1}{C}\textstyle\sum_{c=1}^{C} g_{\mathrm{AR}}(\boldsymbol{\pi}^{c\leftrightharpoons j})_c\right] = 0. \tag{2.7}$$

Thus we can utilize $\frac{1}{C}\sum_{c=1}^{C} g_{\mathrm{AR}}(\boldsymbol{\pi}^{c\leftrightharpoons j})_c$ as a baseline function that is nonzero in general but has zero expectation under $\boldsymbol{\pi}\sim \mathrm{Dir}(\mathbf{1}_C)$. Subtracting (2.7) from (2.6) leads to another unbiased estimator, with category j as the reference, as

$$\begin{aligned} \nabla_{\phi_c}\mathcal{E}(\boldsymbol{\phi}) &= \mathbb{E}_{\boldsymbol{\pi}\sim \mathrm{Dir}(\mathbf{1}_C)}[g_{\mathrm{ARS}}(\boldsymbol{\pi}, j)_c], \\ g_{\mathrm{ARS}}(\boldsymbol{\pi}, j)_c &:= g_{\mathrm{AR}}(\boldsymbol{\pi}^{c\leftrightharpoons j})_c - \tfrac{1}{C}\textstyle\sum_{m=1}^{C} g_{\mathrm{AR}}(\boldsymbol{\pi}^{m\leftrightharpoons j})_m, \\ &= \left[f(z^{c\leftrightharpoons j}) - \tfrac{1}{C}\textstyle\sum_{m=1}^{C} f(z^{m\leftrightharpoons j})\right](1-C\pi_j), \end{aligned} \tag{2.8}$$

which is referred to as the AR-swap (ARS) estimator, due to the use of variable-swapping in its derivation from AR.

2.2.3 ARSM: Augment-REINFORCE-Swap-Merge

For ARS in (2.8), when the reference category j is randomly chosen from $\{1, \ldots, C\}$ and hence is independent of $\boldsymbol{\pi}$ and $\boldsymbol{\phi}$, it is unbiased. Furthermore, we find that it can be further improved, especially when C is large, by adding a merge step to construct the ARS-merge (ARSM) estimator:

Theorem 2 (ARSM estimator). *The gradient of* $\mathcal{E}(\boldsymbol{\phi}) = \mathbb{E}_{z \sim \text{Cat}(\sigma(\boldsymbol{\phi}))}[f(z)]$ *with respect to* ϕ_c*, can be expressed as*

$$\begin{aligned}
&\nabla_{\phi_c} \mathcal{E}(\boldsymbol{\phi}) = \mathbb{E}_{\boldsymbol{\pi} \sim \text{Dir}(\mathbf{1}_C)} \big[g_{\text{ARSM}}(\boldsymbol{\pi})_c \big], \\
&g_{\text{ARSM}}(\boldsymbol{\pi})_c := \tfrac{1}{C} \textstyle\sum_{j=1}^{C} g_{\text{ARS}}(\boldsymbol{\pi}, j)_c \\
&\quad = \textstyle\sum_{j=1}^{C} \big[f(z^{c \rightleftharpoons j}) - \tfrac{1}{C} \textstyle\sum_{m=1}^{C} f(z^{m \rightleftharpoons j}) \big] \big(\tfrac{1}{C} - \pi_j \big).
\end{aligned} \tag{2.9}$$

Note ARSM requires $C(C-1)/2$ swaps to generate pseudo actions, the unique number of which that differ from z is between 0 and $C-1$; a naive implementation requires $O(C^2)$ $\arg\min$ operations, which, however, is totally unnecessary, as in general it can at least be made below $O(2C)$ and hence is scalable even C is very large (*e.g.*, $C = 10,000$); please see Appendix A.2 and the provided code for more details. Note if all pseudo actions $z^{c \rightleftharpoons j}$ are the same as the true action z, then the gradient estimates will be zeros for all ϕ_c.

Corollary 3. *When* $C = 2$*, both the ARS estimator in* (2.8) *and ARSM estimator in* (2.9) *reduce to the unbiased binary ARM estimator introduced in Yin and Zhou [2019].*

Detailed derivations and proofs are provided in Appendix A.1. Note for $C = 2$, Proposition 4 of Yin and Zhou [2019] shows that the ARM estimator is the AR estimator combined with an optimal baseline that is subject to an anti-symmetric constraint. When $C > 2$, however, such type of theoretical analysis becomes very challenging for both the ARS and ARSM estimators. For example, it is even unclear how to define anti-symmetry for categorical variables. Thus in what follows we will focus on empirically evaluating the effectiveness of both ARS and ARSM for variance reduction.

2.3 ARSM Estimator for Multivariate, Hierarchical, and Sequential Settings

This section shows how the proposed univariate ARS and ARSM estimators can be generalized into multivariate, hierarchical, and sequential settings. We summarize ARS and ARSM (stochastic) gradient ascent for various types of categorical latent variables in Algorithms 2-4 of the Appendix.

2.3.1 ARSM for Multivariate Categorical Variables and Stochastic Categorical Network

We generalize the univariate AR/ARS/ARSM estimators to multivariate ones, which can backpropagate the gradient through a K dimensional vector of C-way categorical variables as $\boldsymbol{z} = (z_1, \ldots, z_K)$, where $z_k \in \{1, \ldots, C\}$. We further generalize them to backpropagate the gradient through multiple stochastic categorical layers, the tth layer of which consists of a K_t-dimensional C-way categorical vector as $\boldsymbol{z}_t = (z_{t1}, \ldots, z_{tK_t})' \in \{1, \ldots, C\}^{K_t}$. We defer all the details to Appendix A.3 due to space constraint.

Note for categorical variables, especially in multivariate and/or hierarchical settings, the ARS/ARSM estimators may appear fairly complicated due to their variable-swapping operations. Their implementations, however, are actually relatively straightforward, as shown in Algorithms 2 and 3 of the Appendix, and the provided Python code.

2.3.2 ARSM for Discrete-Action Policy Optimization

In RL with a discrete action space with C possible actions, at time t, the agent with state $\boldsymbol{s}_t$ chooses action $a_t \in \{1, \dots, C\}$ according to policy

$$\pi_{\boldsymbol{\theta}}(a_t \,|\, \boldsymbol{s}_t) := \mathrm{Cat}(a_t; \sigma(\boldsymbol{\phi}_t)), \quad \boldsymbol{\phi}_t := \mathcal{T}_{\boldsymbol{\theta}}(\boldsymbol{s}_t),$$

where $\mathcal{T}_{\boldsymbol{\theta}}(\cdot)$ denotes a neural network parameterized by $\boldsymbol{\theta}$; the agent receives award $r(\boldsymbol{s}_t, a_t)$ at time t, and state $\boldsymbol{s}_t$ transits to state $\boldsymbol{s}_{t+1}$ according to $\mathcal{P}(\boldsymbol{s}_{t+1} \,|\, \boldsymbol{s}_t, a_t)$. With discount parameter $\gamma \in (0, 1]$, policy gradient methods optimize $\boldsymbol{\theta}$ to maximize the expected reward $J(\boldsymbol{\theta}) = \mathbb{E}_{\mathcal{P},\pi_{\boldsymbol{\theta}}}[\sum_{t=0}^{\infty} \gamma^t r(\boldsymbol{s}_t, a_t)]$ [Sutton and Barto, 1998, Sutton et al., 2000, Peters and Schaal, 2008, Schulman et al., 2015a]. With $Q(\boldsymbol{s}_t, a_t) := \mathbb{E}_{\mathcal{P},\pi_{\boldsymbol{\theta}}}[\sum_{t'=t}^{\infty} \gamma^{t'-t} r(\boldsymbol{s}_{t'}, a_{t'})]$ denoted as the action-value functions, $\hat{Q}(\boldsymbol{s}_t, a_t) := \sum_{t'=t}^{\infty} \gamma^{t'-t} r(\boldsymbol{s}_{t'}, a_{t'})$ as their sample estimates, and $\rho_\pi(\boldsymbol{s}) := \sum_{t=0}^{\infty} \gamma^t \mathcal{P}(\boldsymbol{s}_t = \boldsymbol{s} \,|\, \boldsymbol{s}_0, \pi_{\boldsymbol{\theta}})$ as the unnormalized discounted state visitation frequency, the policy gradient via REINFORCE [Williams, 1992b] can be expressed as

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{a_t \sim \pi_{\boldsymbol{\theta}}(a_t | \boldsymbol{s}_t),\, \boldsymbol{s}_t \sim \rho_\pi(\boldsymbol{s})}[\nabla_{\boldsymbol{\theta}} \ln \pi_{\boldsymbol{\theta}}(a_t | \boldsymbol{s}_t) Q(\boldsymbol{s}_t, a_t)].$$

For variance reduction, one often subtracts state-dependent baselines $b(\boldsymbol{s}_t)$ from $\hat{Q}(\boldsymbol{s}_t, a_t)$ [Williams, 1992b, Greensmith et al., 2004]. In addition, several different action-dependent baselines $b(\boldsymbol{s}_t, a_t)$ have been recently proposed Gu et al. [2017], Grathwohl et al. [2018], Wu et al. [2018], Liu et al. [2018], though their promise in appreciable variance reduction without introducing bias for policy gradient has been questioned by Tucker et al. [2018].

Distinct from all previous baseline-based variance reduction methods, in this paper, we develop both the ARS and ARSM policy gradient estimators, which use the action-value functions $Q(\boldsymbol{s}_t, a_t)$ themselves combined with pseudo actions to achieve variance reduction:

Proposition 4 (ARS/ARSM policy gradient)**.** *The policy gradient* $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ *can be expressed as*

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\varpi}_t \sim \mathrm{Dir}(\mathbf{1}_C),\ \boldsymbol{s}_t \sim \rho_\pi(\boldsymbol{s})} \big[\nabla_{\boldsymbol{\theta}} \textstyle\sum_{c=1}^{C} g_{tc} \phi_{tc} \big], \tag{2.10}$$

where $\boldsymbol{\varpi}_t = (\varpi_{t1}, \ldots, \varpi_{tC})'$ *and* ϕ_{tc} *is the cth element of* $\boldsymbol{\phi}_t = \mathcal{T}_{\boldsymbol{\theta}}(\boldsymbol{s}_t) \in \mathbb{R}^C$*; under the ARS estimator, we have*

$$\begin{aligned} g_{tc} &:= f_{t\Delta}^{c \leftrightharpoons j_t}(\boldsymbol{\varpi}_t)(1 - C\varpi_{tj_t}), \\ f_{t\Delta}^{c \leftrightharpoons j_t}(\boldsymbol{\varpi}_t) &:= Q(\boldsymbol{s}_t, a_t^{c \leftrightharpoons j_t}) - \tfrac{1}{C} \textstyle\sum_{m=1}^{C} Q(\boldsymbol{s}_t, a_t^{m \leftrightharpoons j_t}), \\ a_t^{c \leftrightharpoons j_t} &:= \arg\min_{i \in \{1,\ldots,C\}} \varpi_{ti}^{c \leftrightharpoons j_t} e^{-\phi_{ti}}, \end{aligned} \tag{2.11}$$

where $j_t \in \{1, \ldots, C\}$ *is a randomly selected reference category for time step* t*; under the ARSM estimator, we have*

$$g_{tc} := \textstyle\sum_{j=1}^{C} f_{t\Delta}^{c \leftrightharpoons j}(\boldsymbol{\varpi}_t)(\tfrac{1}{C} - \varpi_{tj}). \tag{2.12}$$

Note as the number of unique actions among $a_t^{m\leftrightharpoons j}$ is as few as one, in which case the ARS/ARSM gradient is zero and there is no need at all to estimate the Q function, and as many as C, in which case one needs to estimate the Q function C times. Thus if the computation of estimating Q once is $O(1)$, then the worst computation for an episode that lasts T time steps before termination is $O(TC)$. Usually the number of distinct pseudo actions will decrease dramatically as the training progresses. We illustrate this in Figure A.4, where we show the trace of categorical variable's entropy and number of distinct pseudo actions that differ from the true action. Examining (2.11) and (2.12) shows that the ARS/ARSM policy gradient estimator can be intuitively understood as a "try-and-see self-critic" method, which eliminates the need of constructing baselines and estimating their parameters for variance reduction. To decide the gradient direction of whether increasing the probability of action c at a given state, it compares the pseudo-action reward $Q(\boldsymbol{s}_t, a_t^{c\leftrightharpoons j})$ with the average of all pseudo-action rewards $\{Q(\boldsymbol{s}_t, a_t^{m\leftrightharpoons j})\}_{m=1,C}$. If the current policy is very confident on taking action a_t at state $\boldsymbol{s}_t$, which means ϕ_{ta_t} dominates the other $C-1$ elements of $\boldsymbol{\phi}_t = \mathcal{T}_{\boldsymbol{\theta}}(\boldsymbol{s}_t)$, then it is very likely that $a_t^{m\leftrightharpoons j_t} = a_t$ for all m, which will lead to zero gradient at time t. On the contrary, if the current policy is uncertain about which action to choose, then more pseudo actions that are different from the true action are likely to be generated. This mechanism encourages exploration when the policy is uncertain, and balance the tradeoff of exploration and exploitation intrinsically. It also explains our empirical observations that ARS/ARSM tends to generate a large number of unique pseudo actions in the early stages of training, leading to fast convergence, and significantly reduced number once the policy becomes sufficiently certain, leading to stable performance after convergence.

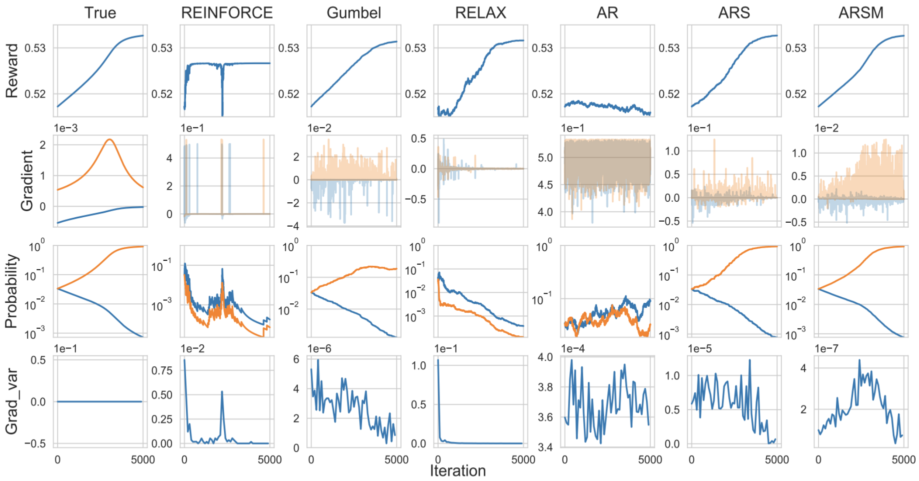


Figure 2.1: Comparison of a variety of gradient estimators in maximizing (2.13). The optimal solution is $\sigma(\boldsymbol{\phi}) = (0, \ldots, 1)$, which means $z = C$ with probability one. The reward is computed analytically by $\mathbb{E}_{z\sim\text{Cat}(\sigma(\boldsymbol{\phi}))}[f(z)]$ with maximum as 0.533. Rows 1, 2, and 3 show the trace plots of reward $\mathbb{E}[f(z)]$, the gradients with respect to ϕ_1 and ϕ_C, and the probabilities $\sigma(\boldsymbol{\phi})_1$ and $\sigma(\boldsymbol{\phi})_C$, respectively. Row 4 shows the gradient variance estimation with 100 Monte Carlo samples at each iteration, averaged over categories 1 to C.

2.4 Experimental Results

In this section, we use a toy example for illustration, demonstrate both multivariate and hierarchical settings with categorical latent variable models, and demonstrate the sequential setting with discrete-action policy optimization. Comparison of gradient variance between various algorithms can be found in Figures 2.1 and 2.3-A.3.

2.4.1 Example Results on Toy Data

To illustrate the working mechanism of the ARSM estimator, we consider learning $\boldsymbol{\phi} \in \mathbb{R}^C$ to maximize

$$\mathbb{E}_{z\sim \text{Cat}(\sigma(\boldsymbol{\phi}))}[f(z)], \quad f(z) := 0.5 + z/(CR), \tag{2.13}$$

where $z \in \{1, \ldots, C\}$. The optimal solution is $\sigma(\boldsymbol{\phi}) = (0, \ldots, 0, 1)$, which leads to the maximum expected reward of $0.5 + 1/R$. The larger the C and/or R are, the more challenging the optimization becomes. We first set $C = R = 30$ that are small enough to allow existing algorithms to perform reasonably well. Further increasing C or R will often fail existing algorithms and ARS, while ARSM always performs almost as good as the true gradient when used in optimization via gradient ascent. We include the results for $C = 1,000$ and $10,000$ in Figures A.1 and A.2 of the Appendix.

We perform an ablation study of the proposed AR, ARS, and ARSM estimators. We also make comparison to two representative low-variance estimators, including the biased Gumbel-Softmax estimator [Jang et al., 2017, Maddison et al., 2017] that applies the reparameterization trick after continuous relaxation of categorical variables, and the unbiased RELAX estimator of Grathwohl et al. [2018] that combines reparameterization and REINFORCE with an adaptively estimated baseline. We compare them in terms of the expected reward as $\sum_{c=1}^{C} \sigma(\boldsymbol{\phi})_c f(c)$, gradients for ϕ_c, probabilities $\sigma(\boldsymbol{\phi})_c$, and gradient variance. Note when $C = 2$, both ARS and ARSM reduce to the ARM estimator, which has been shown in Yin and Zhou [2019] to outperform a wide variety of estimators for binary variables, including the REBAR estimator of Tucker et al. [2017]. The true gradient in this example can be computed analytically as in (2.3). All estimators in comparison use a single Monte Carlo sample for gradient estimation. We initialize $\phi_c = 0$ for all c and fix the gradient-ascent stepsize as one.

As shown in Figure 2.1, without appropriate variance reduction, both AR and REINFORCE either fail to converge or converge to a low-reward solution. We notice RELAX for $C = R = 30$ is not that stable across different runs; in this particular run, it manages to obtain a relatively high reward, but its probabilities converge towards a solution that is different from the optimum $\sigma(\boldsymbol{\phi}) = (0, \ldots, 0, 1)$. By contrast, Gumbel-Softmax, ARS, and ARSM all robustly reach probabilities close to the optimum $\sigma(\boldsymbol{\phi}) = (0, \ldots, 0, 1)$ after 5000 iterations across all random trials. The gradient variance of ARSM is about one to four magnitudes less than these of the other estimators, which helps explain why ARSM is almost identical to the true gradient in moving $\sigma(\boldsymbol{\phi})$ towards the optimum that maximizes the expected reward. The advantages of ARSM become even clearer in more complex settings where analytic gradients become intractable to compute, as shown below.

2.4.2 Categorical Variational Auto-Encoders

For optimization involving expectations with respect to multivariate categorical variables, we consider a variational auto-encoder (VAE) with a single categorical stochastic hidden layer. We further consider a categorical

VAE with two categorical stochastic hidden layers to illustrate optimization involving expectations with respect to hierarchical multivariate categorical variables.

Following Jang et al. [2017], we consider a VAE with a categorical hidden layer to model D-dimensional binary observations. The decoder parameterized by $\boldsymbol{\theta}$ is expressed as $p_{\boldsymbol{\theta}}(\boldsymbol{x} \,|\, \boldsymbol{z}) = \prod_{i=1}^{D} p_{\boldsymbol{\theta}}(x_i \,|\, \boldsymbol{z})$, where $\boldsymbol{z} \in \{1, \ldots, C\}^K$ is a K-dimensional C-way categorical vector and $p_{\boldsymbol{\theta}}(x_i \,|\, \boldsymbol{z})$ is Bernoulli distributed. The encoder parameterized by $\boldsymbol{\phi}$ is expressed as $q_{\boldsymbol{\phi}}(\boldsymbol{z} \,|\, \boldsymbol{x}) = \prod_{k=1}^{K} q_{\boldsymbol{\phi}}(z_k \,|\, \boldsymbol{x})$. We set the prior as $p(z_k = c) = 1/C$ for all c and k. For optimization, we maximize the evidence lower bound (ELBO) as

$$\mathcal{L}(\boldsymbol{x}) = \mathbb{E}_{\boldsymbol{z} \sim q_{\boldsymbol{\phi}}(\boldsymbol{z} \,|\, \boldsymbol{x})} \big[\ln \ \tfrac{p_{\boldsymbol{\theta}}(\boldsymbol{x} \,|\, \boldsymbol{z}) p(\boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z} \,|\, \boldsymbol{x})} \big]. \tag{2.14}$$

We also consider a two-categorical-hidden-layer VAE, whose encoder and decoder are constructed as

$$q_{\boldsymbol{\phi}_{1:2}}(\boldsymbol{z}_1, \boldsymbol{z}_2 \,|\, \boldsymbol{x}) = q_{\boldsymbol{\phi}_1}(\boldsymbol{z}_1 \,|\, \boldsymbol{x}) q_{\boldsymbol{\phi}_2}(\boldsymbol{z}_2 \,|\, \boldsymbol{z}_1),$$
$$p_{\boldsymbol{\theta}_{1:2}}(\boldsymbol{x} \,|\, \boldsymbol{z}_1, \boldsymbol{z}_2) = p_{\boldsymbol{\theta}_1}(\boldsymbol{x} \,|\, \boldsymbol{z}_1) p_{\boldsymbol{\theta}_2}(\boldsymbol{z}_1 \,|\, \boldsymbol{z}_2),$$

where $\boldsymbol{z}_1, \boldsymbol{z}_2 \in \{1, \ldots, C\}^K$. The ELBO is expressed as

$$\mathcal{L}(\boldsymbol{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}_{1:2}}(\boldsymbol{z}_1, \boldsymbol{z}_2 \,|\, \boldsymbol{x})} \Big[\ln \tfrac{p_{\boldsymbol{\theta}_1}(\boldsymbol{x} \,|\, \boldsymbol{z}_1) p_{\boldsymbol{\theta}_2}(\boldsymbol{z}_1 \,|\, \boldsymbol{z}_2) p(\boldsymbol{z}_2)}{q_{\boldsymbol{\phi}_1}(\boldsymbol{z}_1 \,|\, \boldsymbol{x}) q_{\boldsymbol{\phi}_2}(\boldsymbol{z}_2 \,|\, \boldsymbol{z}_1)} \Big]. \tag{2.15}$$

For both categorical VAEs, we set $K = 20$ and $C = 10$. We train them on a binarized MNIST dataset as in van den Oord et al. [2017] by thresholding each pixel value at 0.5. Implementations of the VAEs with one and two

Table 2.1: Comparison of training and testing negative ELBOs (nats) on binarized MNIST between ARSM and various gradient estimators.

Gradient estimator	REINFORCE	RELAX	ST Gumbel-S.	AR	ARS	ARSM	||	Gumbel-S.-2layer	ARSM-2layer
−ELBO (Training)	127.0	117.4	94.1	133.6	97.4	82.0	||	91.3	**78.3**
−ELBO (Testing)	127.6	118.7	96.4	135.0	101.4	**86.7**	||	98.3	89.5

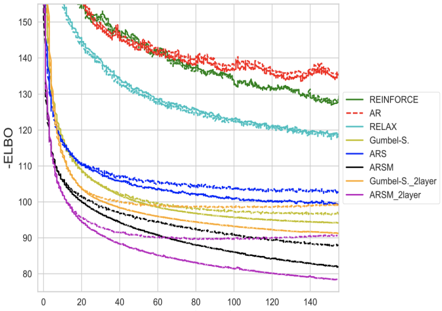


Figure 2.2: Plots of negative ELBOs (nats) on binarized MNIST against training iterations (analogous ones against times are shown in Figure A.5). The solid and dash lines correspond to the training and testing respectively (best viewed in color).

categorical hidden layers are summarized in Algorithms 2 and 3, respectively; see the provided code for more details.

We consider the AR, ARS, and ARSM estimators, and include the REINFORCE [Williams, 1992b], Gumbel-Softmax [Jang et al., 2017], and RELAX [Grathwohl et al., 2018] estimators for comparison. We note that Jang et al. [2017] has already shown Gumbel-Softmax outperforms a wide variety of previously proposed estimators; see Jang et al. [2017] and the references therein for more details.

We present the trace plots of the training and validation negative ELBOs in Figure 2.2 and gradient variance in Figure A.3. The numerical values are summarized in Table 2.1. We use the Gumbel-Softmax code [1] to obtain the results of the VAE with a single categorical hidden layer, and modify it with our best effort for the VAE with two categorical hidden layers; we modify the RELAX code [2] with our best effort to allow it to optimize VAE with a single categorical hidden layer. For the single-hidden-layer VAE, we connect its latent categorical layer $\boldsymbol{z}$ and observation layer $\boldsymbol{x}$ with two nonlinear deterministic layers; for the two-hidden-layer VAE, we add an additional categorical hidden layer $\boldsymbol{z}_2$ that is linearly connected to the first one. See Table A.1 of the Appendix for detailed network architectures. In our experiments, all methods use exactly the same network architectures and data, set the mini-batch size as 200, and are trained by the Adam optimizer Kingma and Ba [2014], whose learning rate is selected from $\{5, 1, 0.5\} \times 10^{-4}$ using the validation set.

The results in Table 2.1 and Figure 2.2 clearly show that for optimizing the single-categorical-hidden-layer VAE, both ARS and ARSM estimators outperform all the other ones in both training and testing ELBOs. In particular, ARSM outperforms all the other estimators by a large margin. We also consider Gumbel-Softmax by computing its gradient with 25 Monte Carlo samples, making it run as fast as the provided ARSM code does per iteration. In this case, both algorithms take similar time but ARSM achieves $-$ELBOs for the training and testing sets as 94.6 and 100.6, respectively, while those of Gumbel-Softmax are 102.5 and 103.6, respectively. The performance gain of ARSM can

---

[1] https://github.com/ericjang/gumbel-softmax

[2] https://github.com/duvenaud/relax

be explained by both its unbiasedness and a clearly lower variance exhibited by its gradient estimates in comparison to all the other estimators, as shown in Figure A.3 of the Appendix. The results on the two-categorical-hidden-layer VAE, which adds a linear categorical layer on top of the single-categorical-hidden-layer VAE, also suggest that ARSM can further improve its performance by adding more stochastic hidden layers and clearly outperforms the biased Gumbel-Softmax estimator.

2.4.3 Maximum Likelihood Estimation for a Stochastic Categorical Network

Denoting $\boldsymbol{x}_l, \boldsymbol{x}_u \in \mathbb{R}^{392}$ as the lower and upper halves of an MNIST digit, respectively, we consider a standard benchmark task of estimating the conditional distribution $p_{\boldsymbol{\theta}_{0:2}}(\boldsymbol{x}_l \,|\, \boldsymbol{x}_u)$ [Raiko et al., 2014, Bengio et al., 2013, Gu et al., 2016a, Jang et al., 2017, Tucker et al., 2017]. We consider a stochastic categorical network with two stochastic categorical hidden layers, expressed as

$$\begin{aligned}
\boldsymbol{x}_l &\sim \text{Bernoulli}(\sigma(\mathcal{T}_{\boldsymbol{\theta}_0}(\boldsymbol{b}_1))),\\
\boldsymbol{b}_1 &\sim \textstyle\prod_{c=1}^{20} \text{Cat}(b_{1c}; \sigma(\mathcal{T}_{\boldsymbol{\theta}_1}(\boldsymbol{b}_2)_{[10(c-1)+(1:10)]})),\\
\boldsymbol{b}_2 &\sim \textstyle\prod_{c=1}^{20} \text{Cat}(b_{2c}; \sigma(\mathcal{T}_{\boldsymbol{\theta}_2}(\boldsymbol{x}_u)_{[10(c-1)+(1:10)]})),
\end{aligned}$$

where both $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ are 20-dimensional 10-way categorical variables, $\mathcal{T}_{\boldsymbol{\theta}}(\cdot)$ denotes linear transform, $\mathcal{T}_{\boldsymbol{\theta}_2}(\boldsymbol{x}_u)_{[10(c-1)+(1:10)]}$ is a 10-dimensional vector consisting of elements $10(c-1)+1$ to $10c$ of $\mathcal{T}_{\boldsymbol{\theta}_2}(\boldsymbol{x}_u) \in \mathbb{R}^{200}$, $\mathcal{T}_{\boldsymbol{\theta}_1}(\boldsymbol{b}_2) \in \mathbb{R}^{200}$, and $\mathcal{T}_{\boldsymbol{\theta}_0} \in \mathbb{R}^{392}$. Thus we can consider the network structure as 392-200-200-392, making the results directly comparable with these in Jang et al. [2017] for

Table 2.2: Comparison of the test negative log-likelihoods between ARSM and various gradient estimators in Jang et al. [2017], for the MNIST conditional distribution estimation benchmark task.

Gradient estimator	ARSM	ST	Gumbel-S.	MuProp
$-\log p(\boldsymbol{x}_l \mid \boldsymbol{x}_u)$	**58.3 ± 0.2**	61.8	59.7	63.0

stochastic categorical network. We approximate $\log p_{\boldsymbol{\theta}_{0:2}}(\boldsymbol{x}_l \mid \boldsymbol{x}_u)$ with K Monte Carlo samples as

$$\log \tfrac{1}{K} \textstyle\sum_{k=1}^{K} \text{Bernoulli}(\boldsymbol{x}_l; \sigma(\mathcal{T}_{\boldsymbol{\theta}_0}(\boldsymbol{b}_1^{(k)}))), \tag{2.16}$$

where $\boldsymbol{b}_1^{(k)} \sim \prod_{c=1}^{20} \text{Cat}(b_{1c}^{(k)}; \sigma(\mathcal{T}_{\boldsymbol{\theta}_1}(\boldsymbol{b}_2^{(k)})_{[10(c-1)+(1:10)]}))$,
$\boldsymbol{b}_2^{(k)} \sim \prod_{c=1}^{20} \text{Cat}(b_{2c}^{(k)}; \sigma(\mathcal{T}_{\boldsymbol{\theta}_2}(\boldsymbol{x}_u)_{[10(c-1)+(1:10)]}))$. We perform training with $K = 1$, which can also be considered as optimizing on a single-Monte-Carlo-sample estimate of the lower bound of the log marginal likelihood. We use Adam [Kingma and Ba, 2014], with the learning rate set as 10^{-4}, mini-batch size as 100, and number of training epochs as 2000. Given the inferred point estimate of $\boldsymbol{\theta}_{0:2}$, we evaluate the accuracy of conditional density estimation by estimating the negative log-likelihood $-\log p_{\boldsymbol{\theta}_{0:2}}(\boldsymbol{x}_l \mid \boldsymbol{x}_u)$ using (2.16), averaging over the test set with $K = 1000$.

