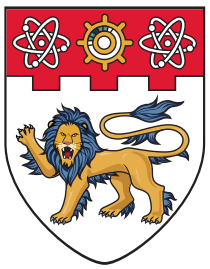
Advanced Topics in Deep Reinforcement Learning and Its Applications



NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

Chen Jianda

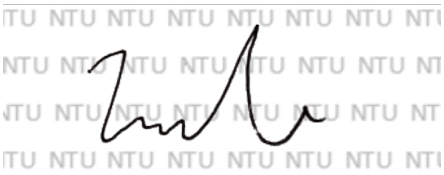
School of Computer Science and Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfilment of the requirement for the degree of
Doctor of Philosophy (Ph.D.)

2022

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.



27 Jul 2022

.....
Date

.....
Chen Jianda

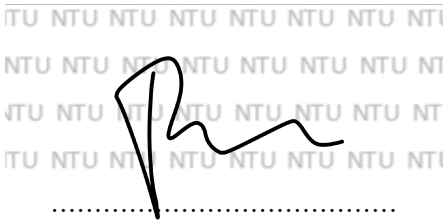
Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

27/07/2022

.....

Date



.....

Sinno Jialin Pan

Authorship Attribution Statement

This thesis contains material from two papers published and one paper under review in the following peer-reviewed conferences in which I am listed as an author.

Chapter 4 is published as **Jianda Chen, Sinno Jialin Pan, "Learning Generalizable Representations for Reinforcement Learning via Adaptive Meta-learner of Behavioral Similarities," International Conference on Learning Representations (ICLR), 2022**.

The contributions of the co-authors are as follows:

- Sinno Jialin Pan fulfilled the idea and revised the paper.

Chapter 5 is under review as **Jianda Chen, Sinno Jialin Pan, "Learning Representations via a Robust Behavioral Metric for Deep Reinforcement Learning," submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022)**.

The contributions of the co-authors are as follows:

- Sinno Jialin Pan fulfilled the idea and revised the paper.

Chapter 6 is published as **Jianda Chen, Shangyu Chen , Sinno Jialin Pan, "Storage Efficient and Dynamic Flexible Runtime Channel Pruning via Deep Reinforcement Learning," 34th Conference on Neural Information Processing Systems (NeurIPS 2020)**.

The contributions of the co-authors are as follows:

- Shangyu Chen and Sinno Jialin Pan fulfilled the idea and revised the paper.
- Shangyu Chen discussed the experiments setup.

27 Jul 2022

.....
Date

.....
Chen Jianda

Abstract

The development of reinforcement learning attracts more and more attention among researchers. Leveraging deep learning, i.e. embedding neural network function approximators, reinforcement learning is empowered to achieve great success on a broad range of tasks including video games, control, Natural Language Processing (NLP), and data center cooling. However, applications of deep reinforcement learning under a real-world setting usually suffer from an overfitting issue, in which performance would likely decrease if the environment is slightly different from the training environment. It turns out that it is crucial to improve deep reinforcement learning's generalization ability such that its techniques can be applied to real-world scenarios more effectively.

My major research work is investigating how to perform reinforcement learning on an important and widely used application, a control task that takes high-dimensional pixels as input. Images are informative, but they can also introduce noisy visual features, such as lighting or color shift in the real scene. Reinforcement learning agents which take input in the form of pixels are easily distracted by task-irrelevant features, resulting in a significant drop in performance in environments that are changed slightly from the training environment. Moreover, compared to low-dimensional inputs, learning to control from pixels suffers from sample inefficiency that requires more interactions with an environment to learn behaviors. My research works aim to improve the data efficiency for accelerating policy training and improve the generalization ability of reinforcement learning agents such that agents are able to perform consistently well across different environments even if they are unseen in training. I propose novel methods that map the high-dimensional visual observations to a low-dimensional representation space where the state abstractions are learned. Behavioral metrics, which compute state-wise similarities according to the properties of the Markov decision process (MDP), e.g. reward and

transition probability, are measured upon the representation space, in order to learn state representation to capture task-relevant features. By learning representation with a behavioral metric, the task-irrelevant features in pixels are omitted and task-specific information is captured. Therefore the generalization ability of an agent is enhanced and the data efficiency is raised.

Besides, another research work of mine is to extend applications for reinforcement learning to deep model compression. Nowadays deep neural networks are potentially overparameterized so there is a big challenge to deploy such networks on computationally constrained devices. I propose a reinforcement learning-based method for pruning convolutional neural networks (CNNs). This method combines runtime channel pruning, where pruning result depends on input data instance, and static pruning, i.e. conventional channel pruning, and develops a trade-off strategy to balance the effect of flexibility and storage efficiency.

Acknowledgments

First and most importantly, I would like to give my warmest thank to my supervisor, Prof. Sinno Jialin Pan, for offering me this opportunity to pursue a Ph.D. and for his continuous support throughout my studies and research. His insightful guidance helps me to keep moving forward on my research problems and to develop my skills of doing research. I have learned so much from him, and he is always a role model for me as a researcher.

I am grateful to have the experience working with Haiyan Yin during my first two years as she is a talented researcher and guided me to reinforcement learning. I enjoyed working with my collaborator Shangyu Chen who contributes to part of the work presented in this thesis. I am also thankful to my undergraduate advisor Prof. Long Chen and Prof. Eun-Young Kang as they initially exposed me to academic research.

I would like to extend my thanks to my friends in Nanyang Technological University, including Long-Kai Huang, Wenya Wang, Yu Chen, Jianjun Zhao, Hangwei Qian, Tianze Luo, Jie Zhang, Zichen Chen, Tianbo Li, Qiang Zhou,Xiang Li, Qiuhao Zeng, Dani Peng, Yue Deng, Aidi Liu, Quanyu Long, Ng Wen Zheng Terence, Zhanfeng Mo, Xinyi Huang, Haosen Shi, Jianhan Mei, Henghui Ding, Jun Liu, and Feixiang He, for their support and companionship throughout my studies.

I express my deepest gratitude to my mother who believes in me and supports my decisions. I love you too!

Last but not least, I would also like to extend my thanks to the world, as even small moments of joy, such as a little white cat, can fill me with love and appreciation.

Contents

List of Figures

List of Tables

List of Notation

$\mathcal{S}$	State space
$\mathcal{A}$	Action space
$\mathcal{R}$	Reward function
P	Transition distribution
γ	Discount factor
π	Policy
$\mathbf{s}$	State
$\mathbf{a}$	Action
r	Reward signal
V	State value function
Q	State-action value function
F_B^π	π-bisimulation metric transformation function
ϕ	State representation function
W	Wasserstein distance
$\hat{P}$	Predicted dynamics model
d_B^π	The unique fixed point of F_B^π
F_M^π	MICo distance transformation function
d_M^π	The unique fixed point of F_M^π
$\tilde{\mathcal{F}}_M^\pi$	Shift MICo distance transformation function
$\tilde{d}_M^\pi$	The unique fixed point of $\tilde{\mathcal{F}}_M^\pi$
F_G^π	RAP distance transformation function
d_G^π	The unique fixed point of F_G^π
$\hat{d}_G$	A distance function regarding angular on representation space
$\hat{\mu}$	Mean vector of predictive next state embedding
$\hat{\sigma}$	Standard deviation vector of predictive next state embedding
$\mathbf{F}_{in}$	In feature map of a conventional layer
$\mathbf{F}_{out}$	Output feature map of a conventional layer
$\mathbf{u}_r$	Runtime channel importance
$\mathbf{u}_s$	Static channel importance
C	Number of channels of a feature map
C_{in}	Number of channels of $\mathbf{F}_{in}$
C_{out}	Number of channels of $\mathbf{F}_{out}$

List of Publications

International Conferences

1. **Jianda Chen**, and Sinno Pan. "Learning Representations via a Robust Behavioral Metric for Deep Reinforcement Learning," in *Advances in Neural Information Processing Systems (NeurIPS-2022)*. 2022.

2. **Jianda Chen**, and Sinno Pan. "Learning Generalizable Representations for Reinforcement Learning via Adaptive Meta-learner of Behavioral Similarities," in *The 10th International Conference on Learning Representations (ICLR-2022)*. Apr. 25-29, 2022.

3. Haiyan Yin, **Jianda Chen**, Sinno Jialin Pan, and Sebastian Tschiatschek. "Sequential Generative Exploration Model for Partially Observable Reinforcement Learning," in *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI-2021)*.

4. **Jianda Chen**, Shangyu Chen, and Sinno Jialin Pan. "Storage Efficient and Dynamic Flexible Runtime Channel Pruning via Deep Reinforcement Learning," in *Advances in Neural Information Processing Systems 33 (NeurIPS-2020)*. Dec. 6-12, 2020. Pages 14747-14758. Feb. 2-9, 2021. Pages 10700-10708.

5. Long-Kai Huang, **Jianda Chen**, and Sinno Jialin Pan. "Accelerate Learning of Deep Hashing With Gradient Attention," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV-2019)*. Oct. 27 - Nov. 2, 2019. Pages 5271-5280.

6. Haiyan Yin, **Jianda Chen**, and Sinno Jialin Pan. "Hashing over Predicted Future Frames for Informed Exploration of Deep Reinforcement Learning," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI'18)*. Jul. 13-19, 2018. Pages 3026-3032

Chapter 1

Introduction

Reinforcement learning (RL) involves training an agent to sequentially interact with an environment in order to maximize rewards. The agent predicts actions based on states given by the environment. The agent receives feedback in the form of rewards or penalties, which it uses to adjust its behavior. Through this process, the agent's performance, as measured by the sum of its rewards, improves as it learns the optimal actions taken in the environment. For example, a robot trained to reach the goal in a maze might receive rewards for reaching the goal and penalties for collisions or other failures. It can use this feedback to learn the optimal path to go through the maze to reach the goal.

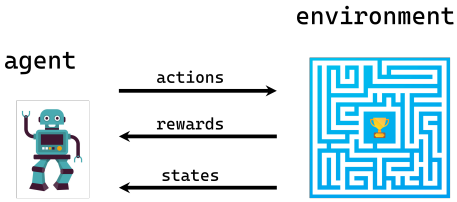


Fig. 1.1: An illustration of RL.

Deep reinforcement learning (deep RL) combines RL with deep learning, involving the training of deep neural networks as function approximators. This allows agents to learn complex behaviors by processing high-dimensional input data. There are many successes in recent years in deep RL, which becomes an emerging technique popular among researchers and industries. Deep RL confirms its powerful ability in a few real-world applications, video games [63], board games [74], controls [43], natural language processing (NLP) [81], recommendation [1], datacenters cooling [50], etc. However, deep

RL algorithms in real-world scenarios are still facing a lot of challenges, reducing the performance of deep RL in existing tasks and limiting the scope of extending applications for deep RL.

Control from pixels, which is to learn an RL agent to control with pixels input. Many interesting applications provide pixel input to RL agents, e.g., playing video games, navigating maze with a first-person camera, and robot grasping objects with a camera. How to learn RL agents to control from pixels becomes an important and attractive application in the RL community. Due to the development of computer vision (CV), images become an important data source in supervised learning meanwhile there are also increasing RL applications constructed on pixels input. As the utilization of deep learning plays a pivotal role in the advancement of computer vision, convolutional neural networks (CNNs) also enable RL algorithms to control with visual inputs. However, similar to the deep RL algorithms in low-dimensional, sample efficiency is still a critical challenge for learning policy on visual control tasks. Moreover, current studies indicate that learning policy control from pixel observations requires more training steps to converge to the same performance as directly learning from low-dimensional states.

Another crucial challenge to control from pixels is the overfitting issue such that deep RL methods perform worse in any real-world controlling scenarios where environments are smoothly changed, i.e., the testing environments are different from training environments. Pixels are high-dimensional space, which is easy to contain noise or redundant visual features that RL algorithms may capture. Such noisy features usually change among environments resulting in overfitting issues of RL agents. A key aspect of robust control in real-world visual scenarios is learning a policy that generalizes to different environments. Generalization has been widely explored in supervised learning tasks such as CV and NLP, but it has not been well-studied for applied tasks in deep RL domains and there is a lack of RL methods for general generalization in real-world scenarios.

A particle way to generalize in RL control with pixels inputs is to learn a representation function to extract generalizable and vectored latent embeddings from visual observations if the representations are able to discard the noisy details and capture the task-relevant features. With such invariant, generalizable, and low-dimensional representations, the RL algorithm is able to build upon for improving data efficiency.

The first two works in this thesis focus on improving generalization and efficiency in applications of control from pixels, and developing a deep encoder for mapping high-dimensional states to low-dimensional representations. These two works incorporate behavioral metrics that are distances between states computed according to the elements of a Markov decision process (MDP), i.e., rewards and transition probability distributions. Behavioral metrics are defined to measure the high-level similarities between states, therefore learning behavioral metrics on state representation spaces is able to capture task-relevant features and embed such features in the representation vectors. The representations learned with behavioral metrics are generalizable because the redundant features regardless of MDPs are removed from the state representations. In the first work, self-learned meta-learners for behavioral metrics are proposed to adaptively balance the effects of rewards and transition probabilities. Instead of specifying a handcrafted distance form for the behavioral metrics, e.g., the L_1 norm that destabilizes the representation learning, it proposes to use deep neural networks to approximate the distances. In the second work, an analysis of existing approximation methods of behavioral metrics is given to show the existence of approximation issues that are introduced by several potential relaxations. To address the approximation issues, a new form of the behavioral metric is proposed and a novel approximation method to such metric is developed. Besides, some theoretical guarantees, e.g., convergency and bounds, are provided to support the proposed behavioral metric.

Beyond the popular applications, how to explore more scopes of applications for deep RL is also challenging. This thesis explores applying a deep RL algorithm to model compression for large neural networks, to be specific, CNNs. In recent years, overparameterized neural networks have shown remarkable performance but they rely on huge computational and storage resources. However, the limited computation, storage, and energy on edge devices become the bottlenecks of the deployment of large neural networks. Model compression is a category of methods that speed up the inference of deployed neural networks by utilizing network pruning, quantization, network distillation, etc. Those methods compress the whole network with every layer compressed to the same compression ratio but ignore the fact that each layer may have different redundancy. Instead of manually setting the compression ratios for each layer, the last work in this

thesis proposes to determine the ratio by learning RL agents. Specifically, the work explores learning deep RL agents to determine the sparsities in a layer-wise manner for channel pruning for CNNs. It designs an algorithm that combines two types of channel pruning, runtime channel pruning, which prunes different channels on different input instances to increase flexibility, and static channel pruning, removing channels permanently to save storage space. Two deep RL agents, corresponding runtime and static channel pruning, determine the sparsity ratios for each network layer, respectively. A trade-off pruner is introduced to provide the unified pruning results of the runtime and static pruning.

1.1 Organization of the Thesis

This thesis is organized as follows.

Chapter 2 defines the concepts and notations of deep RL and provides a brief introduction to the fundamental algorithms for later chapters.

Chapter 3 reviews important algorithms of deep RL for control from pixels and deep RL for model compression.

Chapter 4 presents a meta-learner-based representation learning algorithm regarding behavioral metrics for control with pixels inputs. This method decouples the representation of each state into two different components for measuring the similarities in terms of reward and dynamics, respectively. Furthermore, it designs a learnable strategy that adaptively balances those two components to estimate the similarity between states. The learned representations are empirically verified to improve generalization for continuous action space control from pixels.

Chapter 5 analyzes the potential approximation issues in existing representation learning methods based on behavioral metric approximation. Such issues may harm the training in previous approximation methods. To address the issues, a new form of behavioral metric with a particle approximation algorithm is proposed to reduce approximation gaps. The experiments show that the proposed representation learning is robust and generalizable with environment changes in a few benchmarks.

Chapter 6 proposes a deep RL-based channel pruning framework for large CNN compression. It provides a balanced strategy for runtime channel pruning to trade-off between dynamic flexibility and storage cost.

Chapter 2

Background of Deep Reinforcement Learning

This chapter introduces the background knowledge of deep RL, including the Markov decision process (MDP), the definition of RL, and fundamental RL algorithms, especially for control tasks.

2.1 Markov Decision Process

The RL agent interacts with an environment that is formulated as an MDP, which is defined as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, P, \gamma \rangle$. $\mathcal{S}$ is the state space. $\mathcal{A}$ is the action space. $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function that outputs the reward signal after taking action $\mathbf{a} \in \mathcal{A}$ in state $\mathbf{s} \in \mathcal{S}$. $P(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ is the transition distribution that captures the probability of transitioning from the current state $\mathbf{s} \in \mathcal{S}$ to the next state $\mathbf{s}' \in \mathcal{S}$ by taking an action $\mathbf{a} \in \mathcal{A}$. $\gamma \in [0, 1)$ is the discount factor.

2.2 Reinforcement Learning

In RL, an agent sequentially interacts with an environment and collects rewards. Such an RL agent takes the input of a state $\mathbf{s}_t$ from the environment and predicts an action $\mathbf{a}_t$. The environment receives and executes the action at the current time step t, then feeds back an immediate reward signal r_t to the agent and transits to a new state $\mathbf{s}_{t+1}$. The new state will be captured by the RL agent for the next time step. The output action from the RL agent is decided by a policy $\pi : \mathcal{S} \to \mathcal{A}$ which defines an action

selection strategy. If a stochastic policy is used, then $\pi(\mathbf{a}_t|\mathbf{s}_t)$ denotes the probability of action $\mathbf{a}_t$ given current state $\mathbf{s}_t$ according to the policy. As the agent interacts with the environment, the sequence of states, actions, and rewards is generated by the MDP and the policy. The sequence of tuples denoted as $(\langle \mathbf{s}_0, \mathbf{a}_0, r_0 \rangle, \langle \mathbf{s}_1, \mathbf{a}_1, r_1 \rangle, ..., \langle \mathbf{s}_t, \mathbf{a}_t, r_t \rangle, ...)$ is called the trajectory, where the subscript t denotes the t-th time step and $r_t = \mathcal{R}(\mathbf{s}_t, \mathbf{a}_t)$ is the reward signal at time step t. The objective of reinforcement learning is to find an optimal policy denoted as π^* that maximizes the expected cumulative future rewards. The objective is defined as,

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_t \mathcal{R}(\mathbf{s}_t, \mathbf{a}_t)\right],$$

where $\mathbf{a}_t \sim \pi(\cdot|\mathbf{s}_t)$ is the action sampled from the policy by given state $\mathbf{s}_t$ at time step t.

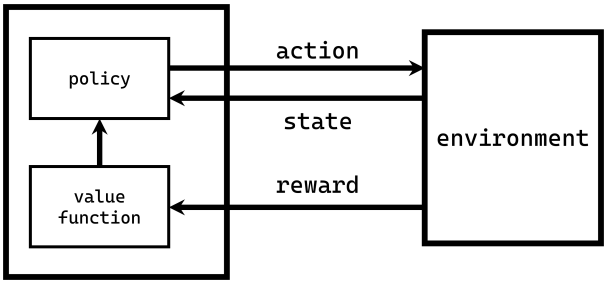


Fig. 2.1: An illustration of RL with policy and value function.

2.2.1 Value Functions

In RL, there are two functions, the state value function and the Q-function (state-action value function), which evaluate the performance of the RL agent according to the future received rewards. The state value function in RL defines the expected sum of rewards in future time steps while using policy π. $V_\pi(\mathbf{s})$ that denotes the value at state $\mathbf{s}$ is defined as,

$$V^\pi(\mathbf{s}) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid \mathbf{s}_0 = \mathbf{s}\right]. \tag{2.1}$$

The Bellman equation for $V^{\pi}(\mathbf{s})$ can be derived as,

$$\begin{aligned} V^{\pi}(\mathbf{s}) &= \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty}\gamma^{t} r_t \mid \mathbf{s}_0 = \mathbf{s}\right] \\ &= \mathbb{E}_{\pi}\left[r_t + \sum_{t=1}^{\infty}\gamma^{t} r_t \mid \mathbf{s}_0 = \mathbf{s}\right] \\ &= \sum_{\mathbf{a}}\pi(\mathbf{a}|\mathbf{s})\sum_{\mathbf{s}'}P(\mathbf{s}'|\mathbf{s},\mathbf{a})\left[\mathcal{R}(\mathbf{s},\mathbf{a}) + \gamma\mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty}\gamma^{t} r_t \mid \mathbf{s}_0 = \mathbf{s}'\right]\right] \\ &= \sum_{\mathbf{a}}\pi(\mathbf{a}|\mathbf{s})\sum_{\mathbf{s}'}P(\mathbf{s}'|\mathbf{s},\mathbf{a})\left[\mathcal{R}(\mathbf{s},\mathbf{a}) + \gamma V^{\pi}(\mathbf{s}')\right] \end{aligned} \tag{2.2}$$

Another one is called Q-function that defines the expected value given an action and a state under policy π. The Q-function denoted by Q_{π} is defined as,

$$Q^{\pi}(\mathbf{s},\mathbf{a}) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty}\gamma^{t} r_t \,|Q\ \mathbf{s}_0 = \mathbf{s}, \mathbf{a}_0 = \mathbf{a}\right]. \tag{2.3}$$

2.3 Algorithms

Value-based and policy-based approaches have been proposed to learn the policy π for RL in recent years. This section will introduce two categories of methods, value-based approaches in Section 2.3.1 and policy-based approaches in Section 2.3.2.

2.3.1 Value-based Approaches

Value-based approaches introduce an optimal Q-function $Q^{*}(\mathbf{s},\mathbf{a})$ that is defined as the maximum Q-function over policy π,

$$Q^{*}(\mathbf{s},\mathbf{a}) = \max_{\pi} Q^{\pi}(\mathbf{s},\mathbf{a}). \tag{2.4}$$

The optimal policy π^{*} in value-based approaches is greedily selecting action according to the optimal Q-value at state $\mathbf{s}$:

$$\pi^{*}(\mathbf{s}) = \arg\max_{\mathbf{a}} Q^{*}(\mathbf{s},\mathbf{a}). \tag{2.5}$$

The Bellman equation for optimal Q-function that helps to learn Q-function practically is derived as

$$Q^{*}(\mathbf{s},\mathbf{a}) = \mathbb{E}_{\mathbf{s}'}\left[\mathcal{R}(\mathbf{s},\mathbf{a}) + \gamma\max_{\mathbf{a}'} Q^{*}(\mathbf{s}',\mathbf{a}')\right]. \tag{2.6}$$

Q-learning [79] updates the optimal Q-function by using Temporal Difference (TD):

$$Q(\mathbf{s}, \mathbf{a}) = Q(\mathbf{s}, \mathbf{a}) + \alpha \left[\mathcal{R}(\mathbf{s}, \mathbf{a}) + \gamma \max_{\mathbf{a}'} Q(\mathbf{s}', \mathbf{a}') - Q(\mathbf{s}, \mathbf{a}) \right], \tag{2.7}$$

where α is the learning rate and with this update $Q(\mathbf{s}, \mathbf{a})$ will converge to optimal Q-function $Q^*(\mathbf{s}, \mathbf{a})$.

To learn the Q-function, the deep learning method, Deep Q-Network (DQN) [63], approximates the Q-function by a neural network that is parameterized by θ, denoted by $Q(\mathbf{s}, \mathbf{a}; \theta)$. Such Q-network is trained by minimizing a regression loss in an iteration manner as follows,

$$\mathcal{L}(\theta_i) = \mathbb{E}_{\mathbf{s},\mathbf{a}} \left[(r + \gamma \max_{\mathbf{a}'} Q(\mathbf{s}', \mathbf{a}'; \theta_{i-1}) - Q(\mathbf{s}, \mathbf{a}; \theta_i))^2 \right], \tag{2.8}$$

where θ_i is the parameter at the i-th iteration.

The training of DQN adopts a replay buffer. Generally, the trajectories are sampled from the environment with the current stage of the neural network, and the same trajectories as training data are used to feed and train the neural network. The high correlation of temporal states in consecutive trajectories would lead to overfitting in training the neural network and the network would not learn efficiently. To break down the temporal correlation, a replay buffer is adopted to store and resample the experience. Specifically, at each time step t, a transition of ($\mathbf{s}_t$, $\mathbf{a}_t$, $\mathbf{s}_{t+1}$, r_t) is stored in the replay buffer, where $\mathbf{s}_t$ is the observed state at time step t, $\mathbf{a}_t$ is the taken action predicted by the policy based on the DQN, r_t is the reward signal returned by environment, and $\mathbf{s}_{t+1}$ is the next state observed at time step $t + 1$. The replay buffer consists of recent transitions and removes the oldest ones if the replay buffer is full. At every training step, a batch of transitions are randomly sampled from the replay buffer and used to update the neural network.

2.3.2 Policy-based Approaches

Policy-based approaches are methods that learn policy π directly, instead of explicitly learning value functions. In deep RL, the policy is usually implemented as a neural network.

Policy gradient is a set of RL algorithms that model the action distribution of the policy and directly estimate the gradients of expected rewards with respect to the

parameter of policy. The policy is usually modeled as function $\pi_\theta(\mathbf{a}|\mathbf{s})$ parameterized by θ. With denoting $J(\theta)$ as the expected accumulated rewards, the policy gradient estimates the gradient of $J(\theta)$ w.r.t. parameter θ is

$$\nabla_\theta J(\theta) = \mathbb{E}\left[Q^\pi(\mathbf{s}, \mathbf{a}) \nabla_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s})\right] \tag{2.9}$$

Since we want to maximize the objective $J(\theta)$, the parameter is updated by gradient ascent.

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \tag{2.10}$$

where $\alpha \in \mathbb{R}^+$ is learning step size.

REINFORCE [87] estimates the Q-value $Q^\pi(\mathbf{s}, \mathbf{a})$ by Monte-Carlo methods, sampling an episode to compute the return. In expectation, the sample gradient is equal to the actual gradient,

$$\nabla_\theta J(\theta) = \mathbb{E}\left[G_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\right] \tag{2.11}$$

where G_t accumulates discounted rewards from time step t in real sampled trajectory. Because $Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = \mathbb{E}\left[G_t|\mathbf{s}_t, \mathbf{a}_t\right]$, G_t is an estimation of Q^π.

Actor-Critic methods consist of two components, policy model and value function. *Actor* provides the policy model, predicting the action given a state. The value function act as *critic*, evaluating the current policy by approximating the state value for the current state. Critic is updated by minimizing the TD error, while actor updates the policy parameter θ for $\pi_\theta(\mathbf{a}|\mathbf{s})$.

Advantage Actor-Critic [64] adopts advantage value to update policy. Advantage value is the difference between Q-value and state-value. It represents how much the specific taken action is better than the average, which is represented by state-value. The advantage value $A(\mathbf{s}, \mathbf{a})$ is formally defined as follows,

$$A(\mathbf{s}_t, \mathbf{a}_t) = Q(\mathbf{s}_t, \mathbf{a}_t) - V(\mathbf{s}_t) \tag{2.12}$$

Instead of introducing two critic networks of both Q-value and state value, the relationship between Q-value and state value can be used for decomposing Q-value. The relationship between Q-value and state value from Bellman equation:

$$Q(\mathbf{s}_t, \mathbf{a}_t) = \mathbb{E}\left[r_t + \gamma V(\mathbf{s}_{t+1})\right] \tag{2.13}$$

Therefore, the advantage can be rewritten.

$$A(\mathbf{s}_t, \mathbf{a}_t) = r_t + \gamma V(\mathbf{s}_{t+1}) - V(\mathbf{s}_t) \tag{2.14}$$

In this way, only the state value needs to be approximated by a neural network. The approximated value $V_\theta(\mathbf{s}_t)$ is trained by minimizing the TD error δ_t,

$$\delta_t = y_t - V_\theta(\mathbf{s}_t) \tag{2.15}$$

where $y_t = r_t + \gamma V(\mathbf{s}_{t+1})$ is the target at time step t. The update of policy parameter θ in advantage actor-critic method becomes,

$$\theta \leftarrow \theta + \alpha A(\mathbf{s}_t, \mathbf{a}_t) \nabla_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s}) \tag{2.16}$$

Proximal Policy Optimization(PPO) [72] constrains the step size of policy updated in policy gradient, overcoming the challenge that policy gradient is sensitive to updating step size. The idea behind PPO and a similar method TRPO [71], is to utilize *trust region*. In trust region methods, it can start from a maximum step size that it wants to explore, and then adjust the step size dynamically. PPO and TRPO limit the update of policy so that it is not too far from the previous policy via KL-divergence. Instead of imposing a hard constraint of KL-divergence, PPO formalizes the constraint as a penalty in the objective function. PPO maximizes a clipped objective which is easier to implement and tune, to achieve better performance. The clipped objective of PPO is formulated as follows,

$$\mathcal{L}^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min(\dot{r}_t(\theta)\hat{A}_t, \text{clip}(\dot{r}_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \right] \tag{2.17}$$

where $\hat{\mathbb{E}}_t$ denotes the empirical expectation over time steps, $\hat{A}_t$ is the estimated advantage at time t and ϵ is a hyper-parameter. $\dot{r}_t(\theta)$ is the ratio of the probability under the new and old policies,

$$\dot{r}_t(\theta) = \frac{\pi_\theta(\mathbf{a}|\mathbf{s})}{\pi_{\theta_{old}}(\mathbf{a}|\mathbf{s})} \tag{2.18}$$

where θ_{old} is the parameter of policy before update.