As shown in Table 2.2, optimizing a stochastic categorical network with the ARSM estimator achieves the lowest test negative log-likelihood, outperforming all previously proposed gradient estimators on the same structured stochastic networks, including straight through (ST) [Bengio et al., 2013] and ST Gumbel-softmax [Jang et al., 2017] that are biased, and MuProp [Gu et al., 2016a] that is unbiased.

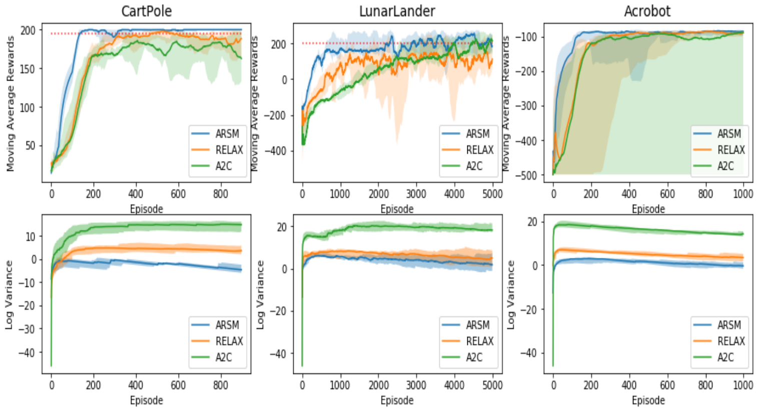


Figure 2.3: *Top row:* Moving average reward curves. *Bottom row:* Log-variance of gradient estimator. In each plot, the solid lines are the median value of ten independent runs (ten different random seeds for random initializations). The opaque bars are 10th and 90th percentiles. Dashed straight lines in Cart Pole and Lunar Lander represent task-completion criteria.

2.4.4 Discrete-Action Policy Optimization

The key of applying the ARSM policy gradient shown in (2.12) is to provide, under the current policy $\pi_{\boldsymbol{\theta}}$, the action-value functions' sample estimates $\hat{Q}(\boldsymbol{s}_t, a_t) := \sum_{t'=t}^{\infty} \gamma^{t'-t} r(\boldsymbol{s}_{t'}, a_{t'})$ for all unique values in $\{a_t^{c\rightleftharpoons j}\}_{c,j}$. Thus ARSM is somewhat related to the *vine* method proposed in Schulman et al. [2015a], which defines a heuristic rollout policy that chooses a subset of the states along the true trajectory as the "rollout set," samples K pseudo actions uniformly at random from the discrete-action set at each state of the rollout set, and performs a single rollout for each state-pseudo-action-pair to estimate its action-value function Q. ARSM chooses its rollout set in the same manner, but is distinct from the *vine* method in having a rigorously derived rollout policy: it swaps the elements of $\boldsymbol{\varpi}_t \sim \mathrm{Dir}(\mathbf{1}_C)$ to generate pseudo actions if state $\boldsymbol{s}_t$ belongs to the rollout set; the number of unique pseudo actions that are different from the true action a_t is a random number, which is positively related to the uncertainty of the policy and hence often negatively related to its convergence; and a single rollout is then performed for each of these unique pseudo actions to estimate its Q.

As ARSM requires the estimation of Q function for each unique state-pseudo-action pair using Monte Carlo rollout, it could have high computational complexity if (1) the number of unique pseudo actions is large, and (2) each rollout takes many expensive steps (interactions with the environments) before termination. However, there exist ready solutions and many potential ones. As given a true trajectory, all the state-pseudo-action rollouts of ARSM can be independently simulated and hence all pseudo-action related Q's can be estimated in an embarrassingly parallel manner. Furthermore, in addition to Monte Carlo estimation, we can potentially adapt for ARSM a wide variety of off-the-shelf action-value function estimation methods [Sutton and Barto, 1998], to either accelerate the estimation of Q or further reduce the variance (though possibly at the expense of introducing bias). In our experiment, for simplicity and clarity, we choose to use Monte Carlo estimation to obtain $\hat{Q}$ for both the true trajectory and all state-pseudo-action rollouts. The results for RELAX and A2C are obtained by running the code provided by Grathwohl et al. [2018][3].

We apply the ARSM policy gradient to three representative RL tasks

---

[3] https://github.com/wgrathwohl/BackpropThroughTheVoidRL

with discrete actions, including the Cart Pole, Acrobot, and Lunar Lander environments provided by OpenAI Gym [Brockman et al., 2016], and compare it with advantage actor-critic algorithm (A2C) [Sutton et al., 2000] and RELAX [Grathwohl et al., 2018]. We report the moving-average rewards and the estimated log-variance of the gradient estimator at every episode; for each episode, the reward score is obtained by running the updated policy on a new random environment; and the variance is obtained by first applying exponential moving averages to the first and second moments of each neural network parameter with decay 0.99, and then taking the average of the estimated variances of all neural network parameters.

Shown in Figure 2.3 are the mean rewards over the last 100 steps; the opaque bar indicates 10th and 90th percentiles obtained by ten independent runs for each method (using 10 different random seeds for random initializations); the solid line is the median value of these ten independent runs. ARSM outperform both baselines in all three tasks in terms of stability, moving average rewards, and log-variance of gradient estimator. All methods are cross validated by optimizers {Adam Optimizer, RMSProp Optimizer} and learning rates $\{1, 3, 10, 30\} \times 10^{-3}$. Both the policy and critic networks for A2C and RELAX have two 10-unit hidden layers with ReLU activation functions [Nair and Hinton, 2010]. The discount factor γ is 0.99 and entropy term is 0.01. The policy network of ARSM is the same as that of A2C and RELAX, and the maximum number of allowed state-pseudo-action rollouts of ARSM is set as 16, 64, and 1024 for Cart Pole, Acrobot, and Lunar Lander, respectively;

see Algorithm 4 and the provided code for more details. Using our current implementation that has not been optimized to fully take the advantage of parallel computing, to finish the number of episodes as in Figure 2.3, ARSM on average takes 677, 425, and 19050 seconds for CartPole, Acrobot, and LunarLander, respectively. For comparison, for these three tasks, RELAX on average takes 139, 172, and 3493 seconds and A2C on average takes 92, 120, and 2708 seconds.

2.5 Conclusion

To backpropagate the gradients through categorical stochastic layers, we propose the augment-REINFORCE-swap-merge (ARSM) estimator that is unbiased and exhibits low variance. The performance of ARSM is almost identical to that of the true gradient when used for optimization involving a C-way categorical variable, even when C is very large (such as $C = 10,000$). For multiple C-way categorical variables organized into a single stochastic layer, multiple stochastic layers, or a sequential setting, the ARSM estimator clearly outperforms state-of-the-art methods, as shown in our experimental results for both categorical latent variable models and discrete-action policy optimization. We attribute the outstanding performance of ARSM to both its unbiasedness and its ability to control variance by simply combing its reward function with randomly generated pseudo actions, where the number of unique pseudo actions is positively related to the uncertainties of categorical distributions and hence negatively correlated to how well the optimization algorithm has converged; there is no more need to construct separate baselines and estimate their parameters, which also help make the optimization more robust. Some natural extensions of the proposed ARSM estimator include applying it to reinforcement learning with high-dimensional discrete-action spaces or multiple discrete-action agents, and various tasks in natural language processing such as sentence generation and machine translation.

Chapter 3

Discrete Action On-Policy Learning with Action-Value Critic

Reinforcement learning (RL) in discrete action space is ubiquitous in real-world applications, but its complexity grows exponentially with the action-space dimension, making it challenging to apply existing on-policy gradient based deep RL algorithms efficiently. To effectively operate in multidimensional discrete action spaces, we construct a critic to estimate action-value functions, apply it on correlated actions, and combine these critic estimated action values to control the variance of gradient estimation. We follow rigorous statistical analysis to design how to generate and combine these correlated actions, and how to sparsify the gradients by shutting down the contributions from certain dimensions. These efforts result in a new discrete action on-policy RL algorithm that empirically outperforms related on-policy algorithms relying on variance control techniques. We demonstrate these properties on OpenAI Gym benchmark tasks, and illustrate how discretizing the action space could benefit the exploration phase and hence facilitate convergence to a better local

---

The content in this chapter was published in Yue et al. [2020a]; I brought up the algorithm, and conducted experiments for CARSM algorithm. Dr. Zhou and I worked together to come up with the toy example for demonstrations.

optimal solution thanks to the flexibility of discrete policy.

3.1 Introduction

There has been significant recent interest in using model-free reinforcement learning (RL) to address complex real-world sequential decision making tasks [Silver et al., 2018, MacAlpine and Stone, 2017, OpenAI, 2018]. With the help of deep neural networks, model-free deep RL algorithms have been successfully implemented in a variety of tasks, including game playing [Silver et al., 2016, Mnih et al., 2013] and robotic controls [Levine et al., 2016]. Among those model-free RL algorithms, policy gradient (PG) algorithms are a class of methods that parameterize the policy function and apply gradient-based methods to make updates. It has been shown to succeed in solving a range of challenging RL tasks [Mnih et al., 2016, Schulman et al., 2015a, Lillicrap et al., 2015, Schulman et al., 2017, Wang et al., 2016, Haarnoja et al., 2018a, Liu et al., 2017b]. Despite directly targeting at maximizing the expected rewards, PG suffers from problems including having low sample efficiency [Haarnoja et al., 2018a] for on-policy PG algorithms and undesirable sensitivity to hyper-parameters for off-policy algorithms [Lillicrap et al., 2015].

On-policy RL algorithms use on-policy samples to estimate the gradients for policy parameters, as routinely approximated by Monte Carlo (MC) estimation that often comes with large variance. A number of techniques have sought to alleviate this problem for continuous action spaces [Gu et al., 2016b, Grathwohl et al., 2018, Liu et al., 2017a, Wu et al., 2018], while relatively fewer have been proposed for discrete action spaces [Grathwohl et al., 2018, Yin et al., 2019]. In practice, RL with discrete action space is ubiquitous in fields including recommendation system [Dulac-Arnold et al., 2015], bidding system [Hu et al., 2018], gaming [Mnih et al., 2013], to name a few. It plays an important role in the early stage of RL development [Sutton and Barto, 1998], and many value-based algorithms [Watkins and Dayan, 1992, Mnih et al., 2013, Van Hasselt et al., 2016] can handle such setup when the action space is not large. However, when the action space is multidimensional, the number of unique actions grows exponentially with the dimension, leading to an intractable combinatorial optimization problem at every single step that prevents the application of most value-based RL methods.

Under the setting of high-dimensional discrete action space, policy-gradient based algorithms can still be applied if we assume the joint distribution over discrete actions to be factorized across dimensions, so that the joint policy is still tractable [Jaśkowski et al., 2018, Andrychowicz et al., 2018]. Then the challenge boils down to obtaining a gradient estimator that can well control its variance. Though many variance reduction techniques have been proposed for discrete variables [Jang et al., 2017, Tucker et al., 2017, Yin and Zhou, 2018a, Raiko et al., 2014], they either provide biased gradients or are not applicable to multidimensional RL settings.

In this paper, we propose Critic-ARSM (CARSM) policy gradient, which improves the recently proposed augment-REINFORCE-swap-merge (ARSM) gradient estimator of Yin et al. [2019] and integrates it with action-value function evaluation, to accomplish three-fold effects: **1)** CARSM sparsifies the ARSM gradient and introduces an action-value Critic to work with multidimensional discrete actions spaces; **2)** By estimating the rewards of a set of correlated discrete actions via the proposed action-value Critic, and combining these rewards for variance reduction, CARSM achieves better sample efficiency compared with other variance-control methods such as A2C [Mnih et al., 2016] and RELAX [Grathwohl et al., 2018]; **3)** CARSM can be easily applied to other RL algorithms using REINFORCE or its variate as the gradient estimator. Although we mainly focus on on-policy algorithms, our algorithm can also be potentially applied to off-policy algorithms with the same principle; we leave this extension for future study.

The paper proceeds as follows. In Section 2, we briefly review existing on-policy learning frameworks and variance reduction techniques for discrete action space. In Section 3, we introduce CARSM from both theoretical and practical perspectives. In Section 4, we first demonstrate the potential benefits of discretizing a continuous control task compared with using a diagonal Gaussian policy, then show the high sample efficiency of CARSM from an extensive range of experiments and illustrate that CARSM can be plugged into state-of-arts on-policy RL learning frameworks such as Trust Region Policy Optimization (TRPO) [Schulman et al., 2015a]. Python (TensorFlow) code is available at `https://github.com/yuguangyue/CARSM`.

3.2 Preliminaries

RL is often formulated as learning under a Markov decision process (MDP). Its action space $\mathcal{A}$ is dichotomized into either discrete (*e.g.*, $\mathcal{A} = \{1, \ldots, 100\}$) or continuous (*e.g.*, $\mathcal{A} = [-1, 1]$). In an MDP, at discrete time $t \geq 0$, an agent in state $\boldsymbol{s}_t \in \mathcal{S}$ takes action $a_t \in \mathcal{A}$, receives instant reward $r(\boldsymbol{s}_t, a_t) \in \mathbb{R}$, and transits to next state $\boldsymbol{s}_{t+1} \sim \mathcal{P}(\cdot \mid \boldsymbol{s}_t, a_t)$. Let $\pi : \mathcal{S} \mapsto \mathcal{P}(\mathcal{A})$ be a mapping from the state to a distribution over actions. We define the expected cumulative rewards under π as

$$J(\pi) = \mathbb{E}_\pi \left[\textstyle\sum_{t=0}^{\infty} \gamma^t r(\boldsymbol{s}_t, a_t) \right], \tag{3.1}$$

where $\gamma \in (0, 1]$ is a discount factor. The objective of RL is to find the (sub-)optimal policy $\pi^* = \arg\max_\pi J(\pi)$. In practice, it is infeasible to search through all policies and hence one typically resorts to parameterizing the policy $\pi_{\boldsymbol{\theta}}$ with $\boldsymbol{\theta}$.

3.2.1 On-Policy Optimization

We introduce on-policy optimization methods from a constrained optimization point of view to unify the algorithms we will discuss in this article. In practice, we want to solve the following constrained optimization problem as illustrated in Schulman et al. [2015a]:

$$\max_{\boldsymbol{\theta}} \ \mathbb{E}_{\pi_{\boldsymbol{\theta}_{\text{old}}}} \left[\tfrac{\pi_{\boldsymbol{\theta}}(a_t \mid \boldsymbol{s}_t)}{\pi_{\boldsymbol{\theta}_{\text{old}}}(a_t \mid \boldsymbol{s}_t)} Q^{\pi_{\boldsymbol{\theta}_{\text{old}}}}(\boldsymbol{s}_t, a_t) \right]$$

$$\text{subject to } D(\boldsymbol{\theta}_{\text{old}}, \boldsymbol{\theta}) \leq \epsilon,$$

where $Q^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}_t, a_t) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[\sum_{t'=t} \gamma^{t'-t} r(s'_t, a'_t)]$ is the action-value function and $D(\cdot, \cdot)$ is some metric that measures the closeness between $\boldsymbol{\theta}_{\text{old}}$ and $\boldsymbol{\theta}$.

A2C Algorithm: One choice of $D(\cdot,\cdot)$ is the L_2 norm, which will lead us to first-order gradient ascent. By applying first-order Taylor expansion on $\pi_{\boldsymbol{\theta}}$ around $\boldsymbol{\theta}_{\text{old}}$, the problem can be re-written as maximizing $\mathbb{E}_{\pi_{\boldsymbol{\theta}_{\text{old}}}}[Q^{\pi_{\boldsymbol{\theta}_{\text{old}}}}(\boldsymbol{s}_t, a_t)] + \nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}_{\text{old}}})^T(\boldsymbol{\theta} - \boldsymbol{\theta}_{\text{old}})$ subject to $||\boldsymbol{\theta} - \boldsymbol{\theta}_{\text{old}}||_2 \leq \epsilon$, which will result in a gradient ascent update scheme; note $\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}_{\text{old}}}) := \nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}})|_{\boldsymbol{\theta}=\boldsymbol{\theta}_{\text{old}}}$. Based on REINFORCE [Williams, 1992a], the gradient of the original objective function (3.1) can be written as

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left[\textstyle\sum_{t=0}^{\infty} Q^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}_t, a_t) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t|\boldsymbol{s}_t)\right]. \tag{3.2}$$

However, a naive Monte Carlo estimation of (3.2) has large variance that needs to be controlled. A2C algorithm [Mnih et al., 2016] adds value function $V^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}) := \mathbb{E}_{a_t \sim \pi_\theta}[Q^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}_t, a_t)]$ as a baseline and obtains a low-variance estimator of $\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}})$ as

$$\boldsymbol{g}_{\text{A2C}} = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left[\textstyle\sum_{t=0}^{\infty} A^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}_t, a_t) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t|\boldsymbol{s}_t)\right], \tag{3.3}$$

where $A^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}_t, a_t) = Q^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}_t, a_t) - V^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}_t)$ is called the Advantage function.

Trust Region Policy Optimization: The other choice of metric $D(\cdot,\cdot)$ could be KL-divergence, and the update from this framework is introduced as TRPO [Schulman et al., 2015a]. In practice, this constrained optimization problem is reformulated as follows:

$$\begin{gathered}\max_{\boldsymbol{\theta}} \ \nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}_{\text{old}}})^T(\boldsymbol{\theta} - \boldsymbol{\theta}_{\text{old}}) \\ \text{subject to } \tfrac{1}{2}(\boldsymbol{\theta}_{\text{old}} - \boldsymbol{\theta})^T H (\boldsymbol{\theta}_{\text{old}} - \boldsymbol{\theta}) \leq \delta,\end{gathered}$$

where H is the second-order derivative $\nabla^2_{\boldsymbol{\theta}} D_{\text{KL}}(\boldsymbol{\theta}_{\text{old}}||\boldsymbol{\theta})|_{\boldsymbol{\theta}=\boldsymbol{\theta}_{\text{old}}}$. An analytic update step for this optimization problem can be expressed as

$$\boldsymbol{\theta} = \boldsymbol{\theta}_{\text{old}} + \sqrt{\tfrac{2\delta}{\boldsymbol{d}^T H^{-1} \boldsymbol{d}}}\boldsymbol{d}, \tag{3.4}$$

where $\boldsymbol{d} = H^{-1}\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}_{\text{old}}})$, and in practice the default choice of $\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}_{\text{old}}})$ is $\boldsymbol{g}_{\text{A2C}}$ as defined at (3.3).

3.2.2 Variance Control Techniques

Besides the technique of using state-dependent baseline to reduce variance as in (3.3), two recent works propose alternative methods for variance reduction in discrete action space settings. For the sake of space, we defer to Grathwohl et al. [2018] for the detail about the RELAX algorithm and briefly introduce ARSM here.

ARSM Policy Gradient: The ARSM gradient estimator can be used to backpropagate unbiased and low-variance gradients through a sequence of unidimensional categorical variables [Yin et al., 2019]. It comes up with a reparametrization formula for discrete random variable, and combines it with a parameter-free self-adjusted baseline to achieve variance reduction.

Instead of manipulating on policy parameters $\boldsymbol{\theta}$ directly, ARSM turns to reduce variance on the gradient with respect to the logits $\boldsymbol{\phi}$, before backpropagating it to $\boldsymbol{\theta}$ using the chain rule. Let us assume

$$\pi_{\boldsymbol{\theta}}(a_t \,|\, \boldsymbol{s}_t) = \text{Categorical}(a_t \,|\, \sigma(\boldsymbol{\phi}_t)),\ \boldsymbol{\phi}_t := \mathcal{T}_{\boldsymbol{\theta}}(\boldsymbol{s}_t),$$

where $\sigma(\cdot)$ denotes the softmax function and $\mathcal{T}_{\boldsymbol{\theta}}(\cdot) \in \mathbb{R}^C$ denotes a neural network, which is parameterized by $\boldsymbol{\theta}$ and has an output dimension of C.

Denote $\boldsymbol{\varpi}^{c \rightleftharpoons j}$ as the vector obtained by swapping the cth and jth elements of vector $\boldsymbol{\varpi}$, which means $\varpi_j^{c \rightleftharpoons j} = \varpi_c$, $\varpi_c^{c \rightleftharpoons j} = \varpi_j$, and $\varpi_i^{c \rightleftharpoons j} = \varpi_i$ if

$i \notin \{c, j\}$. Following the derivation from Yin et al. [2019], the gradient with respect to ϕ_{tc} can be expressed as

$$
\begin{aligned}
\nabla_{\phi_{tc}} J(\boldsymbol{\phi}_{0:\infty}) &= \mathbb{E}_{\mathcal{P}(\boldsymbol{s}_t \,|\, \boldsymbol{s}_0, \pi_{\boldsymbol{\theta}})\mathcal{P}(\boldsymbol{s}_0)} \left\{ \gamma^t \mathbb{E}_{\boldsymbol{\varpi}_t \sim \mathrm{Dir}(\mathbf{1}_C)} \left[g_{tc} \right] \right\}, \\
g_{tc} &:= \sum_{j=1}^{C} \left[Q(\boldsymbol{s}_t, a_t^{c \rightleftharpoons j}) - \frac{1}{C} \sum_{m=1}^{C} Q(\boldsymbol{s}_t, a_t^{m \rightleftharpoons j}) \right] \left(\frac{1}{C} - \varpi_{tj} \right),
\end{aligned}
$$

where $\mathcal{P}(\boldsymbol{s}_t \,|\, \boldsymbol{s}_0, \pi_{\boldsymbol{\theta}})$ is the marginal form of $\prod_{t'=0}^{t-1} \mathcal{P}(\boldsymbol{s}_{t'+1} \,|\, \boldsymbol{s}_{t'}, a_{t'}) \mathrm{Categorical}(a_{t'}; \sigma(\boldsymbol{\phi}_{t'}))$, $\boldsymbol{\varpi}_t \sim \mathrm{Dir}(\mathbf{1}_C)$, and $a_t^{c \rightleftharpoons j} := \arg\min_{i \in \{1,\ldots,C\}} \varpi_{ti}^{c \rightleftharpoons j} e^{-\phi_{ti}}$. In addition, $a_t^{c \rightleftharpoons j}$ is called a *pseudo action* to differentiate it from the true action $a_t =: \arg\min_{i \in \{1,\ldots,C\}} \varpi_{ti} e^{-\phi_{ti}}$.

Applying the chain rule leads to ARSM policy gradient:

$$
\begin{aligned}
\boldsymbol{g}_{\mathrm{ARSM}} &= \textstyle\sum_{t=0}^{\infty} \sum_{c=1}^{C} \frac{\partial J(\boldsymbol{\phi}_{0:\infty})}{\partial \phi_{tc}} \frac{\partial \phi_{tc}}{\partial \boldsymbol{\theta}} \\
&= \mathbb{E}_{\boldsymbol{s}_t \sim \rho_{\pi,\gamma}(\boldsymbol{s})} \left\{ \mathbb{E}_{\boldsymbol{\varpi}_t \sim \mathrm{Dir}(\mathbf{1}_C)} \left[\nabla_{\boldsymbol{\theta}} \textstyle\sum_{c=1}^{C} g_{tc} \phi_{tc} \right] \right\},
\end{aligned}
$$

where $\rho_{\pi,\gamma}(\boldsymbol{s}) := \sum_{t=0}^{\infty} \gamma^t \mathcal{P}(\boldsymbol{s}_t = \boldsymbol{s} \,|\, \boldsymbol{s}_0, \pi_{\boldsymbol{\theta}})$ is the unnormalized discounted state visitation frequency.

In Yin et al. [2019], $Q(\boldsymbol{s}_t, a_t^{c \rightleftharpoons j})$ are estimated by MC integration, which requires multiple MC rollouts at each timestep if there are *pseudo actions* that differ from the true action. This estimation largely limits the implementation of ARSM policy gradient to small action space due to the high computation cost. The maximal number of unique *pseudo actions* grows quadratically with the number of actions along each dimension and a long episodic task will result in more MC rollouts too. To differentiate it from the new algorithm, we refer to it as ARSM-MC.

3.3 CARSM Policy Gradient

In this section, we introduce Critic-ARSM (CARSM) policy gradient for multidimensional discrete action space. CARSM improves ARSM-MC in the following two aspects: 1. ARSM-MC only works for unidimensional RL settings while CARSM generalizes it to multidimensional ones with sparsified gradients. 2. CARSM can be applied to more complicated tasks as it employs an action-value function critic to remove the need of running multiple MC rollouts for a single estimation, which largely improves the sample efficiency.

For an RL task with K-dimensional C-way discrete action space, we assume different dimensions $a_{tk} \in \{1, \dots, C\}$ of the multidimensional discrete action $\boldsymbol{a}_t = (a_{t1}, \dots, a_{tK})$ are independent given logits $\boldsymbol{\phi}_t$ at time t, that is $a_{t1} \perp a_{t2} \cdots \perp a_{tK} \mid \boldsymbol{\phi}_t$. For the logit vector $\boldsymbol{\phi}_t \in \mathbb{R}^{KC}$, which can be decomposed as $\boldsymbol{\phi}_t = (\boldsymbol{\phi}'_{t1}, \dots, \boldsymbol{\phi}'_{tK})'$, $\boldsymbol{\phi}_{tk} = (\phi_{tk1}, \dots, \phi_{tkC})'$, we assume

$$P(\boldsymbol{a}_t \mid \boldsymbol{\phi}_t) = \textstyle\prod_{k=1}^{K} \text{Categorical}(a_{tk}; \sigma(\boldsymbol{\phi}_{tk})).$$

Theorem 5 (Sparse ARSM for multidimensional discrete action space)**.** *The element-wise gradient of $J(\boldsymbol{\phi}_{0:\infty})$ with respect to ϕ_{tkc} can be expressed as*

$$\nabla_{\phi_{tkc}} J(\boldsymbol{\phi}_{0:\infty}) = \mathbb{E}_{\mathcal{P}(\boldsymbol{s}_t \mid \boldsymbol{s}_0, \pi_{\boldsymbol{\theta}})\mathcal{P}(\boldsymbol{s}_0)} \left\{ \gamma^t \mathbb{E}_{\boldsymbol{\Pi}_t \sim \prod_{k=1}^{K} \text{Dir}(\boldsymbol{\varpi}_{tk}; \mathbf{1}_C)} [g_{tkc}] \right\},$$

where $\boldsymbol{\varpi}_{tk} = (\varpi_{tk1}, \dots, \varpi_{tkC})' \sim \text{Dir}(\mathbf{1}_C)$ is the Dirichlet random vector for *dimension k, state t and*

$$
\begin{aligned}
&g_{tkc} := \begin{cases} 0, & \textit{if } a_{tk}^{c\leftrightharpoons j} = a_{tk} \textit{ for all (c,j)} \\ \sum_{j=1}^{C} \left[\Delta_{c,j}(\boldsymbol{s}_t, \boldsymbol{a}_t)\right] \left(\frac{1}{C} - \varpi_{tkj}\right), & \textit{otherwise} \end{cases} \\
&\Delta_{c,j}(\boldsymbol{s}_t, \boldsymbol{a}_t) := Q(\boldsymbol{s}_t, \boldsymbol{a}_t^{c\leftrightharpoons j}) - \tfrac{1}{C} \sum_{m=1}^{C} Q(\boldsymbol{s}_t, \boldsymbol{a}_t^{m\leftrightharpoons j}), \\
&\boldsymbol{a}_t^{c\leftrightharpoons j} := (a_{t1}^{c\leftrightharpoons j}, \ldots, a_{tK}^{c\leftrightharpoons j})', \\
&a_{tk}^{c\leftrightharpoons j} := \arg\min_{i\in\{1,\ldots,C\}} \varpi_{tki}^{c\leftrightharpoons j} e^{-\phi_{tki}}.
\end{aligned}
$$

We defer the proof to Appendix B.1. One difference from the original ARSM [Yin et al., 2019] is the values of g_{tkc}, where we obtain a sparse estimation that shutdowns the kth dimension if $a_{tk}^{c\leftrightharpoons j} = a_{tk}$ for all (c, j) and hence there is no more need to calculate $\nabla_{\boldsymbol{\theta}} \phi_{tkc}$ for all c belonging to dimension k at time t. One immediate benefit from this sparse gradient estimation is to reduce the noise from that specific dimension because the Q function is always estimated with either MC estimation or Temporal Difference (TD) [Sutton and Barto, 1998], which will introduce variance and bias, respectively.

In ARSM-MC, the action-value function is estimated by MC rollouts. Though it returns unbiased estimation, it inevitably decreases the sample efficiency and prevents it from applying to more sophisticated tasks. Therefore, CARSM proposes using an action-value function critic $\hat{Q}_{\boldsymbol{\omega}}$ parameterized by $\boldsymbol{\omega}$ to estimate the Q function. Replacing Q with $\hat{Q}_{\boldsymbol{\omega}}$ in Theorem 5, we obtain $\hat{g}_{tkc} \approx g_{tkc}$ as the empirical estimation of $\nabla_{\phi_{tkc}} J(\pi)$, and hence the CARSM estimation for $\nabla_{\boldsymbol{\theta}} J(\pi) = \sum_{t=0}^{\infty} \sum_{k=1}^{K} \sum_{c=1}^{C} \frac{\partial J(\boldsymbol{\phi}_{0:\infty})}{\phi_{tkc}} \frac{\phi_{tkc}}{\partial \boldsymbol{\theta}}$ becomes

$$\hat{\boldsymbol{g}}_{\text{CARSM}} = \nabla_{\boldsymbol{\theta}} \sum_t \sum_{k=1}^{K} \sum_{c=1}^{C} \hat{g}_{tkc} \phi_{tkc}.$$

Note the number of unique values in $\{\boldsymbol{a}_t^{c\leftrightharpoons j}\}_{c,j}$ that differ from the true action $\boldsymbol{a}_t$ is always between 0 and $C(C-1)/2-1$, regardless of how large K is. The dimension shutdown property further sparsifies the gradients, removing the noise of the dimensions that have no *pseudo actions*.

Design of Critic: A practical challenge of CARSM is that it is notoriously hard to estimate action-value functions for on-policy algorithm because the number of samples are limited and the complexity of the action-value function quickly increases with dimension K. A natural way to overcome the limitation of samples is the reuse of historical data, which has been successfully implemented in previous studies [Gu et al., 2016b, Lillicrap et al., 2015]. The idea is to use the transitions $\{\boldsymbol{s}_\ell, r_\ell, \boldsymbol{a}_\ell, \boldsymbol{s}'_\ell\}$'s from the replay buffer to construct target values for the action-value estimator under the current policy. More specifically, we can use one-step TD to rewrite the target value of critic $\hat{Q}_{\boldsymbol{\omega}}$ network with these off-policy samples as

$$y_\ell^{\text{off}} = r(\boldsymbol{s}_\ell, \boldsymbol{a}_\ell) + \gamma \mathbb{E}_{\tilde{\boldsymbol{a}} \sim \pi(\cdot \,|\, \boldsymbol{s}'_\ell)} \hat{Q}_{\boldsymbol{\omega}}(\boldsymbol{s}'_\ell, \tilde{\boldsymbol{a}}), \tag{3.5}$$

where the expectation part can be evaluated with either an exact computation when the action space size C^K is not large, or with MC integration by drawing random samples from $\tilde{\boldsymbol{a}} \sim \pi_{\boldsymbol{\theta}}(\cdot \,|\, \boldsymbol{s}')$ and averaging $\hat{Q}_{\boldsymbol{\omega}}(\boldsymbol{s}', \tilde{\boldsymbol{a}})$ over these random samples. This target value only uses one-step estimation, and can be extended to n-step TD by adding additional importance sampling weights.

Since we have on-policy samples, it is natural to also include them to construct unbiased targets for $\hat{Q}_{\boldsymbol{\omega}}(\boldsymbol{s}_t, a_t)$:

$$y_t^{\text{on}} = \textstyle\sum_{t'=t}^{\infty} \gamma^{t'-t} r(\boldsymbol{s}_{t'}, \boldsymbol{a}_{t'}).$$

Then we optimize parameters $\boldsymbol{\omega}$ by minimizing the Bellman error between the targets and critic as

$$\textstyle\sum_{\ell=0}^{L}[y_\ell^{\text{off}} - \hat{Q}_{\boldsymbol{\omega}}(\boldsymbol{s}_\ell, \boldsymbol{a}_\ell)]^2 + \sum_{t=0}^{T}[y_t^{\text{on}} - \hat{Q}_{\boldsymbol{\omega}}(\boldsymbol{s}_t, \boldsymbol{a}_t)]^2,$$

where L is the number of off-policy samples and T is the number of on-policy samples. In practice, the performance varies with the ratio between L and T, which reflects the trade-off between bias and variance. We choose $L = T$, which is found to achieve good performance across all tested RL tasks.

Target network update: Another potential problem of CARSM is the dependency between the action-value function and policy. Though CARSM is a policy-gradient based algorithm, the gradient estimation procedure is closely related with the action-value function, which may lead to divergence of the estimation as mentioned in previous studies [Mnih et al., 2016, Lillicrap et al., 2015, Bhatnagar et al., 2009, Maei et al., 2010]. Fortunately, this issue has been addressed, to some extent, with the help of target network update [Mnih et al., 2013, Lillicrap et al., 2015], and we borrow that idea into CARSM for computing policy gradient. In detail, we construct two target networks corresponding to the policy network and Q critic network, respectively; when computing the target of critic network in (3.5), instead of using the current policy network and Q critic, we use a smoothed version of them to obtain the target value, which can be expressed as

$$y_\ell^{\text{off}} = r(\boldsymbol{s}_\ell, \boldsymbol{a}_\ell) + \gamma \mathbb{E}_{\tilde{\boldsymbol{a}} \sim \pi'(\cdot \,|\, \boldsymbol{s}'_\ell)} Q'_{\boldsymbol{\omega}}(\boldsymbol{s}'_\ell, \tilde{\boldsymbol{a}}),$$

where π' and $Q'_{\boldsymbol{\omega}}$ denote the target networks. These target networks are updated every episode in a "soft" update manner, as in Lillicrap et al. [2015], by

$$\boldsymbol{\omega}^{Q'} \leftarrow \tau \boldsymbol{\omega}^{Q} + (1-\tau)\boldsymbol{\omega}^{Q'}, \;\; \boldsymbol{\theta}^{\pi'} \leftarrow \tau \boldsymbol{\theta}^{\pi} + (1-\tau)\boldsymbol{\theta}^{\pi'},$$

which is an exponential moving average of the policy network and action value function network parameters, with τ as the smoothing parameter.

Annealing on entropy term: In practice, maximizing the maximum entropy (ME) objective with an annealing coefficient is often a good choice to encourage exploration and achieve a better sub-optimal solution, and the CARSM gradient estimator for ME would be

$$\boldsymbol{g}_{\text{CARSM}}^{\text{ME}} = \boldsymbol{g}_{\text{CARSM}} + \lambda \textstyle\sum_{k=1}^{K} \nabla_{\boldsymbol{\theta}} \mathbb{H}(\pi_{\boldsymbol{\theta}}(a_t | \boldsymbol{s}_t)),$$

where $\mathbb{H}(\cdot)$ denotes the entropy term and λ is the annealing coefficient. The entropy term can be expressed explicitly because π is factorized over its dimensions and there are finite actions along each dimension.

Delayed update: As an accurate critic plays an important role for ARSM to estimate gradient, it would be helpful to adopt the delayed update trick of Fujimoto et al. [2018]. In practice, we update the critic network several times before updating the policy network.

In addition to the Python (TensorFlow) code in the Supplementary Material, we also provide detailed pseudo code to help understand the implementation of CARSM in Appendix B.3.

3.4 Experiments

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 a large discrete action space? **(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 be integrated into more sophisticated RL learning frameworks such as TRPO to achieve an improved performance? Since we run trials on some discretized continuous control tasks, another fair question would be: **(e)** Will discretization help learning? If so, what are possible explanations?

We consider benchmark tasks provided by OpenAI Gym classic-control and MuJoCo simulators [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., 2018]; 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 its A2C gradient estimator for $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$. Performance evaluation show that a simple plug-in of CARSM estimator can bring the improvement. Details on experimental settings

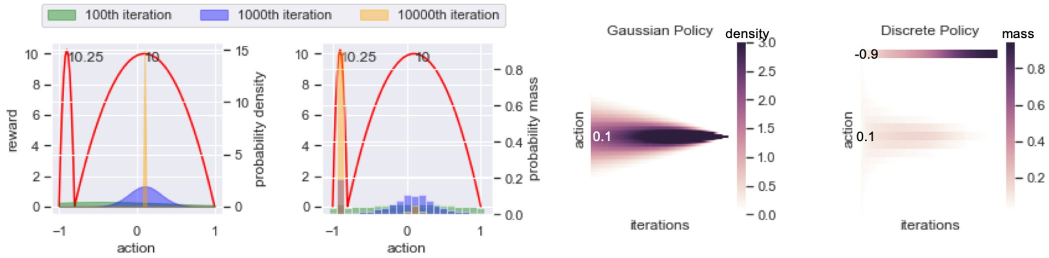


Figure 3.1: **left panel**: Change of policy over iterations in a single random trial between Gaussian policy (left) and discrete policy (right) on a bimodal-reward toy example. **right panel**: Average density on each action along with the training iteration between Gaussian and discrete policies for 100 random trials. Under this setting, the Gaussian policy fails to converge to the global optimum while discrete policy always finds the global optimum.

can be found in Appendix B.2.2.

On our experiments with tasks in continuous control domain, we discretize the continuous action space uniformly to get a discrete action space. More specifically, if the action space is $\mathcal{A} = [-1, 1]^K$, and we discretize it to C actions at each dimension, the action space would become $\tilde{\mathcal{A}} = \{\frac{-C+1}{C-1}, \frac{-C+3}{C-1}, \ldots, \frac{C-1}{C-1}\}^K$.

There are two motivations of discretizing the action space. First, MuJoCo tasks are a set of standard comparable tasks that naturally have multidimensional action spaces, which is the case we are interested in for CARSM. Second, as illustrated in Tang and Agrawal [2019], discrete policy is often more expressive than diagonal-Gaussian policy, leading to better exploration. We will illustrate this point by experiments.

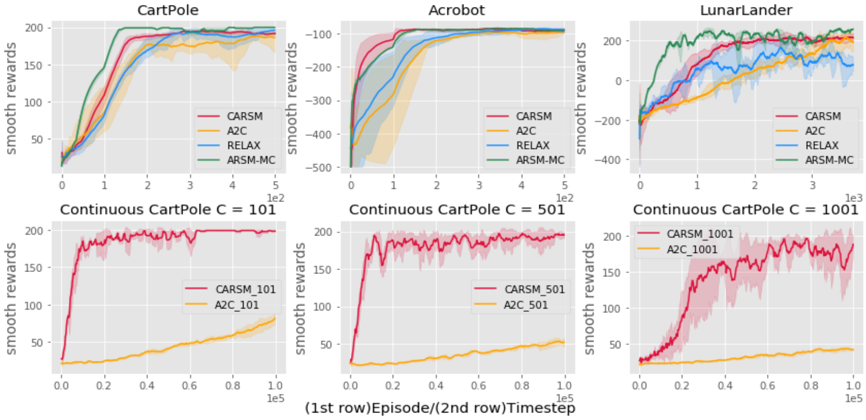


Figure 3.2: **top row**: Performance curves for discrete domains. Comparison between: A2C, RELAX, ARSM-MC, and CARSM. We show the cumulative rewards during training, moving averaged across 100 epochs; the curves show the mean ± std performance across 5 random seeds. **bottom row**: Performance curves on CartPole with very large discrete action space. Comparison between: A2C and CARSM over a range of different discretization scale $C \in \{101, 501, 1001\}$. We show the cumulative rewards during training, moving averaged across 100 epochs; the curves show the mean ± std performance across 5 random seeds.

3.4.1 Motivation and Illustration

One distinction between discrete and Gaussian policies is that a discrete policy can learn multi-modal and skewed distributions while a Gaussian policy can only support uni-modal, symmetric, and bell-shaped distributions. This intrinsic difference could lead to significantly difference on exploration, as reflected by the toy example presented below, which will often lead to different sub-optimal solutions in practice.

To help better understand the connections between multi-modal policy and exploration, we take a brief review of RL objective function from an energy-based distribution point of view. For a bandit problem with reward function $r(a) : \mathcal{A} \rightarrow \mathbb{R}$, we denote the true reward induced distribution as $p(a) \propto e^{r(a)}$. The objective function in (3.1) can be reformulated as

$$\mathbb{E}_{a \sim \pi_\theta(a)}[r(a)] = -\mathbb{KL}(\pi_\theta(a)||p(a)) - \mathbb{H}(\pi_\theta).$$

The KL-divergence term matches the objective function of variational inference (VI) [Blei et al., 2017] in approximating $p(a)$ with distribution $\pi_\theta(a)$, while the second term is the entropy of policy π_θ. Therefore, if we use maximum entropy objective [Haarnoja et al., 2017], which is maximizing $\mathbb{E}_{a \sim \pi_\theta(a)}[r(a)] + \mathbb{H}(\pi_\theta)$, we will get an VI approximate solution. Suppose $p(a)$ is a multi-modal distribution, due to the inherent property of VI [Blei et al., 2017], if π_θ is a Guassian distribution, it will often underestimate the variance of $p(a)$ and capture only one density mode. By contrast, if π_θ is a discrete distribution, it can capture the multi-modal property of $p(a)$, which will lead to more exploration before converging to a more deterministic policy.

We design a simple toy example to reflect these differences. We restrict the action space to $[-1, 1]$, and the true reward function is a concatenation of two quadratic functions (as shown in Figure B.1 left panel red curves) that intersect at a middle point m. We fix the left sub-optimal point as the global optimal one and control the position of m to get tasks with various difficulty levels. More specifically, the closer m to -1, the more explorations needed to converge to the global optimal. We defer the detailed experiment setting to Appendix B.2.1. We run 100 trials of both Gaussian policy and discrete policy and show their behaviors.

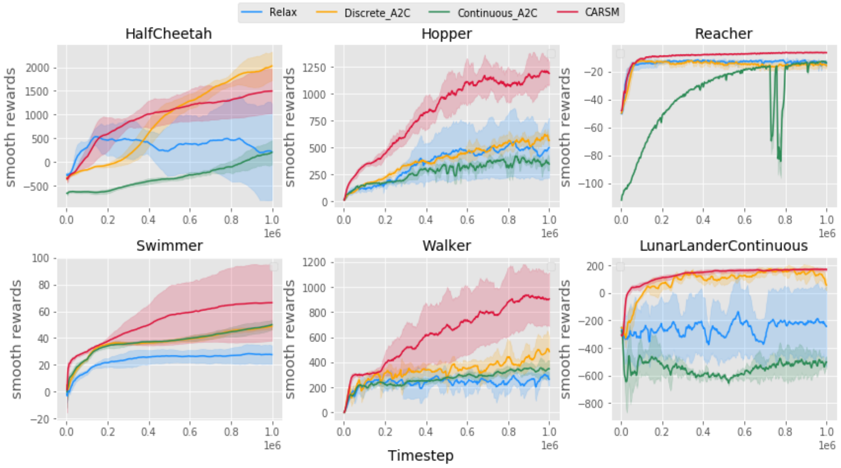


Figure 3.3: Performance curves on six benchmark tasks (all except the last are MuJoCo tasks). Comparison between: continuous A2C (Gaussian policy), discrete A2C, RELAX, and CARSM policy gradient. We show the cumulative rewards during training, moving averaged across 100 epochs; The curves show the mean ± std performance across 5 random seeds.

We show, in Figure B.1 left panel, the learning process of both Gaussian and discrete policies with a quadratic annealing coefficient for the entropy term, and, in right panel, a heatmap where each entry indicates the average density of each action at one iteration. In this case where $m = -0.8$, the signal from the global optimal point has a limited range which requires more explorations during the training process. Gaussian policy can only explore with unimodal distribution and fail to capture the global optimal all the time. By contrast, discrete policy can learn the bi-modal distribution in the early stage, gradually concentrate on both the optimal and sub-optimal peaks before collecting enough samples, and eventually converges to the optimal peak. More explanations can be found in Appendix B.2.1.

3.4.2 Comparing CARSM and ARSM-MC

One major difference between CARSM and ARSM-MC is the usage of Q-Critic. It saves us from running MC rollouts to estimate the action-value functions of all unique pseudo actions, the number of which can be enormous under a multidimensional setting. This saving is at the expense of introducing bias to gradient estimation (not by the gradient estimator per se but by how Q is estimated). Similar to the argument between MC and TD, there is a trade-off between bias and variance. In this set of experiments, we show that the use of Critic in CARSM not only brings us accelerated training, but also helps return good performance.

To make the results of CARSM directly comparable with those of ARSM-MC shown in Yin et al. [2019], we evaluate the performances on an *Episode* basis on discrete classical-control tasks: CartPole, Acrobot, and LunarLander. We follow Yin et al. [2019] to limit the MC rollout sizes for ARSM-MC as 16, 64, and 1024, respectively. From Figure 3.2 top row, ARSM-MC has a better performance than CARSM on both CartPole and LunarLander, while CARSM outperforms the rest on Acrobot. The results are promising in the sense that CARSM only uses one rollout for estimation while ARSM-MC uses up to 16, 64, and 1024, respectively, so CARSM largely improves the sample efficiency of ARSM-MC while maintaining comparable performance. The action space is uni-dimensional with 2, 3, and 4 discrete actions for CartPole, Acrobot, and LunarLander, respectively. We also compare CARSM and ARSM-MC given fixed number of timesteps. Under this setting, CARSM outperforms ARSM-

MC by a large margin on both Acrobot and LunarLander. See Figure B.2 in Appendix B.2.3 for more details.

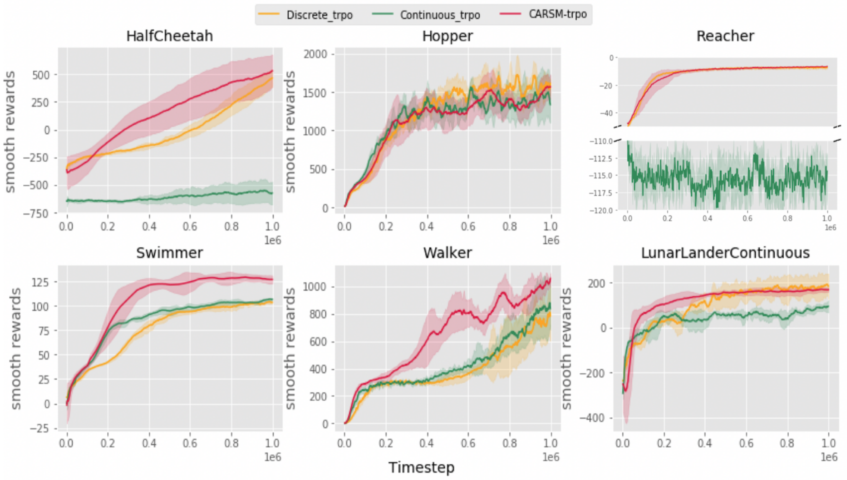
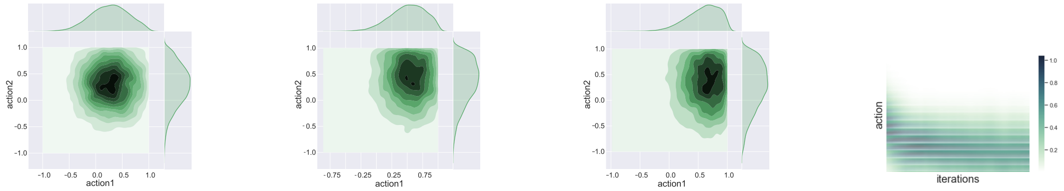


Figure 3.4: Performance curves on six benchmark tasks (all except the last are MuJoCo tasks). Comparison between: continuous TRPO (Gaussian policy), discrete TRPO, and CARSM policy gradient combined with TRPO. We show the cumulative rewards during training, moving averaged across 100 epochs; the curves show the mean ± std performance across 5 random seeds.

3.4.3 Large Discrete Action Space

We want to show that CARSM has better sample efficiency on cases where the number of action C in one dimension is large. We test CARSM along with A2C on a continuous CartPole task, which is a modified version of discrete CartPole. In this continuous environment, we restrict the action space to $[-1, 1]$. Here the action indicates the force applied to the Cart at any time.

The intuition of why CARSM is expected to perform well under a large action space setting is because of the low-variance property. When C is large, the distribution is more dispersed on each action compared with smaller case,

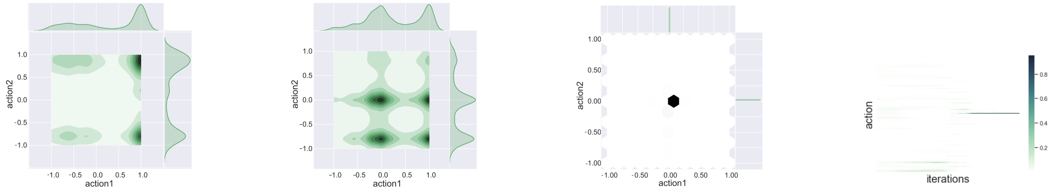


(a) Gaussian policy at an early stage.

(b) Gaussian policy at an intermediate stage.

(c) Gaussian policy at a late stage.

(d) Average density for Gaussian policy.



(e) Discrete policy at an early stage.

(f) Discrete policy at an intermediate stage.

(g) Discrete policy at a late stage.

(h) Average density for Discrete policy.

Figure 3.5: Policy distribution on the Reacher task between discrete policy and Gaussian policy for a given state (discrete action space has 11 actions on each dimension).

which requires the algorithm captures the signal from best action accurately to improve exploitation. In this case, a high-variance gradient estimator will surpass the right signal, leading to a long exploration period or even divergence.

As shown in Figure 3.2 bottom row, CARSM outperforms A2C by a large margin in all three large C settings. Though the CARSM curve exhibits larger variations as C increases, it always learns much more rapidly at an early stage compared with A2C. Note the naive ARSM-MC algorithm will not work on this setting simply because it needs to run as many as tens of thousands MC rollouts to get a single gradient estimate.

3.4.4 OpenAI Gym Benchmark Tasks

In this set of experiments, we compare CARSM with A2C and RELAX, which all share the same underlying idea of improving the sample efficiency by reducing the variance of gradient estimation. For A2C, we compare with both Gaussian and discrete policies to check the intuition presented in Section 3.4.1. In all these tasks, following the results from Tang and Agrawal [2019], the action space is equally divided into $C = 11$ discrete actions at each dimension. Thus the discrete action space size becomes 11^K, where K is the action-space dimension that is 6 for HalfCheetah, 3 Hopper, 2 Reacher, 2 Swimmer, 6 Walker2D, and 2 LunarLander. More details on Appendix B.2.2.

As shown in Figure 3.3, CARSM outperforms the other algorithms by a large margin except on HalfCheetah, demonstrating the high-sample efficiency of CARSM. Moreover, the distinct behaviors of Gaussian and discrete policies in the Reacher task, as shown in both Figures 3.3 and 3.4, are worth thinking, motivating us to go deeper on this task to search for possible explanations. We manually select a state that requires exploration on the early stage, and record the policy evolvement along with training process at that specific state. We show those transition phases in Figure 3.5 for both Gaussian policy (top row) and discrete policy (bottom row).

For discrete policy, plots (e)-(g) in Figure 3.5 bottom row show interesting property: at the early stage, the policy does not put heavy mass at all on the final sub-optimal point $(0, 0)$, but explores around multiple density modes; then it gradually concentrates on several sup-optimal points on an intermediate phase, and converges to the final sub-optimal point. Plot (h) also conveys the same message that during the training process, discrete action can transit explorations around several density modes since the green lines can jump along the iterations. (The heatmaps of (d) and (h) in Figure 3.5 are computed in the same way as that in Figure B.1, and details can be found in Appendix B.2.1.)

By contrast, Gaussian policy does not have the flexibility of exploring based on different density modes, therefore from plots (a)-(c) on the top row of Figure 3.5, the policy moves with a large radius but one center, and on (d), the green lines move consecutively which indicates a smooth but potentially not comprehensive exploration.

3.4.5 Combining CARSM with TRPO

Below we show that CARSM can be readily applied under TRPO to improves its performance. In the update step of TRPO shown in (3.4), the default estimator for $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ is A2C or its variant. We replace it with CARSM estimator and run it on the same set of tasks. As shown in Figure 3.4, Gaussian policy fails to find a good sub-optimal solution under TRPO for both HalfCheetah and Reacher and performs similarly to its discrete counterpart on the other tasks. Meanwhile, CARSM improves the performance of TRPO over discrete policy setting on three tasks and maintains similar performance on the others, which shows evidence that CARSM is an easy plug-in estimator for $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ and hence can potentially improve other algorithms, such as some off-policy ones [Wang et al., 2016, Degris et al., 2012], that need this gradient estimation.

3.5 Conclusion

To solve RL tasks with multidimensional discrete action setting efficiently, we propose Critic-ARSM policy gradient, which is a combination of multidimensional sparse ARSM gradient estimator and an action-value critic, to improve sample efficiency for on-policy algorithm. We show the good performances of this algorithm from perspectives including stability on very large action space cases and comparisons with other standard benchmark algorithms, and show its potential to be combined with other standard algorithms. Moreover, we demonstrate the potential benefits of discretizing continuous control tasks to obtain a better exploration based on multimodal property.

Chapter 4

Implicit Distributional Reinforcement Learning

To improve the sample efficiency of policy-gradient based reinforcement learning algorithms, we propose implicit distributional actor critic (IDAC) that consists of a distributional critic, built on two deep generator networks (DGNs), and a semi-implicit actor (SIA), powered by a flexible policy distribution. We adopt a distributional perspective on the discounted cumulative return and model it with a state-action-dependent implicit distribution, which is approximated by the DGNs that take state-action pairs and random noises as their input. Moreover, we use the SIA to provide a semi-implicit policy distribution, which mixes the policy parameters with a reparameterizable distribution that is not constrained by an analytic density function. In this way, the policy's marginal distribution is implicit, providing the potential to model complex properties such as covariance structure and skewness, but its parameter and entropy can still be estimated. We incorporate these features with an off-policy algorithm framework to solve problems with continuous action space,

---

The content in this chapter was published in Yue et al. [2020b]; I designed the SIA and the twin delayed network parts, and brainstormed with the other coauthors to come up with the DGN part. I was also in charge of the empirical evaluation section.

and compare IDAC with the state-of-art algorithms on representative OpenAI Gym environments. We observe that IDAC outperforms these baselines for most tasks.

4.1 Introduction

There has been significant recent interest in using model-free reinforcement learning (RL) to address complex real-world sequential decision making tasks [MacAlpine and Stone, 2017, Silver et al., 2018, Pachocki et al., 2018]. With the help of deep neural networks, model-free deep RL algorithms have been successfully implemented in a variety of tasks, including game playing [Silver et al., 2016, Mnih et al., 2013] and robotic control [Levine et al., 2016]. Deep Q-network (DQN) [Mnih et al., 2015] enables RL agent with human level performance on Atari games [Bellemare et al., 2013], motivating many follow-up works with further improvements [Wang et al., 2016, Andrychowicz et al., 2017]. A novel idea, proposed by Bellemare et al. [2017a], is to take a distributional perspective for deep RL problems, which models the full distribution of the discounted cumulative return of a chosen action at a state rather than just the expectation of it, so that the model can capture its intrinsic randomness instead of just first-order moment. Specifically, the distributional Bellman operator can help capture skewness and multimodality in state-action value distributions, which could lead to a more stable learning process, and approximating the full distribution may also mitigate the challenges of learning from a non-stationary policy. Under this distributional framework, Bellemare et al. [2017a] propose the C51 algorithm that outperforms previous state-of-art classical Q-learning based algorithms on a range of Atari games. However, some discrepancies exist between the theory and implementation in C51, motivating Dabney et al. [2018b] to introduce QR-DQN that borrows Wasserstein distance and quantile regression related techniques to diminish the theory-practice gap. Later on, the distributional view is also incorporated into the framework of deep deterministic policy gradient (DDPG) [Lillicrap et al., 2015] for continuous control tasks, yielding efficient algorithms such as distributed distributional DDPG (D4PG) [Barth-Maron et al., 2018] and sample-based distributional policy gradient (SDPG) [Singh et al., 2020]. Due to the deterministic nature of the policy, these algorithms always manually add random noises to actions during the training process to avoid getting stuck in poor local optimums. By contrast, stochastic policy takes that randomness as part of the policy, learns it during the training, and achieves state-of-art performance, with soft actor critic (SAC) [Haarnoja et al., 2018a] being a successful case in point.

Motivated by the promising directions from distributional action-value learning and stochastic policy, this paper integrates these two frameworks in hopes of letting them strengthen each other. We model the distribution of the discounted cumulative return of an action at a state with a deep generator network (DGN), whose input consists of a state-action pair and random noise, and applies the distributional Bellman equation to update its parameters. The DGN plays the role of a distributional critic, whose output conditioning on a state-action pair follows an implicit distribution. Intuitively, only modeling the expectation of the cumulative return is inevitably discarding useful information readily available during the training, and modeling the full distribution of it could capture more useful information to help better train and stabilize a stochastic policy. In other words, there are considerable potential gains in guiding the training of a distribution with a distribution rather than its expectation.

For stochastic policy, the default distribution choice under continuous control is diagonal Gaussian. However, having a unimodal and symmetric density at each dimension and assumping independence between different dimensions make it incapable of capturing complex distributional properties, such as skewness, kurtosis, multimodality, and covariance structure. To fully take advantage of the distributional return modeled by the DGN, we thereby propose a semi-implicit actor (SIA) as the policy distribution, which adopts a semi-implicit hierarchical construction [Yin and Zhou, 2018b] that can be made as complex as needed while remaining amenable to optimization via stochastic gradient descent (SGD).

We have now defined an implicit distributional critic, DGN, and a semi-implicit actor, SIA. A naive combination of them within an actor-critic policy gradient framework, however, only delivers mediocre performance, falling short of the promise it holds. We attribute its underachievement to the overestimation issue, commonly existing in classical value-based algorithms [Van Hasselt et al., 2016], that does not automatically go away under the distributional setting. Inspired by previous work in mitigating the over estimation issue in deep

Q-learning [Fujimoto et al., 2018], we come up with a twin delayed DGNs based critic, with which we provide a novel solution that takes the target values as the element-wise minimums of the sorted output values of these two DGNs, stabilizing the training process and boosting performance.

Contributions: The main contributions of this paper include: 1) we incorporate the distributional idea with the stochastic policy setting, and characterize the return distribution with the help of a DGN under a continuous control setup; 2) we introduce the twin delayed structure on DGNs to mitigate the overestimation issue; and 3) we improve the flexibility of the policy by using a SIA instead of a Gaussian or mixture of Gaussian distribution to improve exploration.

Related work: Since the successful implementation of RL problems from a distributional perspective on Atari 2600 games [Bellemare et al., 2017a], there is a number of follow-ups trying to boost existing deep RL algorithms by directly characterizing the distribution of the random return instead of the expectation [Dabney et al., 2018a,b, Barth-Maron et al., 2018, Singh et al., 2020]. On the value-based side, C51 [Bellemare et al., 2017a] represents the return distribution with a categorical distribution defined by attaching $C = 51$ variable parameterized probabilities at $C = 51$ fixed locations. QR-DQN [Dabney et al., 2018b], on the other hand, does so by attaching N variable parameterized locations at N equally-spaced fixed quantiles, and employs a quantile regression loss for optimization. IQN [Dabney et al., 2018a] further extends this idea by learning a full quantile function. On the policy-gradient-based side, D4PG [Barth-Maron et al., 2018] incorporates the distributional perspective into DDPG [Lillicrap et al., 2015], with the return distribution modeled similarly as in C51 [Bellemare et al., 2017a]. On top of that, SDPG [Singh et al., 2020] is proposed to model the quantile function with a generator to overcome the limitation of using variable probabilities at fixed locations, and the same as D4PG, it models the policy as a deterministic transformation of the state representation.