Soft Actor-Critic(SAC) [23] is an actor-critic algorithm that learns stochastic policy in an *off-policy* manner. SAC is originally proposed to solve *offline/batch* RL where

the agent only learns from a dataset and never interacts with the environment. But SAC works well in many continuous control tasks regardless of offline/online RL and becomes a strong baseline. In SAC, the stochastic policy is optimized by a DDPG-style [54] critic network that takes input of the action sampled from the policy and provides gradient backpropagated to the actor network. Another important feature of SAC is maximizing policy entropy which defines the randomness of the policy. The policy is learned to behave as a trade-off between exploitation, maximizing the expected return, and exploration, maximizing policy entropy. To be specific, the overall objective of SAC is defined as follows,

$$\mathcal{J} = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty}\gamma^{t}\Big(\mathcal{R}(\mathbf{s}_t,\mathbf{a}_t) + \alpha\mathcal{H}\left(\pi(\cdot|\mathbf{s}_t)\right)\Big)\right], \tag{2.19}$$

where $\mathcal{H}$ denotes entropy and α is the trade-off coefficient. The Bellman equation of entropy regularized Q-function in SAC can be derived as,

$$\begin{aligned} Q^{\pi}(\mathbf{s},\mathbf{a}) &= \mathbb{E}_{\mathbf{a}'\sim\pi,\mathbf{s}'\sim P}\left[\mathcal{R}(\mathbf{s},\mathbf{a}) + \gamma\Big(Q^{\pi}(\mathbf{s}',\mathbf{a}') + \alpha\mathcal{H}\left(\pi(\cdot|\mathbf{s}')\right)\Big)\right] \\ &= \mathbb{E}_{\mathbf{s}'\sim P}\Big[\mathcal{R}(\mathbf{s},\mathbf{a}) + \gamma V^{\pi}(\mathbf{s}')\Big]. \end{aligned} \tag{2.20}$$

SAC uses a parametric Q-function $Q(\mathbf{s},\mathbf{a})$ to learn the on-policy Q-function $Q^{\pi}(\mathbf{s},\mathbf{a})$. Similar to previous off-policy algorithms, e.g., DQN [63], DDPG [54], and TD3 [20], a replay buffer $\mathcal{D}$ is adopted to store transitions. While learning Q-function $Q(\mathbf{s},\mathbf{a})$, SAC samples transitions from $\mathcal{D}$ with which $Q(\mathbf{s},\mathbf{a})$ is optimized by minimizing a Bellman error,

$$\mathcal{L}(Q) = \mathop{\mathbb{E}}_{(\mathbf{s},\mathbf{a},r,\mathbf{s}')\sim\mathcal{D}}\left[\left(Q(\mathbf{s},\mathbf{a}) - \mathcal{R}(\mathbf{s},\mathbf{a}) - \gamma\bar{V}(\mathbf{s}')\right)^2\right], \tag{2.21}$$

The function $\bar{V}$ is the target value network which is learned by minimizing the following square error,

$$\mathcal{L}(\bar{V}) = \mathbb{E}_{\mathbf{s}\sim\mathcal{D}}\left[\left(\bar{V}(\mathbf{s}) - \mathbb{E}_{\mathbf{a}\sim\pi}\left[Q(\mathbf{s},\mathbf{a}) - \alpha\log\pi(\mathbf{a}|\mathbf{s})\right]\right)^2\right]. \tag{2.22}$$

The policy is evaluated in the parametric $Q(\mathbf{s},\mathbf{a})$ and improved by maximizing the predicted Q-value as well as the entropy. The update of π is maximizing the following objective,

$$\mathcal{J}(\pi) = \mathbb{E}_{\mathbf{s}\sim\mathcal{D}}\Big[\mathbb{E}_{\mathbf{a}\sim\pi}\left[Q(\mathbf{s},\mathbf{a}) - \alpha\log\pi(\mathbf{a}|\mathbf{s})\right]\Big]. \tag{2.23}$$

Note that the action $\mathbf{a}$ is on-policy sampled and the reparameterization trick is utilized in such sampling in order to backpropagate gradients to the policy network.

Chapter 3

Literature Review

The research works presented in this thesis are under the topic of applications of deep RL. Therefore, in this chapter, I review previous existing works of deep RL for control from pixels and deep RL for neural network compression.

3.1 A Review of Deep RL Control from Pixels

In the scope of deep RL, the state space can be categorized into low-dimensional or high-dimensional. If a deep RL agent takes input of low-dimensional state space, e.g. not visual input, stacked fully-connected layers are usually used for encoding the state into a learned policy, or into a latent representation space for further processing. Stacked fully-connected layers, i.e., multilayer perceptron (MLP), which are a class of neural networks that form non-linear functions, have made successful RL applications on control and robotics with sensor inputs.

For the high-dimensional state space, especially for the pixels input, CNNs are widely used to build up the neural networks for learning the policy. CNN is able to encode pixels into low-dimensional representations or combine with MLPs to directly learn policy in an end-to-end manner. The utilization of CNNs in modern RL algorithms has seen much success in playing video games, board games, navigation with visual inputs, and control from pixels.

The end-to-end manner RL algorithms, e.g., DQN [63], IMPALA [17], and SAC [23], are designed to be compatible with both low-dimensional and pixels state spaces. They use MLPs as the function approximators for low-dimensional states and CNNs for pixels

states. No matter the state space, they learn the policy by optimizing the same objectives.

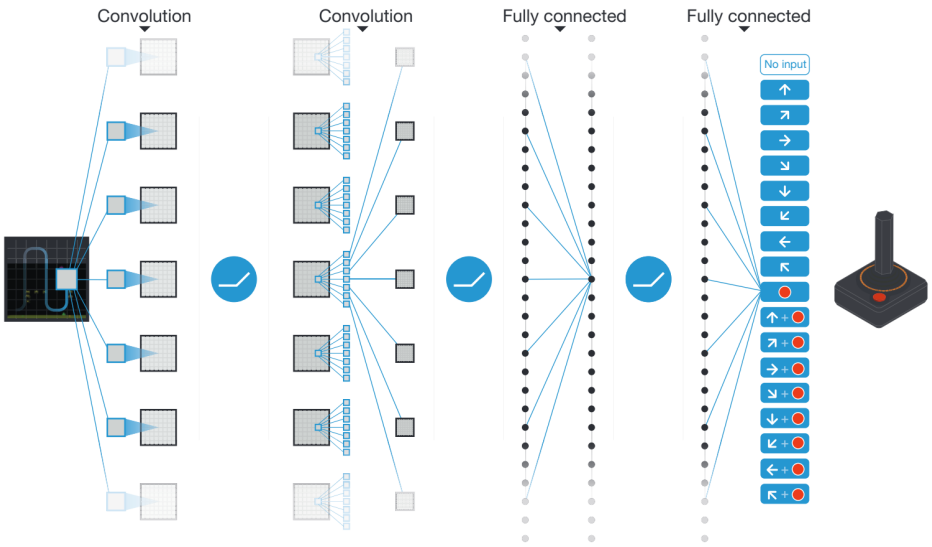


Fig. 3.1: Illustration of neural network architecture for pixels states. Figure is cited from Mnih et al. [63]

However, end-to-end control from pixels has some issues with the learning procedure. The first issue is generalization. Visual input is usually informative and hard to generalize well on some visual features, e.g., textures, backgrounds, and view-point. Such visual features are usually task-irrelevant and easy to be accidentally learned by the RL agents, leading to heavy overfitting problems when the scenes are changed from train environments even though the changes are slight. The second issue is data efficiency. If there exists a mapping in which a low-dimensional state corresponds to a pixel observation, it has been proven that control from low-dimensional states is more data-efficient than control from pixels. That is, RL agents are learning to master the tasks with fewer interactions with the environments when the states are low-dimensional.

Until now, there are many trends to address the above two issues. These approaches can be categorized into four classes: *reconstruction-based methods*, *data augmentation*, *contrastive learning*, and *behavioral metric*. Generally speaking, the reconstruction-based methods can improve data efficiency only, while data augmentation, contrastive learning, and behavioral metric methods are able to address both issues in a unified manner. Note

that recent methods may combine two of above techniques to achieve better results, so that the borders between these categories may not be clear. Most of the approaches utilize the advancement of representation learning, which encodes the pixel state to a low-dimensional representation. Additional objectives are built upon the representations in order to learn representations to satisfy some specific properties. With learned representations, conventional deep RL algorithms are applied to learn the optimal policy. Following is the review of existing methods according to the above categories.

3.1.1 Reconstruction-based Methods

Reconstruction-based methods aim to learn the representation function by autoencoder, where an encoder maps the pixel observation to a vector representation and a decoder reconstructs back to the image given the representation. Such representation is considered as containing sufficient information from the image observation because the reconstruction error is minimized, and it is also considered as a compression because the dimension is much lower than the original image.

Earlier methods learn Variational Autoencoder (VAE) [41] along with the RL algorithms. Watter et al. [85] propose to learn a VAE and learned a dynamics model on the representation space. The generative model, i.e., the decoder of VAE, reconstructs not only the current observation but also predicts the observation of the next state embedding sampled from the learned latent dynamics model. Yarats et al. [90] adopt β-VAE [34], which is a variant of VAE with regularized latent space, to learn representation with robustness to noise. Higgins et al. [35] learn β-VAE on multiple source environments and disentangle the latent space as task features and background features. With the task features, zero-shot transfer to unseen domain is able to apply to a policy that has learned with different background features.

Model-based RL algorithms learn a forward transition model while the policy can be learned by rolling out in the learned model. Hafner et al. [24] learn a latent dynamics model, which is implemented as a recurrent state space model, over deterministic representation that is learned by reconstructing MDP elements including pixel observation. The later works [25, 26] follow the major network architecture but formulate the stochastic latent space as categorical variables. Their policy learns behaviors on the learned dynamics with predictive rewards, in such a way that the sample efficiency is improved.

The ideas of reconstruction-based approaches are straightforward, however, such approaches still face some overfitting problems. The image observation usually contains task-irrelevant visual features such as backgrounds and textures. The reconstruction losses are designed to recover every pixel, including the task-irrelevant features. When their embeddings preserve background features, they are not robust to the distractors that may harmful to policy generalization. Although the VAE or β-VAE that has stochastic latent space regularizes the latent embeddings with Gaussian distribution, the stochasticity increases the difficulty of training and tuning deep networks.

3.1.2 Data Augmentation

Data augmentation has been widely used in computer vision to improve generalization for CNNs [51, 45, 15, 30, 8]. In deep RL, there are only few papers that propose to adopt data augmentation for improving generalization and data efficiency. Cobbe et al. [9] explore various augmentations in deep RL scenarios. They show that simple data augmentation cutout [12] directly applied to pixel observation is able to improve generalization. RAD [48] explores 7 different image transformations as augmentation, such as random crop, random translate, grayscale, cutout, rotate, and color jitter. It provides sufficient experiments of each kind of transformation and combination of any two. The conclusion is that random crop has the best generalization and efficiency.

More recent approaches suggest to introduce new objectives for learning with augmented data. RandConv [53] first proposes in deep RL scope to augment data by feeding the image to a randomly initialized convolutional layer. The random convolutional layer is considered as a random mapping denoted by $\hat{\mathbf{s}} = f(\mathbf{s}; \phi)$, where f is the convolutional layer, ϕ is the random initialized parameter of f, s is the pixel state and $\hat{\mathbf{s}}$ is the augmented state that has the same dimensions as the original state $\mathbf{s}$. Instead of the original state $\mathbf{s}$, the policy π takes input of the augmented state, formulated as $\pi(\mathbf{a}|f(\mathbf{s}; \phi); \theta)$ where θ is the parameter of the policy. In order to learn more invariant features from the randomized state, RandConv re-samples ϕ in each training iteration and minimizes the following loss to match the hidden features between the original state and the corresponding augmented state

$$\mathcal{L}^{FM} = \mathbb{E}_{\mathbf{s}\in\mathcal{D}} \left[|h(f(\mathbf{s}; \phi); \theta) - h(\mathbf{s}; \theta)|^2 \right], \tag{3.1}$$

where $h(\cdot|\theta)$ is the output of the last second layer of policy π, i.e., the hidden features, and $\mathcal{D}$ is the replay buffer. $\mathcal{L}^{FM}$ is optimized jointly with the RL objectives.

DrQ [89, 91] is an extension of RAD, applying image transformation to the input data when optimizing networks. However, DrQ originally proposes embedding the augmented data into the RL objectives. DrQ regularizes the Q-function by making different transformations on the same image have similar Q-values. The image transformation function is denoted by $f(\mathbf{s}, v)$, where v is the randomly sampled parameter of f. In practice, random crop is used as the transformation so that v defines the crop location. The Bellman error of the Q-function regularized by DrQ is defined as,

$$\mathcal{L}(Q) = \mathop{\mathbb{E}}_{(\mathbf{s},\mathbf{a},r,\mathbf{s}')\sim\mathcal{D}} \left[\frac{1}{N} \sum_{k=1}^{N} \left(Q(f(\mathbf{s}, v_k), \mathbf{a}) - y \right)^2 \right], \tag{3.2}$$

where N is the number of transformations that is usually 2, and y is the regression target for Q-function. The Q-values of N transformations are regressed to the same value y so that the Q-function is regularized. The target y is computed by averaging N sampled transformations on the next state,

$$y = \mathcal{R}(\mathbf{s}, \mathbf{a}) + \gamma \frac{1}{N} \sum_{k=1}^{N} Q(f(\mathbf{s}', v_k), \mathbf{a}'_k) \tag{3.3}$$

where $\mathbf{a}'_k \sim \pi(\cdot|f(\mathbf{s}', v_k))$. DrQ demonstrates its advanced performance in generalization and sample efficiency. Besides, DrQ is easy to implement in codes. These advantages make DrQ a strong baseline for RL generalization problems.

3.1.3 Contrastive Learning

Contrastive learning is a self-supervised representation learning approach that raises or discourages the similarities between embeddings of data pairs [82, 30, 8]. The similarities of data pairs that are constructed by augmentations of the same data point are enhanced, while the similarities of representations of dissimilar data pairs are minimized. Because the data augmentations are heuristic and free of domain knowledge, contrastive learning is powerful in both supervised and unsupervised settings. In deep RL, contrastive losses are used to learn state representations without knowledge of MDPs.

CURL [47] is a simple framework for contrastive learning with model-free RL algorithm. Figure 3.2 shows its overall architecture. CURL consists of two encoders that have the same network architectures but different parameters. The *query* encoder, which is demonstrated as f_{θ_q} in the left side of Figure 3.2, learns representations q optimized by passing to both RL objective and contrastive loss. The *key* encoder, namely momentum encoder, of which the parameters are the moving average of *query* encoder, predicts representations k for constructing contrastive loss. Data augmentation is also adopted in CURL. It applies random crop to the observation o and generates different cropped images o_q and o_k, which together form similar data pairs for contrastive learning. With dissimilar data points that are rather than o or cropped from o, the contrastive learning loss is,

$$\mathcal{L} = \log \frac{\exp(q^T W k_+)}{\exp(q^T W k_+) + \sum_{i=0}^{K-1} \exp(q^T W k_i)}, \tag{3.4}$$

where W is the contrastive transformation matrix, k is the similar/dissimilar label and K is the total number of data points in this iteration.

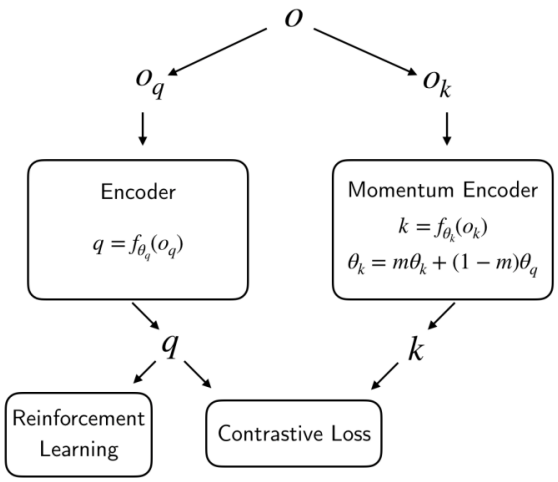


Fig. 3.2: Overall architecture of CURL. Figure is cited from Laskin et al. [47]

Schwarzer et al. [73] and Stooke et al. [77] propose methods that build upon CURL but learn forward dynamics models in representation space. They predict the representation of a state n-steps onward using learned dynamics, and contrastive loss is applied to the predicted future representation and the $t + n$ time step ground truth representation. Since the dynamics is embedded to learn, the representation can also learn the dynamics information and predict future states.

3.1.4 Behavioral Metric

The most important behavioral metric is the bisimulation metric, which in earlier research is a metric defining the similarity between states [18, 19, 4]. The bisimulation metric is defined as the difference between states with regard to immediate rewards and the 1-Wasserstein metric between dynamics models. There are several variants of bisimulation metrics proposed, we take the *on-policy* bisimulation metric, also known as the π-bisimulation metric, as an example. It is defined as

$$d(\mathbf{s}_i, \mathbf{s}_j) = |\, \mathbb{E}_{\mathbf{a}_i \sim \pi}\, r_{\mathbf{s}_i}^{\mathbf{a}_i} - \mathbb{E}_{\mathbf{a}_j \sim \pi}\, r_{\mathbf{s}_j}^{\mathbf{a}_j} \,| + \gamma W_1(d)(P_{\mathbf{s}_i}^{\pi}, P_{\mathbf{s}_j}^{\pi}), \tag{3.5}$$

where $\mathbb{E}_{\mathbf{a}_i \sim \pi}\, r_{\mathbf{s}_i}^{\mathbf{a}_i} = \mathbb{E}_{\mathbf{a}_i \sim \pi(\cdot|\mathbf{s}_i)}\, r_{\mathbf{s}_i}^{\mathbf{a}_i}$, $P_{\mathbf{s}_i}^{\pi} = \mathbb{E}_{a \sim \pi(\cdot|\mathbf{s}_i)}\, P_{\mathbf{s}_i}^{\mathbf{a}}$, $P_{\mathbf{s}_i}^{\mathbf{a}}$ is short for $P(\cdot|\mathbf{s}_i, \mathbf{a})$ and W_1 is 1-Wasserstein distance.

Later researches find that learning state representation by measuring the bisimulation metric can improve generalization against task-irrelevant distractors. Because bisimulation metric measures the rewards and dynamics differences, which are highly related to the MDP that the RL agent needs to learn, learning bisimulation metric on representation space can capture the difference between two observations.

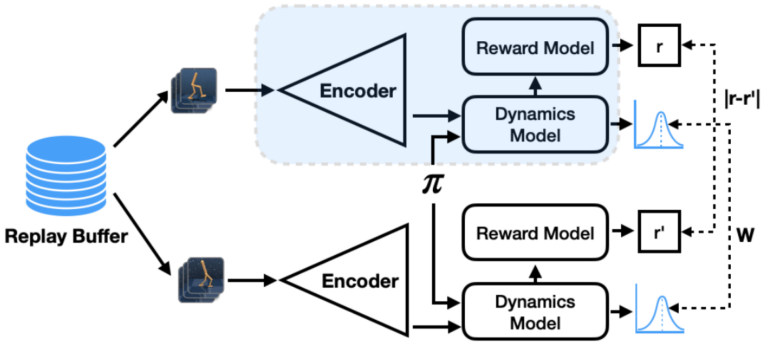


Fig. 3.3: An illustration of the architecture of DBC [94]. This figure is cited from Zhang et al. [94].

DBC [94] learns a state encoder by approximating the bisimulation metric with the L_1 distance on representation space. Since computing the 1-Wasserstein distance is intractable, DBC relaxes it to the 2-Wasserstein distance and approximates the dynamics model with parametric Gaussian distribution. DBC proposes to minimize the mean

square error between the L_1 distance on representation and the relaxed π-bisimulation metric,

$$\mathcal{L}(\phi) = \left(\|\phi(\mathbf{s}_i) - \phi(\mathbf{s}_j)\|_1 - |r_i - r_j| - \gamma W_2 \left(\hat{P}(\cdot|\phi(\mathbf{s}_i), \mathbf{a}_i), \hat{P}(\cdot|\phi(\mathbf{s}_j), \mathbf{a}_j) \right) \right)^2, \quad (3.6)$$

where ϕ is the encoder, and $\hat{P}$ is the learned dynamics model with Gaussian output. W_2 denotes the 2-Wasserstein distance which has a closed-form for Gaussian distributions: $W_2(\mathcal{N}(\mu_i, \mathbf{\Sigma}_i), \mathcal{N}(\mu_j, \mathbf{\Sigma}_j))^2 = \|\mu_i - \mu_j\|_2^2 + \|\mathbf{\Sigma}_i^{\frac{1}{2}} - \mathbf{\Sigma}_j^{\frac{1}{2}}\|_{\mathcal{F}}^2$, where $\mu_* \in \mathbb{R}^{|\mathcal{Z}_d|}$ are the mean vectors of Gaussian distribution, $\mathbf{\Sigma}_* \in \mathbb{R}^{|\mathcal{Z}_d|}$ are the diagonals of covariance matrices, and $\|\cdot\|_{\mathcal{F}}$ is Frobenius norm. DBC demonstrates the generalization ability on DeepMind Control Suite [80] with natural video backgrounds inserted [93], and also shows sample efficiency on clean backgrounds.

Agarwal et al. [2] propose another behavioral metric rather than the bisimulation metric. They replace the reward difference with a distance measuring the policy divergence. Such behavioral metric, called PSM, is defined as

$$d(\mathbf{s}_i, \mathbf{s}_j) = DIST(\pi(\cdot|\mathbf{s}_i), \pi(\cdot|\mathbf{s}_j)) + \gamma W_1(d)(P^{\pi}_{\mathbf{s}_i}, P^{\pi}_{\mathbf{s}_j}), \quad (3.7)$$

where $DIST$ is the total variation distance (TV). This metric is also learned on representation space but has several differences from DBC. The first difference is that it doesn't learn the dynamics model, instead it is derived in a recursion version under a deterministic environment setting. That is,

$$d(\mathbf{s}_i, \mathbf{s}_j) = DIST(\pi(\cdot|\mathbf{s}_i), \pi(\cdot|\mathbf{s}_j)) + \gamma d(\mathbf{s}'_i, \mathbf{s}'_j). \quad (3.8)$$

Another difference is that they do not approximate the exact metric but use SimCLR [8] for representation learning and RandConv [53] for data augmentation. These techniques make this approach able to generalize the policy to environments that have different background distractors.

MICo [5] is a behavioral distance that is estimated under sampling state pairs in a real dynamics model under a stochastic environment. The MICo distance is defined as

$$d(\mathbf{s}_i, \mathbf{s}_j) = |\, \mathbb{E}_{\mathbf{a}_i \sim \pi}\, r^{\mathbf{a}_i}_{\mathbf{s}_i} - \mathbb{E}_{\mathbf{a}_j \sim \pi}\, r^{\mathbf{a}_j}_{\mathbf{s}_j} \,| + \gamma\, \mathbb{E}_{\mathbf{s}'_i \sim P^{\pi}_{\mathbf{s}_i} \mathbf{s}'_j \sim P^{\pi}_{\mathbf{s}_j}}\, d(\mathbf{s}'_i, \mathbf{s}'_j). \quad (3.9)$$

The MICo distance is approximated in representation space, but different from DBC using the L_1 distance, it proposes an angular distance that has non-zero self-distance,

$$d(\phi(\mathbf{s}_i), \phi(\mathbf{s}_j)) = \frac{\|\phi(\mathbf{s}_i)\|_2 + \|\phi(\mathbf{s}_j)\|_2}{2} + \beta\theta(\phi(\mathbf{s}_i), \phi(\mathbf{s}_j)), \tag{3.10}$$

where $\theta(\cdot, \cdot)$ is an angle between two vectors and β is a hyperparameter. MICo is theoretically proven to be computationally efficient, and the non-zero self-distance approximation obtains higher performance in complex visual RL tasks.

The works in Chapter 4 and Chapter 5 of this thesis are highly related to above behavioral metrics approaches. However, these metrics are approximated with an explicit form of distance, e.g., the L_1 distance, which maybe too restricted in learning representations. Besides, the approximation algorithms of the above metrics may encounter some approximation issues so that the robustness of the learning process is reduced.

3.2 A Review of Deep RL for Neural Network Compression

In this section, some approaches to neural network compression and acceleration by RL are presented, and the drawbacks of these approaches will be listed. Because the work in this thesis is built upon channel pruning, which is a popular way toward model compression, I focus on pruning methods in this section.

RNP [55] proposed to train an RL agent to determine which channels of convolutional layers could be pruned. They model the pruning of convolutional layers as a layer-by-layer MDP, with each layer representing a time step. Fig. 3.4 shows the overall framework of their approach. They define state at time step i as the feature map F_i at layer i. In order to reduce computation in inference, since the feature map is too large for the RL agent, *global pooling* is adopted to efficiently reduce the shape of the feature map F_i to a vector while still preserving important information of the feature map. Then a layer-specific encoder converts the vector into a fixed-length embedding, representing the same dimension for states at various layers. Such embedding is fed into an RL agent, which consists of an RNN layer and a following fully-connected layer. The agent predicts the Q-value for each action.

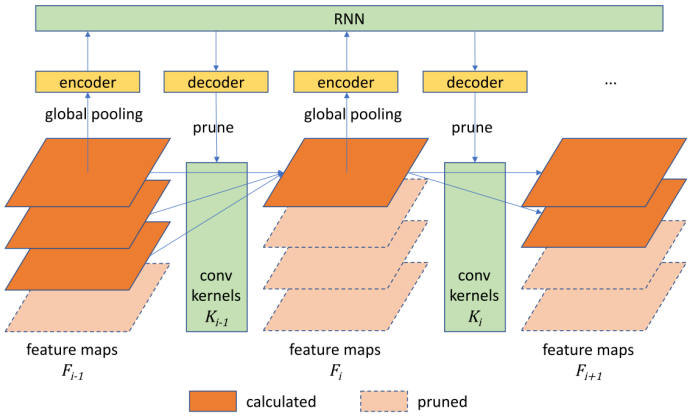


Fig. 3.4: Layer-by-layer MDP in RNP. This figure is cited from Lin et al. [55].

They group the feature maps F_i into 4 sets, denotes as F'_1, F'_2, F'_3, F'_4. With 4 sets of feature maps, they only define 4 actions a_1, a_2, a_3, a_4 for each layer. Each action a_j is defined as reserving the feature map $F'_1, ..F'_j$ and prune $F'_{j+1}, ...F'_4$. In this way, the feature maps with a larger index are more likely to be pruned.

The *reward* function in this approach is defined according to the CNN performance as well as the total computation. The reward at layer t for the taken action a_j is,

$$r_t = \begin{cases} -\alpha L_{cls} + (j-1) \times p, & t \text{ is last layer} \\ (j-1) \times p, , & \text{otherwise,} \end{cases} \tag{3.11}$$

where L_{cls} is the cross-entropy loss of the network, j is the index of the taken action, α and p are hyper-parameters.

This approach pioneered in proposing a practical dynamic channel pruning by an RL agent. However, they only use 4 actions and the action space is very rough, resulting in not achieving comparable performance with the baseline network. Besides, since it is runtime pruning, it keeps all parameters in inference and cannot reduce the storage consumption of the network.

AMC adopts deep RL for static pruning. This approach also follows the layer-by-layer MDP but with different definitions of elements in MDP. Fig. 3.5 illustrates the overview of AMC. Instead of predicting an action given the feature map, it defines a *state* as 11 features of a layer,

$$(t, n, c, h, w, stride, k, FLOPs[t], reduced, rest, a_{t-1}) \tag{3.12}$$

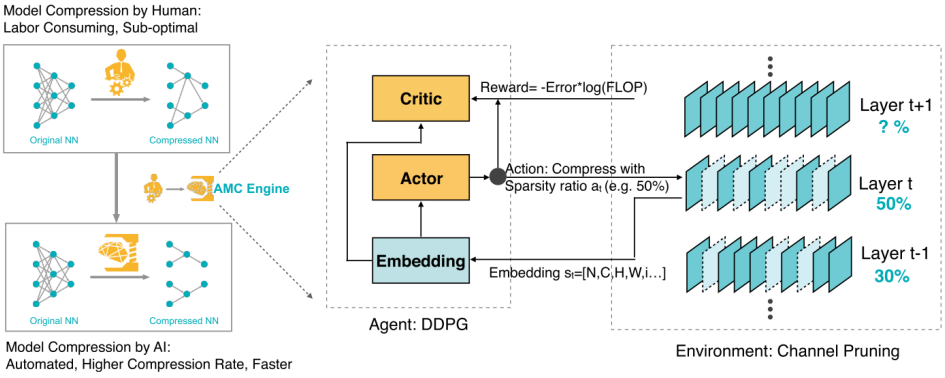


Fig. 3.5: Overview of AMC. This figure is cited from He et al. [33].

where t is the layer index, $c \times h \times w$ is the input dimension, n is the output channels, k is the kernel size, $FLOPs[t]$ is the FLOPs of layer t, *reduced* is the number of pruned layers and *rest* is the number of remaining layers. The *action* in AMC is defined as a real scalar $a_t \in (0, 1]$, representing the target sparsity of this layer after pruning. Given the sparsity a_t, AMC prunes channels with smaller norm. The *reward* is defined with the network accuracy and computation. To provide the reward signal for each episode, AMC samples data and evaluates network accuracy on the pruned network without finetuning, because they assume that higher pre-finetuning performance leads to higher finetuned accuracy. After finishing the learning of RL, AMC prunes the network by the RL predicted action and finetunes network with preserved parameters.

AMC performs static pruning with RL, successfully reducing the model size and preserving the performance compared to the baseline network. However, in the training of the RL agent, they do not consider the post-finetuned network, which provides the true rewards to learn the agent.

Chapter 4

Adaptive Meta-learner of Behavioral Similarities[1]

4.1 Control from Pixels

Control from pixels involves learning an RL agent to control with high-dimensional pixels input. Data efficiency and generalization are still the critical challenges in control from pixels. To efficiently learn a policy to control from pixels observations, prior approaches first learn an encoder to map high-dimensional visual observations to low-dimensional representations, and subsequently train a policy from low-dimensional representations to actions based on various RL algorithms. How to learn low-dimensional representations, which are able to provide semantic abstraction for high-dimensional observations, plays a key role.

Early works on deep reinforcement learning train encoders based on various reconstruction losses [46, 85, 83, 35], which aim to enforce the learned low-dimensional representations to reconstruct the original high-dimensional observations after decoding. Promising results have been achieved in some application domains, such as playing video games, simulated tasks, etc. However, the policy learned on state representations trained with a reconstruction loss may not generalize well to complex environments, which are even though semantically similar to the source environment. The reason is that a reconstruction loss is computed over all the pixels, which results in all the details of the high-dimensional images tending to be preserved in the low-dimensional representations.

[1]The work in this chapter has been published in [6].

However, some of the observed details, such as complex background objects in an image, are task-irrelevant and highly environment-dependent. Encoding such details in representation learning makes the learned representation less effective for a downstream reinforcement learning task of interest and less generalizable to new environments. To address the aforementioned issue, some data augmentation techniques have been proposed to make the learned representations more robust [89, 53, 48]. However, these approaches rarely consider the properties of a Markov Decision Process (MDP), such as conditional transition probabilities between states given an action, in the representation learning procedure.

Previous research [94, 2, 4, 18] has shown that the *bisimulation metric* and its variants are potentially effective to be exploited to learn a more generalizable reinforcement learning agent across semantically similar environments. The bisimulation metric measures the "behavioral similarity" between two states based on two terms: 1) reward difference that considers the difference in immediate task-specific reward signals between two states, and 2) dynamics difference that considers the similarity of the long-term behaviors between two states. A general idea of these approaches is to learn an encoder to map observations to a latent space such that the distance or similarity between two states in the latent space approximates their bisimulation metric. In this way, the learned representations (in the latent space) are task-relevant and invariant to environmental details.

In this thesis, Chapter 4 (this chapter) presents a meta-learner-based representation learning framework for learning behavioral metrics. Chapter 5 (the next chapter) will analyze the potential issues in behavioral metrics approximation and provide solutions to these issues along with theoretical guarantees.

4.2 Preliminaries

We define an environment as a Markov Decision Process (MDP) described by a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, \mathcal{R}, \gamma)$, where $\mathcal{S}$ is the high-dimensional state space (e.g., images), $\mathcal{A}$ is the action space, $P(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ is the transition dynamics model that captures the probability of transitioning to next state $\mathbf{s}' \in \mathcal{S}$ given current state $s \in \mathcal{S}$ and action $\mathbf{a} \in \mathcal{A}$, $\mathcal{R}$ is the reward function yielding a reward signal $r = \mathcal{R}(\mathbf{s}, \mathbf{a}) \in \mathbb{R}$, and $\gamma \in [0, 1)$ is the discounting

factor. The goal of reinforcement learning is to learn a policy $\pi(\mathbf{a}|\mathbf{s})$ that maximizes the expected cumulative rewards: $\max_\pi \mathbb{E}[\sum_t \gamma^t r(\mathbf{s}_t, \mathbf{a}_t)|\mathbf{a}_t \sim \pi(\cdot|\mathbf{s}_t), \mathbf{s}'_t \sim P(\mathbf{s}'|\mathbf{s}, \mathbf{a})]$. In the scope of this chapter, we do not consider partial observability, and use stacked consecutive frames as the fully observed states.

The Bisimulation Metric defines a pseudometric $d : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$,[2] where d quantifies the behavioral similarity of two discrete states [18]. An extension to both continuous and discrete state spaces has also been developed [19]. A variant of the bisimulation metric proposed in [4] defines a metric w.r.t. a policy π, which is known as the on-policy bisimulation metric. It removes the requirement of matching actions in the dynamics model but focuses on the policy π, which is able to better capture behavioral similarity for a specific task. Because of this property, we focus on the *π-bisimulation metric* [4]:

$$F_B^\pi(d)(\mathbf{s}_i, \mathbf{s}_j) = (1-c)|\mathcal{R}^\pi_{\mathbf{s}_i} - \mathcal{R}^\pi_{\mathbf{s}_j}| + cW_1(d)(P^\pi_{\mathbf{s}_i}, P^\pi_{\mathbf{s}_j}). \tag{4.1}$$

where $\mathcal{R}^\pi_{\mathbf{s}} := \sum_{\mathbf{a}} \pi(\mathbf{a}|\mathbf{s})\mathcal{R}(\mathbf{s}, \mathbf{a})$ and $P^\pi_{\mathbf{s}} := \sum_{\mathbf{a}} \pi(\mathbf{a}|\mathbf{s})P(\cdot|\mathbf{s}, \mathbf{a})$. The mapping $F_B^\pi : \mathbb{M} \to \mathbb{M}$, where $\mathbb{M}$ denotes the set of all pseudometrics in state space $\mathcal{S}$ and $W_1(d)(\cdot, \cdot)$ denotes the 1-Wasserstein distance given the pseudometric d. Then F_B^π has a least fixed point denoted by d_*.

Deep Bisimulation for Control (DBC) [94] is to learn latent representations such that the L_1 distances in the latent space are equal to the π-bisimulation metric at the fixed-point in the state space:

$$\arg\min_\phi \Big(\|\phi(\mathbf{s}_i) - \phi(\mathbf{s}_j)\|_1 - |\mathcal{R}^\pi_{\mathbf{s}_i} - \mathcal{R}^\pi_{\mathbf{s}_j}| - \gamma W_2(d)(P^\pi_{\mathbf{s}_i}, P^\pi_{\mathbf{s}_j}) \Big)^2,$$

where ϕ is the state encoder, and $W_2(d)(\cdot, \cdot)$ denotes the 2-Wasserstein distance. DBC is combined with the reinforcement learning algorithm SAC [23], where $\phi(\mathbf{s})$ is the input for SAC.

4.3 Introduction

There are some issues in previous representation learning approaches regarding bisimulation metric: 1) manually specifying a form of distance, e.g., the L_1 norm as used in [94],

[2] If the pseudometric d of two states is 0, then the two states belong to an equivalence relation.

in the latent space may limit the approximation precision for the bisimulation metric and potentially discard some state information that is useful for policy learning; 2) existing approaches rarely explore how to learn an adaptive combination of reward and dynamics differences in the bisimulation metric, which may vary in different tasks or environments.

We propose a novel framework for learning generalizable state representations for RL, named Adaptive Meta-learner of Behavioral Similarities (AMBS). In this framework, we design a network with two encoders that map the high-dimensional observations to two decomposed representations regarding rewards and dynamics. For the purpose of learning behavioral similarity on state representations, we introduce a pair of meta-learners that learn similarities in order to measure the reward and the dynamics similarity between two states over the corresponding decomposed state representations, respectively. The meta-learners are self-learned by approximating the reward difference and the dynamics difference in the bisimulation metric. Then the meta-learners update the state representations according to their behavioral distance to the other state representations. Previous approaches with a hand-crafted form of distance/similarity evaluating state encoding in the L_1 space are difficult to minimize the approximation error for the bisimulation metric, which may lead to important side information being discarded, e.g., information regarding to Q-value but not relevant to states distance/similarity. Instead, our learned similarities measure two states representations via a neural architecture, where side information can be preserved for policy learning. Our experiments also showed that a smaller approximation loss for similarity learning can be obtained by using the meta-learners. This demonstrates that the proposed meta-learners can overcome the approximation precision issue introduced by the L_1 distance in previous approaches and provide more stable gradients for robust learning of state representations for deep RL. Moreover, we explore the impact between the reward and the dynamics terms in the bisimulation metric. We propose a learning-based adaptive strategy to balance the effect between reward and dynamics in different tasks or environments by introducing a learnable importance parameter, which is jointly learned with the state-action value function. Finally, we use a simple but effective data augmentation strategy to accelerate the RL procedure and learn more robust state representations.

The main contributions of this work are 3-fold: 1) we propose a meta-learner-based framework to learn task-relevant and environment-details-invariant state representations;

2) we propose a network architecture that decomposes each state into two different types of representations for measuring the similarities in terms of reward and dynamics, respectively, and design a learnable adaptive strategy to balance them to estimate the bisimulation metric between states; 3) we verify our proposed framework on extensive experiments and demonstrate new state-of-the-art results on background-distraction DeepMind control suite [80, 93, 76] and other visual-based RL tasks.

4.4 Related Work

4.4.1 Representation learning for reinforcement learning from pixels

Various existing deep RL methods have been proposed to address the sample efficiency and the generalization problems for conventional RL from pixel observation. In end-to-end deep RL, neural networks learn representations implicitly by optimizing some RL objective [63, 17]. [85] and [83] proposed to learn an encoder with training a dynamics model jointly to produce observation representations. [52], [24], [25] and [95] aimed to learn an environment dynamics model with the reconstruction loss to compact pixels to latent representations. [22] proposed to learn representations by predicting the dynamics model along with the reward, and analyzed its theoretical connection to the bisimulation metric. [2] proposed to learn representations by state distances based on policy distribution. [40] modified the learning of bisimulation metric with intrinsic rewards and inverse dynamics regularization. [5] replaced the Wasserstein distance in bisimulation metric with a parametrized metric.

4.4.2 Data Augmentation in reinforcement learning

[48] and [9] explored various data augmentation techniques for deep RL, which are limited to transformations on input images. [9] applied data augmentation to domain transfer. [92] studied data augmentation on game environments for zero-shot generalization. RandConv [53] proposed a randomized convolutional neural network to generate randomized observations in order to perform data augmentation. A recent work DrQ [89] performs random crop on image observations and provides regularized formulation for updating

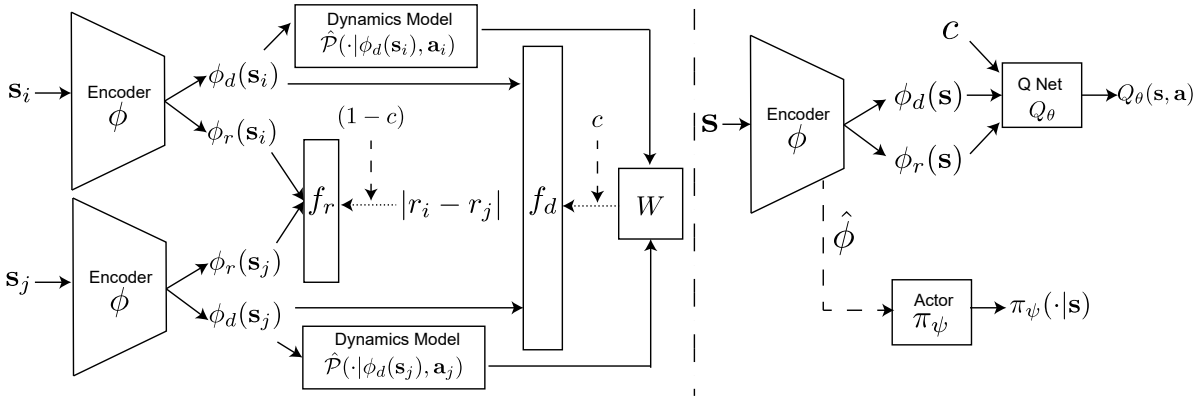


Fig. 4.1: Architecture of our AMBS framework. The dotted arrow represents the regression target and the dash arrow means stop gradient. **Left**: the learning process of meta-learner. **Right**: the model architecture for SAC with adaptive weight c which is jointly learned with SAC objective.

Q-function. [68] proposed to use data augmentation for regularizing on-policy actor-critic RL objectives. [66] proposed a counterfactual data augmentation technique by swapping observed trajectory pairs.

4.5 Adaptive Meta-learner of Behavioral Similarities

In this section, we propose a framework named Adaptive Meta-learner of Behavioral Similarities (AMBS) to learn generalizable states representation regarding the π-bisimulation metric. The learning procedure is demonstrated in Figure 4.1. Observe that the π-bisimulation metric is composed of two terms: $|\mathcal{R}^\pi_{\mathbf{s}_i} - \mathcal{R}^\pi_{\mathbf{s}_j}|$, which computes the difference of rewards between states, and $W_2(d)(P^\pi_{\mathbf{s}_i}, P^\pi_{\mathbf{s}_j})$, which computes the difference of the outputs of dynamics model between states. We propose a network architecture which contains encoders to transform the high-dimensional visual observations to two decomposed encodings regarding rewards and dynamics. We develop two meta-learners, one of which quantifies the reward difference on reward representations and the other captures dynamics distance (Section 4.5.1). Each meta-learner is self-learned to update the corresponding state representations by learning to approximate a term in the π-bisimulation metric, respectively. Rather than enforcing the L_1 distance between embedded states to be equal to the π-bisimulation metric, our meta-learners are able to use a more flexible

form of similarity, i.e., a well-designed non-linear neural network, with each similarity evaluates the reward difference or dynamics difference between states beyond the original Euclidean space. Moreover, we propose a strategy for learning to combine the outputs of the two learned similarities in a specific environment (Section 4.5.2). We introduce a learnable weight for the combination and such weight is adaptively learned together with the policy learning procedure (in this work we use SAC as the base reinforcement learning algorithm). In addition, we also use a data augmentation technique to make the learned representations more robust (Section 4.5.3). The whole learning procedure is trained in an end-to-end manner.

4.5.1 Meta-learners for Decomposed Representations

We design a network architecture with two encoders, ϕ_r and ϕ_d, to encode decomposed features from visual observations as shown in the left side of Figure 4.1. Each encoder maps a high-dimensional observation to a low-dimensional representation, i.e., $\phi_r : \mathcal{S} \to \mathcal{Z}_r$ capturing reward-relevant features and $\phi_d : \mathcal{S} \to \mathcal{Z}_d$ capturing dynamics-relevant features. We design two meta-learners, f_r and f_d, where f_r learns to measure reward difference and f_d aims to measure dynamics difference between two state representations. Specifically, each meta-learner takes two embedded states as input and outputs the corresponding similarity. The procedure is summarized as follows.

$$\mathbf{s}_i, \mathbf{s}_j \to \phi_*(\mathbf{s}_i), \phi_*(\mathbf{s}_j) \to f_*(\phi_*(\mathbf{s}_i), \phi_*(\mathbf{s}_j)), \text{ where } * \in \{r, d\}.$$

Note that the aggregation of the outputs of the two encoders ϕ_r and ϕ_d of each observation (i.e., state)is also fed into SAC to learn a policy (the right side of Figure 4.1). Each self-learned meta-learner learns to capture similarity by approximating a distance term in the π-bisimulation metric, respectively, and provides stable gradients for updating ϕ_r or ϕ_d according to the distance to the other state representations. The details of network architecture can be found in Section 4.7.4.

As discussed in Section 4.1, restricting the form of distance to be the L_1/L_2 norm in the latent space may limit the approximation precision. As shown in Figure 4.2, the L_1 and the L_2 norm for measuring the distance of two latent representations $\phi(\mathbf{s}_i)$ and

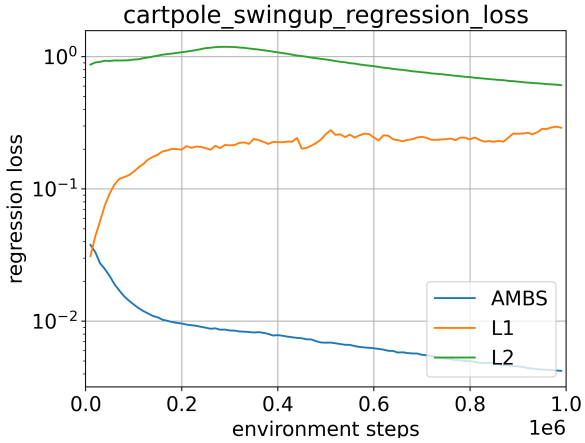


Fig. 4.2: The regression loss among different forms of distances.

$\phi(\mathbf{s}_j)$ lead to large regression losses during RL training, which destabilize the representation learning and consequently decrease final performance. Besides, such hand-crafted distances make semantically similar states encoding close to each other in terms of the L_1 norm in Euclidean space, which however may lose part of useful information, e.g., policy-relevant information but not immediately regarding to the distance.

To overcome the aforementioned limitations, we propose to exploit meta-learners f_r and f_d to learn similarities. By learning with a regression target, the similarity learned by f_* is easier to converge to the target and leads to faster descent tendency of regression loss (as shown in Figure 4.2 where y-axis is log-scale). Such a property provides more stable gradients for state representation comparing to the L_1/L_2 norm, and therefore the meta-learner f_* is able to guide the process of updating the state encoder. Besides, f_* is a non-linear transformation that evaluates the state similarity in a more flexible space instead of the original Euclidean space. Consequently, it is able to preserve more task-relevant information in state representation that is required for further SAC policy learning.

As the goal of meta-learners is to approximate different types of similarities in the π-bisimulation metric, we design the loss functions for the meta-learners using the mean squared error:

$$\ell(f_r, \phi_r) = \Big(f_r(\phi_r(\mathbf{s}_i), \phi_r(\mathbf{s}_j)) - |\mathcal{R}^\pi_{\mathbf{s}_i} - \mathcal{R}^\pi_{\mathbf{s}_j}|\Big)^2, \tag{4.2}$$

$$\ell(f_d, \phi_d) = \Big(f_d(\phi_d(\mathbf{s}_i), \phi_d(\mathbf{s}_j)) - W_2(P^\pi_{\mathbf{s}_i}, P^\pi_{\mathbf{s}_j})\Big)^2. \tag{4.3}$$

To make the resultant optimization problems compatible with SGD updates with transitions sampled from replay buffer, we replace the reward metric term $|\mathcal{R}^{\pi}_{\mathbf{s}_i} - \mathcal{R}^{\pi}_{\mathbf{s}_j}|$ by the distance of rewards in the sampled transitions in (4.2), and use a learned parametric dynamics model $\hat{P}$ to approximate the true dynamics model $P^{\pi}_{\mathbf{s}}$ in (4.3). We also use 2-Wasserstein distance W_2 in (4.3) (as in DBC) because it has a closed form for Gaussian distributions. Denote by $(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i)$ and $(\mathbf{s}_j, \mathbf{a}_j, r_j, \mathbf{s}'_j)$ two transitions sampled from the replay buffer. The loss functions (4.2) and (4.3) can be revised as follows,

$$\ell(f_r, \phi_r) = \left(f_r(\phi_r(\mathbf{s}_i), \phi_r(\mathbf{s}_j)) - |r_i - r_j|\right)^2, \tag{4.4}$$

$$\ell(f_d, \phi_d) = \left(f_d(\phi_d(\mathbf{s}_i), \phi_d(\mathbf{s}_j)) - W_2(\hat{P}(\cdot|\phi_d(\mathbf{s}_i), \mathbf{a}_i), \hat{P}(\cdot|\phi_d(\mathbf{s}_j), \mathbf{a}_j))\right)^2, \tag{4.5}$$

where $\hat{P}(\cdot|\phi_d(\mathbf{s}_*), \mathbf{a}_*)$ is a learned probabilistic dynamics model, which is implemented as a neural network that takes the dynamics representation $\phi_d(\mathbf{s}_*)$ and an action $\mathbf{a}_*$ as input, and predicts the distribution of dynamics representation of next state $\mathbf{s}'_*$. The details of $\hat{P}$ can be found Section 4.7.1.

4.5.2 Balancing Impact of Reward and Dynamics

The π-bisimulation metric (4.1) is defined as a linear combination of the reward and the dynamics difference. Rather than considering the combination weight as a hyperparameter, which needs to be tuned in advance, we introduce a learnable parameter $c \in (0, 1)$ to adaptively balance the impact of the reward and the dynamics in different environments and tasks.

As the impact factor c should be automatically determined when learning in a specific task or environment, we integrate the learning of c into the update of the Q-function in SAC such that the value of c is learned from the state-action value which is highly relevant to a specific task and environment. To be specific, we concatenate the low-dimensional representations $\phi_r(\mathbf{s})$ and $\phi_d(\mathbf{s})$ weighted by $1 - c$ and c, where $1 - c$ and c are output of a softmax to ensure $c \in (0, 1)$. The loss for the Q-function is revised as follows, which takes $\phi_r(\mathbf{s})$, $\phi_d(\mathbf{s})$, weight c and action a as input:

$$J_Q(\theta, c, \phi_r, \phi_d) = \mathbb{E}_{(\mathbf{s},\mathbf{a},\mathbf{s}')\sim\mathcal{D}}\left[\frac{1}{2}\left(Q_\theta(\phi_r(\mathbf{s}), \phi_d(\mathbf{s}), c, \mathbf{a}) - (r(\mathbf{s}, \mathbf{a}) + \gamma V(\mathbf{s}'))\right)^2\right], \tag{4.6}$$

where Q_θ is the Q-function parameterized by θ and $V(\cdot)$ is the target value function.

Moreover, to balance the learning of the two meta-learners f_r and f_d, we jointly minimize the regression losses of f_r and f_d (loss (4.4) and loss (4.5), respectively) with the balance factor c. Thus, the loss function is modified as follows,

$$\ell(\Theta) = (1-c)\ell(f_r, \phi_r) + c\ell(f_d, \phi_d), \text{ where } \Theta = \{f_r, f_d, \phi_r, \phi_d\}. \tag{4.7}$$

Note that c is already learned along with the Q-function, we stop gradient of c when minimizing the loss (4.7). This is because if we perform gradient descent w.r.t. c, it may only capture which loss ($\ell(f_r, \phi_r)$ or $\ell(f_d, \phi_d)$) is larger/smaller and fail to learn the quantities of their importance. We also stop gradient for the dynamics model $\hat{P}$ since $\hat{P}$ only predicts one-step dynamics.

4.5.3 Overall Objective with Data Augmentation

In this section, we propose to integrate a data augmentation strategy into our proposed framework to learn more robust state representations. We follow the notations of an existing work on data augmentation for reinforcement learning, DrQ [89], and define a state transformation $h : \mathcal{S} \times \mathcal{T} \to \mathcal{S}$ that maps a state to a data-augmented state, where $\mathcal{T}$ is the space of parameters of h. In practice, we use a random crop as the transformation h in this work. Then $\mathcal{T}$ contains all possible crop positions and $\mathbf{v} \in \mathcal{T}$ is one crop position drawn from $\mathcal{T}$. We apply DrQ to regularize the objective of the Q-function (see Section 4.7.1 for the full objective of Q-function).

Besides, by using the augmentation transformation h, we rewrite the loss of representation learning and similarity approximation in (4.7) as follows,

$$\begin{aligned}
\ell(\Theta) =& (1-c)\left(f_r\left(\phi_r(\mathbf{s}_i^{(1)}), \phi_r(\mathbf{s}_j^{(1)})\right) - |r_i - r_j|\right)^2 \\
&+ c\left(f_d\left(\phi_d(\mathbf{s}_i^{(1)}), \phi_d(\mathbf{s}_j^{(1)})\right) - W_2\left(\hat{P}(\cdot|\phi_d(\mathbf{s}_i^{(1)}), \mathbf{a}_i), \hat{P}(\cdot|\phi_d(\mathbf{s}_j^{(1)}), \mathbf{a}_j)\right)\right)^2 \\
&+ (1-c)\left(f_r\left(\phi_r(\mathbf{s}_j^{(2)}), \phi_r(\mathbf{s}_i^{(2)})\right) - |r_i - r_j|\right)^2 \\
&+ c\left(f_d\left(\phi_d(\mathbf{s}_j^{(2)}), \phi_d(\mathbf{s}_i^{(2)})\right) - W_2\left(\hat{P}(\cdot|\phi_d(\mathbf{s}_i^{(1)}), \mathbf{a}_i), \hat{P}(\cdot|\phi_d(\mathbf{s}_j^{(1)}), \mathbf{a}_j)\right)\right)^2,
\end{aligned} \tag{4.8}$$

where $\mathbf{s}_*^{(k)} = h(\mathbf{s}_*, \mathbf{v}_*^{(k)})$ is the transformed state with parameters $\mathbf{v}_*^{(k)}$. Specifically, $\mathbf{s}_i^{(1)}$, $\mathbf{s}_j^{(1)}$, $\mathbf{s}_i^{(2)}$ and $\mathbf{s}_j^{(2)}$ are transformed states with parameters of $\mathbf{v}_i^{(1)}$, $\mathbf{v}_j^{(1)}$, $\mathbf{v}_i^{(2)}$ and $\mathbf{v}_j^{(2)}$ respectively, which are all drawn from $\mathcal{T}$ independently. The first two terms in (4.8) are

similar to the two terms in (4.7), while the observation $\mathbf{s}_*$ in (4.7) is transformed to $\mathbf{s}_*^{(1)}$. For the last two terms in (4.8), the observation is transformed by a different parameter $\mathbf{v}_*^{(2)}$ except for the last term, where the parameter $\mathbf{v}_*^{(1)}$ does not change. The reason why the parameter $\mathbf{v}_*^{(1)}$ is used in the last term is that we expect the meta-learner f_d is able to make a consensus prediction on $\mathbf{s}_*^{(1)}$ and $\mathbf{s}_*^{(2)}$ as they are transformed from a same state $\mathbf{s}_*$. Another major difference between the first two terms and the last two terms is the order of the subscripts i and j: i is followed by j in first two terms while j is followed by i in last two terms. The reason behind this is that we aim to make f_r and f_d to be symmetric.

The loss of the actor of SAC, which is based on the output of the encoder, is

$$J_\pi(\psi) = \mathbb{E}_{(\mathbf{s},\mathbf{a})\sim\mathcal{D}}\left[\alpha \log(\pi_\psi(\mathbf{a}|\hat{\phi}(\mathbf{s}))) - Q_\theta(\phi(\mathbf{s}),\mathbf{a})\right], \tag{4.9}$$

where $\hat{\phi}$ denotes the convolutional layers of ϕ and $Q_\theta(\phi(\mathbf{s}),\mathbf{a})$ is a convenient form of the Q-function with c-weighted state representations input. We also stop gradients for $\hat{\phi}$ and ϕ (ϕ_r and ϕ_d) regarding the actor loss. In other words, the loss in (4.9) only calculates the gradients w.r.t. ψ.

The overall learning algorithm is summarized in Algorithm 1. In summary, as illustrated in Figure 4.1, in AMBS, the convnet is shared by the encoders ϕ_r and ϕ_d, which consists of 4 convolutional layers. The final output of the convnet is fed into two branches: 1) one is a fully-connected layer to form reward representation $\phi_r(\mathbf{s})$, and 2) the other is also a fully-connected layer but forms dynamics representation $\phi_d(\mathbf{s})$. The meta-learners f_r and f_d are both MLPs with two layers. More implementation details are provided at Section 4.7.

4.6 Experiments

The major goal of our proposed AMBS is to learn a generalizable representation for RL from high-dimensional raw data. We evaluate AMBS on two environments: 1) the DeepMind Control (DMC) suite [80] with background distraction; and 2) autonomous driving task on CARLA [16] simulator. Several baselines methods are compared with AMBS: 1) Deep Bisimulation Control (DBC) [94] which is recent research on RL representation learning by using the L_1 distance to approximate bisimulation metric; 2)

Algorithm 1 AMBS+SAC

1: **Input:** Replay Buffer $\mathcal{D}$, initialized $\Theta = \{f_r, \phi_r, f_d, \phi_d\}$, Q network Q_θ, actor pi_ψ, target Q network $Q_{\bar{\theta}}$,
2: Sample a batch with size B: $\{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i)\}_{b=1}^{B} \sim \mathcal{D}$.
3: Shuffle batch $\{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i)\}_{b=1}^{B}$ to $\{(\mathbf{s}_j, \mathbf{a}_j, r_j, \mathbf{s}'_j)\}_{b=1}^{B}$.
4: Update Q network by (4.6) with DrQ augmentation.
5: update actor network by (4.9).
6: update alpha α.
7: update encoder Θ by (4.8)
8: update dynamics model $\hat{P}$.

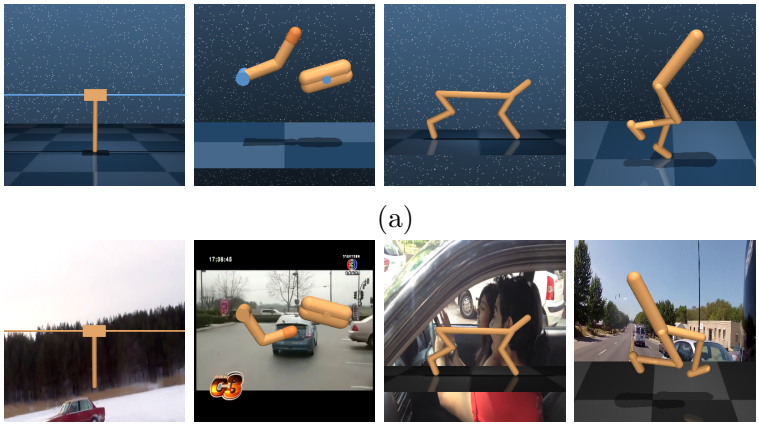
PSEs [2] which proposes a new state-pair metric called policy similarity metric for representation learning and learns state embedding by incorporating a contrastive learning method SimCLR [8]; 3) DrQ [89], a state-of-the-art data augmentation method for deep RL; 4) Stochastic Latent Actor-Critic (SLAC) [52], a state-of-the-art approach for partial observability on controlling by learning a sequential latent model with a reconstruction loss; 5) Soft Actor-Critic (SAC) [23], a commonly used off-policy deep RL method, upon which above baselines and our approach are built.

4.6.1 DMC suite with background distraction

DeepMind Control (DMC) suite [80] is an environment that provides high-dimensional pixel observations for RL tasks. DMC suite with background distraction [93, 76] replaces the background with natural videos which are task-irrelevant to the RL tasks. We perform the comparison experiments in two settings: original background and nature video background. For each setting we evaluate AMBS and baselines in 4 environments, cartpole-swingup, finger-spin, cheetah-run and walker-walk.

Original Background. Figure 4.3a shows the DMC observation of original background which is static and clean. The experiment result is shown in Figure 4.4. Our method AMBS has comparable learning efficiency and converge scores comparing to state-of-the-art pixel-level RL methods DrQ and SLAC. Note that DBC performs worse in original background setting.

Natural Video Background. Figure 4.3b shows the DMC observation with natural video background. The background video is sampled from the Kinetics [39] dataset under



(a)

(b)

Fig. 4.3: **(a)** Pixel observations of DMC suite with original background. **(b)** Pixel observations of DMC suite with natural video background. Videos are sampled from Kinetics [39].

label "driving car". By following the experimental configuration of DBC, we sample 1,000 continuous frames as background video for training, and sample another 1,000 continuous frames for evaluation. The experimental result is shown in Figure 4.5. Our method AMBS outperforms other baselines in terms of both efficiency and scores. It converges to the scores that are similar to the scores on original background setting within 1 million steps. DrQ can only converge to the same scores on cartpole-swingup and finger-spin but the learning is slightly slower than AMBS. The sequential latent model method SLAC performs badly on this setting. This experiment demonstrates that AMBS is robust to the background task-irrelevant information and the AMBS objectives improve the performance when comparing to other RL representation learning methods, e.g., DBC.