There is rich literature aiming to obtain a high-expressive policy to encourage exploration during the training. When a deterministic policy is applied, a random perturb is always added when choosing a continuous action [Silver et al., 2014, Lillicrap et al., 2015]. In Haarnoja et al. [2017], the policy is modeled proportional to its action-value function to guarantee flexibility. In Haarnoja et al. [2018a], SAC is proposed to mitigate the policy's expressiveness issue while retaining tractable optimization; with the policy modeled with either a Gaussian or a mixture of Gaussian, SAC adopts a maximum entropy RL objective function to encourage exploration. The normalizing flow [Rezende and Mohamed, 2015, Dinh et al., 2016] based techniques have been recently applied to design a flexible policy in both on-policy [Tang and Agrawal, 2018] and off-policy settings [Ward et al., 2019].

4.2 Implicit distributional actor critic

We present implicit distributional actor critic (IDAC) as a policy gradient based actor-critic algorithm under the off-policy learning setting, with a semi-implicit actor (SIA) and two deep generator networks (DGNs) as critics. We will start off with the introduction of distributional RL and DGN.

4.2.1 Implicit distributional RL with deep generator network (DGN)

We model the agent-environment interaction by a Markov decision process (MDP) denoted by $(\mathcal{S}, \mathcal{A}, R, P)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ the action space, R a random reward function, and P the dynamic of environment describing $P(\boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{a})$, where $\boldsymbol{a} \in \mathcal{A}$ and $\boldsymbol{s}, \boldsymbol{s}' \in \mathcal{S}$. A policy is defined as a map from the state space to action space $\pi(\cdot \mid \boldsymbol{s}) : \mathcal{S} \to \mathcal{A}$. Let us denote the discounted cumulative return from state-action pair $(\boldsymbol{s}, \boldsymbol{a})$ following policy π as $Z^\pi(\boldsymbol{s}, \boldsymbol{a}) = \sum_{t=0}^{\infty} \gamma^t R(\boldsymbol{s}_t, \boldsymbol{a}_t)$, where γ is the discount factor, $\boldsymbol{s}_0 := \boldsymbol{s}$, and $\boldsymbol{a}_0 := \boldsymbol{a}$. Under a classic RL setting, an action value function Q is used to represent the expected return as $Q^\pi(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}[Z^\pi(\boldsymbol{s}, \boldsymbol{a})]$, where the expectation takes over all sources of intrinsic randomness [Goldstein et al., 1981]. While under the distributional setup, it is the random return $Z^\pi(\boldsymbol{s}, \boldsymbol{a})$ itself rather than its expectation that is being directly modeled. Similar to the classical Bellman equation, we have the *distributional Bellman equation* [Dabney et al., 2018b] as

$$Z^\pi(\boldsymbol{s}, \boldsymbol{a}) \overset{D}{=} R(\boldsymbol{s}, \boldsymbol{a}) + \gamma Z^\pi(\boldsymbol{s}', \boldsymbol{a}'). \tag{4.1}$$

where $\overset{D}{=}$ denotes "equal in distribution" and $\boldsymbol{a}' \sim \pi(\cdot \mid \boldsymbol{s}'),\ \boldsymbol{s}' \sim P(\cdot \mid \boldsymbol{s}, \boldsymbol{a})$.

We propose using a DGN to model the distribution of random return Z^π as

$$Z^\pi(\boldsymbol{s}, \boldsymbol{a}) \overset{D}{\approx} G_{\boldsymbol{\omega}}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\epsilon}),\ \boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon}), \tag{4.2}$$

where $\overset{D}{\approx}$ denotes "approximately equal in distribution," $p(\boldsymbol{\epsilon})$ is a random noise distribution, and $G_{\boldsymbol{\omega}}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\epsilon})$ is a neural network based deterministic function parameterized by $\boldsymbol{\omega}$, whose input consists of $\boldsymbol{s}$, $\boldsymbol{a}$, and $\boldsymbol{\epsilon}$. We can consider $G_{\boldsymbol{\omega}}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\epsilon})$ as a generator that transforms $p(\boldsymbol{\epsilon})$ into an implicit distribution, from which random samples can be straightforwardly generated but the probability density function is in general not analytic (*e.g.*, when $G_{\boldsymbol{\omega}}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\epsilon})$ is not invertible with respect to $\boldsymbol{\epsilon}$). If the distributional equality holds in (4.2), we can approximate the distribution of $Z^{\pi}(\boldsymbol{s}, \boldsymbol{a})$ in a sample-based manner, which can be empirically represented by K independent, and identically distributed (*iid*) random samples as $\{G_{\boldsymbol{\omega}}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\epsilon}_1), \cdots, G_{\boldsymbol{\omega}}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\epsilon}_K)\}$, where $\boldsymbol{\epsilon}_1, \ldots, \boldsymbol{\epsilon}_K \overset{iid}{\sim} p(\boldsymbol{\epsilon})$.

4.2.2 Learning of DGN

Based on (4.1), we desire the DGN to also satisfy the distributional matching that

$$G_{\boldsymbol{\omega}}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\epsilon}) \overset{D}{=} R(\boldsymbol{s}, \boldsymbol{a}) + \gamma G_{\boldsymbol{\omega}}(\boldsymbol{s}', \boldsymbol{a}', \boldsymbol{\epsilon}'), \quad \text{where} \;\; \boldsymbol{\epsilon}, \boldsymbol{\epsilon}' \overset{iid}{\sim} p(\boldsymbol{\epsilon}). \tag{4.3}$$

This requires us to adopt a differential metric to measure the distance between two distributions and use it to guide the learning of the generator parameter $\boldsymbol{\omega}$. While there exist powerful methods to learn high-dimensional data generators, such as generative adverserial nets [Goodfellow et al., 2014, Arjovsky et al., 2017], there is no such need here since there exist simple and stable solutions to estimate the distance between two one-dimensional distributions given *iid* random samples from them.

In particular, the p-Wasserstein distance [Villani, 2008] between the distributions of univariate random variables $X, Y \in \mathbb{R}$ can be approximated by that between their empirical distributions supported on K random samples, which can be expressed as $\hat{X} = \frac{1}{K}\sum_{k=1}^{K}\delta_{x_k}$ and $\hat{Y} = \frac{1}{K}\sum_{k=1}^{K}\delta_{y_k}$, and we have

$$W_p(X,Y)^p \approx W_p(\hat{X},\hat{Y})^p = \tfrac{1}{K}\textstyle\sum_{k=1}^{K} ||\overrightarrow{x}_k - \overrightarrow{y}_k||^p, \tag{4.4}$$

where $\overrightarrow{x}_{1:K}$ and $\overrightarrow{y}_{1:K}$ are obtained by sorting $x_{1:K}$ and $y_{1:K}$ in increasing order, respectively [Villani, 2008, Bernton et al., 2019, Deshpande et al., 2018, Kolouri et al., 2019]. Though seems tempting to use $W_p(\hat{X},\hat{Y})^p$ as the loss function, it has been shown [Bellemare et al., 2017b, Dabney et al., 2018b] that such a loss function may not be theoretically sound when optimized with SGD, motivating the use of a quantile regression loss based on $\hat{X}$ and $\hat{Y}$. In the same sprite as Dabney et al. [2018b], we propose to measure the distributional distance with a quantile regression Huber loss [Huber, 1992] based on empirical samples, defined as

$$\mathcal{L}_{\mathrm{QR}}(X,Y) \approx \mathcal{L}_{\mathrm{QR}}(\hat{X},\hat{Y}) = \tfrac{1}{K^2}\textstyle\sum_{k=1}^{K}\sum_{k'=1}^{K}\rho_{\tau_k}^{\kappa}(y_{k'} - \overrightarrow{x}_k), \tag{4.5}$$

where $\overrightarrow{x}_k$ that are arranged in increasing order are one-to-one mapped to K equally-spaced increasing quantiles $\tau_k = (k-0.5)/K$, κ is a pre-fixed threshold (set as $\kappa = 1$ unless specified otherwise), and

$$\rho_{\tau_k}^{\kappa}(u) = |\tau_k - \mathbf{1}_{[u<0]}|\mathcal{L}_\kappa(u)/\kappa, \quad \mathcal{L}_\kappa(u) = \begin{cases} \frac{1}{2}u^2, & \text{if } |u| \le \kappa \\ \kappa(|u| - \frac{1}{2}\kappa), & \text{otherwise} \end{cases}. \tag{4.6}$$

Note that the reason we map $\overrightarrow{x}_k$ to quantile $\tau_k = (k-0.5)/K$, for $k = 1, \dots, K$, is because $P(X \le \overrightarrow{x}_k) \approx \tau_k$, an approximation that becomes increasingly more accurate as K increases.

Recall the distributional matching objective in (4.3). To train the DGN, we first obtain an empirical distribution $\hat{X}$ of the generator supported on K *iid* random samples as

$$x_{1:K} := \{G_{\boldsymbol{\omega}}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\epsilon}^{(k)})\}_{1:K}, \quad \text{where } \boldsymbol{\epsilon}^{(1)}, \ldots, \boldsymbol{\epsilon}^{(K)} \overset{iid}{\sim} p(\boldsymbol{\epsilon}), \tag{4.7}$$

and similarly an empirical target distribution $\hat{Y}$ supported on

$$y_{1:K} := \{R(\boldsymbol{s}, \boldsymbol{a}) + \gamma G_{\tilde{\boldsymbol{\omega}}}(\boldsymbol{s}', \boldsymbol{a}', \boldsymbol{\epsilon}'^{(k)})\}_{1:K}, \quad \text{where } \boldsymbol{\epsilon}'^{(1)}, \ldots, \boldsymbol{\epsilon}'^{(K)} \overset{iid}{\sim} p(\boldsymbol{\epsilon}), \tag{4.8}$$

where $\boldsymbol{a}' \sim \pi(\cdot \,|\, \boldsymbol{s}')$, $\boldsymbol{s}' \sim P(\cdot \,|\, \boldsymbol{s}, \boldsymbol{a})$, and $\tilde{\boldsymbol{\omega}}$ is the delayed generator parameter, a common practice to stabilize the learning process as used in Lillicrap et al. [2015] and Fujimoto et al. [2018]. Since we use empirical samples to represent the distributions, we first sort $x_{1:K}$ in increasing order, denoted as

$$(\overrightarrow{x}_1, \cdots, \overrightarrow{x}_K) = \text{sort}(x_1, \cdots, x_K),$$

and then map them to increasing quantiles $((k - 0.5)/K)_{1:K}$. The next step is to minimize the quantile regression Huber loss as in (4.5), and the objective function for DGN parameter $\boldsymbol{\omega}$ becomes

$$J(\boldsymbol{\omega}) = \mathcal{L}_{\text{QR}}(\hat{X}, \text{StopGradient}\{\hat{Y}\}) = \tfrac{1}{K^2} \textstyle\sum_{k=1}^{K} \sum_{k'=1}^{K} \rho_{\tau_k}^{\kappa}(\text{StopGradient}\{y_{k'}\} - \overrightarrow{x}_k). \tag{4.9}$$

4.2.3 Twin delayed DGNs

Motivated by the significant improvement shown in Fujimoto et al. [2018], we propose the use of twin DGNs to prevent overestimation of the return distribution. However, it cannot be applied directly. On value-based algorithm, one can directly take the minimum of two estimated Q-values; on the other hand, we have empirical samples from a distribution and we try to avoid overestimation on that distribution which need to be taken care of. Specifically, we design two DGNs $G_{\boldsymbol{\omega}_1}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\epsilon})$ and $G_{\boldsymbol{\omega}_2}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\epsilon})$ with independent initialization of $\boldsymbol{\omega}_1$ and $\boldsymbol{\omega}_2$ and independent input noise. Therefore, we will have two sets of target values as defined in (4.8), which are denoted as $y_{1,1:K}$ and $y_{2,1:K}$, respectively. Since they represent empirical distributions now and each element of them is assigned to one specific quantile, we will need to sort them before taking element-wise minimum so that the distribution is not distorted before mitigating the overestimation issue. In detail, with

$$(\overrightarrow{y}_{1,1}, \cdots, \overrightarrow{y}_{1,K}) = \mathrm{sort}(y_{1,1}, \cdots, y_{1,K}), \quad (\overrightarrow{y}_{2,1}, \cdots, \overrightarrow{y}_{2,K}) = \mathrm{sort}(y_{2,1}, \cdots, y_{2,K}),$$

the new target values for twin DGNs become

$$(\overrightarrow{y}_{1}, \cdots, \overrightarrow{y}_{K}) = (\min(\overrightarrow{y}_{1,1}, \overrightarrow{y}_{2,1}), \cdots, \min(\overrightarrow{y}_{1,K}, \overrightarrow{y}_{2,K}))\,,$$

and with $\boldsymbol{\epsilon}^{(1)}, \ldots, \boldsymbol{\epsilon}^{(K)} \overset{iid}{\sim} p(\boldsymbol{\epsilon})$, the objective function for parameter $\boldsymbol{\omega}_1$ of twin DGNs becomes

$$J(\boldsymbol{\omega}_1) = \tfrac{1}{K^2} \textstyle\sum_{k=1}^{K} \sum_{k'=1}^{K} \rho^{\kappa}_{\tau_k}(\mathrm{StopGradient}\{\overrightarrow{y}_{k'}\} - \overrightarrow{x}_k), \;\; x_{1:K} := \{G_{\boldsymbol{\omega}_1}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\epsilon}^{(k)})\}_{1:K}. \tag{4.10}$$

The objective function for parameter $\boldsymbol{\omega}_2$ is similarly defined under the same set of target values.

4.2.4 Semi-implicit actor (SIA)

Since the return distribution is modeled in a continuous action space, it will be challenging to choose the action that maximizes the critic. We instead turn to finding a flexible stochastic policy that captures the energy landscape of $\mathbb{E}_{\boldsymbol{\epsilon}\sim p(\boldsymbol{\epsilon})}[G_{\boldsymbol{\omega}}(\boldsymbol{s},\boldsymbol{a},\boldsymbol{\epsilon})]$. The default parametric policy for continuous control problems is modeled as a diagonal Gaussian distribution, where the means and variances of all dimensions are obtained from some deterministic transformations of state $\boldsymbol{s}$. Due to the nature of the diagonal Gaussian distribution, it can not capture the dependencies between different action dimensions and has a unimodal and symmetric assumption on its density function at each dimension, limiting its ability to encourage exploration. For example, it may easily get stuck in a bad local mode simply because of its inability to accomodate multi-modality [Yue et al., 2020a].

To this end, we consider a semi-implicit construction [Yin and Zhou, 2018b] that enriches the diagonal Gaussian distribution by randomizing its parameters with another distribution, making the marginal of the semi-implicit hierarchy, which in general has no analytic density function, become capable of modeling much more complex distributional properties, such as skewness, multi-modality, and dependencies between different action dimensions. In addition, its parameters are amenable to SGD based optimization, making it even more attractive as a plug-in replacement of diagonal Gaussian. Specifically, we construct a semi-implicit policy with a hierarchical structure as

$$\pi_{\boldsymbol{\theta}}(\boldsymbol{a} \,|\, \boldsymbol{s}) = \int_{\boldsymbol{\xi}} \pi_{\boldsymbol{\theta}}(\boldsymbol{a} \,|\, \boldsymbol{s}, \boldsymbol{\xi}) p(\boldsymbol{\xi}) d\boldsymbol{\xi}, \text{ where } \pi_{\boldsymbol{\theta}}(\boldsymbol{a} \,|\, \boldsymbol{s}, \boldsymbol{\xi}) = \mathcal{N}(\boldsymbol{a}; \boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{s}, \boldsymbol{\xi}), \text{diag}\{\boldsymbol{\sigma}^2_{\boldsymbol{\theta}}(\boldsymbol{s}, \boldsymbol{\xi})\}), \tag{4.11}$$

where $\boldsymbol{\theta}$ denotes the policy parameter and $\boldsymbol{\xi} \sim p(\boldsymbol{\xi})$ denotes a random noise, which concatenated with state $\boldsymbol{s}$ is transformed by a deep neural network parameterized by $\boldsymbol{\theta}$ to define both the mean and covariance of a diagonal Gaussian policy distribution. Note while we choose $\pi_{\boldsymbol{\theta}}(\boldsymbol{a} \,|\, \boldsymbol{s}, \boldsymbol{\xi})$ to be diagonal Gaussian, it can take any explicit reparameterizable distribution. There is no constraint on $p(\boldsymbol{\xi})$ as long as it is simple to sample from, and is reparameterizable if it contains parameters to learn. This semi-implicit construction balances the tractability and expressiveness of $\pi_{\boldsymbol{\theta}}(\boldsymbol{a} \,|\, \boldsymbol{s})$, where we can get a powerful implicit policy while still be capable of sampling from it and estimating its entropy. Based on previous proofs [Yin and Zhou, 2018b, Molchanov et al., 2019], we present the following Lemma for entropy estimation and defer its proof to the Appendix. The ability of entropy estimation is crucial when solving problems under the maximum entropy RL framework [Todorov, 2007, Ziebart, 2010, Ziebart et al., 2008], which we adopt below to encourage exploration.

Lemma 6. *Assume* $\pi_{\boldsymbol{\theta}}(\boldsymbol{a} \,|\, \boldsymbol{s})$ *is constructed as in Eq.* (4.11)*, the following expectation*

$$\mathcal{H}_L := \mathbb{E}_{\boldsymbol{\xi}^{(0)}, \ldots, \boldsymbol{\xi}^{(L)} \overset{iid}{\sim} p(\boldsymbol{\xi})} \mathbb{E}_{\boldsymbol{a} \sim \pi_{\boldsymbol{\theta}}(\boldsymbol{a} \,|\, \boldsymbol{s}, \boldsymbol{\xi}^{(0)})} [\log \tfrac{1}{L+1} \textstyle\sum_{\ell=0}^{L} \pi_{\boldsymbol{\theta}}(\boldsymbol{a} \,|\, \boldsymbol{s}, \boldsymbol{\xi}^{(\ell)})] \tag{4.12}$$

is an asymptotically tight upper bound of the negative entropy, expressed as

$$\mathcal{H}_\ell \geq \mathcal{H}_{\ell+1} \geq \mathcal{H} := \mathbb{E}_{\boldsymbol{a} \sim \pi_{\boldsymbol{\theta}}(\boldsymbol{a} \,|\, \boldsymbol{s})} [\log \pi_{\boldsymbol{\theta}}(\boldsymbol{a} \,|\, \boldsymbol{s})], \quad \forall \ell \geq 0.$$

4.2.5 Learning of SIA

In IDAC, the action value function can be expressed as $\mathbb{E}_{\boldsymbol{\epsilon}}[G_{\boldsymbol{\omega}}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\epsilon})]$. Related to SAC [Haarnoja et al., 2018a], we learn the policy towards the Boltzman distribution of the action-value function by minimizing a Kullback–Leibler (KL) divergence between them as

$$\pi_{\text{new}} = \operatorname{argmin}_{\pi' \in \boldsymbol{\Pi}} \mathbb{E}_{\boldsymbol{s} \sim \rho(\boldsymbol{s})} \left[\mathrm{KL}\left(\pi_{\boldsymbol{\theta}}(\boldsymbol{a} \,|\, \boldsymbol{s}) \middle\| \frac{\exp(\mathbb{E}_{\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})}[G(\boldsymbol{s},\boldsymbol{a},\boldsymbol{\epsilon})/\alpha])}{\int \exp(\mathbb{E}_{\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})}[G(\boldsymbol{s},\boldsymbol{a},\boldsymbol{\epsilon})/\alpha]) d\boldsymbol{a}} \right) \right], \tag{4.13}$$

where $\rho(\boldsymbol{s})$ denotes the state-visitation frequency, $\alpha > 0$ is a reweard scaling coefficient, and $\boldsymbol{\Pi}$ is the semi-implicit distribution family. Therefore, the loss function for policy parameters is

$$J(\boldsymbol{\theta}) = -\mathbb{E}_{\boldsymbol{s} \sim \rho(\boldsymbol{s})} \mathbb{E}_{\boldsymbol{a} \sim \pi_{\boldsymbol{\theta}}(\cdot \,|\, \boldsymbol{s})} \{ \mathbb{E}_{\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})}[G_{\boldsymbol{\omega}}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\epsilon})] - \alpha \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a} \,|\, \boldsymbol{s}) \}. \tag{4.14}$$

We cannot optimize (4.14) directly since the semi-implicit policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{a} \,|\, \boldsymbol{s})$ does not have an analytic density function and its entropy is not analytic. With the help of Lemma 6, we turn to minimizing an asymptotic upper bound of (4.14) as

$$\begin{aligned} &J(\boldsymbol{\theta}) \leq -\mathbb{E}_{\boldsymbol{s} \sim \rho(\boldsymbol{s})} \mathbb{E}_{\boldsymbol{\xi}^{(1)}, \ldots, \boldsymbol{\xi}^{(L)} \overset{iid}{\sim} p(\boldsymbol{\xi})} \tfrac{1}{J} \textstyle\sum_{j=1}^{J} \mathbb{E}_{\boldsymbol{\xi}_j^{(0)} \sim p(\boldsymbol{\xi})} \mathbb{E}_{\boldsymbol{a}^{(j)} \sim \pi_{\boldsymbol{\theta}}\left(\cdot \,|\, \boldsymbol{s}, \boldsymbol{\xi}_j^{(0)}\right)} \mathbb{E}_{\boldsymbol{\epsilon}^{(j)} \sim p(\boldsymbol{\epsilon})} [\overline{J}_j(\boldsymbol{\theta})] \\ &\overline{J}_j(\boldsymbol{\theta}) := \left(\tfrac{1}{2} \textstyle\sum_{i=1}^{2} G_{\boldsymbol{\omega}_i}(\boldsymbol{s}, \boldsymbol{a}^{(j)}, \boldsymbol{\epsilon}^{(j)}) \right) - \alpha \log \left(\frac{\pi_{\boldsymbol{\theta}}(\boldsymbol{a}^{(j)} \,|\, \boldsymbol{s}, \boldsymbol{\xi}_j^{(0)}) + \sum_{\ell=1}^{L} \pi_{\boldsymbol{\theta}}(\boldsymbol{a}^{(j)} \,|\, \boldsymbol{s}, \boldsymbol{\xi}^{(\ell)})}{L+1} \right), \end{aligned} \tag{4.15}$$

where $\boldsymbol{\omega}_1, \boldsymbol{\omega}_2$ are the parameters of twin DGNs, $\overline{J}_j(\theta)$ is a Monte Carlo estimate of this asymptotic upper bound given a single action, and J is the number of actions that we will use to estimate the objective function. Note we could set

$J = 1$, but then we will still need to sample multiple *iid* $\boldsymbol{\epsilon}$'s to estimate the action-value function. An alternative choice is to sample $J > 1$ actions and sample multiple $\boldsymbol{\epsilon}$'s for each action, which, given the same amount of computational budget, is in general found to be less efficient than simply increasing the number of actions in (4.15). To estimate the gradient, each $\boldsymbol{a}^{(j)} \sim \boldsymbol{\pi_\theta}(\boldsymbol{a} \,|\, \boldsymbol{s}, \boldsymbol{\xi}_j^{(0)})$ is sampled via the reparametrization trick by letting $\boldsymbol{a}^{(j)} = \mathcal{T}_{\boldsymbol{\theta}}(\boldsymbol{s}, \boldsymbol{\xi}_j^{(0)}, \boldsymbol{e}_j),\ \boldsymbol{e}_j \sim p(\boldsymbol{e})$ to ensure low gradient estimation variance, which means it is deterministically transformed from $\boldsymbol{s}$, $\boldsymbol{\xi}_j^{(0)}$, and random noise $\boldsymbol{e}_j \sim p(\boldsymbol{e})$ with a nueral network parameterized by $\boldsymbol{\theta}$. To compute the gradient of $\overline{J}(\boldsymbol{\theta}) := \sum_{j=1}^J \overline{J}_j(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$, we notice that $\nabla_{\boldsymbol{\theta}} \log \left(\frac{\pi_{\boldsymbol{\theta}}(\boldsymbol{a}^{(j)} \,|\, \boldsymbol{s}, \boldsymbol{\xi}_j^{(0)}) + \sum_{\ell=1}^L \pi_{\boldsymbol{\theta}}(\boldsymbol{a}^{(j)} \,|\, \boldsymbol{s}, \boldsymbol{\xi}^{(\ell)})}{L+1} \right)$ can be rewritten as the summation of two terms: the first term is obtained by treating $\boldsymbol{a}^{(j)}$ in $\boldsymbol{\pi_\theta}(\boldsymbol{a}^{(j)} \,|\, -)$ as constants, and the second term by treating $\boldsymbol{\theta}$ in $\boldsymbol{\pi_\theta}(\cdot)$ as constants. Since $\mathbb{E}_{\boldsymbol{a} \sim \boldsymbol{\pi_\theta}(\boldsymbol{a} \,|\, \boldsymbol{s})}[\nabla_{\boldsymbol{\theta}} \log \boldsymbol{\pi_\theta}(\boldsymbol{a} \,|\, \boldsymbol{s})] = 0$, the expectation of the first term becomes zero when $L \to \infty$. For this reason, we omit its contribution to the gradient when computing $\nabla_{\boldsymbol{\theta}} \overline{J}(\boldsymbol{\theta})$, which can then be expressed as

$$\nabla_{\boldsymbol{\theta}} \overline{J}(\boldsymbol{\theta}) = \sum_{j=1}^J \Bigg\{ \Big[\Big(\tfrac{1}{2J} \sum_{i=1}^2 \nabla_{\boldsymbol{a}^{(j)}} G_{\boldsymbol{\omega}_i}(\boldsymbol{s}, \boldsymbol{a}^{(j)}, \boldsymbol{\epsilon}^{(j)}) \Big)\Big)$$
$$- \tfrac{1}{J} \alpha \sum_{\ell=0}^L \frac{\pi_{\boldsymbol{\theta}}(\boldsymbol{a}^{(j)} \,|\, \boldsymbol{s}, \boldsymbol{\xi}_j^{(\ell)})}{\sum_{\ell'=0}^L \pi_{\boldsymbol{\theta}}(\boldsymbol{a}^{(j)} \,|\, \boldsymbol{s}, \boldsymbol{\xi}_j^{(\ell')})} \nabla_{\boldsymbol{a}^{(j)}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}^{(j)} \,|\, \boldsymbol{s}, \boldsymbol{\xi}_j^{(\ell)}) \Big] \Big|_{\boldsymbol{a}^{(j)} = \mathcal{T}_{\boldsymbol{\theta}}(\boldsymbol{s}, \boldsymbol{\xi}_j^{(0)}, \mathrm{e}_j)} \nabla_{\boldsymbol{\theta}} \mathcal{T}_{\boldsymbol{\theta}}(\boldsymbol{s}, \boldsymbol{\xi}_j^{(0)}, \mathrm{e}_j) \Bigg\}, \tag{4.16}$$

where with a slight abuse of notation, we denote $\boldsymbol{\xi}_j^{(\ell)} = \boldsymbol{\xi}^{(\ell)}$ when $\ell > 0$.

We follow Haarnoja et al. [2018b] to adaptively adjust the reward scaling coefficient α. Denote H_{target} as a fixed target entropy, heuristically chosen as $H_{\text{target}} = -\dim(\mathcal{A})$. We update α by performing gradient descent on $\eta := \log(\alpha)$

under the loss

$$J(\eta) = \mathbb{E}_{\boldsymbol{s}\sim\rho(\boldsymbol{s})}[\eta(-\log\pi_{\boldsymbol{\theta}}(\boldsymbol{a}\,|\,\boldsymbol{s}) - H_{\text{target}})], \tag{4.17}$$

where the marginal log-likelihood is estimated by $\log\pi_{\boldsymbol{\theta}}(\boldsymbol{a}\,|\,\boldsymbol{s}) = \log\frac{\sum_{\ell=0}^{L}\pi_{\boldsymbol{\theta}}(\boldsymbol{a}\,|\,\boldsymbol{s},\boldsymbol{\xi}^{(\ell)})}{L+1}$, where $\boldsymbol{a}\sim\pi_{\boldsymbol{\theta}}(\cdot\,|\,\boldsymbol{s},\boldsymbol{\xi}^{(0)})$ and $\boldsymbol{\xi}^{(0)},\boldsymbol{\xi}^{(1)},\dots,\boldsymbol{\xi}^{(L)} \overset{iid}{\sim} p(\boldsymbol{\xi})$.

4.2.6 Off policy learning with IDAC

We incorporate the proposed twin-delayed DGNs and SIA into the off-policy framework. Specifically, the samples are gathered with a SIA based behavior policy and stored in a replay buffer. In detail, for each state $\boldsymbol{s}_t$, the agent will first sample a random noise $\boldsymbol{\xi}_t\sim p(\boldsymbol{\xi})$, then generate an action by $\boldsymbol{a}_t\sim\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t\,|\,\boldsymbol{s}_t,\boldsymbol{\xi}_t)$, and observe a reward r_t and next state $\boldsymbol{s}_{t+1}$ returned by the environment.

We save the tuples $(\boldsymbol{s}_t,\boldsymbol{a}_t,r_t,\boldsymbol{s}_{t+1})$ in a replay buffer and sample them uniformly when training the DGNs based implicit distributional critics and the SIA based semi-implicit policy, therefore all the previous $\rho(\boldsymbol{s})$ is the uniform distribution from the replay buffer. We show an overview of the algorithm here and provide a detailed pseudo code with all implementation details in Appendix B.3.