4.6.2 Ablation Study

Our approach AMBS consists of several major components: meta-learners, a factor to balance the impact between reward and dynamics, and data augmentation on representation learning. To evaluate the importance of each component, we eliminate or replace each component by baseline methods. We conduct experiments on cheetah-run and walker-walk under the natural video background setting. Figure 4.6 shows the performance of AMBS, variants of AMBS and baseline methods. **AMBS w/o data aug** is

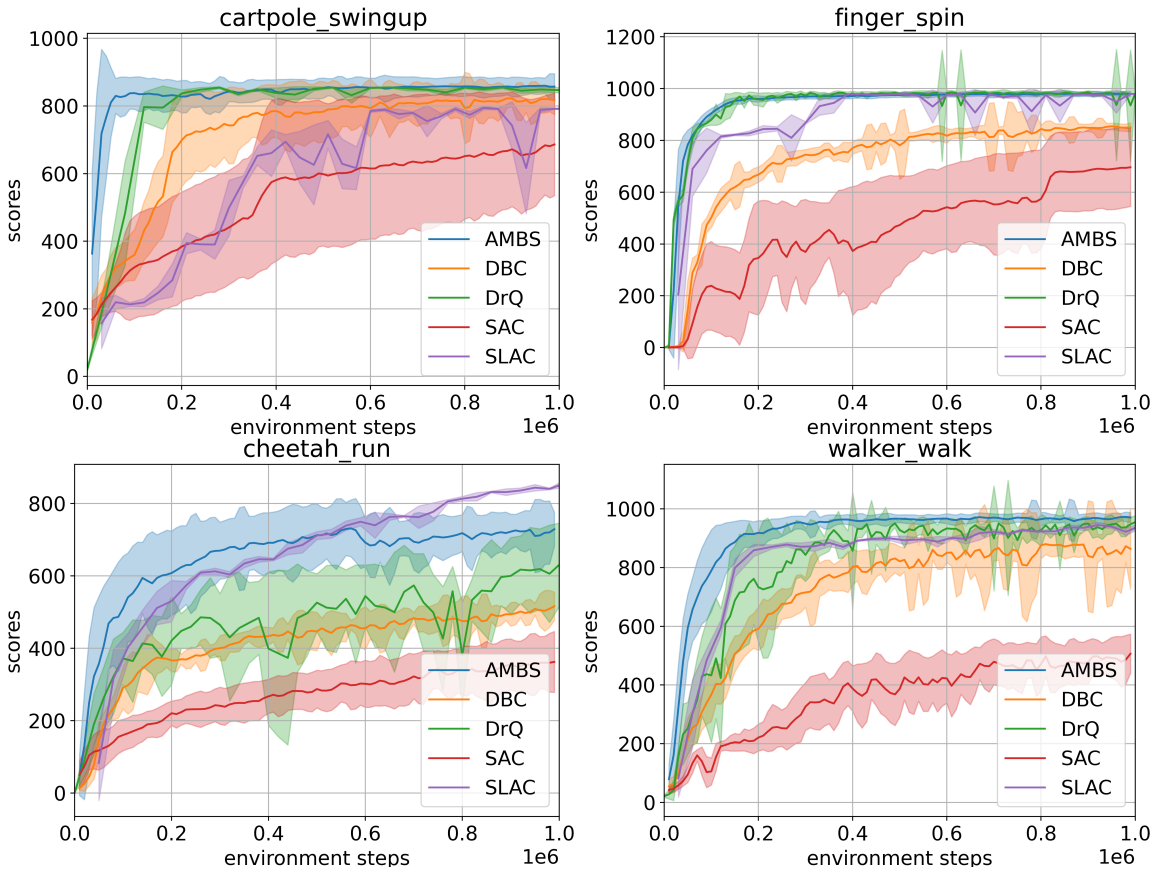


Fig. 4.4: Training curves of AMBS and comparison methods. Each curve is average on 5 runs.

constructed by removing data augmentation in AMBS. **AMBS w/o** f is referred to as replacing the meta-learners f_r and f_d by the L_1 distance, as in DBC. **AMBS w/o** c removes the component of balancing impact between reward and dynamics but encodes the observation into a single embedding. It uses single meta-learner to approximate the sum of reward and dynamics distance. **DBC + DrQ** is DBC with data augmentation applied on representation learning and SAC. **AMBS** $c = 0.5$ denotes AMBS with a fixed weight $c = 0.5$. **AMBS w/** $(1, \gamma)$ replaces c in AMBS by freezing the reward weight to 1 and the dynamics weight to γ. Such a setting of weights is used in DBC. Figure 4.6 demonstrates that AMBS performs better than any of its variant. Comparing to DBC that does not utilize data augmentation, our variant **AMBS w/o data aug** still performs better. This comparison shows that using meta-learners f_r and f_d to learn reward- and dynamics- relevant representations is able to improve RL performance compared

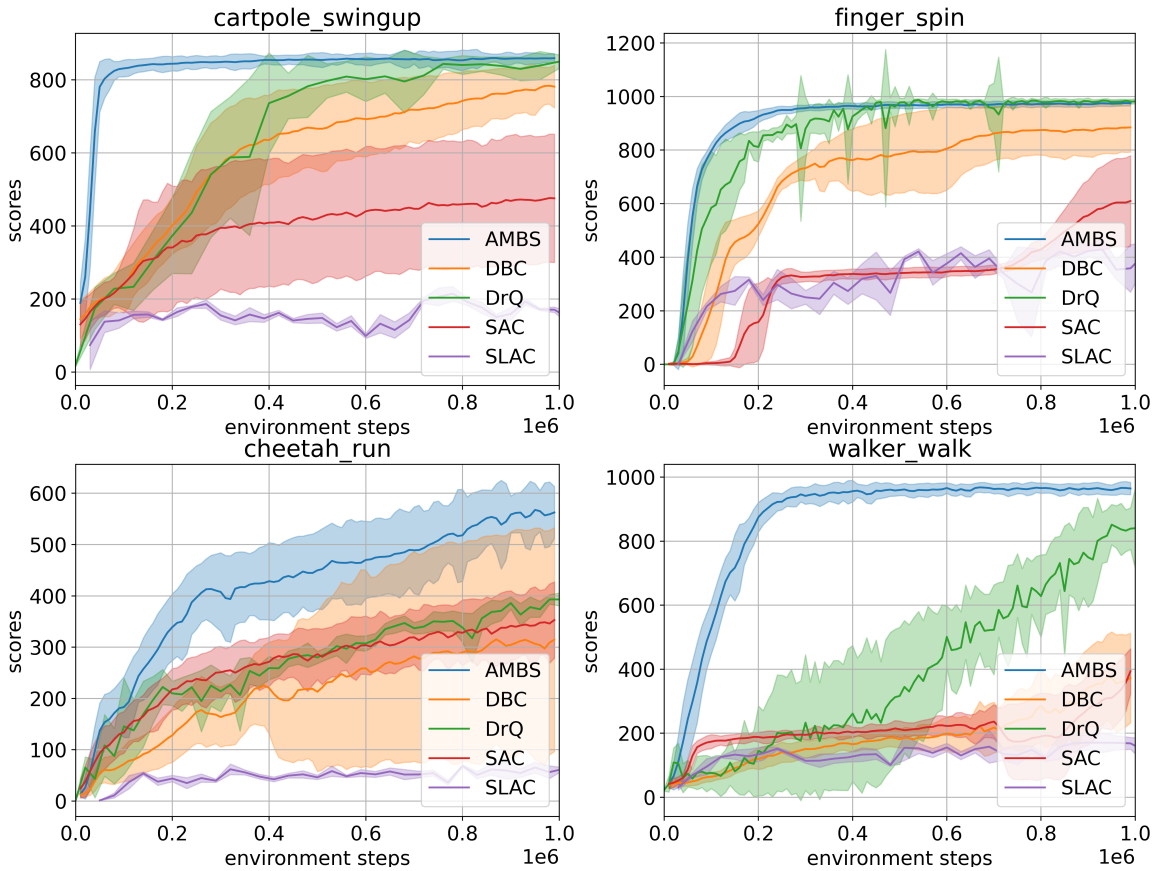


Fig. 4.5: Training curves of AMBS and comparison methods on natural video background setting. Videos are sampled from Kinetics [39] dataset. Each curve is average on 5 runs.

with using the L_1 distance to approximate the whole π-bisimulation metric.

4.6.3 Transfer over Reward Functions

The environments walker-walk, walker-run, and walker-stand share the same dynamics but have different reward functions according to the moving speed. To evaluate the transfer ability of AMBS over reward functions, we transfer the learned model from walker-walk to walker-run and walker-stand. We train agents on walker-run and walker-stand with frozen convolutional layers of encoders that are well-trained in walker-walk. The experiments are done under the natural video setting compared with DBC as shown in Figure 4.7. It shows that the transferring encoder converges faster than training from scratch. Besides our approach has better transfer ability than DBC.

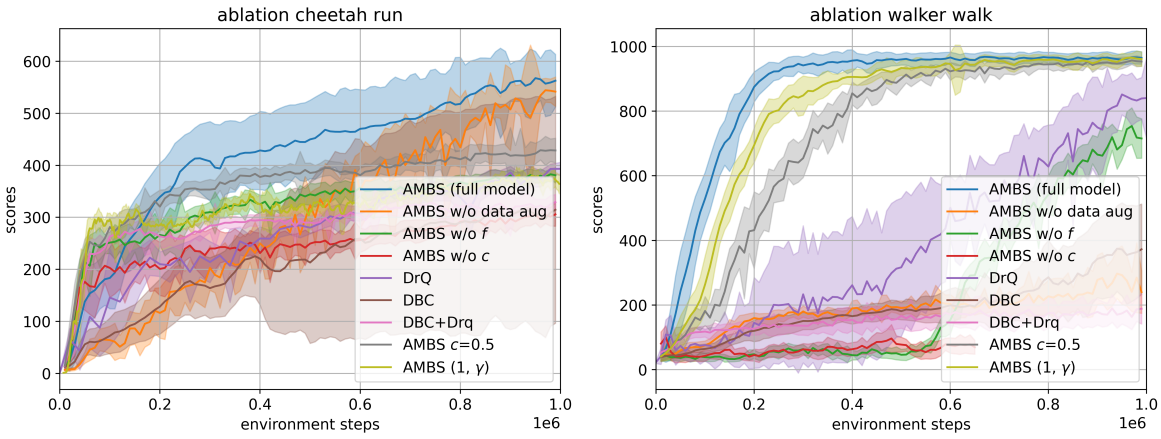


Fig. 4.6: Ablation Study on cheetah-run **(left)** and walker-walk **(right)** in natural video setting.

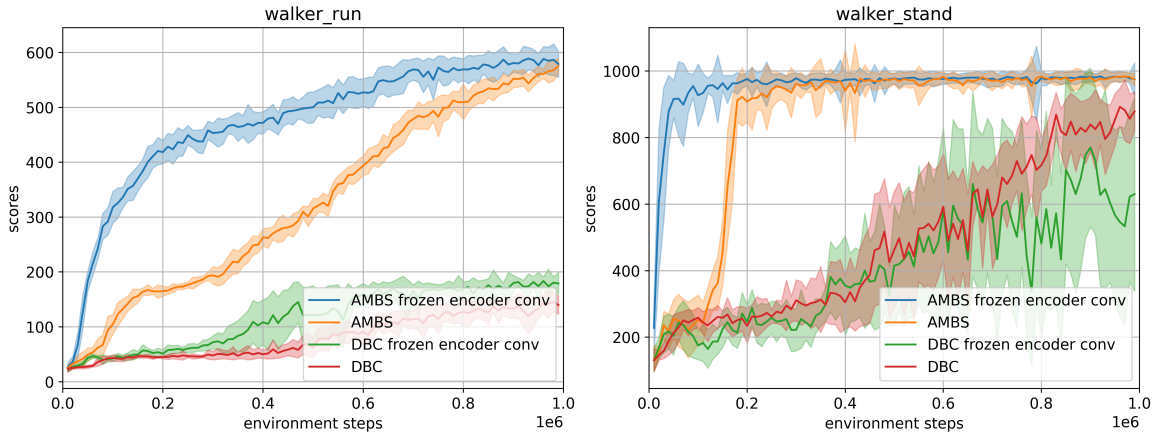


Fig. 4.7: Transfer from walker-walk to **(left)** walker-run and **(right)** walker-stand.

4.6.4 Regression Losses of Approximating Bisimulation Metric

We compare the regression losses of approximating the bisimulation metric over RL environment steps among DBC and our AMBS. DBC uses the L_1 norm to calculate the distances between two states embeddings. AMBS performs meta-learners on state representations to measure the distance with learned similarities. Figure 4.8 demonstrates that AMBS has smaller regression losses and also a faster descent tendency compared to DBC. Meta-learners in AMBS are able to provide more stable gradients to update the state representations.

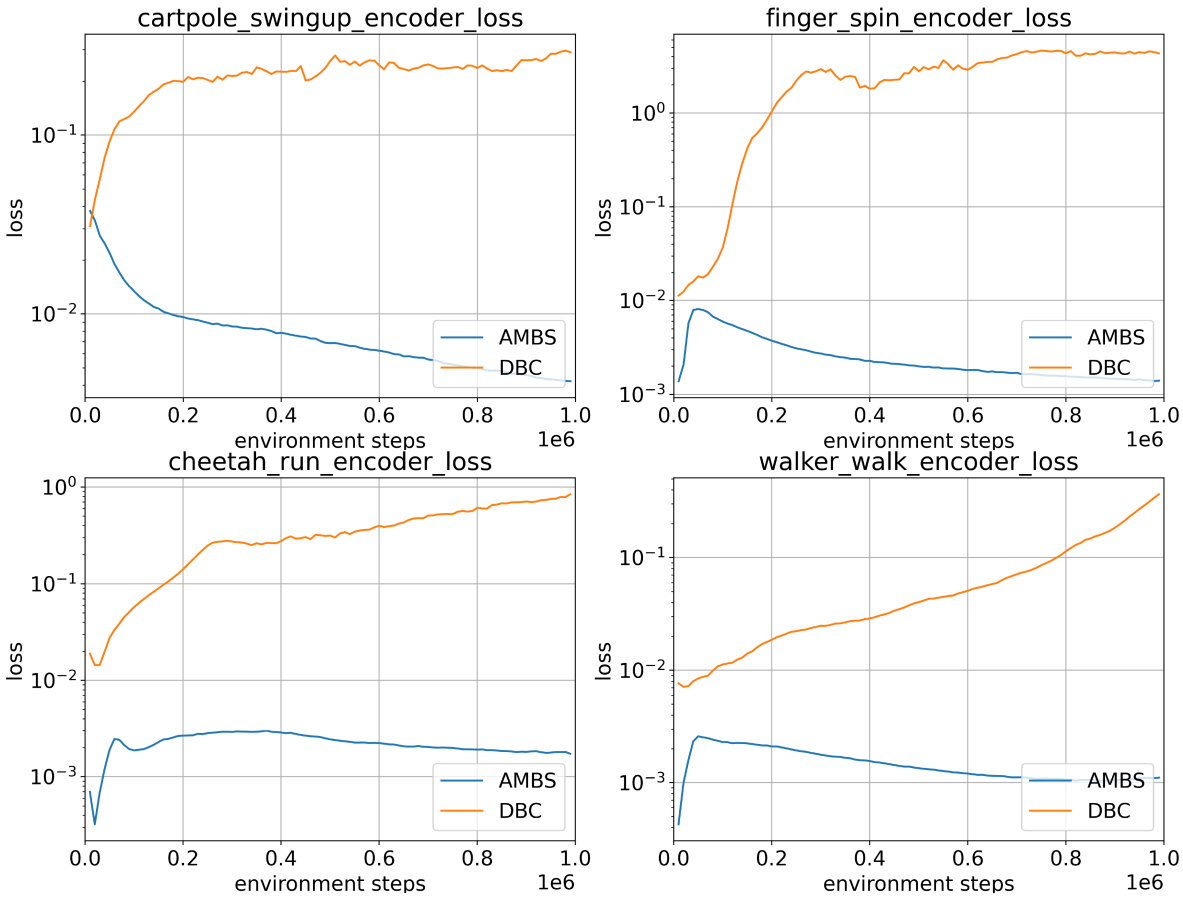


Fig. 4.8: The regression losses over RL environment steps among DBC and AMBS.

4.6.5 Visualizations of Combination Weight c

Figure 4.9 shows the values of combination weight c in AMBS trained on DMC with original background, and Figure 4.9 is on DMC with video background setting. The values of c at 1M steps vary in different tasks. The common trend is that they increase at the beginning of training. In original background setting c changes slowly after the beginning. In natural video background setting, c has a fast drop after the beginning. In the beginning, the agent learns a large weight c for dynamics feature that may boost the learning of the RL agent. Then it drops for natural video background setting because it learns that the video background is distracting the agent. Lower weight is learned for the dynamics features.

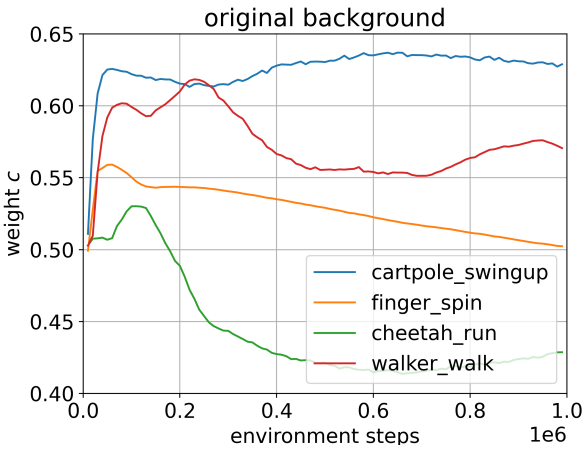


Fig. 4.9: The value of combination weight c over environment steps. AMBS is trained on DMC on Original background.

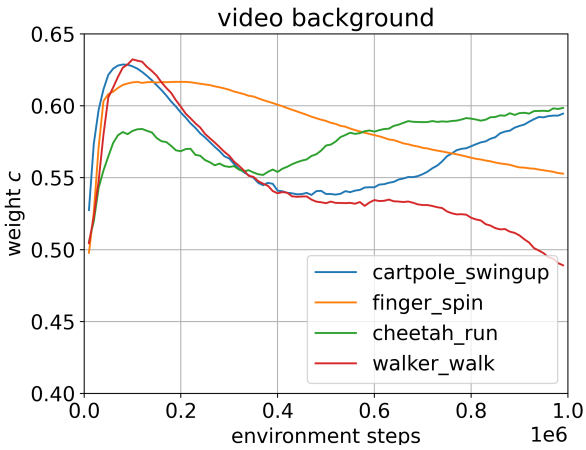


Fig. 4.10: The value of combination weight c over environment steps. AMBS is trained on DMC on video background.

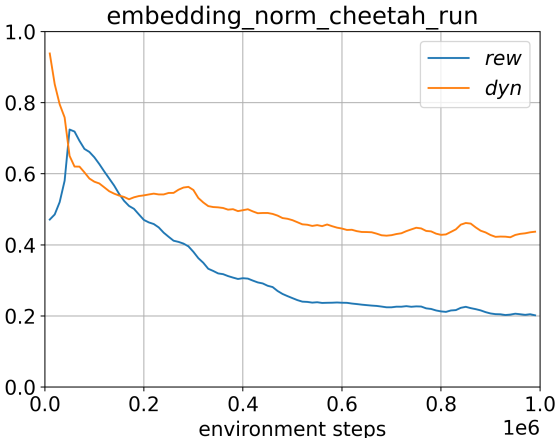


Fig. 4.11: Norms of state embeddings: $\frac{1}{n_r}||\phi_r(s)||_1$ and $\frac{1}{n_d}||\phi_d(s)||_1$.

4.6.6 Norms of State Embeddings

We record the L_1 norms of reward representation $\phi_r(s)$ and dynamics representation $\phi_d(s)$ during training. We record $\frac{1}{n_r}||\phi_r(s)||_1$ for reward representation where n_r is the number of dimensions of $\phi_r(s)$, and $\frac{1}{n_d}||\phi_d(s)||_1$ for dynamics representation where n_r is the number of dimensions of $\phi_d(s)$. Figure 4.11 shows the values averaged on each sampled training batch. Although the L_1 norm of $\phi_d(s)$ decreases during training, it converges around 0.4-0.5. The L_1 norm of $\phi_r(s)$ also decreases in the training procedure. It can verify that dynamics representation $\phi_d(s)$ will not converge to all zeros.

4.6.7 Generalization over Background Video

We evaluate the generalization ability of our method when background video is changed. We follow the experiment setting of PSE [2]. We pick 2 videos, 'bear' and 'bmx-bumps', from the training set of DAVIS 2017 [67] for training. We randomly sample one video and a frame at each episode start, and then play the video forward and backward until episode ends. The RL agent is evaluated on validation environment where the background video is sampled from validation set of DAVIS 2017. Those validation videos are unseen during the training. The evaluation scores at 500K environment interaction steps are shown in Table 4.1. The comparison method DrQ+PSEs is data augmented by DrQ. AMBS outperforms DrQ+PSEs on all the 4 tasks. It demonstrates that AMBS is generalizable on unseen background videos.

Methods	C-swingup	F-spin	C-run	W-walk
DrQ + PSEs	749 ± 19	779 ± 49	308 ± 12	789 ± 28
AMBS	**807 ± 41**	**933 ± 96**	**332 ± 27**	**893 ± 85**

Table 4.1: Generalization to unseen background videos sampled from DAVIS 2017 [67]. Scores of DrQ+PSEs are reported from [2].

(a) (b)

Fig. 4.12: **(a)** Illustration of a third-person view in "Town4" scenario of CARLA. **(b)** A first-person observation for RL agent. It concatenates five cameras for 300 degrees view .

4.6.8 Autonomous Driving Task on CARLA

CARLA is an autonomous driving simulator that provides a 3D simulation for realistic on-road scenario. In real world cases or in realistic simulations, the learning of RL agents may suffer from complex background that contains task-irrelevant details. To

argue that AMBS can address this issue, we perform experiments with CARLA. We follow the setting of DBC to create an autonomous driving task on map "Town04" for controlling a car to drive as far as possible in 1,000 frames. The environment contains a highway with 20 moving vehicles. The reward function prompts the driving distance and penalizes collisions. The observation shape is 84 × 420 pixels which consists of five 84 × 84 cameras. Each camera has 60 degrees view, together they produce a 300 degrees first-person view, as shown in Figure 4.12. The weather, clouds and brightness may change slowly during the simulation. Figure 4.13 shows the training curves of AMBS and other comparison methods. AMBS performs the best, which provides empirical evidence that AMBS can potentially generalize over task-irrelevant details, e.g., weather and clouds, in real world scenarios.

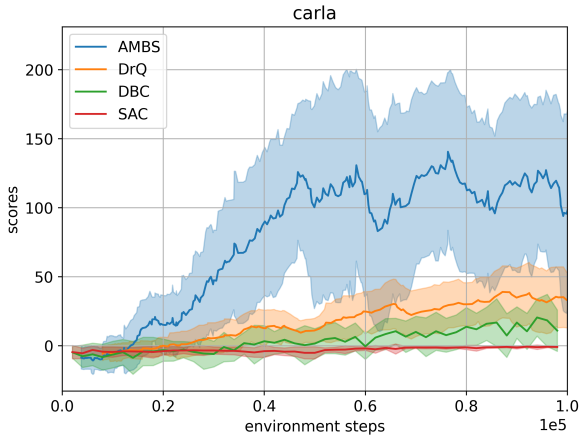


Fig. 4.13: CARLA simulation.

4.7 Implementation Details

4.7.1 Detailed Objectives of AMBS and SAC

We denote two encoders ϕ_r and ϕ_d, where each encoder maps a high-dimensional observation to a low-dimensional representation. The encoder $\phi_r : \mathcal{S} \to \mathcal{Z}_r$ captures reward-relevant features and the encoder $\phi_d : \mathcal{S} \to \mathcal{Z}_d$ captures dynamics-relevant features, where $\mathcal{Z}_r$ and $\mathcal{Z}_d$ are representation spaces. We design two meta-learners $f_r : \mathcal{Z}_r \times \mathcal{Z}_r \to \mathbb{R}$ and $f_d : \mathcal{Z}_d \times \mathcal{Z}_d \to \mathbb{R}$ to output reward difference and dynamics difference, respectively.

We follow DrQ [89] to define the state transformation $h : \mathcal{S} \times \mathcal{T} \to \mathcal{S}$ that maps a state to a data-augmented state, where $\mathcal{T}$ is the space of parameters of h. In practice, we use random crop as transformation h.

With augmentation transformation h, the loss of representation learning is,

$$\begin{aligned}\ell(\Theta) =&(1-c)\left(f_r(\phi_r(\mathbf{s}_i^{(1)}),\phi_r(\mathbf{s}_j^{(1)})) - |r_i - r_j|\right)^2 \\ &+ c\left(f_d(\phi_d(\mathbf{s}_i^{(1)}),\phi_d(\mathbf{s}_j^{(1)})) - W_2(\hat{\mathcal{P}}(\cdot|\phi_d(\mathbf{s}_i^{(1)}),\mathbf{a}_i),\hat{\mathcal{P}}(\cdot|\phi_d(\mathbf{s}_j^{(1)}),\mathbf{a}_j))\right)^2 \\ &+ (1-c)\left(f_r(\phi_r(\mathbf{s}_j^{(2)}),\phi_r(\mathbf{s}_i^{(2)})) - |r_i - r_j|\right)^2 \\ &+ c\left(f_d(\phi_d(\mathbf{s}_j^{(2)}),\phi_d(\mathbf{s}_i^{(2)})) - W_2(\hat{\mathcal{P}}(\cdot|\phi_d(\mathbf{s}_i^{(1)}),\mathbf{a}_i),\hat{\mathcal{P}}(\cdot|\phi_d(\mathbf{s}_j^{(1)}),\mathbf{a}_j))\right)^2,\end{aligned} \tag{4.10}$$

where $\Theta = \{f_r, f_d, \phi_r, \phi_d\}$, $\mathbf{s}_*^{(l)} = h(\mathbf{s}_*, \mathbf{v}_*^{(l)})$ is the transformed state with parameters $\mathbf{v}_*^{(l)}$. Specifically, $\mathbf{s}_i^{(1)}$, $\mathbf{s}_j^{(1)}$, $\mathbf{s}_i^{(2)}$ and $\mathbf{s}_j^{(2)}$ are transformed states with parameters of $\mathbf{v}_i^{(1)}$, $\mathbf{v}_j^{(1)}$, $\mathbf{v}_i^{(2)}$ and $\mathbf{v}_j^{(2)}$ respectively, which are all drawn from $\mathcal{T}$ independently. $\hat{\mathcal{P}}(\cdot|\phi_d(\mathbf{s}_*), \mathbf{a}_*)$ is a learned probabilistic dynamics model, which outputs a Gaussian distribution over $\mathcal{Z}_d$ for predicting dynamics representation of next state $\mathbf{s}'_*$. W_2 denotes the 2-Wasserstein distance which has closed-form for Gaussian distributions: $W_2(\mathcal{N}(\mu_i, \mathbf{\Sigma}_i), \mathcal{N}(\mu_j, \mathbf{\Sigma}_j))^2 = \|\mu_i - \mu_j\|_2^2 + \|\mathbf{\Sigma}_i^{\frac{1}{2}} - \mathbf{\Sigma}_j^{\frac{1}{2}}\|_{\mathcal{F}}^2$, where $\mu_* \in \mathbb{R}^{|\mathcal{Z}_d|}$ are the mean vectors of Gaussian distribution, $\mathbf{\Sigma}_* \in \mathbb{R}^{|\mathcal{Z}_d|}$ are the diagonals of covariance matrices, and $\|\cdot\|_{\mathcal{F}}$ is Frobenius norm. Note that we stop gradients for c as mentioned in Section 4.5.2.

The adaptive weight c and $(1-c)$ are the softmax output of corresponding learnable parameter $\eta_c \in \mathbb{R}^2$. It can be formulated as

$$[c : (1-c)] = softmax(\eta_c). \tag{4.11}$$

where the operation : denotes concatenation of two vectors/scalars.

We integrate the learning of c into the update of Q-function of SAC. The objective of the Q-function with DrQ augmentation and clipped double trick is

$$\begin{aligned}J_Q(\theta_k, \eta_c) = \mathbb{E}_{(\mathbf{s},\mathbf{a},\mathbf{s}')\sim\mathcal{D}}\Bigg[&\frac{1}{2}\left(Q_{\theta_k}([(1-c)\phi_r(\mathbf{s}^{(1)}) : c\phi_d(\mathbf{s}^{(1)})], \mathbf{a}) - (r(\mathbf{s},\mathbf{a}) + \gamma V(\mathbf{s}'))\right)^2 \\ &+\frac{1}{2}\left(Q_{\theta_k}([(1-c)\phi_r(\mathbf{s}^{(2)}) : c\phi_d(\mathbf{s}^{(2)})], \mathbf{a}) - (r(\mathbf{s},\mathbf{a}) + \gamma V(\mathbf{s}'))\right)^2\Bigg],\end{aligned} \tag{4.12}$$

where $\mathbf{s}^{(l)} = h(\mathbf{s}, \mathbf{v}^{(l)})$, $\mathbf{v}^{(l)}$ are all drawn from $\mathcal{T}$ independently, and $\{Q_{\theta_k}\}_{k=1}^{2}$ are the double Q networks. The target value function with DrQ augmentation is

$$V(\mathbf{s}') = \frac{1}{2}\left(\min_{k=1,2} Q_{\bar{\theta}_k}\left([(1-\bar{c})\bar{\phi}_r(\mathbf{s}'^{(1)}) : \bar{c}\bar{\phi}_d(\mathbf{s}'^{(1)})], \mathbf{a}'^{(1)}\right) + \alpha \log \pi_\psi(\mathbf{a}'^{(1)}|\mathbf{s}'^{(1)}) \right.$$
$$\left. + \min_{k=1,2} Q_{\bar{\theta}_k}\left([(1-\bar{c})\bar{\phi}_r(\mathbf{s}'^{(2)}) : \bar{c}\bar{\phi}_d(\mathbf{s}'^{(2)})], \mathbf{a}'^{(2)}\right) + \alpha \log \pi_\psi(\mathbf{a}'^{(2)}|\mathbf{s}'^{(2)})\right), \quad (4.13)$$

where $\mathbf{s}'^{(l)} = h(\mathbf{s}', \mathbf{v}'^{(l)})$, $\mathbf{a}'^{(l)} \sim \pi_\psi(\mathbf{a}'^{(l)}|\mathbf{s}'^{(l)})$, $\bar{\theta}_k$ is the set of parameters of the target Q network, $\bar{\phi}_r$ and $\bar{\phi}_d$ are target encoders specifically for the target Q network and $\bar{c}$ is the adaptive weight for target encoders. The parameters $\bar{\theta}_k$, $\bar{\phi}_r$, $\bar{\phi}_d$ and $\bar{\eta}_c$ are softly updated.

The loss of actor of SAC is,

$$J_\pi(\psi) = \mathbb{E}_{(\mathbf{s})\sim\mathcal{D}}\left[\mathbb{E}_{\mathbf{a}\sim\pi_\psi}[\alpha \log(\pi_\psi(\mathbf{a}|\hat{\phi}(\mathbf{s}^{(1)}))) - \min_{k=1,2} Q_{\theta_k}(\phi(\mathbf{s}^{(1)}), \mathbf{a})]\right]. \quad (4.14)$$

The loss of α of SAC is,

$$J(\alpha) = \mathbb{E}_{(\mathbf{s})\sim\mathcal{D}}\left[\mathbb{E}_{\mathbf{a}\sim\pi_\psi}[\alpha \log(\pi_\psi(\mathbf{a}|\hat{\phi}(\mathbf{s}^{(1)}))) - \alpha\bar{\mathcal{H}}]\right], \quad (4.15)$$

where $\bar{\mathcal{H}} \in \mathbb{R}$ is the target entropy hyper-parameter which in practice is $\bar{\mathcal{H}} = -|\mathcal{A}|$.

The loss for updating of dynamics model $\hat{\mathcal{P}}$ is

$$\ell(\hat{\mathcal{P}}) = \mathbb{E}_{(\mathbf{s},\mathbf{a},\mathbf{s}')\sim\mathcal{D}}\left[\left(\frac{\phi_d(\mathbf{s}'^{(1)}) - \mu(\hat{\mathcal{P}}(\cdot|\phi_d(\mathbf{s}^{(1)}), \mathbf{a}))}{2\sigma(\hat{\mathcal{P}}(\cdot|\phi_d(\mathbf{s}^{(1)}), \mathbf{a}))}\right)^2\right], \quad (4.16)$$

where $\mu(\hat{\mathcal{P}}(\cdot))$ and $\sigma(\hat{\mathcal{P}}(\cdot))$ are mean and standard deviation of $\hat{\mathcal{P}}$ output Gaussian distribution, respectively.

4.7.2 Algorithm

Algorithm 2 shows the algorithm at each learning step.