4.3 Experiments

Our experiments serve to answer the following questions: **(a)** How does IDAC perform when compared to state-of-art baselines, including SAC

Algorithm 1 IDAC: Implicit Distributional Actor Critic (see Appendix B.3 for more implementation details)

```
Require: Learning rate $\lambda$, smoothing factor $\tau$. Initial policy network parameter $\boldsymbol{\theta}$,
  distributional generator network parameters $\boldsymbol{\omega}_1$,$\boldsymbol{\omega}_2$, entropy coefficient $\eta$;
  $\tilde{\boldsymbol{\omega}}_1 \leftarrow \boldsymbol{\omega}_1$, $\tilde{\boldsymbol{\omega}}_2 \leftarrow \boldsymbol{\omega}_2$, $\mathcal{D} \leftarrow \emptyset$
  for Each iteration do
    for Each environment step do
      $\boldsymbol{\xi}_t \sim p(\boldsymbol{\xi}),\ \boldsymbol{a}_t \sim \pi_{\boldsymbol{\theta}}(\cdot \,|\, \boldsymbol{s}_t, \boldsymbol{\xi}_t)$ {Sample noise and then action}
      $\boldsymbol{s}_{t+1} \sim p(\cdot \,|\, \boldsymbol{s}_t, \boldsymbol{a}_t)$ {Observe next state}
      $\mathcal{D} \leftarrow \mathcal{D} \cup (\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1})$ {Store transition tuples}
    end for
    Sample transitions from the replay buffer
    $\boldsymbol{\omega}_i \leftarrow \boldsymbol{\omega}_i - \lambda \nabla_{\boldsymbol{\omega}_i} J(\boldsymbol{\omega}_i)$ for $i = 1, 2$ {Update DGNs, Eq. (4.10)}
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \lambda \nabla_{\boldsymbol{\theta}} \overline{J}(\boldsymbol{\theta})$ {Update SIA, Eq. (4.16)}
    $\eta \leftarrow \eta - \lambda \nabla_{\eta} J(\eta)$, let $\alpha = \exp(\eta)$ {Update entropy coefficient, Eq. (4.17)}
    $\tilde{\boldsymbol{\omega}}_i \leftarrow \tau \boldsymbol{\omega}_i + (1 - \tau) \tilde{\boldsymbol{\omega}}_i$ for $i = 1, 2$ {Soft update delayed networks}
  end for
```

[Haarnoja et al., 2018a], TD3 [Fujimoto et al., 2018], and PPO [Schulman et al., 2017]? **(b)** Can a semi-implicit policy capture complex distributional properties such as skewness, multi-modality, and covariance structure? **(c)** How well is the distributional matching when minimizing the quantile regression Huber loss? **(d)** How important is the type of policy distribution, such as a semi-implicit policy, a diagonal Gaussian policy, or a deterministic policy under this framework? **(e)** How much improvement does distributional critics bring? **(f)** How critical is the twin delayed network? **(g)** Will other baselines (such as SAC) benefit from using multiple actions ($J > 1$) for policy gradient estimation?

We will show two sets of experiments, one for **evaluation study** and the other for **ablation study**, to answer the aforementioned questions. The evaluation study will be addressing questions **(a)**-**(c)** and ablation study will be addressing **(d)**-**(g)**.

As shown in Engstrom et al. [2019], the code-level implementation of different RL algorithms can lead to significant differences in their empirical performances and hence a fair comparison needs to be run on the same codebase. Thus all compared algorithms are either from, or built upon the *stable baselines* codebase (`https://github.com/hill-a/stable-baselines`) of Hill et al. [2018] to minimize the potential gaps caused by the differences of code-level implementations.

IDAC is implemented with a uniform hyperparameter set to guarantee fair comparisons. Specifically, we use three separate fully-connected multilayer perceptrons (MLPs), which all have two 256-unit hidden layers and ReLU nonlinearities, to define the proposed SIA and two DGNs, respectively. Both $p(\boldsymbol{\xi})$ and $p(\boldsymbol{\epsilon})$ are $\mathcal{N}(\mathbf{0}, \mathbf{I}_5)$ and such a random noise, concatenated with the state $\boldsymbol{s}$, will be used as the input of its corresponding network. We fix for all experiments the number of noise $\boldsymbol{\xi}^{(\ell)}$ as $L = 21$. We set the number of equally-spaced quantiles (the same as the number of $\boldsymbol{\epsilon}^{(k)}$) as $K = 51$ and number of auxiliary actions as $J = 51$ by default. A more detailed parameter setting can be found in Appendix C.3. We conduct empirical comparisons on the benchmark tasks provided by OpenAI Gym [Brockman et al., 2016] and MuJoCo simulators [Todorov et al., 2012].

Table 4.1: Comparison of average maximal returns ± 1 std over 4 different random seeds.

ENV	PPO	TD3	SAC	IDAC
BipedalWalker-v2	241.79 ± 36.7	182.80 ± 135.76	312.48 ± 2.81	**328.44** ± **1.23**
Walker2d-v2	1679.39 ± 942.49	3689.48 ± 434.03	4328.95 ± 249.27	**5107.07** ± **351.37**
Hopper-v2	1380.68 ± 899.70	1799.78 ± 1242.63	3138.93 ± 299.62	**3497.86** ± **93.30**
HalfCheetah-v2	1350.37 ± 128.79	10209.65 ± 548.14	10626.34 ± 73.78	**12222.80** ± **157.15**
Ant-v2	141.79 ± 451.10	4905.74 ± 203.09	3732.23 ± 602.83	**4930.73** ± **242.78**
Humanoid-v2	498.88 ± 20.10	105.76 ± 53.65	5055.64 ± 62.96	**5233.43** ± **85.87**

4.3.1 Evaluation study to answer questions (a)-(c)

(a): We compare IDAC with SAC, TD3, and PPO on challenging continuous control tasks; each task is evaluated across 4 random seeds and the evaluation is done per 2000 steps with 5 independent rollouts using the most recent policy (to evaluate IDAC, we first sample $\boldsymbol{\xi} \sim p(\boldsymbol{\xi})$ and then use the mean of $\pi_{\boldsymbol{\theta}}(\boldsymbol{a} \,|\, \boldsymbol{s}, \boldsymbol{\xi})$ as action output). As shown in Fig. 4.1, IDAC outperforms all baseline algorithms with a clear margin across almost all tasks. More detailed numerical comparisons can be found in Table 4.1. For all baselines, we use their default hyperparameter settings from the original papers. Notice that J, K, and L are hyperparameters to set, and making them too small might lead to clearly degraded performance for some tasks. In this paper, to balance performance and computational complexity, we choose moderate values of $J = 51$, $K = 51$, and $L = 21$ for all evaluations.

(b): We also check how well is semi-implicit policy and whether it can capture complex distributional properties. We defer the empirical improvement that semi-implicit policy brings to the ablation study part and only show the

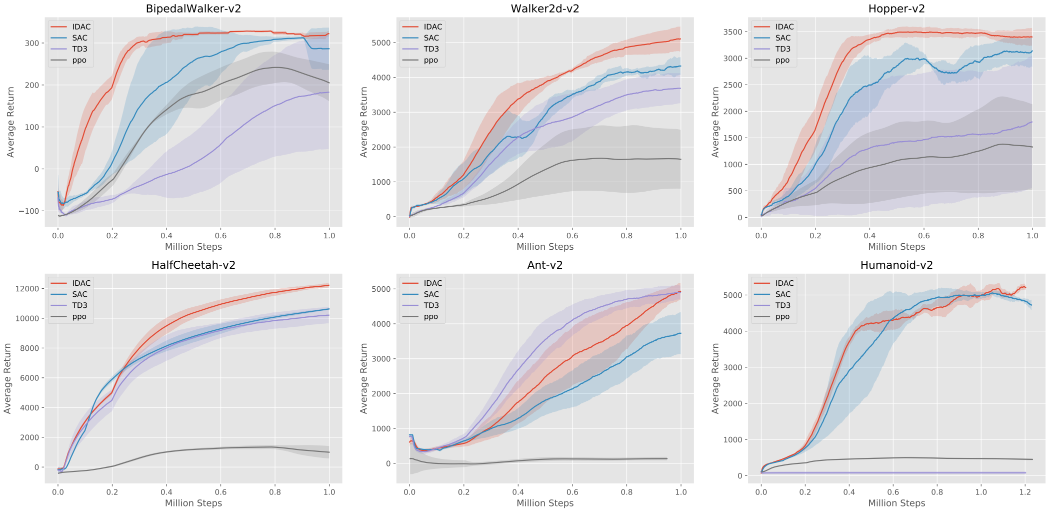
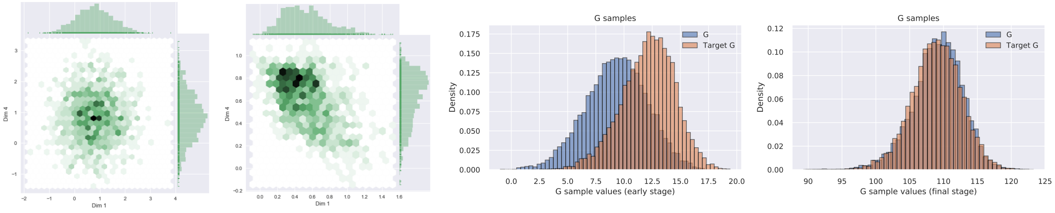


Figure 4.1: Training curves on continuous control benchmarks. The solid line is the average performance over seeds with ± 1 std shaded, and with a smoothing window of length 100.



(a) Gaussian (b) SIA (c) G distributional matching

Figure 4.2: Visualization of Gaussian policy, SIA, and distributional matching for critic generators under SIA. Panels (a) and (b) show the density contour of 1000 random sampled actions at an early training stage, where x-axis and y-axis correspond to dimensions 1 and 4, respectively; Panel (c) shows the empirical density of 10000 DGN samples at an early training stage and the final one, where (target) G samples are in (red) blue.

flexible distribution it supports here. Specifically, we generate this plot by sampling $\boldsymbol{a}_i \sim \pi_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{s})$ for $i = 1, \dots, 1000$, and use these 1000 random actions samples (where $\boldsymbol{\theta}$ is the policy parameters at 10^4 timestep while the total training steps is 10^6), generated given a state $\boldsymbol{s}$, to visualize the empirical joint distribution of two selected dimensions of the action, and the marginal distributions at both dimensions. As shown in the left two panels of Fig. 4.2, the semi-implicit policy is capable of capturing multi-modality, sknewness, and dependencies between different dimensions, none of which are captured by the diagonal Gaussian policy. This flexible policy of SIA can be beneficial to exploration especially during the early training stages. Furthermore, capturing the correlation between action dimensions intuitively will lead to a better policy, *e.g.*, a robot learning to move needs to coordinate the movements of different legs.

(c): Similar to Singh et al. [2020], we check the matching situation of minimizing the quantile regression Huber loss. In detail, we generate 10000 random noises $\boldsymbol{\epsilon}_k \sim p(\boldsymbol{\epsilon})$ to obtain $\{G_{\tilde{\boldsymbol{\omega}}_1}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\epsilon}_k)\}_{k=1}^{10000}$ and $\{r(\boldsymbol{s}, \boldsymbol{a}) + \gamma G_{\tilde{\boldsymbol{\omega}}_1}(\boldsymbol{s}', \boldsymbol{a}', \boldsymbol{\epsilon}_k)\}_{k=1}^{10000}$, and then compare their histograms to check if the empirical distributions are similar to each other. We list the distributions on both early and late stages to demonstrate the evolvement of the DGN. On an early stage, both the magnitude and shape of two distributions are very different, while their differences diminish at a fast pace along with the training process. It illustrates that the DGN is able to represent the distribution well defined by the distributional Bellman equation.

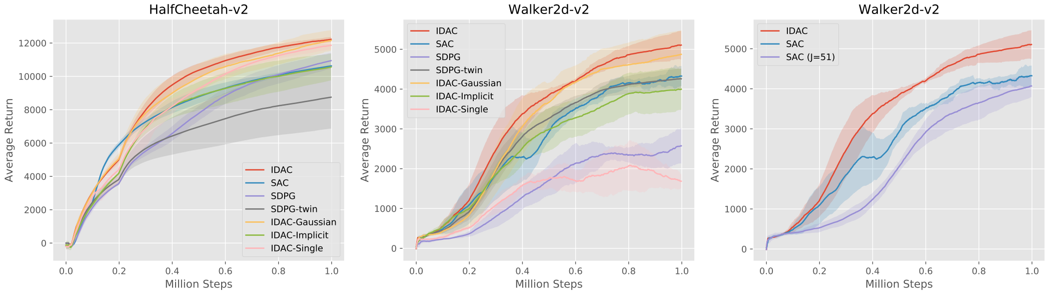


Figure 4.3: Training Curves of Ablation study.

4.3.2 Ablation study to answer questions (d)-(g)

We run a comprehensive set of ablation study to demonstrate the effectiveness of the SIA and DGNs have on the performance. In general, there are three parts that we can control to see the differences they contribute: **(i)**: policy distribution {deterministic policy, Gaussian policy, semi-implicit policy, implicit policy}; **(ii)**: distributional aspect {no: action-value function, yes: distributional critic generator}; **(iii)**: prevent overestimation bias trick {single delayed network, twin delayed network}. Among those possible combinations, we choose a representative subset of them to show that the structure of IDAC is reasonable and bring significant improvement. They also answer the questions **(d)**-**(g)** as we proposed in the beginning. An implicit policy is constructed by concatenating a random variable with state, and obtain an action from the deterministic transformation. In this way, the policy itself is still stochastic, but the log-likelihood is intractable and thus cannot use any entropy regularization trick.

We list all the 8 representative variants in Table 4.2, and evaluate

Table 4.2: Variants for ablation study

Ablations	policy distribution	distributional approach	estimation trick
IDAC	semi-implicit	yes	twin
SAC	Gaussian	no	twin
SAC-J[1]	Gaussian	no	twin
SDPG[2]	deterministic	yes	single
SDPG-twin	deterministic	yes	twin
IDAC-Gaussian	Gaussian	yes	twin
IDAC-Implicit	implicit	yes	twin
IDAC-single	semi-implicit	yes	single

their performances on HalfCheetah and Walker2d environments with the same evaluation process described in Section 4.3.1. [[1] SAC-J refers to SAC with J actions to estimate its objective function. [2] Note that the SDPG paper [Singh et al., 2020] is using a different codebase; the implementation-level differences make their reported results not directly comparable; we use this variant to illustrate how each component works.]

(d): We make comparisons between IDAC, SDPG-twin, IDAC-Gaussian, and IDAC-Implicit to demonstrate the superiority of using a semi-implicit policy. As shown in Fig. 4.3, we have IDAC ¿ IDAC-Gaussian ¿ SDPG-twin ¿ IDAC-Implicit, which not only demonstrates the improvement from semi-implicit policy, but also implies the importance of using a stochastic policy with entropy regularization as shown in Haarnoja et al. [2018a].

(e): The effect of the DGNs can be directly observed by comparing between IDAC-Gaussian and SAC, where IDAC-Gaussian is better than SAC on both tasks as shown in Fig. 4.3.

(f): To understand the importance of twin delayed network structure, we make comparisons between SDPG with SDPG-twin, and IDAC with IDAC-single. As shown in Fig. 4.3, the one with the twin structure significantly outperform its counterpart without the twin structure in both cases, which demonstrate the effectiveness of the twin delayed networks.

(g): Eventually, we want to demonstrate that the improvement of IDAC is not simply by sampling multiple actions for objective function estimation. As shown in the right panel of Fig. 4.3, the implementation of multiple actions on SAC does not boost the performance of SAC.

4.4 Conclusion

In this paper, we present implicit distributional actor-critic (IDAC), an off-policy based actor-critic algorithm incorporated with distributional learning. We model the return distribution with a deep generative network (DGN) and the policy with a semi-implicit actor (SIA), and mitigate the overestimation issue with a twin DGNs structure. We validate the critical roles of these components with a detailed ablation study, and demonstrate that IDAC is capable of state-of-the-art performances on a number of challenging continuous control problems.

Chapter 5

Conclusions and future directions

5.1 Conclusions

In this thesis, we propose methodologies that boost deep RL algorithms with deep probabilistic models and demonstrate the great potential that probabilistic models can contribute to deep RL area. Moreover, most deep RL algorithms cannot work without some heuristic techniques; we hope that by viewing from a statistical way, we can not only improve the empirical performance, but also understand those techniques more deeply so that we can use them in a more systematic way.

5.2 Future directions

5.2.1 Diverse proposal policy for DQN based algorithm

As mentioned in Chapter 1, one feasible way to apply DQN based algorithm in the continuous-action domain is training a proposal policy that can propose candidate actions which are likely to contain the best action. Denote the proposal policy as $\pi(\boldsymbol{a}|\boldsymbol{s})$, we hope that $Q(\boldsymbol{s}, \boldsymbol{a})$ can be large when $\boldsymbol{a} \sim \pi(\boldsymbol{a}|\boldsymbol{s})$ and also be diverse so that it will not degenerate to a deterministic policy. We characterize the policy as $\pi(\boldsymbol{a}|\boldsymbol{s}) = f_{\boldsymbol{\theta}}(\boldsymbol{s}) \odot \boldsymbol{\epsilon}$, $\boldsymbol{\epsilon} \sim p_{\boldsymbol{\phi}}(\boldsymbol{s})$ which is the dot product of a deterministic policy and a contexual dropout mask [Fan et al., 2021]. In this way, we can control the diversity of $\pi(\boldsymbol{a}|\boldsymbol{s})$ to avoid degeneration. The update of $\boldsymbol{\theta}$ is similar to that in DDPG [Lillicrap et al., 2015], and the update of $\boldsymbol{\phi}$ is composed of two parts: 1). $Q^{\pi}(\boldsymbol{s}, \boldsymbol{\epsilon})$ should be large; 2). $p_{\boldsymbol{\phi}}(\boldsymbol{\epsilon}|\boldsymbol{s})$ should be close to a prior to prevent overfitting. We thereby optimize the following objective function to update $\boldsymbol{\phi}$:

$$J(\boldsymbol{\phi}) = Q^{\pi}(\boldsymbol{s}, \pi_{\boldsymbol{\phi}}(\boldsymbol{\epsilon})) - \lambda \mathrm{KL}(p_{\boldsymbol{\phi}}(\boldsymbol{\epsilon}|\boldsymbol{s})||p(\boldsymbol{\epsilon}));$$

this objective function is similar to the Evidence Lower Bound (ELBO) of $p(\boldsymbol{\epsilon}|\boldsymbol{s})$ where the likelihood is proportional to $e^{Q^{\pi}(\boldsymbol{s}, \boldsymbol{\epsilon})}$. This approach is promising since it enforces the proposal policy $\pi(\boldsymbol{a}|\boldsymbol{s})$ to be diverse as long as $p_{\boldsymbol{\phi}}(\boldsymbol{\epsilon}|\boldsymbol{s})$ does not degenerate; and it also has the capability of proposing actions with large action values.

5.2.2 Offline reinforcement learning

Offline RL has become a popular topic recently [Fujimoto et al., 2019, Kumar et al., 2019], which aims to train a policy with few or no interactions with the environment [Levine et al., 2020]. One major challenge in offline RL is the "distributional shift", which describe the mismatch between the offline environment and online environment. Due to this mismatch, training an offline RL task requires more conservative distributional constraints on policy updates compared with classical RL tasks [Kumar et al., 2020], and makes it a promising direction to apply probabilistic models to make a difference.

Appendices

Appendix A

Appendix for ARSM: Augment-REINFORCE-Swap-Merge Gradient for Categorical Variables and Policy Optimization

A.1 Derivation of AR, ARS, and ARSM

A.1.1 Augmentation of a Categorical Variable

Let us denote $\tau \sim \text{Exp}(\lambda)$ as the exponential distribution, with probability density function $p(\tau \,|\, \lambda) = \lambda e^{-\lambda \tau}$, where $\lambda > 0$ and $\tau > 0$. Its mean and variance are $\mathbb{E}[\tau] = \lambda^{-1}$ and $\text{var}[\tau] = \lambda^{-2}$, respectively. It is well known that, e.g. in Ross [2006], if $\tau_i \sim \text{Exp}(\lambda_i)$ are independent exponential random variables for $i = 1, \ldots, C$, then the probability that τ_z, where $z \in \{1, \ldots, C\}$, is the smallest can be expressed as

$$P\big(z = \arg\min_{i \in \{1,\ldots,C\}} \tau_i\big) = P\left(\tau_z < \tau_i, \ \forall \ i \neq z\right) = \tfrac{\lambda_z}{\sum_{i=1}^{C} \lambda_i} \ . \tag{A.1}$$

Note this property, referred to as “exponential racing” in Zhang and Zhou [2018], is closely related to the Gumbel distribution (also known as Type-I extreme-value distribution) based latent-utility-maximization representation of multinomial logistic regression [McFadden, 1974, Train, 2009], as well as the Gumbel-softmax trick [Maddison et al., 2017, Jang et al., 2017]. This is because the exponential random variable $\tau \sim \text{Exp}(\lambda)$ can be reparameterized as $\tau = \epsilon/\lambda,\ \epsilon \sim \text{Exp}(1)$, where $\epsilon \sim \text{Exp}(1)$ can be equivalently generated as $\epsilon = -\log u,\ u \sim \text{Uniform}(0,1)$, and hence we have

$$\arg\min\nolimits_i \tau_i \overset{d}{=} \arg\min\nolimits_i\{-\log u_i/\lambda_i\} = \arg\max\nolimits_i\{\log\lambda_i - \log(-\log u_i)\},$$

where $\tau_i \sim \text{Exp}(\lambda_i)$, "$\overset{d}{=}$" denotes "equal in distribution," and $u_i \overset{iid}{\sim} \text{Uniform}(0,1)$; note that if $u \sim \text{Uniform}(0,1)$, then $-\log(-\log u)$ follows the Gumbel distribution [Train, 2009].

From (A.1) we know that if

$$z = \arg\min\nolimits_{i\in\{1,\dots,C\}} \tau_i \text{ , where } \tau_i \sim \text{Exp}(e^{\phi_i}), \tag{A.2}$$

then $P(z \,|\, \boldsymbol{\phi}) = e^{\phi_z}/\sum_{i=1}^{C} e^{\phi_i}$, and hence (A.2) is an augmented representation of the categorical distribution $z \sim \text{Cat}(\sigma(\boldsymbol{\phi}))$; one may consider $\tau_i \sim \text{Exp}(e^{\phi_i})$ as augmented latent variables, the marginalization of which from $z = \arg\min_{i\in\{1,\dots,C\}} \tau_i$ leads to $P(z \,|\, \boldsymbol{\phi})$. Consequently, the expectation with respect to the categorical variable of C categories can be rewritten as one with respect to C augmented exponential random variables as

$$\mathcal{E}(\boldsymbol{\phi}) = \mathbb{E}_{z\sim\text{Cat}(\sigma(\boldsymbol{\phi}))}[f(z)] = \mathbb{E}_{\tau_1\sim\text{Exp}(e^{\phi_1}),\dots,\tau_C\sim\text{Exp}(e^{\phi_C})}[f(\arg\min\nolimits_i \tau_i)]. \tag{A.3}$$

Since the exponential random variable $\tau \sim \text{Exp}(e^{\phi})$ can be reparameterized as $\tau = \epsilon e^{-\phi},\ \epsilon \sim \text{Exp}(1)$, we also have

$$\mathcal{E}(\boldsymbol{\phi}) = \mathbb{E}_{\epsilon_1,\dots,\epsilon_C \overset{iid}{\sim} \text{Exp}(1)}[f(\arg\min\nolimits_i \epsilon_i e^{-\phi_i})]. \tag{A.4}$$

Note as the $\arg\min$ operator is non-differentiable, the widely used reparameterization trick [Kingma and Welling, 2013, Rezende et al., 2014] is not applicable to computing the gradient of $\mathcal{E}(\boldsymbol{\phi})$ via the reparameterized representation in (A.4).

A.1.2 REINFORCE Estimator in the Augmented Space

Using REINFORCE [Williams, 1992b] on (A.3), we have $\nabla_{\boldsymbol{\phi}}\mathcal{E}(\boldsymbol{\phi}) = [\nabla_{\phi_1}\mathcal{E}(\boldsymbol{\phi}), \ldots, \nabla_{\phi_C}\mathcal{E}(\boldsymbol{\phi})]'$, where

$$\begin{aligned}
\nabla_{\phi_c}\mathcal{E}(\boldsymbol{\phi}) &= \mathbb{E}_{\tau_1\sim\text{Exp}(e^{\phi_1}),\ldots,\tau_C\sim\text{Exp}(e^{\phi_C})}\left[f(\arg\min\nolimits_i \tau_i)\nabla_{\phi_c}\log\textstyle\prod_{i=1}^C \text{Exp}(\tau_i; e^{\phi_i})\right] \\
&= \mathbb{E}_{\tau_1\sim\text{Exp}(e^{\phi_1}),\ldots,\tau_C\sim\text{Exp}(e^{\phi_C})}[f(\arg\min\nolimits_i \tau_i)\nabla_{\phi_c}\log\text{Exp}(\tau_c; e^{\phi_c})] \\
&= \mathbb{E}_{\tau_1\sim\text{Exp}(e^{\phi_1}),\ldots,\tau_C\sim\text{Exp}(e^{\phi_C})}[f(\arg\min\nolimits_i \tau_i)(1-\tau_c e^{\phi_c})]. \qquad \text{(A.5)}
\end{aligned}$$

Below we show how to merge $\nabla_{\phi_c}\mathcal{E}(\boldsymbol{\phi})$ and $-\nabla_{\phi_j}\mathcal{E}(\boldsymbol{\phi})$ by first re-expressing (A.5) into an expectation with respect to *iid* exponential random variables, swapping the indices of these random variables, and then sharing common random numbers [Owen, 2013] to well control the variance of Monte Carlo integration.

A.1.3 Merge of Augment-REINFORCE Gradients

A key observation of the paper is we can re-express the expectation in (A.5) as

$$\nabla_{\phi_c}\mathcal{E}(\boldsymbol{\phi}) = \mathbb{E}_{\epsilon_1,\ldots,\epsilon_C \overset{iid}{\sim} \text{Exp}(1)}[f(\arg\min\nolimits_i \epsilon_i e^{-\phi_i})(1-\epsilon_c)] \qquad \text{(A.6)}$$

Furthermore, we note that $\text{Exp}(1) \overset{d}{=} \text{Gamma}(1,1)$, letting $\epsilon_1, \dots, \epsilon_C \overset{iid}{\sim} \text{Exp}(1)$ is the same (e.g., as proved in Lemma IV.3 of Zhou and Carin [2012]) in distribution as letting

$$\epsilon_i = \pi_i \epsilon, \quad \text{for } i = 1, \dots, C, \quad \text{where } \boldsymbol{\pi} \sim \text{Dirichlet }(\mathbf{1}_C),\ \epsilon \sim \text{Gamma}(C, 1),$$

and $\arg\min_i \pi_i e^{-\phi_i} = \arg\min_i \epsilon \pi_i e^{-\phi_i}$. Thus using Rao-Blackwellization [Casella and Robert, 1996], we can re-express the gradient in (A.5) as

$$\begin{aligned}
\nabla_{\phi_c} \mathcal{E}(\boldsymbol{\phi}) &= \mathbb{E}_{\epsilon \sim \text{Gamma}(C,1),\ \boldsymbol{\pi} \sim \text{Dirichlet}(\mathbf{1}_C)}[f(\arg\min_i \epsilon \pi_i e^{-\phi_i})(1 - \epsilon \pi_c)] \\
&= \mathbb{E}_{\boldsymbol{\pi} \sim \text{Dirichlet}(\mathbf{1}_C)}[f(\arg\min_i \pi_i e^{-\phi_i})(1 - C\pi_c)]. \\
&= \mathbb{E}_{\boldsymbol{\pi} \sim \text{Dirichlet}(\mathbf{1}_C)}[f(\arg\min_i \pi_i^{c \leftrightharpoons j} e^{-\phi_i})(1 - C\pi_j)], \qquad \text{(A.7)}
\end{aligned}$$

where $j \in \{1, \dots, C\}$ is an arbitrarily selected reference category, whose selection does not depends on $\boldsymbol{\pi}$ and $\boldsymbol{\phi}$.

Another useful observation of the paper is that the function

$$b(\boldsymbol{\pi}, \boldsymbol{\phi}, j) = \frac{1}{C} \sum_{m=1}^{C} f(\arg\min_i \pi_i^{m \leftrightharpoons j} e^{-\phi_i})(1 - C\pi_j)$$

has zero expectation, as

$$\mathbb{E}_{\boldsymbol{\pi} \sim \text{Dirichlet}(\mathbf{1}_C)}[b(\boldsymbol{\pi}, \boldsymbol{\phi}, j)] = \mathbb{E}_{\boldsymbol{\pi} \sim \text{Dirichlet}(\mathbf{1}_C)} \left[f(\arg\min_i \pi_i e^{-\phi_i}) \sum_{m=1}^{C} \left(\frac{1}{C} - \pi_m \right) \right] = 0. \tag{A.8}$$

Using $\mathbb{E}[b(\boldsymbol{\pi}, \boldsymbol{\phi}, j)]$ as the baseline function and subtracting it from (A.7) leads to (2.8). We now conclude the proof of Theorem 1 for the AR estimator, and Equation 2.8 for the ARS estimator. Once the ARS estimator is proved, Theorem 2 for the ARSM estimator directly follows.

Proof of Corollary 3. Note that letting $(u, 1-u) \sim \text{Dir}(1,1)$ is the same as letting $u \sim \text{Uniform}(0,1)$. Thus regardless of whether we choose Category 1 or Category 2 for as the reference category, we have

$$\nabla_{\boldsymbol{\phi}_1} \mathcal{E}(\boldsymbol{\phi}) = \mathbb{E}_{u \sim \text{Uniform}(0,1)}[f(\arg\min(u, \sigma(\phi_1 - \phi_2)) - f(\arg\min(1-u, \sigma(\phi_1 - \phi_2))](1/2 - u) \tag{A.9}$$

and $\nabla_{\boldsymbol{\phi}_2} \mathcal{E}(\boldsymbol{\phi}) = -\nabla_{\boldsymbol{\phi}_1} \mathcal{E}(\boldsymbol{\phi})$. Denote $\phi = \phi_1 - \phi_2$ and $\eta = \phi_1 + \phi_2$, we have

$$\nabla_{\phi} \mathcal{E}(\boldsymbol{\phi}) = \nabla_{\phi_1} \mathcal{E}(\boldsymbol{\phi}) \frac{\partial \phi_1}{\partial \phi} + \nabla_{\phi_2} \mathcal{E}(\boldsymbol{\phi}) \frac{\partial \phi_2}{\partial \phi} = \nabla_{\boldsymbol{\phi}_1} \mathcal{E}(\boldsymbol{\phi}).$$

□

A.2 Fast Computation for the Swap Step

Computing the pseudo actions $z^{c \leftrightharpoons j} = \arg\min_i \pi_i^{c \leftrightharpoons j} e^{-\phi_i}$ due to the swap operations can be efficiently realized: we first compute $o_{ij} = \ln \pi_i - \phi_j$, $z = \arg\min_i (\ln \pi_i - \phi_i)$, and $o_{\min} = \ln \pi_z - \phi_z$; then for $m = 1 \ldots, C,\ j < m$, compute

$$z^{m \leftrightharpoons j} = \begin{cases} m, \text{ if } z \notin \{m, j\},\ \min\{o_{mj}, o_{jm}\} < o_{\min},\ o_{mj} \leq o_{jm}; \\ j, \text{ if } z \notin \{m, j\},\ \min\{o_{mj}, o_{jm}\} < o_{\min},\ o_{mj} > o_{jm}; \\ \arg\min_i (\ln \pi_i^{m \leftrightharpoons j} - \phi_i),\ \text{ if } z \in \{m, j\}; \\ z, \text{ otherwise}; \end{cases}$$

and let $z^{j \leftrightharpoons j} = z$ for all j, and $z^{m \leftrightharpoons j} = z^{j \leftrightharpoons m}$ for all $j > m$.

A.3 ARSM for Multivariate, Hierarchical, and Sequential Categorical Variables

A.3.1 ARSM for Multivariate Categorical Variables

Proposition 7 (AR, ARS, and ARSM for multivariate categorical)**.** *Denote* $\boldsymbol{z} = (z_1, \ldots, z_K)$*, where* $z_k \in \{1, \ldots, C\}$*, as a* K *dimensional vector of* C*-way categorical variables. Denote* $\boldsymbol{\Pi} = (\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_K) \in \mathbb{R}^{C \times K}$ *as a matrix obtained by concatenating* K *column vectors* $\boldsymbol{\pi}_k = (\pi_{k1}, \ldots, \pi_{kC})'$*, and* $\boldsymbol{\Phi} = (\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_K) \in \mathbb{R}^{C \times K}$ *by concatenating* $\boldsymbol{\phi}_k = (\phi_{k1}, \ldots, \phi_{kC})'$*. With the multivariate AR estimator, the gradient of*

$$\mathcal{E}(\boldsymbol{\Phi}) = \mathbb{E}_{\boldsymbol{z} \sim \prod_{k=1}^{K} \mathrm{Cat}(z_k; \sigma(\boldsymbol{\phi}_k))}[f(\boldsymbol{z})] \tag{A.10}$$

with respect to ϕ_{kc} *is expressed as*

$$\begin{aligned} \nabla_{\phi_{kc}} \mathcal{E}(\boldsymbol{\Phi}) &= \mathbb{E}_{\boldsymbol{\Pi} \sim \prod_{k=1}^{K} \mathrm{Dir}(\boldsymbol{\pi}_k; \mathbf{1}_C)}[f(\boldsymbol{z})(1 - C\pi_{kc})], \\ z_k &:= \arg\min\nolimits_{i \in \{1, \ldots, C\}} \pi_{ki} e^{-\phi_{ki}}. \end{aligned} \tag{A.11}$$

Denoting $\boldsymbol{j} = (j_1, \ldots, j_K)$*, where* $j_k \in \{1, \ldots, C\}$ *is a randomly selected reference category for dimension* k*, the multivariate ARS estimator is expressed as*

$$\begin{aligned} \nabla_{\phi_{kc}} \mathcal{E}(\boldsymbol{\Phi}) &= \mathbb{E}_{\boldsymbol{\Pi} \sim \prod_{k=1}^{K} \mathrm{Dir}(\boldsymbol{\pi}_k; \mathbf{1}_C)}[f_{\Delta}^{c \rightleftharpoons \boldsymbol{j}}(\boldsymbol{\Pi})(1 - C\pi_{kj_k})], \\ f_{\Delta}^{c \rightleftharpoons \boldsymbol{j}}(\boldsymbol{\Pi}) &:= f(\boldsymbol{z}^{c \rightleftharpoons \boldsymbol{j}}) - \tfrac{1}{C} \textstyle\sum_{m=1}^{C} f(\boldsymbol{z}^{m \rightleftharpoons \boldsymbol{j}}), \\ \boldsymbol{z}^{c \rightleftharpoons \boldsymbol{j}} &:= (z_1^{c \rightleftharpoons j_1}, z_2^{c \rightleftharpoons j_2}, \ldots, z_K^{c \rightleftharpoons j_K}), \\ z_k^{c \rightleftharpoons j_k} &:= \arg\min\nolimits_{i \in \{1, \ldots, C\}} \pi_{ki}^{c \rightleftharpoons j_k} e^{-\phi_{ki}}. \end{aligned} \tag{A.12}$$

Setting $\boldsymbol{j} = j\mathbf{1}_K$ and averaging over all $j \in \{1, \dots, C\}$, the multivariate ARSM estimator is expressed as

$$\nabla_{\phi_{kc}}\mathcal{E}(\mathbf{\Phi}) = \mathbb{E}_{\mathbf{\Pi}\sim\prod_{k=1}^{K}\mathrm{Dir}(\boldsymbol{\pi}_k;\mathbf{1}_C)}\big[\textstyle\sum_{j=1}^{C} f_{\Delta}^{c\leftrightharpoons(j\mathbf{1}_K)}(\mathbf{\Pi})(\frac{1}{C} - \pi_{kj})\big]. \tag{A.13}$$

Note to obtain $\nabla_{\phi_{kc}}\mathcal{E}(\mathbf{\Phi})$ for all k and c based on the ARS estimator in (A.12), we only need to evaluate $f(\boldsymbol{z}^{1\leftrightharpoons\boldsymbol{j}}), \dots, f(\boldsymbol{z}^{C\leftrightharpoons\boldsymbol{j}})$. Thus regardless of how large K is, to obtain a single Monte Carlo sample estimate of the true gradient, one needs to evaluate the reward function $f(\cdot)$ as few as zero time, which happens when the number of unique vectors in $\{\boldsymbol{z}^{c\leftrightharpoons\boldsymbol{j}}\}_{c=1,C}$ is one, and as many as C times, which happens when all $\boldsymbol{z}^{c\leftrightharpoons\boldsymbol{j}}$ are different from each other. Similarly, if the ARSM estimator in (A.13) is used, the number of times one needs to evaluate $f(\cdot)$ is between zero and $C(C-1)/2+1$. In the multivariate setting where $\boldsymbol{z} \in \{1, \dots, C\}^K$, we often choose a relatively small C, such as $C = 10$, but allows K to be as large as necessary, such as $K = 100$. Thus even C^K, the number of unique $\boldsymbol{z}$'s, could be enormous when K is large, both the ARS and ARSM estimators remain computationally efficient; this differs them from estimators, such as the one in Titsias and Lázaro-Gredilla [2015], that are not scalable in the dimension K.

A.3.2 ARSM for Categorical Stochastic Networks

Let us construct a T-categorical-stochastic-layer network as

$$\begin{aligned} q_{\mathbf{\Phi}_{1:T}}(\boldsymbol{z}_{1:T}\,|\,\boldsymbol{x}) = \textstyle\prod_{t=1}^{T} q(\boldsymbol{z}_t\,|\,\mathbf{\Phi}_t),\ \mathbf{\Phi}_t := \mathcal{T}_{\boldsymbol{w}_t}(\boldsymbol{z}_{1:t-1}),\\ q(\boldsymbol{z}_t\,|\,\mathbf{\Phi}_t) := \textstyle\prod_{k=1}^{K_t}\mathrm{Cat}(z_{tk};\sigma(\boldsymbol{\phi}_{tk})), \end{aligned} \tag{A.14}$$

where $\boldsymbol{z}_0 := \boldsymbol{x}$, $\boldsymbol{z}_t := (z_{t1}, \dots, z_{tK_t})' \in \{1, \dots, C\}^{K_t}$ is a K_t-dimensional C-way categorical vector at layer t, $\boldsymbol{\phi}_{tk} := (\phi_{tk1}, \dots, \phi_{tkC})' \in \mathbb{R}^C$ is the parameter vector for dimension k at layer t, $\boldsymbol{\Phi}_t := \big(\boldsymbol{\phi}_{t1}, \dots, \boldsymbol{\phi}_{tK_t}\big) \in \mathbb{R}^{C \times K_t}$, and $\mathcal{T}_{\boldsymbol{w}_t}(\cdot)$ represents a function parameterized by $\boldsymbol{w}_t$ that deterministically transforms $\boldsymbol{z}_{t-1}$ to $\boldsymbol{\Phi}_t$. In this paper, we will define $\mathcal{T}_{\boldsymbol{w}_t}(\cdot)$ with a neural network.

Proposition 8. *For the categorical stochastic network defined in* (A.14)*, the ARSM gradient of the objective*

$$\mathcal{E}(\boldsymbol{\Phi}_{1:T}) = \mathbb{E}_{\boldsymbol{z}_{1:T} \sim q_{\boldsymbol{\Phi}_{1:T}}(\boldsymbol{z}_{1:T} \,|\, \boldsymbol{x})} \left[f(\boldsymbol{z}_{1:T}) \right] \tag{A.15}$$

with respect to $\boldsymbol{w}_t$ *can be expressed as* $\nabla_{\boldsymbol{w}_t} \mathcal{E}(\boldsymbol{\Phi}_{1:T}) = \nabla_{\boldsymbol{w}_t} \big(\sum_{k=1}^{K_t} \sum_{c=1}^{C} (\nabla_{\phi_{tkc}} \mathcal{E}(\boldsymbol{\Phi}_{1:T})) \phi_{tkc} \big)$, *where*

$$\nabla_{\phi_{tkc}} \mathcal{E}(\boldsymbol{\Phi}_{1:T}) = \mathbb{E}_{\boldsymbol{\Pi}_t \sim \prod_{k=1}^{K_t} \mathrm{Dir}(\boldsymbol{\pi}_{tk}; \boldsymbol{1}_C)} \big[\textstyle\sum_{j=1}^{C} f_{t\Delta}^{c \leftrightharpoons j}(\boldsymbol{\Pi}_t) \big(\frac{1}{C} - \pi_{tkj} \big) \big], \tag{A.16}$$

where $\boldsymbol{\pi}_{tk} = (\pi_{tk1}, \dots, \pi_{tkC})'$ *is the Dirichlet distributed probability vector for dimension* k *at layer* t *and*

$$\begin{aligned}
f_{t\Delta}^{c \leftrightharpoons j}(\boldsymbol{\Pi}_t) &: = f(Z_t^{c \leftrightharpoons j}) - \tfrac{1}{C} \textstyle\sum_{m=1}^{C} f(Z_t^{m \leftrightharpoons j}), \\
Z_t^{c \leftrightharpoons j} &: = \{\boldsymbol{z}_{1:t-1}, \boldsymbol{z}_{t:T}^{c \leftrightharpoons j}\}, \quad \boldsymbol{z}_{1:t-1} \sim q_{\boldsymbol{\Phi}_{1:t-1}}(\boldsymbol{z}_{1:t-1} \,|\, \boldsymbol{x}), \\
\boldsymbol{z}_t^{c \leftrightharpoons j} &: = (z_{t1}^{c \leftrightharpoons j}, \dots, z_{tK_t}^{c \leftrightharpoons j})', \\
z_{tk}^{c \leftrightharpoons j} &: = \arg\min\nolimits_{i \in \{1, \dots, C\}} \pi_{tki}^{c \leftrightharpoons j} e^{-\phi_{tki}}, \\
\boldsymbol{z}_{t+1:T}^{c \leftrightharpoons j} &\sim q_{\boldsymbol{\Phi}_{t+1:T}}(\boldsymbol{z}_{t+1:T} \,|\, \boldsymbol{z}_{1:t-1}, \boldsymbol{z}_t^{c \leftrightharpoons j}).
\end{aligned}$$

A.3.3 Proofs

Below we show how to generalize Theorem 2 for a univariate categorical variable to Proposition 7 for multivariate categorical variables, and Proposition 8 for hierarchical multivariate categorical variables.

Proof of Proposition 7. For the expectation in (A.10), since z_k are conditionally independent given $\boldsymbol{\phi}_k$, we have

$$\nabla_{\phi_{kc}}\mathcal{E}(\boldsymbol{\Phi}) = \mathbb{E}_{\boldsymbol{z}_{\backslash k}\sim\prod_{k'\neq k}\text{Discrete}(z_{k'};\sigma(\boldsymbol{\phi}_{k'}))}\big[\nabla_{\phi_{kc}}\mathbb{E}_{z_k\sim\text{Cat}(\sigma(\boldsymbol{\phi}_k))}[f(\boldsymbol{z})]\big]. \tag{A.17}$$

Using Theorem 2 to compute the gradient in the above equation directly leads to

$$\nabla_{\phi_{kc}}\mathcal{E}(\boldsymbol{\Phi}) =\mathbb{E}_{\boldsymbol{z}_{\backslash k}\sim\prod_{k'\neq k}\text{Discrete}(z_{k'};\sigma(\boldsymbol{\phi}_{k'}))}\Big\{\mathbb{E}_{\boldsymbol{\pi}_k\sim\text{Dirichlet}(\mathbf{1}_C)}\Big[(f(\boldsymbol{z}_{\backslash k},\boldsymbol{z}_k^{c\leftrightharpoons j})- \tag{A.18}$$

$$\frac{1}{C}\sum_{m=1}^{C}f(\boldsymbol{z}_{\backslash k},\boldsymbol{z}_k^{m\leftrightharpoons j}))(1-C\pi_{kj})\Big]\Big\}, \tag{A.19}$$

The term inside $[\cdot]$ of (A.19) can already be used to estimate the gradient, however, in the worst case scenario that all the elements of $\{\boldsymbol{z}_k^{c\leftrightharpoons j}\}_{j=1,C}$ are different, it needs to evaluate the function $f(\boldsymbol{z}_{\backslash k},\boldsymbol{z}_k^{c\leftrightharpoons j})$ for $j=1,\ldots,C$, and hence C times for each k and KC times in total. To reduce computation and simplify implementation, exchanging the order of the two expectations in (A.19), we have

$$\nabla_{\phi_{kc}}\mathcal{E}(\boldsymbol{\Phi}) = \mathbb{E}_{\boldsymbol{\pi}_k\sim\text{Dirichlet}(\mathbf{1}_C)}\Big\{(1-C\pi_{kj})\mathbb{E}_{\boldsymbol{z}_{\backslash k}\sim\prod_{k'\neq k}\text{Discrete}(z_{k'};\sigma(\boldsymbol{\phi}_{k'}))}\big[\tag{A.20}$$

$$f(\boldsymbol{z}_{\backslash k}, \boldsymbol{z}_k^{c\leftrightharpoons j}) - \frac{1}{C}\sum_{m=1}^{C} f(\boldsymbol{z}_{\backslash k}, \boldsymbol{z}_k^{m\leftrightharpoons j})]\Big\}$$

Note that

$$\begin{aligned}
&\mathbb{E}_{\boldsymbol{z}_{\backslash k}\sim\prod_{k'\neq k}\text{Discrete}(z_{k'};\sigma(\boldsymbol{\phi}_{k'}))}[f(\boldsymbol{z}_{\backslash k}, \boldsymbol{z}_k^{c\leftrightharpoons j})] \\
&= \mathbb{E}_{\boldsymbol{\epsilon}_{\backslash k}\sim\prod_{k'\neq k}\prod_{i=1}^{C}\text{Exp}(\epsilon_{k'i};e^{\phi_{k'i}})}\big[f\big((z_{k'} = \arg\min_{i\in\{1,\dots,C\}} \epsilon_{k'i}e^{-\phi_{k'i}})_{k'\neq k},\ \boldsymbol{z}_k^{c\leftrightharpoons j}\big)\big] \\
&= \mathbb{E}_{\boldsymbol{\epsilon}_{\backslash k}\sim\prod_{k'\neq k}\prod_{i=1}^{C}\text{Exp}(\epsilon_{k'i};e^{\phi_{k'i}})}\big[f\big((z_{k'} = \arg\min_{i\in\{1,\dots,C\}} \epsilon_{k'i}^{(c\leftrightharpoons j)}e^{-\phi_{k'i}})_{k'\neq k},\ \boldsymbol{z}_k^{c\leftrightharpoons j}\big)\big] \\
&= \mathbb{E}_{\boldsymbol{\Pi}_{\backslash k}\sim\prod_{k'\neq k}\text{Dirichlet}(\boldsymbol{\pi}_{k'};\mathbf{1}_C)}\big[f\big((z_{k'} = \arg\min_{i\in\{1,\dots,C\}} \pi_{k'i}^{(c\leftrightharpoons j)}e^{-\phi_{k'i}})_{k'\neq k},\ \boldsymbol{z}_k^{c\leftrightharpoons j}\big)\big] \\
&= \mathbb{E}_{\boldsymbol{\Pi}_{\backslash k}\sim\prod_{k'\neq k}\text{Dirichlet}(\boldsymbol{\pi}_{k'};\mathbf{1}_C)}\big[f\big(\boldsymbol{z}_1^{c\leftrightharpoons j}, \dots, \boldsymbol{z}_K^{c\leftrightharpoons j}\big)\big]
\end{aligned}$$

Plugging the above equation into (A.20) leads to a simplified representation as (A.13) shown in Proposition 7, with which, regardless of the dimensions C, we draw $\boldsymbol{\Pi} = \{\boldsymbol{\pi}_1, \dots, \boldsymbol{\pi}_K\}$ once to produce correlated $\boldsymbol{z}^{c\leftrightharpoons j}$'s, and evaluate the function $f(\cdot)$ at most C times. □

Proof of Proposition 8. For multi-layer stochastic network $q_{\boldsymbol{\Phi}_{1:T}}(\boldsymbol{z}_{1:T}\,|\,\boldsymbol{x}) = q_{\boldsymbol{\Phi}_1}(\boldsymbol{z}_1\,|\,\boldsymbol{x})\Big[\prod_{t=1}^{T-1} q_{\boldsymbol{\Phi}_{t+1}}(\boldsymbol{z}_{t+1}\,|\,\boldsymbol{z}_t)\Big]$, the gradient of the t-th layer parameter $\boldsymbol{\Phi}_t$ is

$$\nabla_{\boldsymbol{\Phi}_t}\mathcal{E}(\boldsymbol{\Phi}_{1:T}) = \mathbb{E}_{\boldsymbol{z}_{1:t-1}\sim q(\boldsymbol{z}_{1:t-1}|\boldsymbol{x})}\nabla_{\boldsymbol{\Phi}_t}\mathbb{E}_{q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})}f_t(\boldsymbol{z}_{1:t})$$

where $f_t(\boldsymbol{z}_{1:t}) = \mathbb{E}_{q(\boldsymbol{z}_{t+1:T}|\boldsymbol{z}_t)}[f(\boldsymbol{z}_{1:T})]$. To compute the ARSM gradient estimator, first draw a single sample $\boldsymbol{z}_{1:t-1} \sim q(\boldsymbol{z}_{1:t-1}\,|\,\boldsymbol{x})$ if $t > 1$ and compute the pseudo action vector for the t-th layer according to Proposition 7 as

$$z_{tk}^{c\leftrightharpoons j} := \arg\min_{i\in\{1,\dots,C\}} \pi_{tki}^{c\leftrightharpoons j}e^{-\phi_{tki}}$$

for $c, j \in \{1, \dots, C\}$. For each pseudo action vector $\boldsymbol{z}_t^{c \leftrightharpoons j}$, sample $\boldsymbol{z}_{t+1:T}^{c \leftrightharpoons j} \sim q(\boldsymbol{z}_{t+1:T} \,|\, \boldsymbol{z}_t^{c \leftrightharpoons j})$ and compute $f_t(\boldsymbol{z}^{c \leftrightharpoons j}) = f(\boldsymbol{z}_{1:t-1}, \boldsymbol{z}_{t:T}^{c \leftrightharpoons j})$. Replacing $f(\boldsymbol{z}^{c \leftrightharpoons j})$ in Proposition 7 with the $f_t(\boldsymbol{z}^{c \leftrightharpoons j})$ leads to the gradient estimator in Proposition 8.

$\square$

Proof of Proposition 4. We first write the objective function $J(\boldsymbol{\theta})$ in terms of the intermediate parameters $\boldsymbol{\phi}_t = \mathcal{T}_{\boldsymbol{\theta}}(\boldsymbol{s}_t)$, and then apply the chain rule to obtain the policy gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$. Since

$$J(\boldsymbol{\phi}_{0:\infty}) = \mathbb{E}_{\mathcal{P}(\boldsymbol{s}_0)\prod_{t=0}^{\infty} \mathcal{P}(\boldsymbol{s}_{t+1} \,|\, \boldsymbol{s}_t, a_t)\mathrm{Cat}(a_t; \sigma(\boldsymbol{\phi}_t))}\left[\sum_{t=0}^{\infty} \gamma^t r(\boldsymbol{s}_t, a_t)\right]$$

we have

$$\begin{aligned}
J(\boldsymbol{\phi}_{0:\infty}) &= \mathbb{E}_{\mathcal{P}(\boldsymbol{s}_0)\left[\prod_{t'=0}^{t-1} \mathcal{P}(\boldsymbol{s}_{t'+1} \,|\, \boldsymbol{s}_{t'}, a_{t'})\mathrm{Cat}(a_{t'}; \sigma(\boldsymbol{\phi}_{t'}))\right]}\left\{\mathbb{E}_{a_t \sim \mathrm{Cat}(\sigma(\boldsymbol{\phi}_t))}\left[\sum_{t'=0}^{t-1} \gamma^{t'} r(s_{t'}, a_{t'}) + \gamma^t Q(\boldsymbol{s}_t, a_t)\right]\right\} \\
&= \mathbb{E}_{\mathcal{P}(\boldsymbol{s}_0)\left[\prod_{t'=0}^{t-1} \mathcal{P}(\boldsymbol{s}_{t'+1} \,|\, \boldsymbol{s}_{t'}, a_{t'})\mathrm{Cat}(a_{t'}; \sigma(\boldsymbol{\phi}_{t'}))\right]}\left\{\mathbb{E}_{a_t \sim \mathrm{Cat}(\sigma(\boldsymbol{\phi}_t))}\left[\sum_{t'=0}^{t-1} \gamma^{t'} r(s_{t'}, a_{t'})\right]\right\} \\
&\quad + \mathbb{E}_{\mathcal{P}(\boldsymbol{s}_0)\left[\prod_{t'=0}^{t-1} \mathcal{P}(\boldsymbol{s}_{t'+1} \,|\, \boldsymbol{s}_{t'}, a_{t'})\mathrm{Cat}(a_{t'}; \sigma(\boldsymbol{\phi}_{t'}))\right]}\left\{\mathbb{E}_{a_t \sim \mathrm{Cat}(\sigma(\boldsymbol{\phi}_t))}\left[\gamma^t Q(\boldsymbol{s}_t, a_t)\right]\right\},
\end{aligned} \tag{A.21}$$

where $Q(\boldsymbol{s}_t, a_t)$ is the discounted action-value function defined as

$$Q(\boldsymbol{s}_t, a_t) := \mathbb{E}_{\prod_{t'=t}^{\infty} \mathrm{Cat}(a_{t'+1}; \sigma(\boldsymbol{\phi}_{t'+1}))\mathcal{P}(\boldsymbol{s}_{t'+1} \,|\, \boldsymbol{s}_{t'}, a_{t'})}\left[\sum_{t'=t}^{\infty} \gamma^{t'-t} r(\boldsymbol{s}_{t'}, a_{t'})\right].$$

The first summation term in (A.21) can be ignored for computing $\nabla_{\boldsymbol{\phi}_t} J(\boldsymbol{\phi}_{0:\infty})$, and the second one can be re-expressed as

$$\mathbb{E}_{\mathcal{P}(\boldsymbol{s}_t \,|\, \boldsymbol{s}_0, \pi_{\boldsymbol{\theta}})\mathcal{P}(\boldsymbol{s}_0)}\left\{\mathbb{E}_{a_t \sim \mathrm{Cat}(\sigma(\boldsymbol{\phi}_t))}\left[\gamma^t Q(\boldsymbol{s}_t, a_t)\right]\right\}, \tag{A.22}$$

where $\mathcal{P}(\boldsymbol{s}_t \,|\, \boldsymbol{s}_0, \pi_{\boldsymbol{\theta}})$ is the marginal form of the joint distribution $\prod_{t'=0}^{t-1} \mathcal{P}(\boldsymbol{s}_{t'+1} \,|\, \boldsymbol{s}_{t'}, a_{t'}) \text{Cat}(a_{t'}; \sigma(\boldsymbol{\phi}_{t'}))$. Applying Theorem 2 to (A.22), we have

$$\begin{aligned}
\nabla_{\boldsymbol{\phi}_{tc}} J(\boldsymbol{\phi}_{0:\infty}) &= \mathbb{E}_{\mathcal{P}(\boldsymbol{s}_t \,|\, \boldsymbol{s}_0, \pi_{\boldsymbol{\theta}})\mathcal{P}(\boldsymbol{s}_0)} \left\{ \gamma^t \nabla_{\boldsymbol{\phi}_{tc}} \mathbb{E}_{a_t \sim \text{Cat}(\sigma(\boldsymbol{\phi}_t))} \left[Q(\boldsymbol{s}_t, a_t) \right] \right\} \\
&= \mathbb{E}_{\mathcal{P}(\boldsymbol{s}_t \,|\, \boldsymbol{s}_0, \pi_{\boldsymbol{\theta}})\mathcal{P}(\boldsymbol{s}_0)} \left\{ \gamma^t \mathbb{E}_{\boldsymbol{\varpi}_t \sim \text{Dir}(\mathbf{1}_C)} \left[g_{tc} \right] \right\}, \qquad \text{(A.23)}
\end{aligned}$$

where

$$\begin{aligned}
g_{tc} &:= \sum_{j=1}^{C} f_{t\Delta}^{c \leftrightharpoons j}(\boldsymbol{\varpi}_t) \left(\frac{1}{C} - \varpi_{tj} \right), \\
f_{t\Delta}^{c \leftrightharpoons j}(\boldsymbol{\varpi}_t) &:= Q(s_t, a_t^{c \leftrightharpoons j}) - \frac{1}{C} \sum_{m=1}^{C} Q(s_t, a_t^{m \leftrightharpoons j}), \\
a_t^{c \leftrightharpoons j} &:= \arg\min_{i \in \{1, \dots, C\}} \varpi_{ti}^{c \leftrightharpoons j} e^{-\phi_{ti}}.
\end{aligned}$$

Applying the chain rule, we obtain the gradient as

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &= \sum_{t=0}^{\infty} \sum_{c=1}^{C} \frac{\partial J(\boldsymbol{\phi}_{0:\infty})}{\partial \phi_{tc}} \frac{\partial \phi_{tc}}{\partial \boldsymbol{\theta}} \\
&= \sum_{t=0}^{\infty} \sum_{c=1}^{C} \mathbb{E}_{\mathcal{P}(\boldsymbol{s}_0)\mathcal{P}(\boldsymbol{s}_t \,|\, \boldsymbol{s}_0, \pi_{\boldsymbol{\theta}})} \left\{ \gamma^t \mathbb{E}_{\boldsymbol{\varpi}_t \sim \text{Dir}(\mathbf{1}_C)} \left[g_{tc} \right] \nabla_{\boldsymbol{\theta}} \phi_{tc} \right\} \\
&= \sum_{t=0}^{\infty} \mathbb{E}_{\mathcal{P}(\boldsymbol{s}_0)\mathcal{P}(\boldsymbol{s}_t \,|\, \boldsymbol{s}_0, \pi_{\boldsymbol{\theta}})} \left\{ \gamma^t \mathbb{E}_{\boldsymbol{\varpi}_t \sim \text{Dir}(\mathbf{1}_C)} \left[\nabla_{\boldsymbol{\theta}} \sum_{c=1}^{C} g_{tc} \phi_{tc} \right] \right\} \\
&= \mathbb{E}_{\boldsymbol{s}_t \sim \rho_\pi(\boldsymbol{s})} \left\{ \mathbb{E}_{\boldsymbol{\varpi}_t \sim \text{Dir}(\mathbf{1}_C)} \left[\nabla_{\boldsymbol{\theta}} \sum_{c=1}^{C} g_{tc} \phi_{tc} \right] \right\}, \qquad \text{(A.24)}
\end{aligned}$$

where $\rho_\pi(\boldsymbol{s}) := \sum_{t=0}^{\infty} \gamma^t \mathcal{P}(\boldsymbol{s}_t = \boldsymbol{s} \,|\, \boldsymbol{s}_0, \pi_{\boldsymbol{\theta}})$ is the unnormalized discounted state visitation frequency. This concludes the proof of the ARSM policy gradient estimator. The proof of the ARS policy gradient estimator can be similarly derived, omitted here for brevity. □

A.4 Additional Figures and Tables

Table A.1: The constructions of variational auto-encoders. The following symbols "→", "]",)", and "⇝" represent deterministic linear transform, leaky rectified linear units (LeakyReLU) [Maas et al., 2013] nonlinear activation, softmax nonlinear activation, and discrete stochastic activation, respectively, in the encoder; their reversed versions are used in the decoder.

	One layer	Two layers
Encoder	784→512]→256]→200)⇝200	784→512]→256]→200)⇝200 → 200) ⇝200
Decoder	784⇜(784←[512←[256←200	784⇜(784←[512←[256←200 ⇜ (200 ← 200

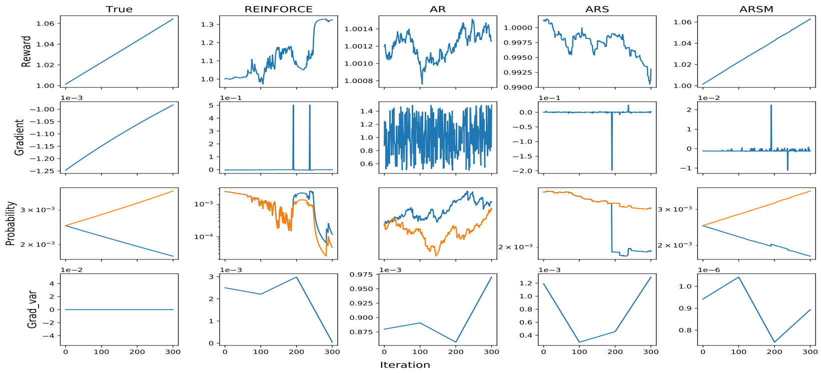


Figure A.1: Analogous plots to these in Figure 2.1, obtained with $C = 1,000$.