Algorithm 2 AMBS+SAC

Input: Replay Buffer $\mathcal{D}$, initialized $\Theta = \{f_r, \phi_r, f_d, \phi_d\}$, Q network Q_{θ_k}, actor pi_ψ, target Q network $Q_{\bar{\theta}_k}$,
Sample a batch with size B: $\{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i)\}_{b=1}^{B} \sim \mathcal{D}$.
Shuffle batch $\{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i)\}_{b=1}^{B}$ to $\{(\mathbf{s}_j, \mathbf{a}_j, r_j, \mathbf{s}'_j)\}_{b=1}^{B}$.
Update Q network by (4.12) with DrQ augmentation.
Update actor network by (4.14) .
Update alpha α by (4.15) .
Update encoder Θ by (4.10)
Update dynamics model $\hat{\mathcal{P}}$ by (4.16).
Softly update target Q network: $\bar{\theta}_k = \tau_Q \theta_k + (1 - \tau_Q)\bar{\theta}_k$.
Softly update target encoder: $\bar{\phi} = \tau_\phi \phi + (1 - \tau_\phi)\bar{\phi}$.
Softly update target $\bar{c}$: $\bar{\eta}_c = \tau_\phi \eta_c + (1 - \tau_\phi)\bar{\eta}_c$.

4.7.3 Pixels Processing

We stack 3 consecutive frames as an observation, where each frame is 84×84 RGB images in DeepMind control suite. Each pixel is divided by 255 to downscale to $[0, 1]$. We consider the stacked frames as a fully observed state.

4.7.4 Network Architecture

We share the convnet for encoder ϕ_r and encoder ϕ_d. The shared convnet consists of four convolutional layers with 3×3 kernels and 32 output channels. We set stride to 1 everywhere, except for the first convolutional layer, which has stride 2. We use ReLU activation for each convolutional layer output. The final output of convnet is fed into two branches: 1) one is a fully-connected layer with 50 dimensions to form reward representation $\phi_r(\mathbf{s})$, and 2) the other one is also a fully-connected layer with 50 dimensions but forms dynamics representation $\phi_d(\mathbf{s})$. The meta-learners f_r and f_d are both MLPs with two layers with 50 hidden dimensions.

The critic network takes $\phi_r(\mathbf{s})$, $\phi_d(\mathbf{s})$ and action as input, and feeds them into three stacked fully-connected layers with 1024 hidden dimensions. The actor network takes the shared convnet output as input and feeds it into four fully-connected layers, where the first layer has 100 hidden dimensions and the other layers have 1024 hidden dimensions. The dynamics model is an MLP with two layers with 512 hidden dimensions. ReLU activations are used in every hidden layer.

4.7.5 Hyperparameters

The hyperparameters used in our algorithm are listed in Table 4.2.

Parameter name	Value
Replay buffer capacity	10^6
Discount factor γ	0.99
Minibatch size	128
Optimizer	Adam
Learning rate for α	10^{-4}
Learning rate except α	5×10^{-4}
Target update frequency	2
Actor update frequency	2
Actor log stddev bounds	$[-10, 2]$
τ_Q	0.01
τ_ϕ	0.05
Init temperature	0.1

Table 4.2: Hyperparameters used in our algorithm.

4.7.6 Action Repeat for DMC

We use different action repeat hyper-parameters for each task in DeepMind control suite, which are listed in Table 4.3.

Task name	Action repeat
Cartpole Swingup	8
Finger Spin	2
Cheetah Run	4
Walker Walk	2

Table 4.3: Action repeat used for each task in DeepMind control suite.

4.8 Conclusion

This chapter presents a novel framework AMBS with meta-learners for learning task-relevant and environment-detail-invariant state representation for deep RL. In AMBS,

we propose state encoders that decompose an observation into rewards and dynamics representations for measuring behavioral similarities by two self-learned meta-learners, and design a learnable strategy to balance the two types of representations. AMBS achieves new state-of-the-art results in various environments and demonstrates its transfer ability on RL tasks. Experimental results verify that our framework is able to effectively learn a generalizable state representation.

This work studies a meta-learners-based method in a deep learning manner, but rarely explores the theoretical properties of behavioral metrics. Besides, AMBS also lacks theoretical guarantees. In the next chapter, I will discuss the current challenges in learning behavioral metrics and present another approach regarding theoretical guarantees toward data efficiency and generalization in RL.

Chapter 5

Reducing Approximation Gap (RAP) Distance[1]

5.1 Issues of Behavioral Metric-based RL

Behavioral metrics, such as the bisimulation metric and its variants [18, 19, 4], are originally proposed to measure the behavioral similarity between states in terms of rewards and dynamics models, e.g., state transition probabilities. The high-level idea is to learn state embeddings by preserving the behavioral similarity between states based on a specific behavioral metric [94, 2, 40, 5]. As behavioral metrics provide a theoretical bound on the difference between the outputs of value function of a pair of states, the learned representation enjoys a theoretical guarantee to capture behavioral structure in the environment for policy learning. However, as behavioral metrics are expensive or even intractable to compute, different approximation approaches and learning objectives have been proposed to make behavioral metric-based representation learning for RL agents more efficient [94, 2, 40, 5].

Though behavioral metric-based representation learning methods have achieved state-of-the-art results on some benchmark RL problems, they suffer from at least one of the following three issues: loss function mismatch, relaxation of dynamics model divergence, and the L_1/L_2 norm limitation.

- **Loss function mismatch**. In behavioral metric-based representation learning, a loss function is to measure the difference between the distance between states on

[1]The work in this chapter has been published in [6].

the embedding space and their behavioral metric. As behavioral metrics are expensive or even intractable to computers, approximation or relaxation are necessary. However, based on our analysis, state-of-the-art methods adopt improper approximations, which may introduce a bias and make the bound of the value function looser.

- **Relaxation of dynamics model divergence**. The bisimulation metric or its on-policy invariant is one of the most widely used behavioral metrics, which requires estimating the 1-Wasserstein distance between dynamics models. As the 1-Wasserstein distance is usually difficult or intractable to estimate, prior methods propose some relaxations to replace the estimation of the 1-Wasserstein, which may break some theoretical guarantees of the bisimulation metric.

- **The L_1/L_2 norm limitation**. The L_1 and the L_2 norms are commonly-used distances with zero self-distance. However, due to the two gaps mentioned above, the approximations or relaxations of behavior metrics in prior approaches are potentially non-zero self-distance. Using the L_1 or L_2 distance on embedding space is difficult to learn robust representation to preserve the behavioral similarity between states.

5.2 Introduction

To address the aforementioned issues or gaps, in this paper, we first introduce a new behavior metric namely the *Reducing Approximation Gap* (RAP) distance, and then develop a practical approximation algorithm with consistency to its theoretical prototype. In this way, our algorithm is guaranteed to learn a robust state representation to capture the behavioral similarity between states. We conduct extensive experiments on DeepMind Control Suite (DMC) [80] with background distraction, Robosuite [99], and autonomous driving simulator CARLA [16] to demonstrate new state-of-the-art results compared with other behavior metric-based representation learning methods.

The contributions of this work are two-fold: 1) we analyze the potential approximation gaps for existing behavioral metric-based representation learning approaches, and 2) we introduce a new behavior distance RAP and develop its practical approximation algorithm with theoretical guarantees.

5.3 Related Work

In RL, early research work has focused on learning state representations by designing and optimizing some auxiliary objectives in addition to the RL task of interest. For instance, Hafner et al. [25] propose to learn a dynamics model to predict future states with a reconstruction loss. Gelada et al. [22] aim to learn state representations by predicting a dynamics model and a reward function on an embedding space. Laskin et al. [47] apply contrastive learning with samples generated by the momentum encoder. Hansen et al. [27] predict an inverse model as an auxiliary task for representation learning. Another type of approaches aims to apply data augmentation techniques to improve representation learning. For instance, Laskin et al. [48] conduct an extensive study of data augmentation for deep RL with pixel-based input. Yarats et al. [89] adopt random crops on pixel-based input and add regularization terms on the Q-function objectives. Lee et al. [53] introduce convolutional neural networks with randomized parameters. Stooke et al. [77] use augmented samples for representation contrastive learning. In contrast to these works, our proposed method aims to encode state representations into a structural metric space based on behavioral metrics.

Recently, behavioral metric-based representation learning has attracted more and more attention in the RL community. For instance, Zhang et al. [94] aim to learn representations by approximating the bisimulation metric [18, 4] on an embedding space. Agarwal et al. [2] propose a behavioral metric considering a distance between action distributions given states for representation learning.

Kemertas and Aumentado-Armstrong [40] propose to improve the robustness of the representation learning method proposed in [94] by adding norm constraints on the embedding space and intrinsic rewards. Castro et al. [5] introduce a new behavioral distance and develop a sampling-based approach to preserve the behavioral similarity between states on the embedding space.

Chen and Pan [6] propose to learn neural networks to approximate components in the bisimulation metric on the state embedding space.

5.4 Preliminaries

5.4.1 Reinforcement Learning

We consider a Markov Decision Process (MDP) defined by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, P, \gamma \rangle$, where $\mathcal{S}$ is the high-dimensional state space, $\mathcal{A}$ is the action space, $P(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ is the transition distribution that captures the probability of entering a next state $\mathbf{s}' \in \mathcal{S}$ given a current state $\mathbf{s} \in \mathcal{S}$ and an action $a \in \mathcal{A}$, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function and $\gamma \in [0, 1)$ is the discount factor. In the sequel, we use $P^a_{\mathbf{s}}$ and $r^a_{\mathbf{s}}$ to denote $P(\cdot|\mathbf{s}, a)$ and $\mathcal{R}(\mathbf{s}, a)$, respectively. A policy $\pi(a|\mathbf{s})$ is a probability distribution over each action a conditioned on a state $\mathbf{s}$. The value function $V^\pi : \mathcal{S} \to \mathbb{R}$ for a given policy π at a state $\mathbf{s}$ is defined as the expected sum of discounted future rewards,

$$V^\pi(\mathbf{s}) = \mathop{\mathbb{E}}_{\substack{\mathbf{a}_t \sim \pi(\cdot|\mathbf{s}_t) \\ \mathbf{s}_{t+1} \sim P^{\mathbf{a}_t}_{\mathbf{s}_t}}} \left[\sum_{t=0}^{\infty} \gamma^t r^{\mathbf{a}_t}_{\mathbf{s}_t} \mid \mathbf{s}_0 = \mathbf{s} \right].$$

The goal of reinforcement learning is to find an optimal policy $\pi^* = \arg\max_\pi V^\pi$ that maximizes the expected future rewards. In the scope of representation learning for deep RL, a state encoder $\phi : \mathcal{S} \to \mathbb{R}^n$ maps a high-dimensional state to a low-dimensional vector, with which a policy $\pi(a|\phi(\mathbf{s}))$ is learned.

5.4.2 Bisimulation Metrics

The bisimulation metric [18, 19] defines a pseudometric $d : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$ to measure the behavioral distance between states. Recently, a variant of the bisimulation metric, *on-policy* bisimulation metric (or π-bisimulation metric) is proposed [4], which focuses on behaviors relative to a particular policy π. The π-bisimulation metric consists of a reward difference term and a Wasserstein distance in dynamics models between states.

Theorem 5.4.1 (π-bisimulation metric [4])**.** *Let $\mathbb{M}$ be the set of all pseudometrics on space $\mathcal{S}$. A pseudometric transformation function $\mathcal{F}^\pi_B : \mathbb{M} \to \mathbb{M}$ is defined as,*

$$\mathcal{F}^\pi_B(d)(\mathbf{s}_i, \mathbf{s}_j) = |\, \mathbb{E}_{\mathbf{a}_i \sim \pi}\, r^{\mathbf{a}_i}_{\mathbf{s}_i} - \mathbb{E}_{\mathbf{a}_j \sim \pi}\, r^{\mathbf{a}_j}_{\mathbf{s}_j} | + \gamma W_1(d)(P^\pi_{\mathbf{s}_i}, P^\pi_{\mathbf{s}_j}) \tag{5.1}$$

where $\mathbb{E}_{\mathbf{a}_i \sim \pi}\, r^{\mathbf{a}_i}_{\mathbf{s}_i} = \mathbb{E}_{\mathbf{a}_i \sim \pi(\cdot|\mathbf{s}_i)}\, r^{\mathbf{a}_i}_{\mathbf{s}_i}$, $P^\pi_{\mathbf{s}_i} = \mathbb{E}_{a \sim \pi(\cdot|\mathbf{s}_i)}\, P^a_{\mathbf{s}_i}$ and W_1 is the 1-Wasserstein distance. $\mathcal{F}^\pi_B$ has a least fixed point d^π_B and d^π_B is a π-bisimulation metric.

The following theorem shows that the difference in value function is bounded by d_B^π.

Theorem 5.4.2 (Value difference bound [4]). *Given states $\mathbf{s}_i$ and $\mathbf{s}_j$, and policy π, we have*

$$|V^\pi(\mathbf{s}_i) - V^\pi(\mathbf{s}_j)| \leq d_B^\pi(\mathbf{s}_i, \mathbf{s}_j). \tag{5.2}$$

MICo distance The MICo distance [5] is a variant of the π-bisimulation metric, which measures the distribution distance between dynamics models by computing the distance between sampled next states from the dynamics models in order to avoid the computation of the Wasserstein distance.

Theorem 5.4.3 (MICo distance [5]). *Let $\mathbb{M}$ be the space of distance function $d : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$, if the MICo metric function $\mathcal{F}_M^\pi : \mathbb{M} \to \mathbb{M}$ is defined as,*

$$\mathcal{F}_M^\pi(d)(\mathbf{s}_i, \mathbf{s}_j) = |\, \mathbb{E}_{\mathbf{a}_i \sim \pi}\, r_{\mathbf{s}_i}^{\mathbf{a}_i} - \mathbb{E}_{\mathbf{a}_j \sim \pi}\, r_{\mathbf{s}_j}^{\mathbf{a}_j}| + \gamma \mathbb{E}_{\substack{\mathbf{s}_i' \sim P_{\mathbf{s}_i}^\pi \\ \mathbf{s}_j' \sim P_{\mathbf{s}_j}^\pi}} d(\mathbf{s}_i', \mathbf{s}_j'), \tag{5.3}$$

then $\mathcal{F}_M^\pi$ has a unique fixed point d_M^π.

Theorem 5.4.4 (Value difference bound [5]). *Given states $\mathbf{s}_i$ and $\mathbf{s}_j$, and policy π, we have*

$$|V^\pi(\mathbf{s}_i) - V^\pi(\mathbf{s}_j)| \leq d_M^\pi(\mathbf{s}_i, \mathbf{s}_j). \tag{5.4}$$

5.5 Approximation Gaps in Behavioral Metric-based Representation Learning

The high-level idea of behavioral metric-based representation learning is to learn an embedding space such that after mapping states onto the embedding space, the behavioral similarity can be preserved. We denote the state encoder by $\phi_\omega : \mathcal{S} \to \mathbb{R}^n$ with parameters ω and the distance between states on the embedding space $\mathbb{R}^n$ by $\hat{d}(\phi_\omega(\mathbf{s}_i), \phi_\omega(\mathbf{s}_j))$, e.g., $\hat{d}$ can be the L_2 norm distance. The problem of representation learning in terms of ω can be cast as a minimization problem of the following expected squared difference or loss between the distance on the embedding space, $\hat{d}(\phi_\omega(\mathbf{s}_i), \phi_\omega(\mathbf{s}_j))$, and the corresponding behavior metric, $d^\pi(\mathbf{s}_i, \mathbf{s}_j)$, between any pair of states $\mathbf{s}_i$ and $\mathbf{s}_j$:

$$\mathcal{L}(\phi_\omega) = \mathbb{E}\left[\left(\hat{d}(\phi_\omega(\mathbf{s}_i), \phi_\omega(\mathbf{s}_j)) - d^\pi(\mathbf{s}_i, \mathbf{s}_j)\right)^2\right]. \tag{5.5}$$

To develop a practical algorithm to minimize the above objective, prior approaches adopt different approximation or relaxation strategies to make the resultant optimization problem computationally tractable. As discussed above, there are three common issues underlying most prior approaches, which introduce gaps between the practical algorithms and their theoretical prototypes, i.e., (5.5), namely loss function mismatch, relaxation of dynamics model divergence, and the L_1 or the L_2 norm limitation. We discuss these 3 gaps in detail in the following sections.

5.5.1 Loss Function Mismatch

In general, to learn the encoder ϕ_ω by minimizing (5.5) is computationally intractable or expensive because of the computation of the behavior metric d^π. We take MICo (MICo-based representation learning [5]) as an example. By specifying the term $d^\pi(\mathbf{s}_i, \mathbf{s}_j)$ in (5.5) as the MICo distance defined in (5.3), the loss of MICo becomes,

$$\mathcal{L}(\phi_\omega) = \mathbb{E}\left[\left(\hat{d}(\phi_\omega(\mathbf{s}_i), \phi_\omega(\mathbf{s}_j)) - \left|\mathbb{E}_{\mathbf{a}_i\sim\pi} r_{\mathbf{s}_i}^{\mathbf{a}_i} - \mathbb{E}_{\mathbf{a}_j\sim\pi} r_{\mathbf{s}_j}^{\mathbf{a}_j}\right| - \gamma \mathbb{E}_{\substack{\mathbf{s}'_i\sim P_{\mathbf{s}_i}^\pi \\ \mathbf{s}'_j\sim P_{\mathbf{s}_j}^\pi}} \hat{d}(\phi_{\bar{\omega}}(\mathbf{s}_i), \phi_{\bar{\omega}}(\mathbf{s}_j))\right)^2\right], \tag{5.6}$$

where $\bar{\omega}$ is a copy of parameters for the target network. However, the reward expectations $\mathbb{E}_{\mathbf{a}_i\sim\pi} r_{\mathbf{s}_i}^{\mathbf{a}_i}$ and $\mathbb{E}_{\mathbf{a}_j\sim\pi} r_{\mathbf{s}_j}^{\mathbf{a}_j}$ in the 2nd term in (5.6) are computationally intractable and also difficult to estimate even based on sampling. Thus, Castro et al. [5] propose to approximate the loss in (5.6) with the following alternative loss,

$$\mathcal{L}(\phi_\omega) = \mathop{\mathbb{E}}_{\substack{\mathbf{a}_i\sim\pi, \mathbf{a}_j\sim\pi \\ \mathbf{s}'_i\sim P_{\mathbf{s}_i}^{\mathbf{a}_i}, \mathbf{s}'_j\sim P_{\mathbf{s}_j}^{\mathbf{a}_j}}} \left[\left(\hat{d}(\phi_\omega(\mathbf{s}_i), \phi_\omega(\mathbf{s}_j)) - \left|r_{\mathbf{s}_i}^{\mathbf{a}_i} - r_{\mathbf{s}_j}^{\mathbf{a}_j}\right| - \gamma \hat{d}(\phi_{\bar{\omega}}(\mathbf{s}_i), \phi_{\bar{\omega}}(\mathbf{s}_j))\right)^2\right]. \tag{5.7}$$

Note that in (5.7), the expectation operator on rewards is moved out of the absolute value operator of the difference between rewards to avoid the estimation of expectation over rewards, $\mathbb{E}_{\mathbf{a}_i\sim\pi} r_{\mathbf{s}_i}^{\mathbf{a}_i}$ and $\mathbb{E}_{\mathbf{a}_j\sim\pi} r_{\mathbf{s}_j}^{\mathbf{a}_j}$, and enable sampling more efficient. However, such a revision introduces a gap between solutions of minimizing (5.7) and (5.6), because in (5.7) the reference behavioral metric is no longer the MICo distance but the "shift" MICo distance defined as follows.

Definition 5.5.1 (Shift MICo distance)**.** The shift MICo distance function $\tilde{\mathcal{F}}_M^\pi$ is defined as

$$\tilde{\mathcal{F}}_M^\pi(d)(\mathbf{s}_i, \mathbf{s}_j) = \mathbb{E}_{\substack{\mathbf{a}_i\sim\pi \\ \mathbf{a}_j\sim\pi}} |r_{\mathbf{s}_i}^{\mathbf{a}_i} - r_{\mathbf{s}_j}^{\mathbf{a}_j}| + \gamma \mathbb{E}_{\substack{\mathbf{s}'_i\sim P_{\mathbf{s}_i}^\pi \\ \mathbf{s}'_j\sim P_{\mathbf{s}_j}^\pi}} d(\mathbf{s}'_i, \mathbf{s}'_j).$$

Lemma 5.5.2 (Fixed-point of shift MICo)**.** *The shift MICo distance function $\tilde{\mathcal{F}}^{\pi}_{M}$ has a unique fixed-point $\tilde{d^{\pi}_{M}}$.*

Proof: This can be proved by following the proof of Theorem 5.4.3 by using Banach's fixed-point theorem. Let $d, d' \in \mathbb{M}$. We have

$$\begin{aligned}
&\left|\tilde{\mathcal{F}}^{\pi}_{M}(d)(\mathbf{s}_i, \mathbf{s}_j) - \tilde{\mathcal{F}}^{\pi}_{M}(d')(\mathbf{s}_i, \mathbf{s}_j)\right| \\
=&\left|\gamma \sum_{\mathbf{s}'_i, \mathbf{s}'_j} \pi(\mathbf{a}_i|\mathbf{s}_i)\pi(\mathbf{a}_j|\mathbf{s}_j) P^{\mathbf{a}_i}_{\mathbf{s}_i}(\mathbf{s}'_i) P^{\mathbf{a}_j}_{\mathbf{s}_j}(\mathbf{s}'_j)(d - d')(s'_i, s'_j)\right| \\
\leq& \gamma \|d - d'\|_{\infty}.
\end{aligned}$$

$\tilde{\mathcal{F}}^{\pi}_{M}$ is a contraction mapping w.r.t. the L_∞ norm and there exists a unique fixed-point $\tilde{d^{\pi}_{M}}$ for $\tilde{\mathcal{F}}^{\pi}_{M}$ due to Banach's fixed-point theorem. This completes the proof.

The above lemma shows that the approximated distance on the embedding space $\hat{d}(\phi_\omega(\mathbf{s}_i), \phi_\omega(\mathbf{s}_j))$ still converges to the Shift MICo distance, $\tilde{d^{\pi}_{M}}$, by minimizing (5.7). However, as $\mathbb{E}_{\substack{\mathbf{a}_i \sim \pi \\ \mathbf{a}_j \sim \pi}} |r^{\mathbf{a}_i}_{\mathbf{s}_i} - r^{\mathbf{a}_j}_{\mathbf{s}_j}| \geq |\mathbb{E}_{\mathbf{a}_i \sim \pi} r^{\mathbf{a}_i}_{\mathbf{s}_i} - \mathbb{E}_{\mathbf{a}_j \sim \pi} r^{\mathbf{a}_j}_{\mathbf{s}_j}|$, we have the following theorem, whose proof can be found in Section 5.9.1.

Theorem 5.5.3 (Looser value difference bound)**.** *Given states $\mathbf{s}_i$ and $\mathbf{s}_j$, and a policy π, we have*

$$|V^{\pi}(\mathbf{s}_i) - V^{\pi}(\mathbf{s}_j)| \leq d^{\pi}_{M}(\mathbf{s}_i, \mathbf{s}_j) \leq \tilde{d^{\pi}_{M}}(\mathbf{s}_i, \mathbf{s}_j). \tag{5.8}$$

Based on the above theorem, as the Shift MICo distance has looser value difference bound, it may be less relevant to the value function. As a result, the learned representation may not be able to encode the behavioral similarity between states accurately. Apart from MICo, other behavioral metric-based methods, which consist of the on-policy reward difference term, such as DBC [94, 40] and AMBS [6], also suffer from a similar approximation gap.

5.5.2 Relaxation of Dynamics Models Divergence

The π-bisimulation metric needs to compute the 1-Wasserstein distance W_1 between dynamics models to measure the distribution distance. In π-bisimulation metric-based representation learning, one can learn the encoder ϕ_ω by minimizing the following loss,

$$\mathcal{L}(\phi_\omega) = \mathbb{E}\left[\left(\hat{d}(\phi_\omega(\mathbf{s}_i), \phi_\omega(\mathbf{s}_j)) - \left|\mathbb{E}_{\mathbf{a}_i \sim \pi} r^{\mathbf{a}_i}_{\mathbf{s}_i} - \mathbb{E}_{\mathbf{a}_j \sim \pi} r^{\mathbf{a}_j}_{\mathbf{s}_j}\right| - \gamma W_1(\hat{d})(P^{\pi}_{\phi_\omega(\mathbf{s}_i)}, P^{\pi}_{\phi_\omega(\mathbf{s}_j)})\right)^2\right],$$

However, the 1-Wasserstein distance is computationally expensive or intractable. In DBC [94] the 2-Wasserstein distance W_2 is proposed to replace W_1, as W_2 has a convenient closed-form of a Gaussian distribution with respect to the L_2 distance. Specifically, the loss function of DBC with batched sampled transitions is defined as,

$$\mathcal{L}(\phi_\omega) = \mathbb{E}\left[\left(\hat{d}(\phi_\omega(\mathbf{s}_i), \phi_\omega(\mathbf{s}_j)) - \left|r_{\mathbf{s}_i}^{\mathbf{a}_i} - r_{\mathbf{s}_j}^{\mathbf{a}_j}\right| - \gamma W_2(\|\cdot\|_2)(\widehat{P}^{\bar{\pi}}_{\phi_\omega(\mathbf{s}_i)}, \widehat{P}^{\bar{\pi}}_{\phi_\omega(\mathbf{s}_j)})\right)^2\right],$$

where $\|\cdot\|_2$ is the L_2 norm, $\widehat{P}$ is a dynamics model on the representation space, and $\bar{\pi}$ is the expected policy output. The use of W_2 almost breaks all theoretical guarantees for the bisimulation metric. The existence of unique fixed-point in the bisimulation metric requires the continuity and monotonicity of W_1 with respect to d [18]. The properties of continuity and monotonicity do not hold with W_2. Therefore, there is no more guarantee about the fixed-point existence in DBC except that both the dynamics model and the policy π are deterministic, in which case $W_2(d)$ degenerates to d and Banach's fixed-point exists [40]. However, this assumption may be too strong to hold in practice.

Instead of using the family of Wasserstein distances, in MICo as shown in (5.3) the sample-based distribution divergence, $\mathbb{E}_{\mathbf{s}'_i \sim P^\pi_{\mathbf{s}_i}, \mathbf{s}'_j \sim P^\pi_{\mathbf{s}_j}} d(\mathbf{s}'_i, \mathbf{s}'_j)$, is used to measure the difference between dynamics models. This sample-based distribution divergence can be considered as a Łukaszyk-Karmowski metric. While a Wasserstein distance has zero self-distance, a Łukaszyk-Karmowski metric does not satisfy the identity of indiscernibles. As a result, the approximated distance on the learned embedding space based on the MICo distance, which involves a Łukaszyk-Karmowski metric to measure distance between dynamics models, may also suffer from the violation issue of the identity of indiscernibles.

5.5.3 Limitation of Using the L_1 or the L_2 Norm on the Embedding Space

As mentioned in Section 5.5.1, to avoid the expensive or intractable computation of the expectation over rewards, prior approaches, such as MICo, and DBC, use an alternative term to measure the reward difference between states, $\mathbb{E}_{\substack{\mathbf{a}_i \sim \pi \\ \mathbf{a}_j \sim \pi}} |r_{\mathbf{s}_i}^{\mathbf{a}_i} - r_{\mathbf{s}_j}^{\mathbf{a}_j}|$. In the following, we discuss the case that state $\mathbf{s}_i$ and $\mathbf{s}_j$ are identical.

Lemma 5.5.4. *If $\mathbf{s}_i = \mathbf{s}_j$, then $\mathbb{E}_{\substack{\mathbf{a}_i \sim \pi \\ \mathbf{a}_j \sim \pi}} |r_{\mathbf{s}_i}^{\mathbf{a}_i} - r_{\mathbf{s}_j}^{\mathbf{a}_j}| \geq 0$. The equality holds only if the reward function r is constant w.r.t. action a or the policy π is deterministic.*

Note that in most RL tasks, a stochastic policy is widely used for exploration and a reward function is rarely constant w.r.t. actions. Therefore, in practice, $\mathbb{E}_{\substack{\mathbf{a}_i\sim\pi \\ \mathbf{a}_j\sim\pi}}|r_{\mathbf{s}_i}^{\mathbf{a}_i} - r_{\mathbf{s}_j}^{\mathbf{a}_j}| > 0$ for most RL tasks. Besides, as mentioned in Section 5.5.2, a Łukaszyk-Karmowski metric measuring distance between dynamics models does not satisfy the identity of indiscernibles. Therefore, the behavioral metric, which is a sum of the reward difference term and the dynamics model distance term (no matter a Wasserstein distance or a Łukaszyk-Karmowski metric), can be greater than zero on a pair of identical states.

Both the L_1 and the L_2 norms satisfy the identity of indiscernibles, i.e. when $\mathbf{x}_i = \mathbf{x}_j$, $\|\mathbf{x}_i - \mathbf{x}_j\|_1 = \|\mathbf{x}_i - \mathbf{x}_j\|_2 = 0$. If we leverage the L_1 or the L_2 norm as the form of distance $\hat{d}$ on the embedding space to approximate the behavioral metric, then the identical states pair has zero distance on embedding space. However, the regression target, i.e., the behavioral metric, is greater than zero on identical states. As a result, when minimizing regression loss between the L_1 / L_2 norm and the behavioral metric, the representations for similar states, especially identical ones, will be pushed apart in the embedding space.

Recently, AMBS [6] proposes to use neural networks to measure distance on state embedding space rather than using the L_1 or the L_2 norm. However, in this case, the learned "behavior metric" does not have theoretical support such as the fixed-point convergence guarantee. Besides, MICo proposes a new form of distance on the embedding space, which is a sum of angular distance between embeddings and the L_2 norm of the embeddings. Note that the proposed distance has non-zero self-distance.

5.6 The Proposed RAP Distance

We firstly propose a new behavioral metric to measure the state similarity without computing the Wasserstein distance between dynamics models in Section 5.6.1. The proposed behavioral metric namely the RAP distance enjoys theoretical properties such as fixed-point existence and a value difference bound. We then present a practical algorithm to learn the state encoder by approximating the RAP distance on the state embedding space in Section 5.6.2. Our algorithm uses the learned estimation of reward functions and dynamics models to provide distance approximation which is consistent to the behavioral metric. It addresses all the aforementioned approximation gaps and preserves

the theoretical guarantee about the value function difference bound. Particularly, the approximation gap of loss function mismatch is addressed in Section 5.6.2, the relaxation of dynamics model divergence is addressed in Section 5.6.1, and the limitation of the L_1 or the L_2 norm on the embedding space is addressed in Section 5.6.3.