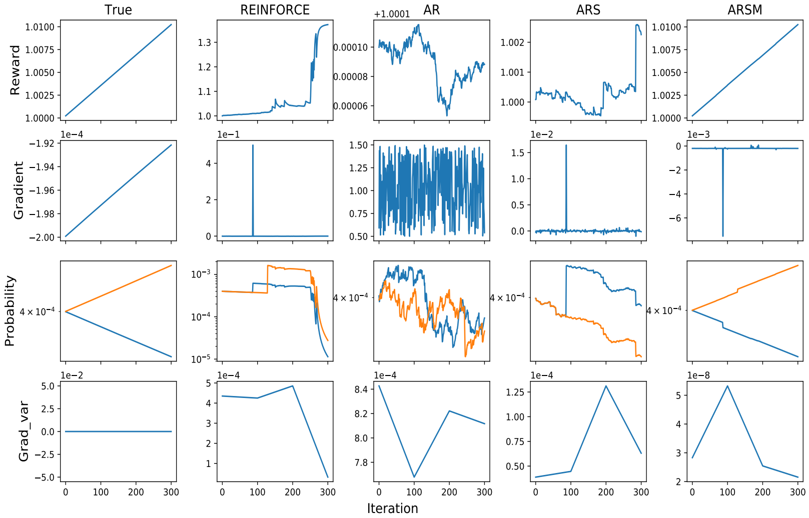


Figure A.2: Analogous plots to these in Figure 2.1, obtained with $C = 10,000$.

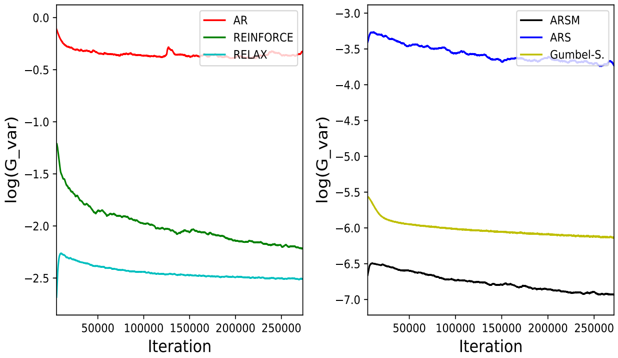
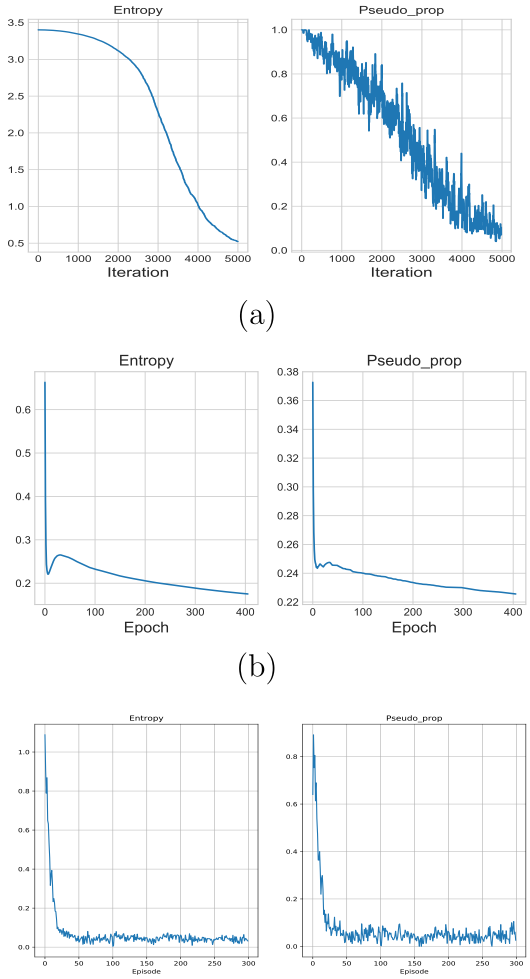


Figure A.3: Trace plots of the log variance of the gradient estimators for categorical VAE on MNIST. The variance is estimated by exponential moving averages of the first and second moments with a decay factor of 0.999. The variance is averaged over all elements of the gradient vector.



(a)

(b)

(c)

Figure A.4: The entropy of latent categorical distributions and the number of distinct pseudo actions, which differ from their corresponding true actions, both decrease as the training progresses. We plot the average entropy for $\{z_{tk}\}$ for all $t = 1 : T$ and $k = 1 : K$. The pseudo action proportion for the k-th categorical random variable at the t-th stochastic layer is calculated as the number of unique values in $\{z_{tk}^{c \rightleftharpoons j}\}_{c=1:C, j=1:C} \backslash z_{tk}$ divided by $C-1$, the maximum number of distinct pseudo actions that differ from the true action z_{tk}. We plot the average pseudo action proportion for $\{z_{tk}\}$ for all $t = 1 : T$ and $k = 1 : K$. Subplots (a), (b), and (c) correspond to the Toy data ($T = K = 1$, $C = 30$), VAE with a single stochastic layer ($T = 1$, $K = 20$, $C = 10$), and Acrobot RL task ($0 \leq T \leq 500$, $K = 1$, $C = 3$); other settings yield similar trace plots.

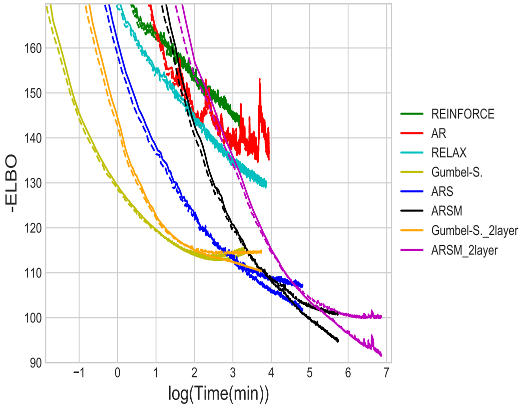


Figure A.5: Plots of −ELBOs (nats) on binarized MNIST against wall clock times on NVIDIA Tesla V100 GPU (analogous ones against training iterations are shown in Figure 2.2). The solid and dash lines correspond to the training and testing respectively (best viewed in color).

A.5 Algorithm

Algorithm 2 ARS/ARSM gradient for K-dimensional C-way categorical vector $\boldsymbol{z} = (z_1, \cdots, z_K)$, where $z_k \in \{1, \dots, C\}$.

```
input  : Reward function $f(\boldsymbol{z};\boldsymbol{\theta})$ parameterized by $\boldsymbol{\theta}$;
output : Distribution parameter $\boldsymbol{\Phi} = (\boldsymbol{\phi}_1, \cdots, \boldsymbol{\phi}_K) \in \mathbb{R}^{C\times K}$ and reward function parameter $\boldsymbol{\theta}$ that maximize the expected reward as $\mathcal{E}(\boldsymbol{\Phi},\boldsymbol{\theta}) := \mathbb{E}_{\boldsymbol{z}\sim\prod_{k=1}^K \mathrm{Cat}(z_k;\sigma(\boldsymbol{\phi}_k))}[f(\boldsymbol{z};\boldsymbol{\theta})]$;
Initialize $\boldsymbol{\Phi}$ and $\boldsymbol{\theta}$ randomly;
while not converged do
    Sample $\boldsymbol{\pi}_k \sim \mathrm{Dirichlet}(\mathbf{1}_C)$ for $k = 1, \dots, K$;
    Let $z_k = \arg\min_{i\in\{1,\dots,C\}}(\ln \pi_{ki} - \phi_{ki})$ for $k = 1, \dots, K$ to obtain the true action vector $\boldsymbol{z} = (z_1, \dots, z_k)$;
    if Using the ARS estimator then
        Using a single reference vector $\boldsymbol{j} = (j_1, \dots, j_K)$ for the variable-swapping operations, where all $j_k$ are uniformly at random selected from $\{1, \dots, C\}$;
        for $c = 1, \dots, C$ (in parallel) do
            Let $z_k^{c\leftrightharpoons j_k} = \arg\min_{i\in\{1,\dots,C\}}(\ln \pi_{ki}^{c\leftrightharpoons j_k} - \phi_{ki})$ for $k = 1, \dots, K$;
            Denote $\boldsymbol{z}^{c\leftrightharpoons \boldsymbol{j}} = (z_1^{c\leftrightharpoons j_1}, \dots, z_K^{c\leftrightharpoons j_K})$ as the $c$th pseudo action vector;
        end
        Let $\bar{f} = \frac{1}{C}\sum_{c=1}^{C} f(\boldsymbol{z}^{c\leftrightharpoons \boldsymbol{j}})$
        Let $g_{\phi_{kc}} = \big(f(\boldsymbol{z}^{c\leftrightharpoons \boldsymbol{j}}) - \bar{f}\big)(1 - C\pi_{kj_k})$ for all $(k,c) \in \{(k,c)\}_{k=1:K,\ c=1:C}$;
    end
    if Using the ARSM estimator then
        Initialize the diagonal of reward matrix $F \in \mathbb{R}^{C\times C}$ with $f(\boldsymbol{z})$, which means letting $F_{cc} = f(\boldsymbol{z})$ for $c = 1, \dots, C$;
        for $(c,j) \in \{(c,j)\}_{c=1:C,\ j<c}$ (in parallel) do
            Let $\boldsymbol{j} = j\mathbf{1}_K$, which means $j_k \equiv j$ for all $k \in \{1, \dots, K\}$;
            Let $z_k^{c\leftrightharpoons j} = \arg\min_{i\in\{1,\dots,C\}}(\ln \pi_{ki}^{c\leftrightharpoons j} - \phi_{ki})$ for $t = 1, \dots, K$;
            Denote $\boldsymbol{z}^{c\leftrightharpoons \boldsymbol{j}} = (z_1^{c\leftrightharpoons j}, \dots, z_K^{c\leftrightharpoons j})$ as the $(c,j)$th pseudo action vector;
            Let $F_{cj} = F_{jc} = f(\boldsymbol{z}^{c\leftrightharpoons \boldsymbol{j}})$;
        end
        Let $\bar{F}_{\cdot j} = \frac{1}{C}\sum_{c=1}^{C} F_{cj}$ for $j = 1, \dots, C$;
        Let $g_{\phi_{kc}} = \sum_{j=1}^{C}(F_{cj} - \bar{F}_{\cdot j})(\frac{1}{C} - \pi_{kj})$ for all $(t,c) \in \{(t,c)\}_{k=1:K,\ c=1:C}$;
    end
    $\boldsymbol{\Phi} = \boldsymbol{\Phi} + \rho_\phi \{g_{\phi_{kc}}\}_{k=1:T,\ c=1:C}$,      with step-size $\rho_\phi$;
    $\boldsymbol{\theta} = \boldsymbol{\theta} + \eta_\theta \nabla_{\boldsymbol{\theta}} f(\boldsymbol{z};\boldsymbol{\theta})$,      with step-size $\eta_\theta$
end
*Note if the categorical distribution parameter $\boldsymbol{\Phi}$ itself is defined by neural networks with parameter $\boldsymbol{w}$, standard backpropagation can be applied to compute the gradient with $\frac{\partial\mathcal{E}(\boldsymbol{\Phi},\boldsymbol{\theta})}{\partial\boldsymbol{w}} = \frac{\partial\mathcal{E}(\boldsymbol{\Phi},\boldsymbol{\theta})}{\partial\boldsymbol{\Phi}}\frac{\partial\boldsymbol{\Phi}}{\partial\boldsymbol{w}} \approx \nabla_{\boldsymbol{w}}\big(\sum_{k=1}^{K}\sum_{c=1}^{C} g_{\phi_{kc}}\phi_{kc}\big)$.
```

Algorithm 3 ARS/ARSM gradient for T layer K-dimensional C-way categorical vector $\boldsymbol{z}_t = (z_{t1}, \cdots, z_{tK})$, where $t \in \{1, \dots, T\}$, $z_{tk} \in \{1, \dots, C\}$.

```
input  : Reward function $f(\boldsymbol{z}_{1:T}; \boldsymbol{\theta})$ parameterized by $\boldsymbol{\theta}$;
output : Distribution parameter $\boldsymbol{\Phi}_t = (\boldsymbol{\phi}_{t1}, \cdots, \boldsymbol{\phi}_{tK})' \in \mathbb{R}^{K \times C}$ and parameter $\boldsymbol{\theta}$ that maximize the expected reward as $\mathcal{E}(\boldsymbol{\Phi}_{1:T}, \boldsymbol{\theta}) := \mathbb{E}_{\boldsymbol{z} \sim q_{\boldsymbol{\Phi}_1}(\boldsymbol{z}_1 \,|\, \boldsymbol{x})[\prod_{t=1}^{T-1} q_{\boldsymbol{\Phi}_{t+1}}(\boldsymbol{z}_{t+1} \,|\, \boldsymbol{z}_t)]}[f(\boldsymbol{z}; \boldsymbol{\theta})]$; $q_{\boldsymbol{\Phi}_t}(\boldsymbol{z}_t \,|\, \boldsymbol{z}_{t-1}) = \prod_{k=1}^{K} \text{Categorical}(z_{tk} | \sigma(\boldsymbol{\phi}_{tk}(\boldsymbol{z}_{t-1})))$;
Initialize $\boldsymbol{\Phi}_{1:T}$ and $\boldsymbol{\theta}$ randomly;
while not converged do
    for $t = 1 : T$ do
        Sample $\boldsymbol{\pi}_{tk} \sim \text{Dirichlet}(\mathbf{1}_C)$ for $k = 1, \dots, K$; Let $z_{tk} = \arg\min_{i \in \{1, \dots, C\}} (\ln \pi_{tki} - \phi_{tki})$ for $k = 1, \dots, K$ to obtain the true action vector $\boldsymbol{z}_t = (z_{t1}, \dots, z_{tK})$;
        if Using the ARS estimator then
            Let $\boldsymbol{j}_t = (j_{t1}, \dots, j_{tK})$, where $j_{tk} \in \{1, \dots, C\}$ is a randomly selected reference category for dimension $k$ at layer $t$.
            for $c = 1, \dots, C$ (in parallel) do
                Let $z_{tk}^{c \leftrightharpoons j_{tk}} := \arg\min_{i \in \{1, \dots, C\}} \pi_{tki}^{c \leftrightharpoons j_{tk}} e^{-\phi_{tki}}$ for $k = 1, \dots, K$;
                Denote $\boldsymbol{z}_t^{c \leftrightharpoons \boldsymbol{j}_t} = (z_{t1}^{c \leftrightharpoons j_{t1}}, \dots, z_{tK}^{c \leftrightharpoons j_{tK}})$ as the $c$th pseudo action vector;
            end
            Let $\bar{f}_t = \frac{1}{C} \sum_{c=1}^{C} f(\boldsymbol{z}_t^{c \leftrightharpoons \boldsymbol{j}_t})$
            Let $g_{\phi_{tkc}} = \big(f(\boldsymbol{z}_t^{c \leftrightharpoons \boldsymbol{j}_t}) - \bar{f}_t\big)(1 - C\pi_{kj_{tk}})$ for all $(k, c) \in \{(k, c)\}_{k=1:K,\ c=1:C}$;
        end
        if Using the ARSM estimator then
            Let $F^{(t)} \in \mathbb{R}^{C \times C}$
            If $t > 1$, sample $\boldsymbol{z}_{1:t-1} \sim q(\boldsymbol{z}_{1:t-1} | \boldsymbol{x})$ ;
            for $(c, j) \in \{(c, j)\}_{c=1:C,\ j \leq c}$ (in parallel) do
                Let $\boldsymbol{j} = j\mathbf{1}_K$, which means $j_k \equiv j$ for all $k \in \{1, \dots, K\}$;
                Let $z_{tk}^{c \leftrightharpoons j} := \arg\min_{i \in \{1, \dots, C\}} \pi_{tki}^{c \leftrightharpoons j} e^{-\phi_{tki}}$ for all $k \in \{1, \dots, K\}$;
                Denote $\boldsymbol{z}_t^{c \leftrightharpoons j} = (z_{t1}^{c \leftrightharpoons j}, \dots, z_{tK}^{c \leftrightharpoons j})$ as the $(c, j)$th pseudo action vector;
                If $t < T$, sample $\boldsymbol{z}_{t+1:T}^{c \leftrightharpoons j} \sim q(\boldsymbol{z}_{t+1:T} | \boldsymbol{z}_t^{c \leftrightharpoons j})$;
                Let $F_{cj}^{(t)} = F_{jc}^{(t)} = f(\boldsymbol{z}_{1:t-1}, \boldsymbol{z}_{t:T}^{c \leftrightharpoons j})$;
                Let $\bar{F}_{\cdot j}^{(t)} = \frac{1}{C} \sum_{c=1}^{C} F_{cj}^{(t)}$ for $j = 1, \dots, C$;
                Let $g_{\phi_{tkc}} = \sum_{j=1}^{C} (F_{cj}^{(t)} - \bar{F}_{\cdot j}^{(t)})(\frac{1}{C} - \pi_{kj})$ for all $(k, c) \in \{(k, c)\}_{k=1:K,\ c=1:C}$;
            end
        end
        $\boldsymbol{\Phi}_t = \boldsymbol{\Phi}_t + \rho_{\boldsymbol{\Phi}_t} \{g_{\phi_{tkc}}\}_{k=1:K,\ c=1:C}$,    with step-size $\rho_{\boldsymbol{\Phi}_t}$;
    end
    $\boldsymbol{\theta} = \boldsymbol{\theta} + \eta_\theta \nabla_{\boldsymbol{\theta}} f(\boldsymbol{z}; \boldsymbol{\theta})$,    with step-size $\eta_\theta$
end
```

Algorithm 4 ARSM policy gradient for reinforcement learning with a discrete-action space of C actions.

```
input  : Maximum number of state-pseudo-action rollouts $S_{\max}$ allowed in a single iteration;
output : Optimized policy parameter $\boldsymbol{\theta}$;
while not converged do
    Given a random state $\boldsymbol{s}_0$ and environment dynamics $\mathcal{P}(\boldsymbol{s}_{t+1} \mid a_t, \boldsymbol{s}_t)$, we run an episode till its termination (or a predefined number of steps) by sampling a true-action trajectory $(a_0, \boldsymbol{s}_1, a_1, \boldsymbol{s}_2, \dots)$ given policy $\pi_{\boldsymbol{\theta}}(a_t \mid \boldsymbol{s}_t) := \mathrm{Cat}(a_t; \sigma(\boldsymbol{\phi}_t))$, $\boldsymbol{\phi}_t := \mathcal{T}_{\boldsymbol{\theta}}(\boldsymbol{s}_t)$, where we sample each $a_t$ by first sampling $(\varpi_{t1}, \dots, \varpi_{tc}) \sim \mathrm{Dir}(\mathbf{1}_C)$ and then letting $a_t = \arg\min_{i \in \{1,\dots,C\}}(\ln \varpi_{ti} - \phi_{ti})$;
    Record the termination time step of the episode as $T$, and set the rollout set as $H = []$ and $S_0 = 0$;
    for $t \in$ RandomPermute$(0, \dots, T)$ do
        Let $A_t = \{(c, j)\}_{c=1:C,\ j<c}$
        Initialize $a_t^{c \leftrightharpoons j} = a_t$ for all $c$ and $j$;
        for $(c, j) \in A_t$ (in parallel) do
            Let $a_t^{c \leftrightharpoons j} = a_t^{j \leftrightharpoons c} = \arg\min_{i \in \{1,\dots,C\}}(\ln \varpi_{ti}^{c \leftrightharpoons j} - \phi_{ti})$
        end
        Let $S_t = \mathrm{unique}(\{a_t^{c \leftrightharpoons j}\}_{c,j}) \backslash a_t$, which means $S_t$ is the set of all unique values in $\{a_t^{c \leftrightharpoons j}\}_{c,j}$ that are different from the true action $a_t$; Denote the cardinality of $S_t$ as $|S_t|$, where $0 \leq |S_t| \leq C - 1$ ;
        if $S_0 + |S_t| \leq S_{\max}$ then
            $S_0 = S_0 + |S_t|$
            Append $t$ to $H$
        else
            break
        end
    end
    for $t \in H$ (in parallel) do
        Initialize $R_{tmj} = \hat{Q}(\boldsymbol{s}_t, a_t) = \sum_{t'=t}^{T} \gamma^{t'-t} r(\boldsymbol{s}_{t'}, a_{t'})$ for all $m, j \in \{1, \dots, C\}$ for $k \in \{1, \dots, |S_t|\}$ (in parallel) do
            Let $\tilde{a}_{tk} = S_t(k)$ be the $k$th unique pseudo action at time $t$;
            Evaluate $\hat{Q}(\boldsymbol{s}_t, \tilde{a}_{tk})$, which in this paper is set as $r(\boldsymbol{s}_t, \tilde{a}_{tk}) + \gamma \sum_{t'=t+1}^{\infty} \gamma^{t'-(t+1)} r(\tilde{\boldsymbol{s}}_{t'}, \tilde{a}_{t'})$, where $(\boldsymbol{s}_t, \tilde{a}_{tk}, \tilde{\boldsymbol{s}}_{t+1}, \tilde{a}_{t+1}, \dots)$ is a state-pseudo-action rollout generated by taking pseudo action $\tilde{a}_{tk}$ at state $\boldsymbol{s}_t$ and then following the environment dynamics and policy $\pi_{\boldsymbol{\theta}}$;
            Let $R_{tmj} = \hat{Q}(\boldsymbol{s}_t, \tilde{a}_{tk})$ for all $(m, j)$ in $\{(m, j) : a_t^{m \leftrightharpoons j_t} = \tilde{a}_{tk}\}$;
        end
    end
    Esimate the ARSM policy gradient as
        $$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \approx \nabla_{\boldsymbol{\theta}} \left\{ \sum_{t \in H} \sum_{c=1}^{C} \left[ \sum_{j=1}^{C} \left( R_{tcj} - \frac{1}{C} \sum_{m=1}^{C} R_{tmj} \right) \left( \frac{1}{C} - \varpi_{tj} \right) \right] \phi_{tc} \right\},$$
    $\boldsymbol{\theta} = \boldsymbol{\theta} + \eta_\theta J(\boldsymbol{\theta})$,      with step-size $\eta_\theta$;
end
```

Appendix B

Appendix for Discrete Action On-Policy Learning with Action-Value Critic

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

Supplementary Material

B.1 Proof of Theorem 5

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)

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

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

$$\nabla_{\phi_{kc}} J(\boldsymbol{\phi}) = \mathbb{E}_{\boldsymbol{a}_{\backslash k} \sim \prod_{k' \neq k} \mathrm{Discrete}(a_{k'};\sigma(\boldsymbol{\phi}_{k'}))} \Big\{ \mathbb{E}_{\boldsymbol{\varpi}_k \sim \mathrm{Dir}(\mathbf{1}_C)} \Big[(Q([\boldsymbol{a}_{\backslash k}, a_k^{c \leftrightharpoons j}], \boldsymbol{s}) - \frac{1}{C} \sum_{m=1}^{C} Q([\boldsymbol{a}_{\backslash k}, a_k^{m \leftrightharpoons j}], \boldsymbol{s}))(1 - C\varpi_{kj}) \Big] \Big\}$$

$$= \mathbb{E}_{\boldsymbol{\varpi}_k \sim \mathrm{Dir}(\mathbf{1}_C)} \Big\{ \mathbb{E}_{\boldsymbol{a}_{\backslash k} \sim \prod_{k' \neq k} \mathrm{Discrete}(a_{k'};\sigma(\boldsymbol{\phi}_{k'}))} \Big[(Q([\boldsymbol{a}_{\backslash k}, a_k^{c \leftrightharpoons j}], \boldsymbol{s}) - \frac{1}{C} \sum_{m=1}^{C} Q([\boldsymbol{a}_{\backslash k}, a_k^{m \leftrightharpoons j}], \boldsymbol{s}))(1 - C\varpi_{kj}) \Big] \Big\} \tag{B.1}$$

$$= \mathbb{E}_{\boldsymbol{\varpi}_k \sim \mathrm{Dir}(\mathbf{1}_C)} \Big\{ \mathbb{E}_{\prod_{k' \neq k} \mathrm{Dir}(\boldsymbol{\varpi}_{k'};\mathbf{1}_C)} \Big[(Q(\boldsymbol{a}^{c \leftrightharpoons j}, \boldsymbol{s}) - \frac{1}{C} \sum_{m=1}^{C} Q(\boldsymbol{a}^{m \leftrightharpoons j}, \boldsymbol{s}))(1 - C\varpi_{kj}) \Big] \Big\}, \tag{B.2}$$

where (B.1) is derived by changing the order of two expectations and (B.2) can be derived by following the proof of Proposition 5 in Yin et al. [2019]. Therefore, if given $\boldsymbol{\varpi}_k \sim \mathrm{Dir}(\mathbf{1}_C)$, it is true that $a_k^{c \leftrightharpoons j} = a_k$ for all (c, j) pairs, then the inner expectation term in (B.1) 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 $\boldsymbol{\varpi}_k \sim \mathrm{Dir}(\mathbf{1}_C)$, there exist (c, j) that $a_k^{c \leftrightharpoons j} \neq a_k$, we can use (B.2) to provide

$$g_{kc} = \sum_{j=1}^{C} \left[Q(\boldsymbol{s}, \boldsymbol{a}^{c \leftrightharpoons j}) - \frac{1}{C} \sum_{m=1}^{C} Q(\boldsymbol{s}, \boldsymbol{a}^{m \leftrightharpoons j}) \right] \left(\frac{1}{C} - \varpi_{kj} \right) \tag{B.3}$$

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

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

$$= \mathbb{E}_{\mathcal{P}(\boldsymbol{s}_0)\left[\prod_{t'=0}^{t-1} \mathcal{P}(\boldsymbol{s}_{t'+1} \,|\, \boldsymbol{s}_{t'}, \boldsymbol{a}_{t'}) \mathrm{Cat}(\boldsymbol{a}_{t'};\sigma(\boldsymbol{\phi}_{t'}))\right]} \left\{ \mathbb{E}_{\boldsymbol{a}_t \sim \mathrm{Cat}(\sigma(\boldsymbol{\phi}_t))} \left[\sum_{t'=0}^{t-1} \gamma^{t'} r(\boldsymbol{s}_{t'}, \boldsymbol{a}_{t'}) \right] \right\}$$

$$+ \mathbb{E}_{\mathcal{P}(\boldsymbol{s}_0)\left[\prod_{t'=0}^{t-1} \mathcal{P}(\boldsymbol{s}_{t'+1} \,|\, \boldsymbol{s}_{t'}, \boldsymbol{a}_{t'}) \mathrm{Cat}(\boldsymbol{a}_{t'};\sigma(\boldsymbol{\phi}_{t'}))\right]} \left\{ \mathbb{E}_{\boldsymbol{a}_t \sim \mathrm{Cat}(\sigma(\boldsymbol{\phi}_t))} \left[\gamma^t Q(\boldsymbol{s}_t, \boldsymbol{a}_t) \right] \right\},$$

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

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

.

With the result from (B.3), the statements in Theorem5 follow.

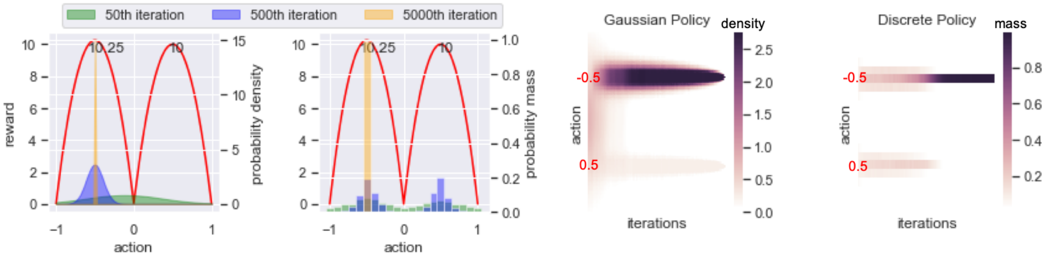


Figure B.1: **left panel**: Change of policy over iterations between Gaussian policy (left) and discrete policy (right) on toy example setting. **right panel**: Average density on each action along with the training iterations between Gaussian policy and discrete policy for 100 experiments.(The Gaussian 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.2 Experiment setup

B.2.1 Toy example setup

Assume the true reward is a bi-modal distribution (as shown in Figure B.1 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 3.4.1 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 B.1 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 3.2.

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 B.1. The *p-value* for this proportion test is 0.001056, which shows strong evidence that discrete policy outperforms Gaussian policy on this example.

B.2.2 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., 2018]; 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 β_{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 $\pi(a_k \,|\, \boldsymbol{s}, \boldsymbol{\theta}) = \sigma(\boldsymbol{\phi}_k)$,

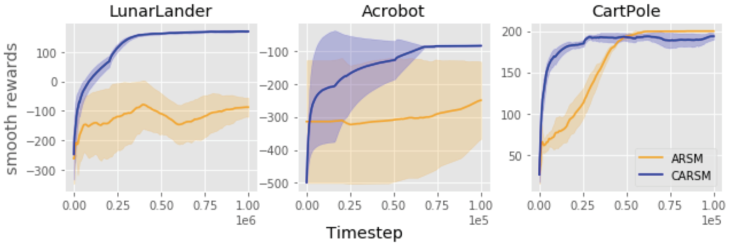


Figure B.2: Performance curves for comparison between ARSM-MC and CARSM given fix timesteps

where $\boldsymbol{\phi}_k = (\phi_{k1}, \dots, \phi_{kC})'$. 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.2.3 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 B.2

B.3 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 5 and 6, respectively.

Algorithm 5 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
```

Algorithm 6 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 $a_{tk}^{c \leftrightharpoons j} = \arg\min_{i \in \{1,\dots,C_k\}}(\ln \varpi_{tki}^{c \leftrightharpoons j} - \phi_{tki})$
            if $a_{tk}^{c \leftrightharpoons j}$ not equals to $a_{tk}$ then
                Assign $a_{tk}^{c \leftrightharpoons j}$ to $P^{tk}(c, j)$
            end
        end
    end
    Let $S_t = \mathrm{unique}(P^{t1} \otimes P^{t2} \cdots \otimes P^{tK}) \backslash \{a_{t1} \otimes a_{t2} \cdots \otimes a_{tK}\}$, 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
    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 $c$th 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
    end
end
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 $\boldsymbol{\theta}$.

Appendix C

Appendix for Implicit Distributional Reinforcement Learning

Implicit Distributional Reinforcement Learning: Appendix

C.1 Proof of Lemma 6

Denote

$$\mathcal{H} = \mathbb{E}_{\boldsymbol{a} \sim \pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s})} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s}),$$

and

$$\mathcal{H}_L = \mathbb{E}_{\boldsymbol{\xi}^{(0),\dots,(L)} \sim p(\boldsymbol{\xi})} \mathbb{E}_{\boldsymbol{a} \sim \pi_{\boldsymbol{\theta}}(\boldsymbol{a} \,|\, \boldsymbol{s}, \boldsymbol{\xi}^{(0)})} \log \frac{1}{L+1} \sum_{\ell=0}^{L} \pi_{\boldsymbol{\theta}}(\boldsymbol{a} \,|\, \boldsymbol{s}, \boldsymbol{\xi}^{(\ell)}),$$

and

$$\pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{\xi}^{(0):(L)}) = \frac{1}{L+1} \sum_{\ell=0}^{L} \pi_{\boldsymbol{\theta}}(\boldsymbol{a} \,|\, \boldsymbol{s}, \boldsymbol{\xi}^{(\ell)}).$$

Notice that $\boldsymbol{\xi}$s are from the same distribution, so we have

$$\begin{aligned}
\mathcal{H}_L &= \frac{1}{L+1} \sum_{i=0}^{L} \mathbb{E}_{\boldsymbol{\xi}^{(0),\dots,(L)} \sim p(\boldsymbol{\xi})} \mathbb{E}_{\boldsymbol{a} \sim \pi_{\boldsymbol{\theta}}(\boldsymbol{a} \,|\, \boldsymbol{s}, \boldsymbol{\xi}^{(i)})} \log \frac{1}{L+1} \sum_{\ell=0}^{L} \pi_{\boldsymbol{\theta}}(\boldsymbol{a} \,|\, \boldsymbol{s}, \boldsymbol{\xi}^{(\ell)}) \\
&= \mathbb{E}_{\boldsymbol{\xi}^{(0),\dots,(L)} \sim p(\boldsymbol{\xi})} \mathbb{E}_{\boldsymbol{a} \sim \pi_{\boldsymbol{\theta}}(\boldsymbol{a} \,|\, \boldsymbol{s}, \boldsymbol{\xi}^{(0):(L)})} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{\xi}^{(0):(L)}).
\end{aligned}$$

Use the identity that $\mathbb{E}_{\boldsymbol{a}\sim\pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s})} = \mathbb{E}_{\boldsymbol{\xi}^{(0),..,(L)}\sim p(\boldsymbol{\xi})}\mathbb{E}_{\boldsymbol{a}\sim\pi_{\boldsymbol{\theta}}(\boldsymbol{a}\,|\,\boldsymbol{s},\boldsymbol{\xi}^{(0):(L)})}$, we can rewrite $\mathcal{H}$ as

$$\mathcal{H} = \mathbb{E}_{\boldsymbol{\xi}^{(0),..,(L)}\sim p(\boldsymbol{\xi})}\mathbb{E}_{\boldsymbol{a}\sim\pi_{\boldsymbol{\theta}}(\boldsymbol{a}\,|\,\boldsymbol{s},\boldsymbol{\xi}^{(0):(L)})} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s}).$$

Therefore, we have

$$\begin{aligned}\mathcal{H}_L - \mathcal{H} &= \mathbb{E}_{\boldsymbol{\xi}^{(0),..,(L)}\sim p(\boldsymbol{\xi})}\mathbb{E}_{\boldsymbol{a}\sim\pi_{\boldsymbol{\theta}}(\boldsymbol{a}\,|\,\boldsymbol{s},\boldsymbol{\xi}^{(0):(L)})} \log \frac{\pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s},\boldsymbol{\xi}^{(0):(L)})}{\pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s})} \\ &= \mathrm{KL}(\pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s},\boldsymbol{\xi}^{(0):(L)})||\pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s})) \geq 0.\end{aligned}$$

To compare between $\mathcal{H}_L$ and $\mathcal{H}_{L+1}$, rewrite $\mathcal{H}_L$ as

$$\mathcal{H}_L = \mathbb{E}_{\boldsymbol{\xi}^{(0),..,(L),(L+1)}\sim p(\boldsymbol{\xi})}\mathbb{E}_{\boldsymbol{a}\sim\pi_{\boldsymbol{\theta}}(\boldsymbol{a}\,|\,\boldsymbol{s},\boldsymbol{\xi}^{(0):(L)})} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s},\boldsymbol{\xi}^{(0):(L)})$$

and $\mathcal{H}_{L+1}$ as

$$\begin{aligned}\mathcal{H}_{L+1} &= \mathbb{E}_{\boldsymbol{\xi}^{(0),..,(L),(L+1)}\sim p(\boldsymbol{\xi})}\mathbb{E}_{\boldsymbol{a}\sim\pi_{\boldsymbol{\theta}}(\boldsymbol{a}\,|\,\boldsymbol{s},\boldsymbol{\xi}^{(0)})} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s},\boldsymbol{\xi}^{(0):(L+1)}) \\ &= \mathbb{E}_{\boldsymbol{\xi}^{(0),..,(L),(L+1)}\sim p(\boldsymbol{\xi})}\mathbb{E}_{\boldsymbol{a}\sim\pi_{\boldsymbol{\theta}}(\boldsymbol{a}\,|\,\boldsymbol{s},\boldsymbol{\xi}^{(0):(L)})} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s},\boldsymbol{\xi}^{(0):(L+1)})\end{aligned}$$

and the difference will be

$$\begin{aligned}\mathcal{H}_L - \mathcal{H}_{L+1} &= \mathbb{E}_{\boldsymbol{\xi}^{(0),..,(L),(L+1)}\sim p(\boldsymbol{\xi})}\mathbb{E}_{\boldsymbol{a}\sim\pi_{\boldsymbol{\theta}}(\boldsymbol{a}\,|\,\boldsymbol{s},\boldsymbol{\xi}^{(0):(L)})} \big[\log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s},\boldsymbol{\xi}^{(0):(L)}) - \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s},\boldsymbol{\xi}^{(0):(L+1)})\big] \\ &= \mathbb{E}_{\boldsymbol{\xi}^{(0),..,(L),(L+1)}\sim p(\boldsymbol{\xi})}\mathrm{KL}(\pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s},\boldsymbol{\xi}^{(0):(L)})||\pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s},\boldsymbol{\xi}^{(0):(L+1)})) \geq 0.\end{aligned}$$

Finally, we arrive at the conclusion that for any ℓ, we have

$$\mathcal{H} \leq \mathcal{H}_{\ell+1} \leq \mathcal{H}_\ell.$$

C.2 Detailed pseudo code

Algorithm 7 Implicit Distributional Actor Critic (IDAC)

```
Require: Learning rate $\lambda$, batch size $M$, quantile number $K$, action number $J$ and noise number $L$, target entropy $\mathcal{H}_t$.
  Initial policy network parameter $\boldsymbol{\theta}$, action-value function network parameter $\boldsymbol{\omega}_1$,$\boldsymbol{\omega}_2$, entropy parameter $\eta$. Initial target network parameter $\tilde{\boldsymbol{\omega}}_1 = \boldsymbol{\omega}_1$,$\tilde{\boldsymbol{\omega}}_2 = \boldsymbol{\omega}_2$.
  for the number of environment steps do
      Sample $M$ number of transitions $\{\boldsymbol{s}_t^i, \boldsymbol{a}_t^i, r_t^i, \boldsymbol{s}_{t+1}^i\}_{i=1}^M$ from the replay buffer
      Sample $\boldsymbol{\epsilon}_t^{i,(k)}, \boldsymbol{\epsilon}_{t+1}^{i,(k)}, \boldsymbol{\xi}_{t+1}^{i,(\ell)}$ from $\mathcal{N}(\mathbf{0}, \mathbf{I})$ for $i = 1 \cdots M$ and $k = 1 \cdots K$ and $\ell = 0 \cdots L$.
      Sample $\boldsymbol{a}_{t+1}^i \sim \pi_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{s}_{t+1}^i, \boldsymbol{\xi}_{t+1}^{i,(0)}) = \mathcal{N}(\mathcal{T}_{\boldsymbol{\theta}}^1(\boldsymbol{s}_{t+1}^i, \boldsymbol{\xi}_{t+1}^{i,(0)}), \mathcal{T}_{\boldsymbol{\theta}}^2(\boldsymbol{s}_{t+1}^i, \boldsymbol{\xi}_{t+1}^{i,(0)}))$ for $i = 1 \cdots M$.

      Apply Bellman update to create samples (of return distribution)

          $y_{1,i,k} = r_t^i + \gamma G_{\tilde{\boldsymbol{\omega}}_1}(\boldsymbol{s}_{t+1}^i, \boldsymbol{a}_{t+1}^i, \boldsymbol{\epsilon}_{t+1}^{i,(k)})$

          $y_{2,i,k} = r_t^i + \gamma G_{\tilde{\boldsymbol{\omega}}_2}(\boldsymbol{s}_{t+1}^i, \boldsymbol{a}_{t+1}^i, \boldsymbol{\epsilon}_{t+1}^{i,(k)})$

      and let

          $(\overrightarrow{y}_{1,i,1}, \dots, \overrightarrow{y}_{1,i,K}) = \text{StopGradient}(\text{sort}(y_{1,i,1}, \dots, y_{1,i,K})$
          $(\overrightarrow{y}_{2,i,1}, \dots, \overrightarrow{y}_{2,i,K}) = \text{StopGradient}(\text{sort}(y_{2,i,1}, \dots, y_{2,i,K})$
          $\overrightarrow{y}_{i,k} = \min(\overrightarrow{y}_{1,i,k}, \overrightarrow{y}_{2,i,k}), \text{ for } i = 1 \cdots M; k = 1 \cdots K$

      Generate samples $x_{1,i,k} = G_{\boldsymbol{\omega}_1}(\boldsymbol{s}_t^i, \boldsymbol{a}_t^i, \boldsymbol{\epsilon}_t^{i,(k)})$ and $x_{2,i,k} = G_{\boldsymbol{\omega}_2}(\boldsymbol{s}_t^i, \boldsymbol{a}_t^i, \boldsymbol{\epsilon}_t^{i,(k)})$, and let

          $(\overrightarrow{x}_{1,i,1}, \dots, \overrightarrow{x}_{1,i,K}) = \text{sort}(x_{1,i,1}, \dots, x_{1,i,K})$

          $(\overrightarrow{x}_{2,i,1}, \dots, \overrightarrow{x}_{2,i,K}) = \text{sort}(x_{2,i,1}, \dots, x_{2,i,K})$

      Update action-value function parameter $\boldsymbol{\omega}_1$ and $\boldsymbol{\omega}_2$ by minimizing the quantile loss

          $J(\boldsymbol{\omega}_1, \boldsymbol{\omega}_2) = \frac{1}{M} \sum_{i=1}^{M} \frac{1}{K^2} \sum_{k=1}^{K} \sum_{k'=1}^{K} \rho_{\tau_k}^{\kappa}(\overrightarrow{y}_{i,k} - \overrightarrow{x}_{1,i,k'}) + \frac{1}{M} \sum_{i=1}^{M} \frac{1}{K^2} \sum_{k=1}^{K} \sum_{k'=1}^{K} \rho_{\tau_k}^{\kappa}(\overrightarrow{y}_{i,k} - \overrightarrow{x}_{2,i,k'}).$

      Sample $\Xi_t^{i,h}$, $\boldsymbol{\epsilon}_t^{i,(j)}$ from $\mathcal{N}(\mathbf{0}, \mathbf{I})$, for $i = 1 \cdots M, j = 1 \cdots J$ and $h = 0 \cdots L + J$, and form $\boldsymbol{\xi}_t^{i,(j,\ell)}$ from $\Xi_t^{i,h}$ by concatenating $L$ of them to the rest of $J$s. Sample $\boldsymbol{a}_t^{i,(j)} \sim \pi_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{s}_t^i, \boldsymbol{\xi}_t^{i,(j,0)}) = \mathcal{N}(\mathcal{T}_{\boldsymbol{\theta}}^1(\boldsymbol{s}_t^i, \boldsymbol{\xi}_t^{i,(j,0)}), \mathcal{T}_{\boldsymbol{\theta}}^2(\boldsymbol{s}_t^i, \boldsymbol{\xi}_t^{i,(j,0)}))$ using
          $\boldsymbol{a}_t^{i,(j)} = \mathcal{T}_{\boldsymbol{\theta}}(\boldsymbol{s}_t^i, \boldsymbol{\xi}_t^{i,(j,0)}, \boldsymbol{e}_t^i) = \mathcal{T}_{\boldsymbol{\theta}}^1(\boldsymbol{s}_t^i, \boldsymbol{\xi}_t^{i,(j,0)}) + \boldsymbol{e}_t^i \odot \mathcal{T}_{\boldsymbol{\theta}}^2(\boldsymbol{s}_t^i, \boldsymbol{\xi}_t^{i,(j,0)}),\ \boldsymbol{e}_t^i \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

      for $i = 1, \cdots, M$.
      Update the policy function parameter $\boldsymbol{\theta}$ by minimizing

          $J(\boldsymbol{\theta}) = -\frac{1}{M} \sum_{i=1}^{M} \left\{ \frac{1}{2J} \sum_{z=1}^{2} \sum_{j=1}^{J} G_{\boldsymbol{\omega}_z}(\boldsymbol{s}_t^i, \boldsymbol{a}_t^{i,(j)}, \boldsymbol{\epsilon}_t^{i,(j)}) - \exp(\eta) \sum_{j=1}^{J} \frac{1}{J} \left[ \log \frac{\sum_{\ell=0}^{L} \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t^{i,(j)} | \boldsymbol{s}_t^i, \boldsymbol{\xi}_t^{i,(j,\ell)})}{L+1} \right] \right\}.$

      We also use stop gradient on $(\mathcal{T}_{\boldsymbol{\theta}}^1(\boldsymbol{s}_t^i, \boldsymbol{\xi}_t^{i,(j)}), \mathcal{T}_{\boldsymbol{\theta}}^2(\boldsymbol{s}_t^i, \boldsymbol{\xi}_t^{i,(j)}))$ to reduce variance on gradient as mentioned in Eq (4.16).
      Update the log entropy parameter $\eta$ by minimizing

          $J(\eta) = \frac{1}{M} \sum_{i=1}^{M} [\text{StopGradient}(-\log \frac{\sum_{\ell=0}^{L} \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t^{i,(0)} | \boldsymbol{s}_t^i, \boldsymbol{\xi}_t^{i,(\ell)})}{L+1} - \mathcal{H}_t)\eta]$

  end for
```

C.3 Hyperparameters of IDAC

Parameter	Value
Optimizer	Adam
learning rate	3e-4
discount	0.99
replay buffer size	10^6
number of hidden layers (all networks)	2
number of hidden units per layer	256
number of samples per minibatch	256
entropy target	-dim($\mathcal{A}$) (e.g. , -6 for HalfCheetah-v2)
nonlinearity	ReLU
target smoothing coefficient	0.005
target update interval	1
gradient steps	1
distribution of $\boldsymbol{\xi}$	$\mathcal{N}(\mathbf{0}, \mathbf{I}_5)$
distribution of $\boldsymbol{\epsilon}$	$\mathcal{N}(\mathbf{0}, \mathbf{I}_5)$
J	51
K	51
L	21

Table C.1: IDAC hyperparameters

Bibliography

Evgeny Andriyash, Arash Vahdat, and Bill Macready. Improved gradient-based optimization over discrete distributions. *arXiv preprint arXiv:1810.00116*, 2018.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in neural information processing systems*, pages 5048–5058, 2017.

Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *arXiv preprint arXiv:1808.00177*, 2018.

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

Gabriel Barth-Maron, Matthew W Hoffman, David Budden, Will Dabney, Dan Horgan, Dhruva Tb, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. Distributed distributional deterministic policy gradients. *arXiv preprint arXiv:1804.08617*, 2018.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 449–458. JMLR. org, 2017a.

Marc G Bellemare, Ivo Danihelka, Will Dabney, Shakir Mohamed, Balaji Lakshminarayanan, Stephan Hoyer, and Rémi Munos. The Cramer distance as a solution to biased Wasserstein gradients. *arXiv preprint arXiv:1705.10743*, 2017b.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Espen Bernton, Pierre E Jacob, Mathieu Gerber, and Christian P Robert. Approximate Bayesian computation with the Wasserstein distance. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 81(2): 235–269, 2019.

Shalabh Bhatnagar, Doina Precup, David Silver, Richard S Sutton, Hamid R Maei, and Csaba Szepesvári. Convergent temporal-difference learning with arbitrary smooth function approximation. In *Advances in Neural Information Processing Systems*, pages 1204–1212, 2009.

David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, 2017.

Léon Bottou. Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*, pages 421–436. Springer, 2012.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv preprint arXiv:1606.01540*, 2016.

George Casella and Christian P Robert. Rao-Blackwellisation of sampling schemes. *Biometrika*, 83(1):81–94, 1996.

Pengyu Cheng, Chang Liu, Chunyuan Li, Dinghan Shen, Ricardo Henao, and Lawrence Carin. Straight-through estimator as projected Wasserstein gradient flow. In *NeurIPS 2018 Bayesian Deep Learning Workshop*, 2018.

Will Dabney, Georg Ostrovski, David Silver, and Remi Munos. Implicit quantile networks for distributional reinforcement learning. In *International Conference on Machine Learning*, pages 1096–1105, 2018a.

Will Dabney, Mark Rowland, Marc Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018b.

Thomas Degris, Martha White, and Richard S Sutton. Off-policy actor-critic. *arXiv preprint arXiv:1205.4839*, 2012.

Ishan Deshpande, Ziyu Zhang, and Alexander G Schwing. Generative modeling using the sliced Wasserstein distance. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3483–3491, 2018.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

Gabriel Dulac-Arnold, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679*, 2015.

Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation matters in deep RL: A case study on ppo and trpo. In *International Conference on Learning Representations*, 2019.

Xinjie Fan, Shujian Zhang, Korawat Tanwisuth, Xiaoning Qian, and Mingyuan Zhou. Contextual dropout: An efficient sample-dependent dropout module. *arXiv preprint arXiv:2103.04181*, 2021.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pages 2052–2062. PMLR, 2019.

Sheldon Goldstein, B Misra, and M Courbage. On intrinsic randomness of dynamical systems. *Journal of Statistical Physics*, 25(1):111–126, 1981.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

Will Grathwohl, Dami Choi, Yuhuai Wu, Geoff Roeder, and David Duvenaud. Backpropagation through the Void: Optimizing control variates for black-box gradient estimation. In *ICLR*, 2018.

Evan Greensmith, Peter L Bartlett, and Jonathan Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. *J. Mach. Learn. Res.*, 5(Nov):1471–1530, 2004.

Karol Gregor, Ivo Danihelka, Andriy Mnih, Charles Blundell, and Daan Wierstra. Deep autoregressive networks. In *ICML*, pages 1242–1250, 2014.

Shixiang Gu, Sergey Levine, Ilya Sutskever, and Andriy Mnih. MuProp: Unbiased backpropagation for stochastic neural networks. In *ICLR*, 2016a.

Shixiang Gu, Timothy Lillicrap, Zoubin Ghahramani, Richard E Turner, and Sergey Levine. Q-Prop: Sample-efficient policy gradient with an off-policy critic. *arXiv preprint arXiv:1611.02247*, 2016b.

Shixiang Gu, Timothy Lillicrap, Zoubin Ghahramani, Richard E Turner, and Sergey Levine. Q-Prop: Sample-efficient policy gradient with an off-policy critic. In *ICLR*, 2017.

Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1352–1361. JMLR. org, 2017.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018a.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018b.

Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. `https://github.com/hill-a/stable-baselines`, 2018.

Geoffrey Hinton. Neural networks for machine learning coursera video lectures - Geoffrey Hinton. 2012.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *arXiv preprint arXiv:1606.03476*, 2016.

Yujing Hu, Qing Da, Anxiang Zeng, Yang Yu, and Yinghui Xu. Reinforcement learning to rank in e-commerce search engine: Formalization, analysis, and application. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 368–377. ACM, 2018.

Peter J Huber. Robust estimation of a location parameter. In *Breakthroughs in statistics*, pages 492–518. Springer, 1992.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with Gumbel-softmax. In *ICLR*, 2017.

Wojciech Jaśkowski, Odd Rune Lykkebø, Nihat Engin Toklu, Florian Trifterer, Zdeněk Buk, Jan Koutník, and Faustino Gomez. Reinforcement learning to run... fast. In *The NIPS'17 Competition: Building Intelligent Systems*, pages 155–167. Springer, 2018.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Soheil Kolouri, Kimia Nadjahi, Umut Simsekli, Roland Badeau, and Gustavo Rohde. Generalized sliced Wasserstein distances. In *Advances in Neural Information Processing Systems*, pages 261–272, 2019.

Alp Kucukelbir, Dustin Tran, Rajesh Ranganath, Andrew Gelman, and David M Blei. Automatic differentiation variational inference. *Journal of Machine Learning Research*, 18(14):1–45, 2017.

Aviral Kumar, Justin Fu, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. *arXiv preprint arXiv:1906.00949*, 2019.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *arXiv preprint arXiv:2006.04779*, 2020.

Sergey Levine and Vladlen Koltun. Guided policy search. In *International conference on machine learning*, pages 1–9. PMLR, 2013.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Hao Liu, Yihao Feng, Yi Mao, Dengyong Zhou, Jian Peng, and Qiang Liu. Action-depedent control variates for policy optimization via Stein's identity. *arXiv preprint arXiv:1710.11198*, 2017a.

Hao Liu, Yihao Feng, Yi Mao, Dengyong Zhou, Jian Peng, and Qiang Liu. Action-dependent control variates for policy optimization via Stein identity. In *ICLR*, 2018.

Yang Liu, Prajit Ramachandran, Qiang Liu, and Jian Peng. Stein variational policy gradient. *arXiv preprint arXiv:1704.02399*, 2017b.

Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *ICML*, 2013.

Patrick MacAlpine and Peter Stone. UT Austin Villa: Robocup 2017 3d simulation league competition and technical challenges champions. In *Robot World Cup*, pages 473–485. Springer, 2017.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *ICLR*, 2017.

Hamid Reza Maei, Csaba Szepesvári, Shalabh Bhatnagar, and Richard S Sutton. Toward off-policy learning control with function approximation. In *ICML*, pages 719–726, 2010.

Daniel McFadden. Conditional Logit Analysis of Qualitative Choice Behavior. In P. Zarembka, editor, *Frontiers in Econometrics*, pages 105–142. Academic Press, New York, 1974.

Andriy Mnih and Karol Gregor. Neural variational inference and learning in belief networks. In *ICML*, pages 1791–1799, 2014.

Andriy Mnih and Danilo J Rezende. Variational inference for monte carlo objectives. *arXiv preprint arXiv:1602.06725*, 2016.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, pages 1928–1937, 2016.

Dmitry Molchanov, Valery Kharitonov, Artem Sobolev, and Dmitry Vetrov. Doubly semi-implicit variational inference. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 2593–2602, 2019.

Christian Naesseth, Francisco Ruiz, Scott Linderman, and David Blei. Reparameterization gradients through acceptance-rejection sampling algorithms. In *AISTATS*, pages 489–498, 2017.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

OpenAI. OpenAI Five. `https://blog.openai.com/openai-five/`, 2018.

Art B. Owen. *Monte Carlo Theory, Methods and Examples*, chapter 8 Variance Reduction. 2013.

Jakub Pachocki, Greg Brockman, Jonathan Raiman, Susan Zhang, Henrique Pondé, Jie Tang, Filip Wolski, Christy Dennison, Rafal Jozefowicz, Przemyslaw Debiak, et al. Openai five, 2018. *URL https://blog. openai. com/openai-five*, 2018.

John Paisley, David M Blei, and Michael I Jordan. Variational Bayesian inference with stochastic search. In *ICML*, pages 1363–1370, 2012.

Jan Peters and Stefan Schaal. Natural actor-critic. *Neurocomputing*, 71(7-9): 1180–1190, 2008.

Tapani Raiko, Mathias Berglund, Guillaume Alain, and Laurent Dinh. Techniques for learning binary stochastic feedforward neural networks. *arXiv preprint arXiv:1406.2989*, 2014.

Rajesh Ranganath, Sean Gerrish, and David Blei. Black box variational inference. In *AISTATS*, pages 814–822, 2014.

Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *ICML*, pages 1530–1538, 2015.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *ICML*, pages 1278–1286, 2014.

Sheldon M. Ross. *Introduction to Probability Models*. Academic Press, 10th edition, 2006.

Francisco J. R. Ruiz, Michalis K. Titsias, and David M. Blei. The generalized reparameterization gradient. In *NIPS*, pages 460–468, 2016.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015a.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015b.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. 2014.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419): 1140–1144, 2018.

Rahul Singh, Keuntaek Lee, and Yongxin Chen. Sample-based distributional policy gradient. *arXiv preprint arXiv:2001.02652*, 2020.

Hao Sun, Ziping Xu, Yuhang Song, Meng Fang, Jiechao Xiong, Bo Dai, Zhengyou Zhang, and Bolei Zhou. Zeroth-order supervised policy improvement. *arXiv preprint arXiv:2006.06600*, 2020.

Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction.* 1998.

Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, pages 1057–1063, 2000.

Yunhao Tang and Shipra Agrawal. Implicit policy for reinforcement learning. *arXiv preprint arXiv:1806.06798*, 2018.

Yunhao Tang and Shipra Agrawal. Discretizing continuous action space for on-policy optimization. *arXiv preprint arXiv:1901.10500*, 2019.

Martin A Tanner and Wing Hung Wong. The calculation of posterior distributions by data augmentation. *J. Amer. Statist. Assoc.*, 82(398):528–540, 1987.

Michalis K Titsias and Miguel Lázaro-Gredilla. Local expectation gradients for black box variational inference. In *NIPS*, pages 2638–2646. MIT Press, 2015.

Emanuel Todorov. Linearly-solvable markov decision problems. In *Advances in neural information processing systems*, pages 1369–1376, 2007.

Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

Kenneth E. Train. *Discrete Choice Methods with Simulation.* Cambridge University Press, 2nd edition, 2009.

George Tucker, Andriy Mnih, Chris J Maddison, John Lawson, and Jascha Sohl-Dickstein. Rebar: Low-variance, unbiased gradient estimates for discrete latent variable models. In *NIPS*, pages 2624–2633, 2017.

George Tucker, Surya Bhupatiraju, Shixiang Gu, Richard Turner, Zoubin Ghahramani, and Sergey Levine. The mirage of action-dependent baselines in reinforcement learning. In *ICML*, pages 5015–5024, 2018.

Aaron van den Oord, Oriol Vinyals, et al. Neural discrete representation learning. In *Advances in Neural Information Processing Systems*, pages 6306–6315, 2017.

David A Van Dyk and Xiao-Li Meng. The art of data augmentation. *Journal of Computational and Graphical Statistics*, 10(1):1–50, 2001.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double Q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

Cédric Villani. *Optimal transport: old and new*, volume 338. Springer Science & Business Media, 2008.

Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*, 2016.

Patrick Nadeem Ward, Ariella Smofsky, and Avishek Joey Bose. Improving exploration in soft-actor-critic with normalizing flows policies. *arXiv preprint arXiv:1906.02771*, 2019.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8 (3-4):279–292, 1992.

Lex Weaver and Nigel Tao. The optimal reward baseline for gradient-based reinforcement learning. In *UAI*, pages 538–545, 2001.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Reinforcement Learning*, pages 5–32. Springer, 1992a.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Reinforcement Learning*, pages 5–32. Springer, 1992b.

Cathy Wu, Aravind Rajeswaran, Yan Duan, Vikash Kumar, Alexandre M Bayen, Sham Kakade, Igor Mordatch, and Pieter Abbeel. Variance reduction for policy gradient with action-dependent factorized baselines. In *ICLR*, 2018.

Mingzhang Yin and Mingyuan Zhou. ARM: Augment-REINFORCE-merge gradient for discrete latent variable models. *Preprint*, May 2018a.

Mingzhang Yin and Mingyuan Zhou. Semi-implicit variational inference. *arXiv preprint arXiv:1805.11183*, 2018b.

Mingzhang Yin and Mingyuan Zhou. ARM: Augment-REINFORCE-merge gradient for stochastic binary networks. In *ICLR*, 2019.

Mingzhang Yin, Yuguang Yue, and Mingyuan Zhou. ARSM: Augment-REINFORCE-swap-merge estimator for gradient backpropagation through categorical variables. In *International Conference on Machine Learning*, 2019.

Yuguang Yue, Yunhao Tang, Mingzhang Yin, and Mingyuan Yin. Discrete action on-policy learning with action-value critic. In *International Conference on Artificial Intelligence and Statistics*, 2020a.

Yuguang Yue, Zhendong Wang, and Mingyuan Zhou. Implicit distributional reinforcement learning. *arXiv preprint arXiv:2007.06159*, 2020b.

Quan Zhang and Mingyuan Zhou. Nonparametric Bayesian Lomax delegate racing for survival analysis with competing risks. In *NeurIPS*, pages 5002–5013, 2018.

M. Zhou. Beta-negative binomial process and exchangeable random partitions for mixed-membership modeling. In *NIPS*, pages 3455–3463, 2014.

Mingyuan Zhou and Lawrence Carin. Negative binomial process count and mixture modeling. *arXiv preprint arXiv:1209.3442v1*, 2012.

Brian D Ziebart. Modeling purposeful adaptive behavior with the principle of maximum causal entropy. 2010.

Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *AAAI*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

Vita

Yuguang Yue was born in Beijing, China. He received his Bachelor of Science degree in mathematics from Fudan University in Shanghai, and his Master of Science degree in Biostatistics from University of California, Los Angeles. He then moved to The University of Texas at Austin to pursue a doctoral degree in Statistics. His PhD study is focused on Bayesian statistics and reinforcement learning.

Permanent address: 7205 Hart Ln
Austin, Texas 78731

This dissertation was typeset by the author.