5.6.1 Definition and Properties of the RAP Distance

In order to avoid the high computational cost of W_1 or the approximation gap introduced by relaxation to W_2 as described in Section 5.5.2, we consider a distance measure between dynamics models $P^{\pi}_{\mathbf{s}_i}$ and $P^{\pi}_{\mathbf{s}_j}$ with sampling. To be specific, our on-policy behavioral distance is defined as follows.

Definition 5.6.1 (the RAP distance)**.** Let $\mathbb{M}$ be the space of distance function $d : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$, the RAP distance function $\mathcal{F}^{\pi}_{G} : \mathbb{M} \to \mathbb{M}$ is defined as,

$$\mathcal{F}^{\pi}_{G}(d)(\mathbf{s}_i, \mathbf{s}_j) = \left| \mathbb{E}_{\mathbf{a}_i \sim \pi} r^{\mathbf{a}_i}_{\mathbf{s}_i} - \mathbb{E}_{\mathbf{a}_j \sim \pi} r^{\mathbf{a}_j}_{\mathbf{s}_j} \right| + \gamma \mathbb{E}_{\substack{\mathbf{a}_i \sim \pi \\ \mathbf{a}_j \sim \pi}} d(\mathbb{E}[s'_i], \mathbb{E}[s'_j]), \tag{5.9}$$

where $\mathbb{E}[s'_i] = \mathbb{E}_{s'_i \sim P^{\mathbf{a}_i}_{\mathbf{s}_i}}[s'_i]$ is the expectation value of next state over the dynamics model $P^{\mathbf{a}_i}_{\mathbf{s}_i}$.

We design the behavioral distance by measuring the expected states over dynamics models recursively, which removes the requirement of sampling on $P^{\mathbf{a}_i}_{\mathbf{s}_i}$ and $P^{\mathbf{a}_j}_{\mathbf{s}_j}$ but only performs sampling on the policy π. In practice, the expected next states are generated by a learnable approximated dynamics model as described in Section 5.6.2.

Theorem 5.6.2. *$\mathcal{F}^{\pi}_{G}$ is a contraction mapping w.r.t. the L_∞ norm and has a unique fixed-point d^{π}_{G}.*

Proof: Let $d, d' \in \mathbb{M}$. We have

$$|\mathcal{F}^{\pi}_{G}(d)(\mathbf{s}_i, \mathbf{s}_j) - \mathcal{F}^{\pi}_{G}(d')(\mathbf{s}_i, \mathbf{s}_j)| = \left| \gamma \sum_{\mathbf{a}_i, \mathbf{a}_j} \pi(\mathbf{a}_i|\mathbf{s}_i)\pi(\mathbf{a}_j|\mathbf{s}_j)(d - d')(\mathbb{E}[s'_i], \mathbb{E}[s'_j]) \right| \leq \gamma \|d - d'\|_\infty.$$

Therefore, $\mathcal{F}^{\pi}_{G}$ is a contraction mapping w.r.t. the L_∞ norm and there exists a unique fixed-point for $\mathcal{F}^{\pi}_{G}$ due to Banach's fixed-point theorem. This completes the proof.

Theorem 5.6.2 provides a convergence guarantee for the RAP distance that by iterating $\mathcal{F}_G^\pi$, distance d will converge to the fixed-point d_G^π.

Theorem 5.6.3 (Value function difference bound)**.** *Given states* $\mathbf{s}_i$ *and* $\mathbf{s}_j$*, and a policy* π*, we have*

$$|V^\pi(\mathbf{s}_i) - V^\pi(\mathbf{s}_j)| \leq d_G^\pi(\mathbf{s}_i, \mathbf{s}_j). \tag{5.10}$$

The proof can be found in Section 5.9.2. Theorem 5.6.3 demonstrates that the RAP distance between states upper-bounds the difference of their states values.

5.6.2 Approximation of RAP

A straightforward way to learn a representation encoder to approximate the RAP distance on the embedding space is to minimize the loss in (5.5). However, the approximation gap of loss function mismatch as mentioned in Section 5.5.1 still exists if we relax the first term in (5.9) as $\mathbb{E}_{\substack{\mathbf{a}_i\sim\pi\\ \mathbf{a}_j\sim\pi}}|r_{\mathbf{s}_i}^{\mathbf{a}_i} - r_{\mathbf{s}_j}^{\mathbf{a}_j}|$. This leads to the learned metric having a looser value difference bound than the original behavioral metric as proved in Section 5.5.1. Here, we propose another alternative relaxation of $|\,\mathbb{E}_{\mathbf{a}_i\sim\pi} r_{\mathbf{s}_i}^{\mathbf{a}_i} - \mathbb{E}_{\mathbf{a}_j\sim\pi} r_{\mathbf{s}_j}^{\mathbf{a}_j}|$ to address this issue. Let $r_{\mathbf{s}}$ be a random variable over the action distribution defined by $p(r_{\mathbf{s}} = r_{\mathbf{s}}^{\mathbf{a}}) = \pi(\mathbf{a}|\mathbf{s})$. We first analyze the difference between $\mathbb{E}_{\substack{\mathbf{a}_i\sim\pi\\ \mathbf{a}_j\sim\pi}}\left[\left|r_{\mathbf{s}_i}^{\mathbf{a}_i} - r_{\mathbf{s}_j}^{\mathbf{a}_j}\right|^2\right]$ and $|\,\mathbb{E}_{\mathbf{a}_i\sim\pi} r_{\mathbf{s}_i}^{\mathbf{a}_i} - \mathbb{E}_{\mathbf{a}_j\sim\pi} r_{\mathbf{s}_j}^{\mathbf{a}_j}|^2$,

$$\begin{aligned}
&\mathbb{E}_{\substack{\mathbf{a}_i\sim\pi\\ \mathbf{a}_j\sim\pi}}\left[\left|r_{\mathbf{s}_i}^{\mathbf{a}_i} - r_{\mathbf{s}_j}^{\mathbf{a}_j}\right|^2\right] - |\,\mathbb{E}_{\mathbf{a}_i\sim\pi} r_{\mathbf{s}_i}^{\mathbf{a}_i} - \mathbb{E}_{\mathbf{a}_j\sim\pi} r_{\mathbf{s}_j}^{\mathbf{a}_j}|^2 \\
&= \mathbb{E}_{\mathbf{a}_i\sim\pi}\left[(r_{\mathbf{s}_i}^{\mathbf{a}_i})^2\right] + \mathbb{E}_{\mathbf{a}_j\sim\pi}\left[(r_{\mathbf{s}_j}^{\mathbf{a}_j})^2\right] - 2\mathbb{E}_{\substack{\mathbf{a}_i\sim\pi\\ \mathbf{a}_j\sim\pi}}\left[r_{\mathbf{s}_i}^{\mathbf{a}_i} r_{\mathbf{s}_j}^{\mathbf{a}_j}\right] - \left[\mathbb{E}_{\mathbf{a}_i\sim\pi} r_{\mathbf{s}_i}^{\mathbf{a}_i}\right]^2 - \left[\mathbb{E}_{\mathbf{a}_j\sim\pi} r_{\mathbf{s}_j}^{\mathbf{a}_j}\right]^2 + 2\left[\mathbb{E}_{\mathbf{a}_i\sim\pi} r_{\mathbf{s}_i}^{\mathbf{a}_i}\right]\left[\mathbb{E}_{\mathbf{a}_j\sim\pi} r_{\mathbf{s}_j}^{\mathbf{a}_j}\right] \\
&= \mathbb{E}_{\mathbf{a}_i\sim\pi}\left[(r_{\mathbf{s}_i}^{\mathbf{a}_i})^2\right] - \left[\mathbb{E}_{\mathbf{a}_i\sim\pi} r_{\mathbf{s}_i}^{\mathbf{a}_i}\right]^2 + \mathbb{E}_{\mathbf{a}_j\sim\pi}\left[(r_{\mathbf{s}_j}^{\mathbf{a}_j})^2\right] - \left[\mathbb{E}_{\mathbf{a}_j\sim\pi} r_{\mathbf{s}_j}^{\mathbf{a}_j}\right]^2 - 2\mathbb{E}_{\substack{\mathbf{a}_i\sim\pi\\ \mathbf{a}_j\sim\pi}}\left[r_{\mathbf{s}_i}^{\mathbf{a}_i} r_{\mathbf{s}_j}^{\mathbf{a}_j}\right] + 2\left[\mathbb{E}_{\mathbf{a}_i\sim\pi} r_{\mathbf{s}_i}^{\mathbf{a}_i}\right]\left[\mathbb{E}_{\mathbf{a}_j\sim\pi} r_{\mathbf{s}_j}^{\mathbf{a}_j}\right] \\
&= Var[r_{\mathbf{s}_i}] + Var[r_{\mathbf{s}_j}] - 2Cov[r_{\mathbf{s}_i}, r_{\mathbf{s}_j}].
\end{aligned} \tag{5.11}$$

where $Var[\cdot]$ is the variance and $Cov[\cdot,\cdot]$ is the covariance. Since $r_{\mathbf{s}_i}$ and $r_{\mathbf{s}_j}$ are independent, $Cov[r_{\mathbf{s}_i}, r_{\mathbf{s}_j}] = 0$. Therefore, we have the reward difference term

$$|\,\mathbb{E}_{\mathbf{a}_i\sim\pi} r_{\mathbf{s}_i}^{\mathbf{a}_i} - \mathbb{E}_{\mathbf{a}_j\sim\pi} r_{\mathbf{s}_j}^{\mathbf{a}_j}| = \sqrt{\mathbb{E}_{\substack{\mathbf{a}_i\sim\pi\\ \mathbf{a}_j\sim\pi}}\left[\left|r_{\mathbf{s}_i}^{\mathbf{a}_i} - r_{\mathbf{s}_j}^{\mathbf{a}_j}\right|^2\right] - Var[r_{\mathbf{s}_i}] - Var[r_{\mathbf{s}_j}]}$$

and the revised RAP distance at fixed-point as

$$d_G^\pi(\mathbf{s}_i, \mathbf{s}_j) = \sqrt{\mathbb{E}_{\substack{\mathbf{a}_i\sim\pi \\ \mathbf{a}_j\sim\pi}}\left[\left|r_{\mathbf{s}_i}^{\mathbf{a}_i} - r_{\mathbf{s}_j}^{\mathbf{a}_j}\right|^2\right] - Var[r_{\mathbf{s}_i}] - Var[r_{\mathbf{s}_j}]} \\ + \gamma\mathbb{E}_{\substack{\mathbf{a}_i\sim\pi \\ \mathbf{a}_j\sim\pi}} d_G^\pi(\mathbb{E}_{s_i'\sim P_{\mathbf{s}_i}^{\mathbf{a}_i}}[s_i'], \mathbb{E}_{s_j'\sim P_{\mathbf{s}_j}^{\mathbf{a}_j}}[s_j']). \quad (5.12)$$

In (5.12), we successfully move the expectation operator on rewards out of the absolute value operator without introducing any approximation gap caused by loss function mismatch. However, there are three issues that need to be further addressed: 1) the square root introduces new bias under sampling, 2) the variances $Var[r_{\mathbf{s}_i}]$ and $Var[r_{\mathbf{s}_j}]$ are intractable to compute, and 3) how to estimate the expected next states $\mathbb{E}_{s_i'\sim P_{\mathbf{s}_i}^{\mathbf{a}_i}}[s_i']$ and $\mathbb{E}_{s_j'\sim P_{\mathbf{s}_j}^{\mathbf{a}_j}}[s_j']$.

In order to reduce the bias issue introduced by the square root, we try to remove the square root in the loss of learning the RAP distance. We move the dynamics term in (5.12) to the left-hand side, then take square on both sides and get

$$\left(d_G^\pi(\mathbf{s}_i, \mathbf{s}_j) - \gamma\mathbb{E}_{\substack{\mathbf{a}_i\sim\pi \\ \mathbf{a}_j\sim\pi}} d_G^\pi(\mathbb{E}_{s_i'\sim P_{\mathbf{s}_i}^{\mathbf{a}_i}}[s_i'], \mathbb{E}_{s_j'\sim P_{\mathbf{s}_j}^{\mathbf{a}_j}}[s_j'])\right)^2 = \mathbb{E}_{\substack{\mathbf{a}_i\sim\pi \\ \mathbf{a}_j\sim\pi}}\left[\left|r_{\mathbf{s}_i}^{\mathbf{a}_i} - r_{\mathbf{s}_j}^{\mathbf{a}_j}\right|^2\right] - Var[r_{\mathbf{s}_i}] - Var[r_{\mathbf{s}_j}]. \quad (5.13)$$

Let $\hat{d}(\phi_\omega(\mathbf{s}_i), \phi_\omega(\mathbf{s}_j))$ be the approximated RAP distance between $\mathbf{s}_i$ and $\mathbf{s}_j$ parameterized by ω. The loss for learning $\hat{d}(\phi_\omega(\mathbf{s}_i), \phi_\omega(\mathbf{s}_j))$ is to minimize the mean squared error between the left-hand side and the right-hand side in (5.13):

$$\mathcal{L} = \mathbb{E}\Bigg[\left(\hat{d}(\phi_\omega(\mathbf{s}_i), \phi_\omega(\mathbf{s}_j)) - \gamma\mathbb{E}_{\substack{\mathbf{a}_i\sim\pi \\ \mathbf{a}_j\sim\pi}} d_G^\pi(\mathbb{E}_{s_i'\sim P_{\mathbf{s}_i}^{\mathbf{a}_i}}[s_i'], \mathbb{E}_{s_j'\sim P_{\mathbf{s}_j}^{\mathbf{a}_j}}[s_j'])\right)^2 \\ - \left(|r_{\mathbf{s}_i}^{\mathbf{a}_i} - r_{\mathbf{s}_j}^{\mathbf{a}_j}|^2 - Var[r_{\mathbf{s}_i}] - Var[r_{\mathbf{s}_j}]\right)\Bigg]^2. \quad (5.14)$$

Such a loss consists of a pair of squared difference terms as the regression target, which approximate the squared difference between distances of current states and distances of dynamics models, respectively.

The reward variance $Var[r_{\mathbf{s}}]$ is computationally intractable, but we can learn a neural network approximator to estimate it by assuming that the reward $r_{\mathbf{s}}$ on state $\mathbf{s}$ is Gaussian distributed. Let r_ψ be the learned reward function approximation parameterized by ψ, which outputs a Gaussian distribution, $r_\psi(\mathbf{s}) = \{\widehat{\mu}(r_{\mathbf{s}}), \widehat{\sigma}(r_{\mathbf{s}})\}$, where $\widehat{\mu}(r_{\mathbf{s}})$ and $\widehat{\sigma}(r_{\mathbf{s}})$ are the mean and the standard deviation, respectively.

Similarly, in order to estimate the expected next states $\mathbb{E}_{s' \sim P^{\mathbf{a}}_{\mathbf{s}}}[s']$ with a neural network approximator on the embedding space, we learn a dynamics model $\widehat{P}$ taking input of state embedding $\phi(\mathbf{s})$ and action $\mathbf{a}$ and outputs a Gaussian distribution over the next state embedding, $\widehat{P}(\phi_\omega(\mathbf{s}), \mathbf{a}) = \{\widehat{\mu}(\widehat{P}^{\mathbf{a}}_{\phi_\omega(\mathbf{s})}), \widehat{\sigma}(\widehat{P}^{\mathbf{a}}_{\phi_\omega(\mathbf{s})})\}$, where $\widehat{\mu}(\widehat{P}^{\mathbf{a}}_{\phi_\omega(\mathbf{s})})$ and $\widehat{\sigma}(\widehat{P}^{\mathbf{a}}_{\phi_\omega(\mathbf{s})})$ are the mean vector and the standard deviation vector for predictive embeddings of the next state, respectively.

The RAP distance between expected next states, $d^{\pi}_{G}(\mathbb{E}[s'_i], \mathbb{E}[s'_j])$, can be approximated by $\hat{d}(\widehat{\mu}(\widehat{P}^{\mathbf{a}_i}_{\phi_\omega(\mathbf{s}_i)}), \widehat{\mu}(\widehat{P}^{\mathbf{a}_j}_{\phi_\omega(\mathbf{s}_j)}))$. The variance $Var[r_{\mathbf{s}}]$ can be replaced by the learned reward function output as $(\widehat{\sigma}(r_{\mathbf{s}}))^2$. By replacing the dynamics term and reward variance terms in (5.14), we propose the RAP loss defined as

$$\mathcal{L}_{RAP}(\phi_\omega) = \mathbb{E}_{\mathcal{D}}\Bigg[\Big(\hat{d}(\phi_\omega(\mathbf{s}_i), \phi_\omega(\mathbf{s}_j)) - \gamma \hat{d}(\widehat{\mu}(\widehat{P}^{\mathbf{a}_i}_{\phi_{\bar{\omega}}(\mathbf{s}_i)}), \widehat{\mu}(\widehat{P}^{\mathbf{a}_j}_{\phi_{\bar{\omega}}(\mathbf{s}_j)}))\Big)^2 - \Big(|r^{\mathbf{a}_i}_{\mathbf{s}_i} - r^{\mathbf{a}_j}_{\mathbf{s}_j}|^2 - (\widehat{\sigma}(r_{\mathbf{s}_i}))^2 - \big(\widehat{\sigma}(r_{\mathbf{s}_j})\big)^2\Big) \Bigg]^2, \quad (5.15)$$

where $\bar{\omega}$ is a copy of parameter with stop gradient and $\phi_{\bar{\omega}}$ is the encoder in the target network and $\mathcal{D}$ is the replay buffer or the set of data that RL algorithm, e.g. SAC [23], learns from. The loss (5.15) is trained over the transitions sampled from $\mathcal{D}$.

5.6.3 Explicit Form of Distance on the Embedding Space

As discussed in Section 5.5.3, the behavioral metric is usually with non-zero self-distance. Besides, the distance measured between the next-state distributions in the RAP distance is a Łukaszyk-Karmowski metric [59] which also has non-zero self-distance. Here, we adopt the approximation form of distance on the embedding space as proposed in MICo [5].

Definition 5.6.4 (MICo approximation)**.** Let $\hat{d}_G : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ be a distance function on representation space $\mathbb{R}^n$, $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ and $k > 0$. $\hat{d}_G$ is defined as $\hat{d}_G(\mathbf{x}, \mathbf{y}) = \|\mathbf{x}\|_2^2 + \|\mathbf{y}\|_2^2 + k\theta(\mathbf{x}, \mathbf{y})$, where θ is angular function evaluating the absolute angle between vectors $\mathbf{x}$ and $\mathbf{y}$, and k is a hyperparameter. In practice, we set $k = 10^{-5}$ for our method.

As the above form of distance produces non-zero self-distance, with $\hat{d}_G$ the approximation of the RAP distance will not push apart similar states on the embedding space.

Lemma 5.6.5 (Non-zero self-distance)**.** *The self-distance of $\hat{d}_G$ is not restrict to zero: $\hat{d}_G(\mathbf{x}, \mathbf{x}) = 2\|\mathbf{x}\|_2^2 \geq 0$. The equality holds only if all the elements of $\mathbf{x}$ are zero.*

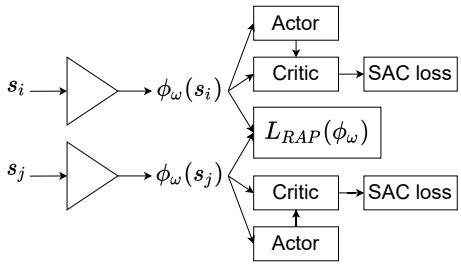


Fig. 5.1: Network architecture of our method.

5.7 Implementation

The network architecture of our method is shown in Figure 5.1. Our method is built upon SAC [23]. Actor and critic networks take input of state representation $\phi_\omega(\mathbf{s})$. The SAC objectives and the RAP regression loss $\mathcal{L}_{RAP}(\phi_\omega)$ are optimized jointly.

5.7.1 Objectives of Soft Actor-Critic

Our approach is built upon Soft Actor-Critic (SAC) [23]. SAC is an actor-critic algorithm that optimizes a stochastic policy in an off-policy manner. We use the version of SAC that incorporates clipped double-Q trick. Let Q_{θ_i} denote the Q-networks where $i = 1, 2$, $Q_{\bar{\theta}_i}$ be the target Q-networks, and π_θ be the actor network. All Q-networks, target Q-networks and actor network take input of state representation $\phi_\omega(\mathbf{s})$ instead of state $\mathbf{s}$. The loss functions for the Q-networks are

$$\mathcal{L}_Q(Q_{\theta_i}, \phi_\omega) = \mathop{\mathbb{E}}_{(\mathbf{s},\mathbf{a},r,\mathbf{s}')\sim\mathcal{D}} \left[\left(Q_{\theta_i}(\phi_\omega(\mathbf{s}), \mathbf{a}) - r - \gamma V_{target}(\mathbf{s}') \right)^2 \right],$$

where $\mathcal{D}$ is the replay buffer, and V_{target} is the target value function. We stop gradients for V_{target} and V_{target} is defined as

$$V_{target}(\mathbf{s}') = \min_{j=1,2} Q_{\bar{\theta}_j}(\phi_{\bar{\omega}}(\mathbf{s}'), \mathbf{a}') - \alpha \log \pi_\theta(\mathbf{a}'|\phi_\omega(\mathbf{s}'))$$

where $\mathbf{a}' \sim \pi_\theta(\cdot|\phi_\omega(\mathbf{s}'))$ is the action at next state. The policy improvement is minimizing the following objective for actor network,

$$\mathcal{J}_\pi(\pi_\theta) = \mathop{\mathbb{E}}_{\mathbf{s}\sim\mathcal{D}} \left[\mathop{\mathbb{E}}_{\mathbf{a}\sim\pi_\theta(\cdot|\phi_\omega(\mathbf{s}))} \left[\alpha \log \pi_\theta(\mathbf{a}|\phi_\omega(\mathbf{s})) - \min_{i=1,2} Q_{\theta_i}(\phi_\omega(\mathbf{s}), \mathbf{a}) \right] \right].$$

The loss function for α of SAC is,

$$\mathcal{J}_\alpha(\alpha) = \mathop{\mathbb{E}}_{\mathbf{s}\sim\mathcal{D}}\left[\mathop{\mathbb{E}}_{\mathbf{a}\sim\pi_\theta(\cdot|\phi_\omega(\mathbf{s}))}[\alpha \log \pi_\theta(\mathbf{a}|\phi_\omega(\mathbf{s})) - \alpha\bar{\mathcal{H}}]\right], \tag{5.16}$$

where $\bar{\mathcal{H}} \in \mathbb{R}$ is the target entropy and $\bar{\mathcal{H}} = -|\mathcal{A}|$ in practice.

5.7.2 Objectives of Reward and Dynamics Model in RAP Distance

We learn the reward function and dynamics model by neural network approximators, in order to approximate RAP distance. Let r_ψ be the learned reward function parameterized by ψ, which outputs a Gaussian distribution, $r_\psi(\mathbf{s}) = \{\widehat{\mu}(r_\mathbf{s}), \widehat{\sigma}(r_\mathbf{s})\}$, where $\widehat{\mu}(r_\mathbf{s})$ and $\widehat{\sigma}(r_\mathbf{s})$ are the mean and the standard deviation, respectively. The loss function for learning r_ψ is

$$\mathcal{L}_r(r_\psi) = \mathop{\mathbb{E}}_{(\mathbf{s},r,)\sim\mathcal{D}}\left[\left(\frac{r - \widehat{\mu}(r_\mathbf{s})}{2\widehat{\sigma}(r_\mathbf{s})}\right)^2\right].$$

We learn a dynamics model $\widehat{P}_\eta$ parameterized by η to take input of state embedding $\phi(\mathbf{s})$ and action $\mathbf{a}$, and output a Gaussian distribution over the next state embedding, $\widehat{P}_\eta(\phi_\omega(\mathbf{s}), \mathbf{a}) = \{\widehat{\mu}(\widehat{P}^\mathbf{a}_{\phi_\omega(\mathbf{s})}), \widehat{\sigma}(\widehat{P}^\mathbf{a}_{\phi_\omega(\mathbf{s})})\}$, where $\widehat{\mu}(\widehat{P}^\mathbf{a}_{\phi_\omega(\mathbf{s})})$ and $\widehat{\sigma}(\widehat{P}^\mathbf{a}_{\phi_\omega(\mathbf{s})})$ are the mean vector and the standard deviation vector for the predictive next state embedding, respectively. The loss function for $\widehat{P}_\eta$ is

$$\mathcal{L}_P(\widehat{P}_\eta) = \mathop{\mathbb{E}}_{(\mathbf{s},\mathbf{a},\mathbf{s}')\sim\mathcal{D}}\left[\left(\frac{\phi_\omega(\mathbf{s}') - \widehat{\mu}(\widehat{P}^\mathbf{a}_{\phi_\omega(\mathbf{s})})}{2\widehat{\sigma}(\widehat{P}^\mathbf{a}_{\phi_\omega(\mathbf{s})})}\right)^2\right].$$

5.7.3 Learning Algorithm

Algorithm shows the learning steps for learning SAC and RAP jointly.

Algorithm 3 Leaning step for SAC and RAP

Input: Replay Buffer $\mathcal{D}$, Q network Q_{θ_j}, actor pi_ψ, target Q network $Q_{\bar{\theta}_j}$, encoder ϕ_ω. reward function r_ψ, dynamics model $\widehat{P}$
Sample a batch with size B: $\{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i)\}_{b=1}^{B} \sim \mathcal{D}$
Update Q network by minimizing $\mathcal{L}_Q(Q_{\theta_i}, \phi_\omega)$
Update actor network by minimizing $\mathcal{J}_\pi(\pi_\theta)$
Update α by minimizing $\mathcal{J}_\alpha(\alpha)$
Update dynamics model $\widehat{P}_\eta$ by minimizing $\alpha_P \mathcal{L}_P(\widehat{P}_\eta)$
Update reward function r_ψ by minimizing $\mathcal{L}_r(r_\psi)$
Update encoder ϕ_ω by minimizing $\alpha_{RAP}\mathcal{L}_{RAP}(\phi_\omega)$
Softly update target Q network: $\bar{\theta}_j = \tau_Q \theta_j + (1 - \tau_Q)\bar{\theta}_j$
Softly update target encoder $\phi_{\bar{\omega}}$: $\bar{\omega} = \tau_\phi \omega + (1 - \tau_\phi)\bar{\omega}$

5.7.4 Networks and Hyperparameters

We stack convolutional layers followed by a fully-connected layer as state encoder ϕ_ω. The encoder takes the input of a state, where 3 frames are stacked, and outputs a vector, the state representation $\phi_\omega(\mathbf{s})$. The detailed dimensions of each convolutional layer and fully-connected layer are described in Table 5.1. The Q-network consists of 3 stacked fully-connected layers with 1024 hidden dimensions, and takes the input of state representation $\phi_\omega(\mathbf{s})$ and action $\mathbf{a}$. The actor network takes the input of $\phi_\omega(\mathbf{s})$ and feeds it into three stacked fully-connected layers with 1024 hidden dimensions to output policy π. Both the approximated dynamics model $\widehat{P}$ and the approximated reward function r_ψ are two-layers MLPs with 512 hidden dimensions. ReLU activation is used for every layer. We train the whole model on one NVIDIA A100 GPU. Other hyperparameters are listed in Table 5.1.

5.8 Experiment

In this section, we evaluate the efficiency, robustness and generalization ability of the representation learned by our method on three RL benchmarks: 1) Distracting DeepMind Control Suite (DMC) [80], 2) Robosuite [99], and 3) CARLA [16]. These three domains are continuous action space control tasks with visual input. We compare our methods with several baselines and state-of-the-art methods including metric-based representation learning and data augmentation: 1) **SAC** [23], a baseline RL method for continuous

Hyperparameter	DMC	Robosuite	CARLA
Episode length	1000	500	1000
Training steps	1M	500K	100K
Replay buffer capacity	1M	100K	100K
Batch size	128	128	128
Discount factor γ	0.99	0.99	0.99
State dims	9×84×84	9×128×128	9×420×84
Encoder conv kernels	[3,3,3,3]	[3,3,3,3]	[5, 3, 3, 3]
Encoder conv channels	[32,32,32,32]	[32,32,32,32]	[64,64,64,64]
Encoder conv strides	[2,1,1,1]	[2,1,1,1]	[2,2,2,2]
Encoder output dims	100	100	100
Optimizer	Adam	Adam	Adam
Q-networks learning rate	5e-4	1e-3	1e-3
Actor network learning rate	5e-4	1e-3	1e-3
Encoder learning rate	5e-4	1e-3	1e-3
Dynamics model learning rate	5e-4	1e-3	1e-3
Reward function learning rate	5e-4	1e-3	1e-3
log α learning rate	1e-4	1e-4	1e-4
τ_ϕ	0.05	0.05	0.05
τ_Q	0.01	0.01	0.01
Target Q-network update frequency	2	1	2
Actor network update frequency	2	1	2
α_{RAP}	0.5	0.5	0.5
α_P	1e-4	1e-4	1e-4
Actor log std bound	[-10, 2]	[-10, 2]	[-10, 2]

Table 5.1: Networks dimensions and hyperparameters.

control, 2) **MICo** [5], a sample-based behavioral metric-based representation learning method for RL, 3) **DBC** [94], a representation learning method by approximating the bisimulation metric with the L_1 norm, 4) **RobustDBC** [40], a DBC-styled method with intrinsic rewards and inverse dynamics, and 5) **DrQ** [89], an image augmentation method on pixel inputs RL.

5.8.1 Distracting DeepMind Control Suite

To evaluate the generalization ability and robustness of our method, we perform experiments on 2 settings, original background and natural video background settings, in DMC [80]. We use the same experiment set up as Section 4.6.1 for such two settings.

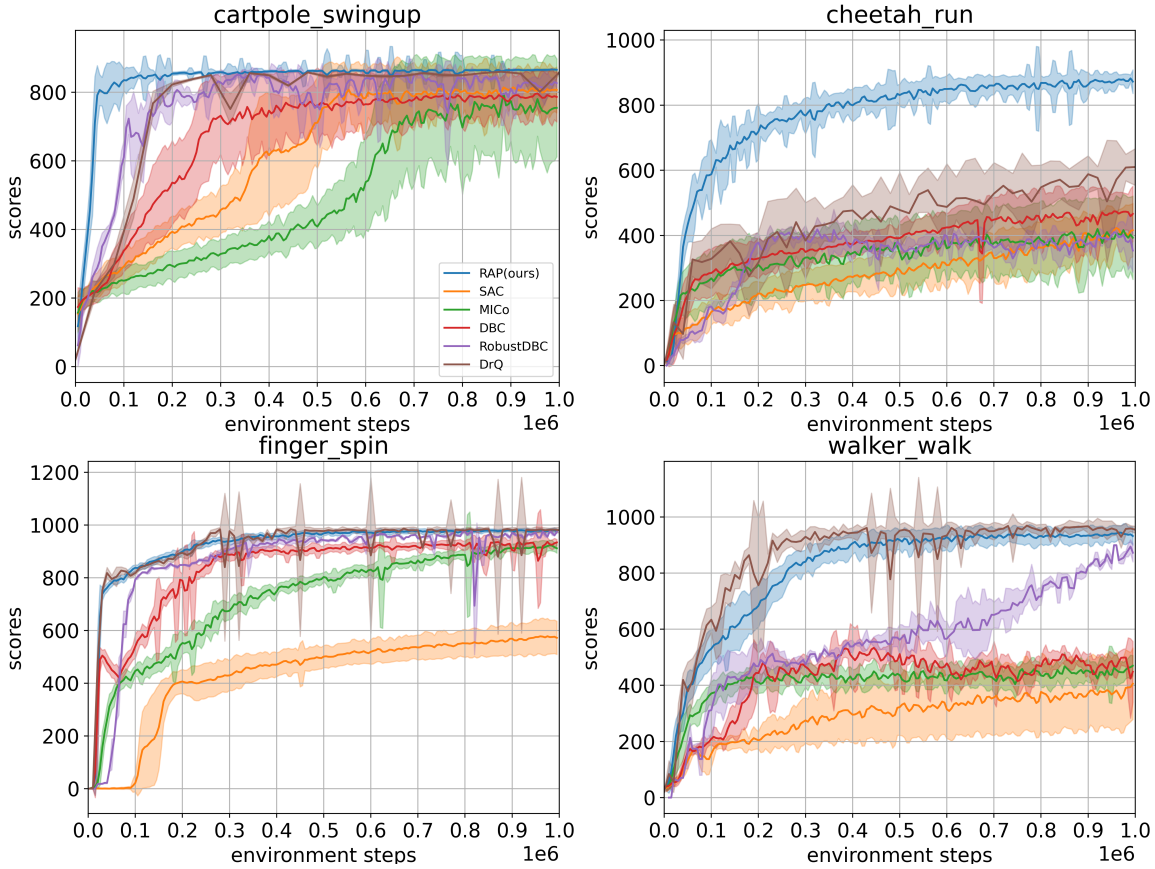


Fig. 5.2: Experimental results on DMC with original background. Each curve is average on 3 runs.

Figure 5.2 is the training curves in original background settings. It confirms that our method RAP can accelerate the training and has comparable results to data augmentation method DrQ. Figure 5.3 is the training curves in natural video background settings. Our method RAP outperforms DBC and MICo on all 4 tasks. It also converges to higher score than DrQ in 1 task and learns much faster in 3 tasks. The new SOTA results verify that our method of reducing the gaps in approximation of behavioral distance can improve the efficiency and robustness for deep RL and they also confirm that our method is able to learn generalizable state representation in complex environments.

5.8.2 Robosuite

Robosuite [99] is a robotic simulator with various manipulation tasks based on MuJoCo engine. To evaluate the robustness of methods, we use the distraction settings

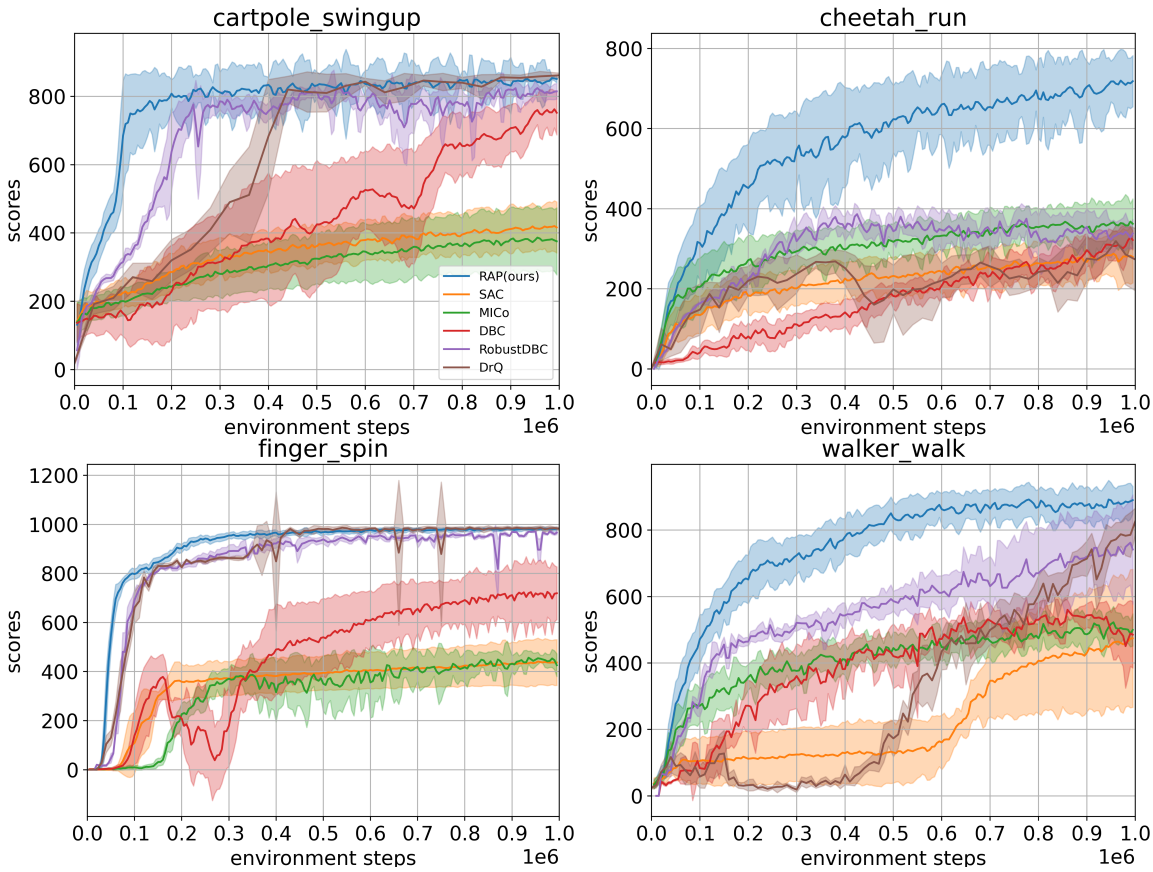


Fig. 5.3: Results on natural video background settings in DMC. Each curve is average on 3 runs.

for Robosuite by randomizing the colors of robot arms and table, the lighting source and luminance, and the camera position. The color, lighting and camera position are changed at the beginning of an episode but are consistent within the episode. Our experiment is performed by controlling the Panda arms. The control rate is 20Hz and the episode length is 500 steps. We evaluate on two tasks: Door Opening and Two Arm Peg-In-Hole. Figure 5.4 shows illustrations of observations of two tasks. We compare our method with behavioral metric methods MICo and DBC, and data augmentation method DrQ. Table 5.2 shows the training scores in 500k environment steps. Our method RAP outperforms the others. The converge scores are around 7× and 1.4× compared to the second-best method in Door Opening and Two Arm Peg-In-Hole, respectively. It shows the robustness of our RAP in environments with random distractions.

Fig. 5.4: Illustrations in Robosuite. **Left**: Door Opening. **right**: Two Arm Peg-In-Hole.

Task	RAP(Ours)	SAC	MICo	DBC	DrQ
Door Opening	**102.19 ± 26.11**	8.84±9.89	6.06±6.57	3.15±3.54	14.63±19.57
Two Arm Peg-In-Hole	**307.27±25.70**	191.29±34.88	123.69±23.25	219.56±36.87	156.86±33.57

Table 5.2: Experimental results on Robosuite trained with 500K environment steps.

5.8.3 CARLA

In order to validate the generalization ability and learning efficiency in natural scenarios, we construct experiments on an autonomous driving simulator CARLA [16], which provides 3D realistic on-world scenarios. The goal of this task is to control a vehicle driving as far as possible on a high-way map (Town 4) in 1000 time steps. The reward function followed [94] is designed to encourage driving far and avoid collisions with other vehicles. The observation is formed as 420×84 pixels of a 300 degrees ego-centric view, constructed by concatenating five cameras. Figure 5.5 shows an illustration of observation in CARLA. In order to evaluate the generalization ability, we randomly sample a weather at each episode starts. The weather (sunlight, rain, etc.), which affects the visual observation, can be considered as real-world distraction to RL agents. Note that there are some differences with Section 4.6.8 where the weather, clouds and brightness may change slowly during the simulation.

Figure 5.6 shows the comparison results of our RAP and other SOTA methods. Our method achieves comparable scores with data augmentation method DrQ, and its score is higher than behavioral metric DBC and MICo. It shows that our RAP is able to generalize well in tasks with real-world scenarios.

5.9 Proofs

5.9.1 Proof of Theorem 5.5.3

Lemma 5.9.1. $\mathbb{E}_{\substack{\mathbf{a}_i\sim\pi \\ \mathbf{a}_j\sim\pi}} |r^{\mathbf{a}_i}_{\mathbf{s}_i} - r^{\mathbf{a}_j}_{\mathbf{s}_j}| \geq |\,\mathbb{E}_{\mathbf{a}_i\sim\pi}\, r^{\mathbf{a}_i}_{\mathbf{s}_i} - \mathbb{E}_{\mathbf{a}_j\sim\pi}\, r^{\mathbf{a}_j}_{\mathbf{s}_j}|\ .$

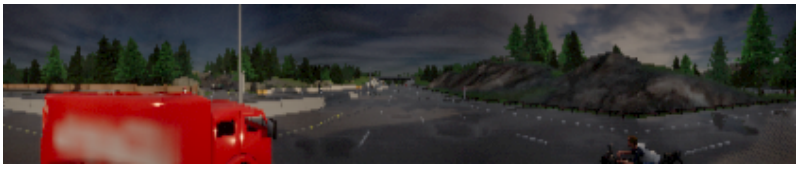


Fig. 5.5: Illustration of observation in CARLA.

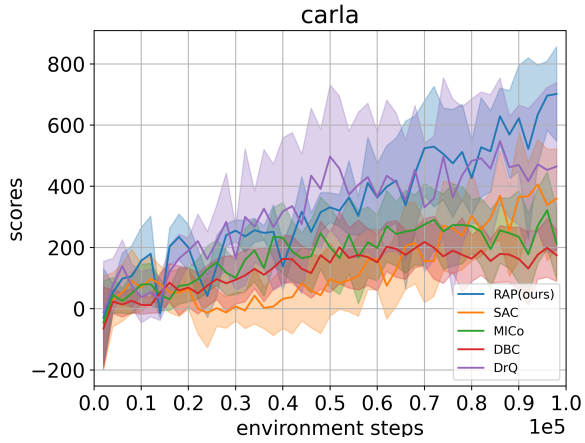


Fig. 5.6: Training curves on CARLA.

Proof: Since $r_{\mathbf{s}_i}^{\mathbf{a}_i}$ and $r_{\mathbf{s}_j}^{\mathbf{a}_j}$ are rewards which are scalars, we have $|r_{\mathbf{s}_i}^{\mathbf{a}_i} - r_{\mathbf{s}_j}^{\mathbf{a}_j}| \geq r_{\mathbf{s}_i}^{\mathbf{a}_i} - r_{\mathbf{s}_j}^{\mathbf{a}_j}$, and $|r_{\mathbf{s}_i}^{\mathbf{a}_i} - r_{\mathbf{s}_j}^{\mathbf{a}_j}| \geq r_{\mathbf{s}_j}^{\mathbf{a}_j} - r_{\mathbf{s}_i}^{\mathbf{a}_i}$ by symmetric. By taking expectation over $\mathbf{a}_i$ and $\mathbf{a}_j$, we get,

$$\mathbb{E}_{\substack{\mathbf{a}_i\sim\pi \\ \mathbf{a}_j\sim\pi}} |r_{\mathbf{s}_i}^{\mathbf{a}_i} - r_{\mathbf{s}_j}^{\mathbf{a}_j}| \geq \mathbb{E}_{\substack{\mathbf{a}_i\sim\pi \\ \mathbf{a}_j\sim\pi}} \left[r_{\mathbf{s}_i}^{\mathbf{a}_i} - r_{\mathbf{s}_j}^{\mathbf{a}_j} \right], \tag{5.17}$$

and

$$\mathbb{E}_{\substack{\mathbf{a}_i\sim\pi \\ \mathbf{a}_j\sim\pi}} |r_{\mathbf{s}_i}^{\mathbf{a}_i} - r_{\mathbf{s}_j}^{\mathbf{a}_j}| \geq \mathbb{E}_{\substack{\mathbf{a}_i\sim\pi \\ \mathbf{a}_j\sim\pi}} \left[r_{\mathbf{s}_j}^{\mathbf{a}_j} - r_{\mathbf{s}_i}^{\mathbf{a}_i} \right]. \tag{5.18}$$

Combine (5.17) and (5.18), we have

$$\mathbb{E}_{\substack{\mathbf{a}_i\sim\pi \\ \mathbf{a}_j\sim\pi}} |r_{\mathbf{s}_i}^{\mathbf{a}_i} - r_{\mathbf{s}_j}^{\mathbf{a}_j}| \geq \left| \mathbb{E}_{\substack{\mathbf{a}_i\sim\pi \\ \mathbf{a}_j\sim\pi}} \left[r_{\mathbf{s}_i}^{\mathbf{a}_i} - r_{\mathbf{s}_j}^{\mathbf{a}_j} \right] \right| = |\, \mathbb{E}_{\mathbf{a}_i\sim\pi}\, r_{\mathbf{s}_i}^{\mathbf{a}_i} - \mathbb{E}_{\mathbf{a}_j\sim\pi}\, r_{\mathbf{s}_j}^{\mathbf{a}_j}|.$$

Lemma 5.9.2 (Lifted MDP of MICo [5])**.** *The MICo metric function $\mathcal{F}_M^\pi$ is the Bellman operator for a Lifted MDP.*

Proof: Let the MDP for RL defined by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, P, \gamma \rangle$, we consider a lifted MDP constructed by a tuple $\langle \hat{\mathcal{S}}, \hat{\mathcal{A}}, \hat{\mathcal{R}}, \hat{P}, \gamma \rangle$, where state space $\hat{\mathcal{S}} = \mathcal{S} \times \mathcal{S}$, action space $\hat{\mathcal{A}} = \mathcal{A} \times \mathcal{A}$, transition distribution $\hat{P}_{(\mathbf{s}_i,\mathbf{s}_j)}^{(\mathbf{a}_i,\mathbf{a}_j)} = P_{\mathbf{s}_i}^{\mathbf{a}_i} P_{\mathbf{s}_j}^{\mathbf{a}_j}$, and reward function $\hat{\mathcal{R}}((\mathbf{s}_i, \mathbf{s}_j)) =$

$|\mathbb{E}_{\mathbf{a}_i\sim\pi} r^{\mathbf{a}_i}_{\mathbf{s}_i} - \mathbb{E}_{\mathbf{a}_j\sim\pi} r^{\mathbf{a}_j}_{\mathbf{s}_j}|$. The Bellman operator $T^{\hat{\pi}}_M$ under policy $\hat{\pi}(\mathbf{a}_i,\mathbf{a}_j|\mathbf{s}_i,\mathbf{s}_j) = \pi(\mathbf{a}_i|\mathbf{s}_i)\pi(\mathbf{a}_j|\mathbf{s}_j)$ is:

$$\begin{aligned}
T^{\hat{\pi}}_M(d^\pi_M)((\mathbf{s}_i,\mathbf{s}_j)) &= \sum_{(\mathbf{a}_i,\mathbf{a}_j)} \hat{\pi}(\mathbf{a}_i,\mathbf{a}_j|\mathbf{s}_i,\mathbf{s}_j) \sum_{(\mathbf{s}'_i,\mathbf{s}'_j)} \hat{P}^{(\mathbf{a}_i,\mathbf{a}_j)}_{(\mathbf{s}_i,\mathbf{s}_j)}(\mathbf{s}'_i,\mathbf{s}'_j)\left[\hat{\mathcal{R}}((\mathbf{s}_i,\mathbf{s}_j)) + \gamma d^\pi_M(\mathbf{s}'_i,\mathbf{s}'_j)\right] \\
&= |\mathbb{E}_{\mathbf{a}_i\sim\pi} r^{\mathbf{a}_i}_{\mathbf{s}_i} - \mathbb{E}_{\mathbf{a}_j\sim\pi} r^{\mathbf{a}_j}_{\mathbf{s}_j}| + \gamma \mathbb{E}_{\substack{\mathbf{s}'_i\sim P^\pi_{\mathbf{s}_i} \\ \mathbf{s}'_j\sim P^\pi_{\mathbf{s}_j}}} d^\pi_M(\mathbf{s}'_i,\mathbf{s}'_j) \\
&= \mathcal{F}^\pi_M(d^\pi_M)(\mathbf{s}_i,\mathbf{s}_j).
\end{aligned}$$

Lemma 5.9.3 (Lifted MDP of shift MICo)**.** *The shift MICo metric function $\tilde{\mathcal{F}}^\pi_M$ is the Bellman operator for a lifted MDP but with different reward function.*

Proof: We construct the lifted MDP of shift MICo by a tuple $\langle\hat{\mathcal{S}},\hat{\mathcal{A}},\tilde{\hat{\mathcal{R}}},\hat{P},\gamma\rangle$ which is same as the lifted MDP in Lemma 5.9.2 except for reward function $\tilde{\hat{\mathcal{R}}}((\mathbf{s}_i,\mathbf{s}_j),(\mathbf{a}_i,\mathbf{a}_j)) = |r^{\mathbf{a}_i}_{\mathbf{s}_i} - r^{\mathbf{a}_j}_{\mathbf{s}_j}|$. The Bellman operator $\tilde{T}^{\hat{\pi}}_M$ under policy $\hat{\pi}$ is:

$$\begin{aligned}
&\tilde{T}^{\hat{\pi}}_M(\tilde{d}^\pi_M)((\mathbf{s}_i,\mathbf{s}_j)) \\
=&\sum_{(\mathbf{a}_i,\mathbf{a}_j)} \hat{\pi}(\mathbf{a}_i,\mathbf{a}_j|\mathbf{s}_i,\mathbf{s}_j) \sum_{(\mathbf{s}'_i,\mathbf{s}'_j)} \hat{P}^{(\mathbf{a}_i,\mathbf{a}_j)}_{(\mathbf{s}_i,\mathbf{s}_j)}(\mathbf{s}'_i,\mathbf{s}'_j)\left[\tilde{\hat{\mathcal{R}}}((\mathbf{s}_i,\mathbf{s}_j),(\mathbf{a}_i,\mathbf{a}_j)) + \gamma\tilde{d}^\pi_M(\mathbf{s}'_i,\mathbf{s}'_j)\right] \\
=&\mathbb{E}_{\substack{\mathbf{a}_i\sim\pi \\ \mathbf{a}_j\sim\pi}} |r^{\mathbf{a}_i}_{\mathbf{s}_i} - r^{\mathbf{a}_j}_{\mathbf{s}_j}| + \gamma \mathbb{E}_{\substack{\mathbf{s}'_i\sim P^\pi_{\mathbf{s}_i} \\ \mathbf{s}'_j\sim P^\pi_{\mathbf{s}_j}}} \tilde{d}^\pi_M(\mathbf{s}'_i,\mathbf{s}'_j) \\
=&\tilde{\mathcal{F}}^\pi_M(\tilde{d}^\pi_M)(\mathbf{s}_i,\mathbf{s}_j).
\end{aligned}$$

Lemma 5.9.4. *Consider an auxiliary MDP specified by a tuple $\langle\hat{\mathcal{S}},\hat{\mathcal{A}},\hat{\mathcal{R}}_\Delta,\hat{P},\gamma\rangle$ where $\hat{\mathcal{R}}_\Delta((\mathbf{s}_i,\mathbf{s}_j),(\mathbf{a}_i,\mathbf{a}_j)) = |r^{\mathbf{a}_i}_{\mathbf{s}_i} - r^{\mathbf{a}_j}_{\mathbf{s}_j}| - |\mathbb{E}_{\mathbf{a}_i\sim\pi} r^{\mathbf{a}_i}_{\mathbf{s}_i} - \mathbb{E}_{\mathbf{a}_j\sim\pi} r^{\mathbf{a}_j}_{\mathbf{s}_j}|$. Let $V^{\hat{\pi}}_\Delta$ denote the value function of such MDP over policy $\hat{\pi}$, then*

$$V^{\hat{\pi}}_\Delta((\mathbf{s}_i,\mathbf{s}_j)) \geq 0.$$

Proof: The value function $V^{\hat{\pi}}_\Delta$ can be expanded as,

$$\begin{aligned}
&V^{\hat{\pi}}_\Delta((\mathbf{s}_i,\mathbf{s}_j)) \\
=&\mathbb{E}_{\hat{\pi}}[\sum_t \gamma^t \hat{\mathcal{R}}_\Delta((\mathbf{s}^{(t)}_i,\mathbf{s}^{(t)}_j),(\mathbf{a}^{(t)}_i,\mathbf{a}^{(t)}_j))|\mathbf{s}^{(0)}_i = \mathbf{s}_i, \mathbf{s}^{(0)}_j = \mathbf{s}_j] \\
=&\mathbb{E}_{\hat{\pi}}\left[\sum_t \gamma^t \left(\mathbb{E}_{\substack{\mathbf{a}^{(t)}_i\sim\pi \\ \mathbf{a}^{(t)}_j\sim\pi}} \left|r^{\mathbf{a}^{(t)}_i}_{\mathbf{s}^{(t)}_i} - r^{\mathbf{a}^{(t)}_j}_{\mathbf{s}^{(t)}_j}\right| - \left|\mathbb{E}_{\mathbf{a}^{(t)}_i\sim\pi} r^{\mathbf{a}^{(t)}_i}_{\mathbf{s}^{(t)}_i} - \mathbb{E}_{\mathbf{a}^{(t)}_j\sim\pi} r^{\mathbf{a}^{(t)}_j}_{\mathbf{s}^{(t)}_j}\right|\right) \middle| \mathbf{s}^{(0)}_i = \mathbf{s}_i, \mathbf{s}^{(0)}_j = \mathbf{s}_j\right].
\end{aligned}$$

Due to Lemma 5.9.1, $\mathbb{E}_{\substack{\mathbf{a}_i^{(t)}\sim\pi \\ \mathbf{a}_j^{(t)}\sim\pi}} \left| r^{\mathbf{a}_i^{(t)}}_{\mathbf{s}_i^{(t)}} - r^{\mathbf{a}_j^{(t)}}_{\mathbf{s}_j^{(t)}} \right| - \left| \mathbb{E}_{\mathbf{a}_i^{(t)}\sim\pi} r^{\mathbf{a}_i^{(t)}}_{\mathbf{s}_i^{(t)}} - \mathbb{E}_{\mathbf{a}_j^{(t)}\sim\pi} r^{\mathbf{a}_j^{(t)}}_{\mathbf{s}_j^{(t)}} \right| \geq 0$, therefore,

$$V^{\hat{\pi}}_{\Delta}((\mathbf{s}_i, \mathbf{s}_j)) \geq 0.$$

This Lemma is proven.

Lemma 5.9.1 indicates that the reward difference term in shift MICo is larger or equal to MICo. Lemma 5.9.2, 5.9.3 and 5.9.4 help to prove the following theorem that demonstrates the value difference bound of Shift MICo is looser than MICo.

Theorem 5.5.3 (Looser value difference bound)**.** *Given states* $\mathbf{s}_i$ *and* $\mathbf{s}_j$*, and a policy* π*, we have*

$$|V^{\pi}(\mathbf{s}_i) - V^{\pi}(\mathbf{s}_j)| \leq d^{\pi}_M(\mathbf{s}_i, \mathbf{s}_j) \leq \tilde{d}^{\pi}_M(\mathbf{s}_i, \mathbf{s}_j).$$

Proof: Due to Theorem 5.4.4, which has been proven by Castro et al. [5] in Proposition 4.8 of their paper, we already have the leftmost two terms,

$$|V^{\pi}(\mathbf{s}_i) - V^{\pi}(\mathbf{s}_j)| \leq d^{\pi}_M(\mathbf{s}_i, \mathbf{s}_j). \tag{5.19}$$

Lemma 5.9.2 shows that d^{π}_M is the value function of lifted MDP $\langle \hat{\mathcal{S}}, \hat{\mathcal{A}}, \hat{\mathcal{R}}, \hat{P}, \gamma \rangle$. d^{π}_M can be expanded as sum of discounted future rewards,

$$d^{\pi}_M(\mathbf{s}_i, \mathbf{s}_j) = \mathbb{E}_{\hat{\pi}} \left[\sum_t \gamma^t \left(\left| \mathbb{E}_{\mathbf{a}_i^{(t)}\sim\pi} r^{\mathbf{a}_i^{(t)}}_{\mathbf{s}_i^{(t)}} - \mathbb{E}_{\mathbf{a}_j^{(t)}\sim\pi} r^{\mathbf{a}_j^{(t)}}_{\mathbf{s}_j^{(t)}} \right| \right) \middle| \mathbf{s}_i^{(0)} = \mathbf{s}_i, \mathbf{s}_j^{(0)} = \mathbf{s}_j \right].$$

Similarly, due to Lemma 5.9.3, $\tilde{d}^{\pi}_M$ can be expanded as,

$$\tilde{d}^{\pi}_M(\mathbf{s}_i, \mathbf{s}_j) = \mathbb{E}_{\hat{\pi}} \left[\sum_t \gamma^t \left(\mathbb{E}_{\substack{\mathbf{a}_i^{(t)}\sim\pi \\ \mathbf{a}_j^{(t)}\sim\pi}} \left| r^{\mathbf{a}_i^{(t)}}_{\mathbf{s}_i^{(t)}} - r^{\mathbf{a}_j^{(t)}}_{\mathbf{s}_j^{(t)}} \right| \right) \middle| \mathbf{s}_i^{(0)} = \mathbf{s}_i, \mathbf{s}_j^{(0)} = \mathbf{s}_j \right].$$

Then we analyze the difference between $\tilde{d}^{\pi}_M$ and d^{π}_M. Since $\tilde{d}^{\pi}_M$ and d^{π}_M are defined on the same policy $\hat{\pi}$ and dynamics model $\hat{P}$, we have

$$\begin{aligned}
&\tilde{d}^{\pi}_M(\mathbf{s}_i, \mathbf{s}_j) - d^{\pi}_M(\mathbf{s}_i, \mathbf{s}_j) \\
=&\mathbb{E}_{\hat{\pi}} \left[\sum_t \gamma^t \left(\mathbb{E}_{\substack{\mathbf{a}_i^{(t)}\sim\pi \\ \mathbf{a}_j^{(t)}\sim\pi}} \left| r^{\mathbf{a}_i^{(t)}}_{\mathbf{s}_i^{(t)}} - r^{\mathbf{a}_j^{(t)}}_{\mathbf{s}_j^{(t)}} \right| - \left| \mathbb{E}_{\mathbf{a}_i^{(t)}\sim\pi} r^{\mathbf{a}_i^{(t)}}_{\mathbf{s}_i^{(t)}} - \mathbb{E}_{\mathbf{a}_j^{(t)}\sim\pi} r^{\mathbf{a}_j^{(t)}}_{\mathbf{s}_j^{(t)}} \right| \right) \middle| \mathbf{s}_i^{(0)} = \mathbf{s}_i, \mathbf{s}_j^{(0)} = \mathbf{s}_j \right] \\
=&V^{\hat{\pi}}_{\Delta}((\mathbf{s}_i, \mathbf{s}_j)).
\end{aligned}$$

where $V_{\Delta}^{\hat{\pi}}((\mathbf{s}_i, \mathbf{s}_j))$ is the value function in Lemma 5.9.4 and $V_{\Delta}^{\hat{\pi}}((\mathbf{s}_i, \mathbf{s}_j)) \geq 0$. Therefore,

$$d_M^{\pi}(\mathbf{s}_i, \mathbf{s}_j) \leq \tilde{d}_M^{\pi}(\mathbf{s}_i, \mathbf{s}_j). \tag{5.20}$$

Combine (5.19) and (5.20), finally prove that,

$$|V^{\pi}(\mathbf{s}_i) - V^{\pi}(\mathbf{s}_j)| \leq d_M^{\pi}(\mathbf{s}_i, \mathbf{s}_j) \leq \tilde{d}_M^{\pi}(\mathbf{s}_i, \mathbf{s}_j).$$

5.9.2 Proof of Theorem 5.6.3

Theorem 5.6.3 (Value function difference bound)**.** *Given states* $\mathbf{s}_i$ *and* $\mathbf{s}_j$*, and a policy* π*, we have*

$$|V^{\pi}(\mathbf{s}_i) - V^{\pi}(\mathbf{s}_j)| \leq d_G^{\pi}(\mathbf{s}_i, \mathbf{s}_j). \tag{5.21}$$

Proof: We follow Castro et al. [5] (the proof of Proposition 4.8 in their paper) to prove the value function difference bound. We will show that if $|V^{\pi}(\mathbf{s}_i) - V^{\pi}(\mathbf{s}_j)| \leq d(\mathbf{s}_i, \mathbf{s}_j), \forall \mathbf{s}_i, \mathbf{s}_j \in \mathcal{S}$, then $|V^{\pi}(\mathbf{s}_i) - V^{\pi}(\mathbf{s}_j)| \leq \mathcal{F}_G^{\pi}(d)(\mathbf{s}_i, \mathbf{s}_j)$. Suppose $|V^{\pi}(\mathbf{s}_i) - V^{\pi}(\mathbf{s}_j)| \leq d(\mathbf{s}_i, \mathbf{s}_j)$ holds, then

$$\begin{aligned}
&V^{\pi}(\mathbf{s}_i) - V^{\pi}(\mathbf{s}_j) \\
=&\mathbb{E}_{\mathbf{a}_i \sim \pi} r_{\mathbf{s}_i}^{\mathbf{a}_i} - \mathbb{E}_{\mathbf{a}_j \sim \pi} r_{\mathbf{s}_j}^{\mathbf{a}_j} + \sum_{\mathbf{a}_i} \pi(\mathbf{a}_i|\mathbf{s}_i) \sum_{\mathbf{s}_i'} P_{\mathbf{s}_i}^{\mathbf{a}_i}(\mathbf{s}_i') V^{\pi}(\mathbf{s}_i') - \sum_{\mathbf{a}_j} \pi(\mathbf{a}_j|\mathbf{s}_j) \sum_{\mathbf{s}_j'} P_{\mathbf{s}_j}^{\mathbf{a}_j}(\mathbf{s}_j') V^{\pi}(\mathbf{s}_j') \\
\leq& \left| \mathbb{E}_{\mathbf{a}_i \sim \pi} r_{\mathbf{s}_i}^{\mathbf{a}_i} - \mathbb{E}_{\mathbf{a}_j \sim \pi} r_{\mathbf{s}_j}^{\mathbf{a}_j} \right| + \mathbb{E}_{\substack{\mathbf{a}_i \sim \pi \\ \mathbf{a}_j \sim \pi}} \left(\sum_{\mathbf{s}_i'} P_{\mathbf{s}_i}^{\mathbf{a}_i}(\mathbf{s}_i') V^{\pi}(\mathbf{s}_i') - \sum_{\mathbf{s}_j'} P_{\mathbf{s}_j}^{\mathbf{a}_j}(\mathbf{s}_j') V^{\pi}(\mathbf{s}_j') \right).
\end{aligned}$$

We make an assumption that $\sum_{\mathbf{s}'} P_{\mathbf{s}}^{\mathbf{a}}(\mathbf{s}') V^{\pi}(\mathbf{s}') = V^{\pi}(\mathbb{E}_{s' \sim P_{\mathbf{s}}^{\mathbf{a}}}[s'])$. We make this assumption because we use learned dynamics model to predict the next states and the learned dynamics model is assumed as Gaussian distribution with small standard deviation. If this assumption holds, then

$$\begin{aligned}
&\left| \mathbb{E}_{\mathbf{a}_i \sim \pi} r_{\mathbf{s}_i}^{\mathbf{a}_i} - \mathbb{E}_{\mathbf{a}_j \sim \pi} r_{\mathbf{s}_j}^{\mathbf{a}_j} \right| + \mathbb{E}_{\substack{\mathbf{a}_i \sim \pi \\ \mathbf{a}_j \sim \pi}} \left(\sum_{\mathbf{s}_i'} P_{\mathbf{s}_i}^{\mathbf{a}_i}(\mathbf{s}_i') V^{\pi}(\mathbf{s}_i') - \sum_{\mathbf{s}_j'} P_{\mathbf{s}_j}^{\mathbf{a}_j}(\mathbf{s}_j') V^{\pi}(\mathbf{s}_j') \right) \\
=& \left| \mathbb{E}_{\mathbf{a}_i \sim \pi} r_{\mathbf{s}_i}^{\mathbf{a}_i} - \mathbb{E}_{\mathbf{a}_j \sim \pi} r_{\mathbf{s}_j}^{\mathbf{a}_j} \right| + \mathbb{E}_{\substack{\mathbf{a}_i \sim \pi \\ \mathbf{a}_j \sim \pi}} \left(V^{\pi}(\mathbb{E}_{s_i' \sim P_{\mathbf{s}_i}^{\mathbf{a}_i}}[s_i']) - V^{\pi}(\mathbb{E}_{s_j' \sim P_{\mathbf{s}_j}^{\mathbf{a}_j}}[s_j']) \right) \\
\leq& \left| \mathbb{E}_{\mathbf{a}_i \sim \pi} r_{\mathbf{s}_i}^{\mathbf{a}_i} - \mathbb{E}_{\mathbf{a}_j \sim \pi} r_{\mathbf{s}_j}^{\mathbf{a}_j} \right| + \mathbb{E}_{\substack{\mathbf{a}_i \sim \pi \\ \mathbf{a}_j \sim \pi}} d(\mathbb{E}_{s_i' \sim P_{\mathbf{s}_i}^{\mathbf{a}_i}}[s_i'], \mathbb{E}_{s_j' \sim P_{\mathbf{s}_j}^{\mathbf{a}_j}}[s_j']) \\
=& \mathcal{F}_G^{\pi}(d)(\mathbf{s}_i, \mathbf{s}_j).
\end{aligned}$$

By symmetric,

$$V^\pi(\mathbf{s}_j) - V^\pi(\mathbf{s}_i) \leq \mathcal{F}_G^\pi(d)(\mathbf{s}_i, \mathbf{s}_j).$$

Therefore,

$$|V^\pi(\mathbf{s}_i) - V^\pi(\mathbf{s}_j)| \leq \mathcal{F}_G^\pi(d)(\mathbf{s}_i, \mathbf{s}_j).$$

We have an initial distance $d_0(\mathbf{s}_i, \mathbf{s}_j) = 2\max_{\mathbf{s},\mathbf{a}} |r_{\mathbf{s}}^{\mathbf{a}}|/(1-\gamma)$ that satisfies $|V^\pi(\mathbf{s}_i) - V^\pi(\mathbf{s}_j)| \leq d(\mathbf{s}_i, \mathbf{s}_j)$, and $\mathcal{F}_G^\pi$ is contraction mapping on d. By repeatedly applying $\mathcal{F}_G^\pi$ on d, d will eventually converge to the fixed-point d_G^π. Because for each iteration $\mathcal{F}_G^\pi(d)$ satisfies $|V^\pi(\mathbf{s}_j) - V^\pi(\mathbf{s}_i)| \leq \mathcal{F}_G^\pi(d)(\mathbf{s}_i, \mathbf{s}_j)$, the fixed-point d_G^π satisfies

$$|V^\pi(\mathbf{s}_i) - V^\pi(\mathbf{s}_j)| \leq d_G^\pi(\mathbf{s}_i, \mathbf{s}_j). \tag{5.22}$$

5.10 Conclusion

Representation learning is one of the most critical problems in high-dimensional deep RL. In this chapter, we propose a new behavioral metric and a practical representation learning algorithm on top of the new behavioral metric for deep RL. We provide theoretical analysis for our proposed metric as well as the representation learning algorithm. We conduct empirical studies on multiple RL domains to verify the effectiveness of our proposed method.

Chapters 4 and 5 present two behavioral metric-based methods for the applications of control from pixels. The next chapter (Chapter 6) will present a work on CNN compression using deep RL-based channel pruning.

Chapter 6

Deep RL for Storage Efficient Runtime Pruning[1]

6.1 Introduction

Apart from the applications of control with visual input, this thesis introduces an application that performs channel pruning on CNNs utilizing deep RL algorithm. CNNs have demonstrated remarkable performance in various computer vision tasks [45, 75, 29, 28, 97, 49, 28, 98, 13]. However, since most state-of-the-art CNNs require expensive computation power for inference and huge storage space to store a large amount of parameters, the limitation of energy, computation and storage on mobile or edge devices have become the major bottleneck on real-world deployments of CNNs. Existing studies have focused on speeding up the execution of CNNs for inference on edge devices by model compression using matrix decomposition [11, 62], network quantization [10], network pruning [14], etc. Among these approaches, channel pruning, which discards an entire input or output channel and keeps the rest of the model with structures, has shown promising performance [32, 61, 100, 65].

Most channel pruning approaches can be categorized into two types: runtime and static. Static approaches aim to evaluate the importance of each channel over the whole training dataset and remove the least important channels to minimize the loss of performance after pruning. By permanently pruning a number of channels, the computation and storage cost of a CNN can be dramatically reduced when being deployed, and

[1]The work in this chapter has been published in [7].

the inference execution can be accelerated consequently. On the other hand, runtime approaches have been recently proposed to achieve dynamic channel pruning on each individual instance [21]. To be specific, the goal of runtime approaches aims to evaluate the channel importance at runtime, which is assumed to be different on different input instances. By pruning channels dynamically, different pruned structures can be considered as different routing of data stream inside CNNs. This kind of approaches is able to significantly improve the representation capability of a CNN, and thus achieve better performance in terms of prediction accuracy compared with static approaches. However, previous runtime approaches trade storage cost off dynamic flexibility. To achieve dynamic pruning on different individual instances, all parameters of kernels are required to be stored (or even more parameters are introduced). This makes runtime approaches not applicable on resource-limited edge devices. Moreover, most of previous runtime approaches only evaluate the importance among channels in each single layer independently, without considering the difference in efficiency among layers.

In this chapter, to address the aforementioned issues of runtime channel pruning approaches, we propose a deep reinforcement learning-based pruning framework. Basically, we aim to apply deep RL to prune CNNs by maximizing received rewards, which are designed to satisfy the overall budget constraints along side with the network's training accuracy. Note that automatic channel pruning by deep RL is a challenging task because the action space is usually very huge. Specifically, the discrete action space for the deep RL agent is as large as the number of channels at each layer, and the action space may vary among layers since there are different numbers of channels in different layers. To facilitate pruning CNNs by deep RL, for each layer, we first design a novel prediction component to estimate the importance of channels, and then develop a deep RL-based component to learn the sparsity ratio of the layer, i.e., how many channels should be pruned.

Different from previous runtime channel pruning approaches, which only learn runtime importance of each channel, we propose to learn both runtime importance and additionally static importance for each channel. While runtime importance maintains the saliency of specific channels for each individual input, the static importance captures the overall saliency of the corresponding channel among the whole dataset. According

to each type of channel importance, we further design different deep RL agents (i.e., a runtime agent and a static agent) to learn a sparsity ratio in a layer-wise manner. The sparsity ratio learned by the runtime agent together with the estimated runtime importance of channels is used to generate runtime pruning structures, while the sparsity ratio learned by the static agent together with the estimated static importance of channels is used to generate static (permanent) pruning structures. By considering both the pruning structures, our framework is able to provide a trade-off between storage efficiency and dynamic flexibility for runtime channel pruning.

In summary, the contributions of this work are 2-fold. First, we propose to prune channels by taking both runtime and static information of the environment into consideration. Runtime information endows pruning with flexibility based on different input instances while static information reduces the number of parameters in deployment, leading to storage reduction, which cannot be achieved by conventional runtime pruning approaches. Second, we propose to use deep RL to determine sparsity ratios, which is different from the previous pruning approaches that manually set sparsity ratios.

6.2 Related Work and Preliminary

6.2.1 Structure Pruning

Wen et al. [86] pioneered structure pruning in deep neural network by imposing the $L_{2,1}$ norm in training. Under the same framework, Liu et al. [58] regarded parameters in batch normalization as channel selection signal, which is minimized to achieve pruning during training. He et al. [32] formulated channel pruning into a two-step iterative process including LASSO regression based channel selection and least square reconstruction. Luo et al. [61] formulated channel pruning as minimization of difference of output features, which is solved by greedy selection. Zhuang et al. [100] further considered early prediction, reconstruction loss and final loss to select importance channels. Luo and Wu [60] proposed to use layer input to learn channel importance, which is then binarized for pruning. Overall, structure pruning methods accelerate inference by producing regular and compact model. However, this brought regularness requires preserving more parameters to ensure performance.

6.2.2 Dynamic Pruning

Dynamic pruning provides different pruning strategies according to input data. Wang et al. [84] proposed to reduce computation by skipping layers or channels based on the analysis of input features. Gao et al. [21] applied the same framework while extended features selection in both input and output features. Similarly, Liu and Deng [56] introduced multiple branches for runtime inference according to inputs. A gating module is learnt to guide the flow of feature maps. Bolukbasi et al. [3] learned to choose the components of a deep network to be evaluated for each input adaptively. Early exit is introduced to accelerate computation. Dynamic pruning adaptively takes different actions for different inputs, which is able to accelerate the overall inference time. However, the original high-precision model needs to be stored, together with extra parameters for making specified pruning actions. Rosenbaum et al. [69] proposed to learn routers to route layers output to different next layers, in order to adjust a network to multi-task learning.

6.2.3 Deep RL for Pruning

Channel selection is on trial using deep RL. Lin et al. [55] trained a LSTM model to remember and provide channel pruning strategy for backbone CNN model, which is conducted using reinforcement learning techniques. He et al. [33] proposed to determine the compression ratio in each layer by training an agent regarding the pruning-retraining process as an environment.

6.2.4 Reinforcement Learning

We consider a standard setup of reinforcement learning: an agent sequentially takes actions over a sequence of time steps in an environment, in order to maximize the cumulative reward [78]. This problem can be formulated as a Markov Decision Process (MDP) of a tuple ($\mathcal{S}$, $\mathcal{A}$, $\mathcal{R}$, P, γ), where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, P is the transition probability distribution where each $P(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ specifies the probability of transiting to the next state $\mathbf{s}'$ from the current state $\mathbf{s}$ given an action $\mathbf{a}$, and $\gamma \in [0, 1)$ is the discount factor. The goal of reinforcement

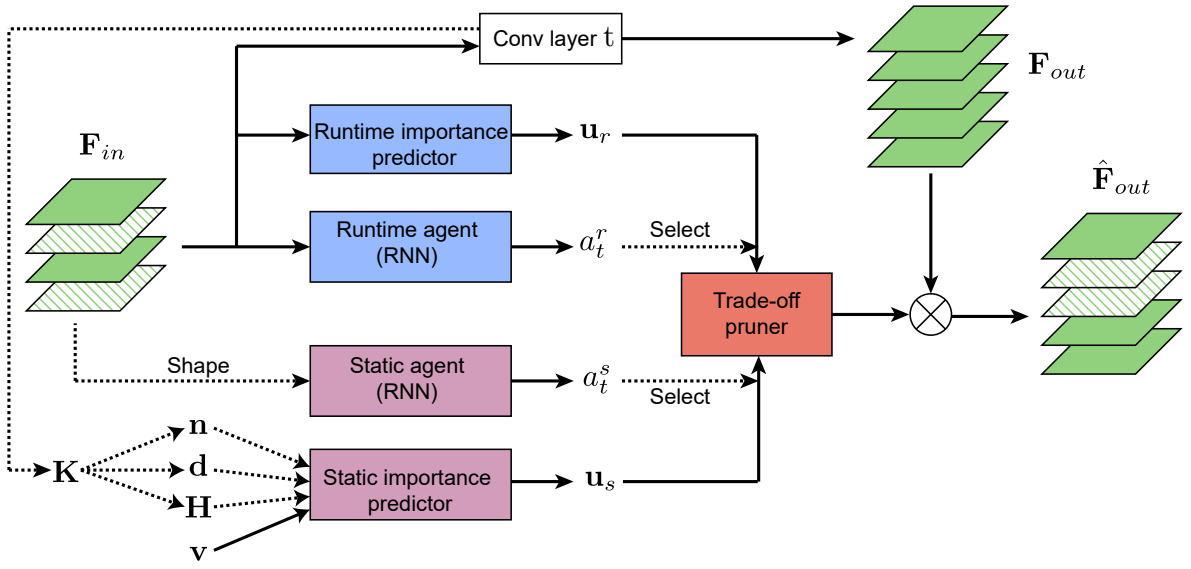


Fig. 6.1: Our proposed deep RL-based runtime pruning framework. Solid arrows indicate differentiable dataflow. Dash arrows indicate dataflow that is non-differentiable or does not require differentiation.

learning is to learn a policy $\pi(a|s)$ that maximizes the objective of discounted cumulative rewards over finite time steps as $\max_\pi \sum_{t=0}^{T} \gamma^t \mathcal{R}(s_t, a_t)$, where $s_t \in \mathcal{S}$ and $a_t \in \mathcal{A}$ are state and taken action at time step t respectively.

6.3 Deep RL-based Runtime Pruning Framework

The overview of our proposed framework is presented in Fig. 6.1, which shows our pruning procedure in the t-th convolutional layer. To prune convolutional layer t, we 1) learn two types of *channel importance*: *static* channel importance $\mathbf{u}_s$ and *runtime* channel importance $\mathbf{u}_r$; 2) learn two deep RL agents, the *runtime agent* and the *static agent*, producing action a_t^r and a_t^s which are defined as sparsity ratios of runtime and static pruning, respectively; 3) perform a *trade-off pruner* to balance the runtime and static pruning results.

Both runtime channel importance $\mathbf{u}_r \in \mathbb{R}^C$ and static channel importance $\mathbf{u}_s \in \mathbb{R}^C$ indicate the channel importance of the full precision output feature map $\mathbf{F}_{out}$ of a convolution layer, where C is the number of channels in $\mathbf{F}_{out}$. Channels are selected to be pruned according to the values of each element in $\mathbf{u}_r$ and $\mathbf{u}_s$, and how many channels to

be selected is decided by the sparsity ratios a_t^r and a_t^s, respectively. The balanced pruning result is represented by two items, a decision mask $\mathbf{m}$ of binary values (1/0) to indicates which channels to be pruned (1: pruned, 0: preserved) and a unified channel importance vector $\mathbf{u}$. $\mathbf{m}$ and $\mathbf{u}$ are generated by trade-off pruner g as $[\mathbf{m}, \mathbf{u}] = g(\mathbf{u}_r, \mathbf{u}_s, a_t^r, a_t^s)$, where $a_t^r, a_t^s \in \mathbb{R}$, $\mathbf{u} \in \mathbb{R}^C$ and $\mathbf{m} \in \{0,1\}^C$. The final output after pruning is constructed by multiplying the full precision output feature map $\mathbf{F}_{out}$, by $1 - \mathbf{m}$ and $\mathbf{u}$ as,

$$\hat{\mathbf{F}}_{out} = \mathbf{F}_{out} \otimes (\mathbf{1} - \mathbf{m}) \otimes \mathbf{u}, \tag{6.1}$$

where $\otimes$ is the broadcast element-wise multiplier, and $\mathbf{1}$ is the matrix of the same size as $\mathbf{m}$ with all the elements being 1. In the following, we introduce how to learn the runtime channel importance vector $\mathbf{u}_r$ and the static channel importance vector $\mathbf{u}_s$ in Sec. 6.3.1, how to construct the trade-off pruner $g(\cdot)$ in Sec. 6.3.2, and how to design the two deep RL agents to predict a_t^r and a_t^s in Sec. 6.3.3.

6.3.1 Learnable Channel Importance

We consider that a convolutional layer takes input of feature map $\mathbf{F}_{in} \in \mathbb{R}^{C_{in} \times H_{in} \times W_{in}}$ and generates an output feature map $\mathbf{F}_{out} \in \mathbb{R}^{C_{out} \times H_{out} \times W_{out}}$, where C_*, H_* and W_* are the number of channels, width and height of the feature map $\mathbf{F}_*$, respectively. Each element of the channel importance vectors $\mathbf{u}_r \in \mathbb{R}^{C_{out}}$ and $\mathbf{u}_s \in \mathbb{R}^{C_{out}}$ represents the importance value of the corresponding channel, respectively. In the following, we drop the subscript $_{out}$ for simplicity in presentation.

6.3.1.1 Runtime Channel Importance

The runtime channel importance $\mathbf{u}_r$ of output feature $\mathbf{F}_{out}$ is predicted by a *runtime importance predictor* $f(\cdot)$, which takes $\mathbf{F}_{in}$ as input. Therefore, $\mathbf{u}_r$ can be considered as a function of $\mathbf{F}_{in}$, whose values vary over different input instances. In this chapter, we design a subnetwork to approximate $f(\cdot)$, which is expected to be small and computationally efficient. Similar to many existing dynamic pruning methods [21, 37], we use global pooling as the first layer in $f(\cdot)$, because global pooling is computationally efficient and it can reduce the dimension of $\mathbf{F}_{in}$ dramatically. We then feed the output of global pooling into a fully-connected layer without any activation function. The output of fully-connected layer is the runtime channel importance vector $\mathbf{u}_r$.

6.3.1.2 Static Channel Importance

The static channel importance $\mathbf{u}_s$ is to capture the global information for pruning, and thus is learned from the whole dataset. An *static importance predictor* $h(\cdot)$ is introduced to predict $\mathbf{u}_s$ by taking convolutional filters' features and a learnable vector $\mathbf{v} \in \mathbb{R}^{C_{out}}$ as input. We denotes $\mathbf{K} \in \mathbb{R}^{C_{out} \times C_{in} \times k \times k}$ as the convolutional filters in layer t, where k is kernel size. Three types of features are designed upon convolutional filter $\mathbf{K}$: norm $\mathbf{n} \in \mathbb{R}^{C_{out}}$, filter distance $\mathbf{d} \in \mathbb{R}^{C_{out}}$ and filter Hessian diagonal $\mathbf{H} \in \mathbb{R}^{C_{out}}$. The norm value $\mathbf{n}^{(i)}$ for i-th output channel is defined as Frobenius norm of i-th filter $\mathbf{K}^{(i)}$: $\mathbf{n}^{(i)} = \left\|\mathbf{K}^{(i)}\right\|_F$, where $\|\cdot\|_F$ stands for Frobenius norm. Inspired by FPGM [31], filter distance is an import feature for pruning. We define distance $\mathbf{d}^{(i)}$ as the Euclidean distance between the i-th filter $\mathbf{K}^{(i)}$ and a "mean" filter: $\mathbf{d}^{(i)} = \left\|\mathbf{K}^{(i)} - \frac{1}{C_{out}} \sum_{i=1}^{C_{out}} \mathbf{K}^{(i)}\right\|_F$. Due to computation complexity of Hessian matrix, we approximate Hessian diagonal using square of first-order Taylor expansion. Therefore our filter Hessian diagonal $\mathbf{H}^{(i)}$ for i-th filter is defined as the sum of approximated Hessian diagonal of all weights in i-th filter $\mathbf{K}^{(i)}$: $\mathbf{H}^{(i)} = \left\|\frac{\partial \mathcal{L}_{CNN}}{\partial \mathbf{K}^{(i)}} \otimes \mathbf{K}^{(i)}\right\|_F^2$, where $\otimes$ is element-wise multiplication and $\mathcal{L}_{CNN}$ is the CNN loss. The first-order gradient $\frac{\partial \mathcal{L}_{CNN}}{\partial \mathbf{K}^{(i)}}$ is evaluated on the whole training dataset.

The static channel importance $\mathbf{u}_s$ is then generated by the static importance predictor $h(\cdot)$ taking input of $\mathbf{n}$, $\mathbf{d}$, $\mathbf{H}$ and $\mathbf{v}$ as $\mathbf{u}_s^{(i)} = h(\mathbf{n}^{(i)}, \mathbf{d}^{(i)}, \mathbf{H}^{(i)}, \mathbf{v}^{(i)})$, where $\mathbf{u}_s^{(i)}$ represents the static importance value of i-th output channel and $\mathbf{u}_s$ is concatenation of $\mathbf{u}_s^{(i)}$. $h(\cdot)$ is approximated by a subnetwork which consists of two fully-connected layers with one SeLU [42] activation layer in the middle. The reason of not choosing ReLU is "dead ReLU", which indicates it may always output zeros and stops backpropagating gradient. The three feature vectors $\mathbf{n}$, $\mathbf{d}$ and $\mathbf{H}$ are generated by pretrained backbone CNN as introduced above and fixed during further training. Meanwhile $\mathbf{v}$ is randomly initialized and learned through backpropagation. The subnetwork $h(\cdot)$ shares parameters among all convolutional layers in order to learn importance globally, while $\mathbf{v}$ is layer specific to learn locally.

6.3.2 The Trade-off Pruner

We propose a *trade-off pruner* to generate a unified channel pruning decision. The goal of the trade-off pruner is to 1) prune those channels that are decided to be pruned by

both of the runtime and the static pruning components, and 2) prune a portion of the rest channels by weighted votes from the two components.

Which channels to be preserved / pruned at runtime are determined according to the values of runtime channel importance $\mathbf{u}_r$. Let $\mathbf{m}_r \in \{0,1\}^C$ denotes a decision mask for pruning, where if the value is 0, then the corresponding channel is preserved, otherwise pruned. For now, suppose a sparsity ratio a_t^r for runtime pruning has already been generated via the dynamic deep RL agent, which will be introduced in Sec. 6.3.3. We then prune $(C - \lceil a_t^r C \rceil)$ channels with the smallest importance values in $\mathbf{u}_r$. Accordingly, the value of an element in $\mathbf{m}_r$ is set to be 1 if the corresponding channel is pruned, otherwise 0. Similarly, for static pruning, given static importance $\mathbf{u}_s$ and a sparsity ratio a_t^s learned by the static deep RL agent, $(C - \lceil a_t^s C \rceil)$ channels with smallest importance values in $\mathbf{u}_s$ are pruned, and a mask $\mathbf{m}_s \in \{0,1\}^C$ is generated to indicate the static pruning results.

With the runtime and the static pruning decisions, $\mathbf{m}_r$ and $\mathbf{m}_s$, our trade-off pruner generates a unified pruning result. To be specific, we define the mask representing channels pruned by both decisions as $\mathbf{m}_o = \mathbf{m}_s \wedge \mathbf{m}_r$, where $\wedge$ is element-wise logical AND and 1/0 in mask represents logical *true* or *false*. The channels indicated to be pruned by $\mathbf{m}_o$ (i.e., the corresponding values are 1) are pruned in final. The channels which are determined to be pruned by $\mathbf{m}_r$ but not by $\mathbf{m}_s$ can be represented by a new mask $\overline{\mathbf{m}}_r = \mathbf{m}_r - \mathbf{m}_o$. Similarly, the channels which are determined to be pruned by $\mathbf{m}_r$ but not by $\mathbf{m}_s$ can be represented by another new mask $\overline{\mathbf{m}}_s = \mathbf{m}_s - \mathbf{m}_o$.

To control the trade-off between $\mathbf{m}_r$ and $\mathbf{m}_s$, we define a rate R_r denoting how much we trust the pruning decision made by $\mathbf{m}_r$, while $1 - R_r$ is for $\mathbf{m}_s$. That means the channels selected by $\overline{\mathbf{m}}_r$ will be finally pruned with the rate R_r. Specifically, the number of channels which are selected by $\overline{\mathbf{m}}_r$ and finally will be pruned is $C_r' = \lfloor R_r(\mathbf{1}^\top \overline{\mathbf{m}}_r) \rfloor$, where $\mathbf{1}^\top \overline{\mathbf{m}}_r$ returns the number of channels selected by $\overline{\mathbf{m}}_r$. We then select the first C_r'-smallest important channels which are recommended to be pruned by $\overline{\mathbf{m}}_r$ to form a mask $\widehat{\mathbf{m}}_r$. Similarly, for static pruning, we select the first C_s'-smallest important channels which are recommended to be pruned by $\overline{\mathbf{m}}_s$ to form another mask $\widehat{\mathbf{m}}_s$, where $C_s' = \lfloor (1 - R_r)(\mathbf{1}^\top \overline{\mathbf{m}}_s) \rfloor$.

The final trade-off pruning mask is defined as $\mathbf{m} = \mathbf{m}_o + \widehat{\mathbf{m}}_r + \widehat{\mathbf{m}}_s$, and the unified channel importance is simply defined as $\mathbf{u} = \mathbf{u}_r \otimes \mathbf{u}_s$. With the trade-off pruning mask $\mathbf{m}$

and the unified channel importance $\mathbf{u}$, the pruned output feature $\hat{\mathbf{F}}_{out}$ can be generated by Eq. (6.1).

6.3.3 Deep RL-based Sparsity Ratio Estimation

In this section, we present how to formulate the problems of learning the ratios a_t^s and a_t^r for static pruning and runtime pruning, as a MDP, and solve it via deep RL, respectively.

6.3.3.1 The Runtime Deep RL Agent

In the MDP for runtime pruning, we consider the t-th layer of the network as the t-th timestamp. The details of the MDP are listed as follows.

State Given an input feature map $\mathbf{F}_{in}$ of layer t, we pass it to a global pooling layer to reduce its dimension to $\mathbb{R}^{C_{in}}$, where C_{in} is the number of input channels of layer t. Since C_{in} varies among layers, we feed the output of global pooling to a layer-dependent encoder to project it to a fix-length vector $\mathbf{s}_t^r$ with dimensions of 128, which is considered as as a *state* representation of deep RL in the context of runtime pruning.

Action The *action* a_t^r is defined as the sparsity ratio at layer t. Existing deep RL-base pruning method RNP [55] uses a unified discrete actions space with k actions which are too coarse to achieve high accuracy. However, fine-grained discrete action space as large as number of channels suffers from exploration difficulty. Therefore, instead of using discrete action spaces, we propose a continuous action space with action $a_t^r \in (0, 1]$. To avoid over-pruning the filters and crashing in training, we set a minimum sparsity ratio $+\alpha$ such that $a_t^r \in (+\alpha, 1]$.

Reward The reward function is proposed to consider both network accuracy and computation budget. We define the accuracy relative reward based on the loss of pruned backbone network as $R_{acc}^r = -\mathcal{L}_{CNN}$, where $\mathcal{L}_{CNN}$ is the CNN loss, and it may vary in scale among different training stage, i.e. large at beginning of training and small near convergence. To avoid the instability brought by the reward scale, R_{acc}^r is normalized by a moving average via $R_{acc}^{r'} = R_{acc}^r / \beta_b$, where $\beta_b = \lambda\beta_{b-1} + (1-\lambda)R_{acc}^r$ is the moving average at the b-th training batch and λ is the weight.

To force computation of the pruned network under a given computation budget, we define a exponential reward function of budget regarding reward $R^r_{bud} = 1 - \exp(\alpha_1(B_{com} - \overline{B}_{com}))$ if $B_{com} > \overline{B}_{com}$, otherwise, $R^r_{bud} = 0$, where B_{com} is the computation consumption, which is calculated based on the current of pruned strategy, and $\overline{B}_{com}$ is the given computation budget constraint. Finally we sum up the two rewards to form sparse rewards $R^r_t = R^{r'}_{acc} + R^r_{bud}$ if $t = T$, and $R^r_t = 0$ if $t < T$.

Actor-Critic Agent To solve the continuous action space problem, we choose a commonly used actor-critic agent with a Gaussian policy. Actor-critic agent consists of two components: 1) actor outputs the mean and variance to form a Gaussian policy where the continuous action is sampled from; 2) critic outputs a scalar predicting the future discounted accumulated reward and assists the policy training. Actor network and Critic network share one-layer RNN with hidden state size of 128 which takes state $\mathbf{s}^r_t$ as input. The output of RNN is fed into actor specific network constructed by two branches of fully-connected layers, leading to the mean and variance of the Gaussian policy. The action is sampled for the Gaussian distribution outputed by the actor: $a^r_t \sim \mathcal{N}(\mu(\mathbf{s}^r_t; \theta^r), \sigma(\mathbf{s}^r_t; \theta^r))$, where $\mu(\mathbf{s}^r_t; \theta^r)$ and $\sigma(\mathbf{s}^r_t; \theta^r)$ is the mean and variance outputed from actor network. The Critic specific network has one fully-connected layer after the shared RNN, and outputs the predictive value $V(\mathbf{s}^r_t; \theta^r)$.

To optimize the actor-critic agent, Proximal Policy Optimization (PPO) [72] is used. Note that we relax the action a^r_t to $(-\infty, +\infty)$ in PPO, and use truncate function to clip a^r_t in $(+\alpha, 1]$ when perform pruning. Besides, an additional regularizer $\mathcal{L}_a$ is introduced to restrict the relaxed a^r_t staying in range $(+\alpha, 1]$ as $\mathcal{L}_a = \frac{1}{2}||a^r_t - \max(\min(a^r_t, 1), +\alpha)||^2_2$.

6.3.3.2 The Static Deep RL Agent

Similar to runtime pruning, the MDP in static pruning is also formulated layer-by-layer. The difference against runtime pruning is the definition of state and reward. The **state** $\mathbf{s}^s_t$ in static pruning is defined as the full shape of $\mathbf{F}_{out}$, and does not depend on $\mathbf{F}_{out}$ and the current input data. **Action** a^s_t is sampled from actor's outputed Gaussian policy, and it represents the sparsity ratio at layer t. The **reward** function takes both network accuracy and parameters budget into consideration. The accuracy relative is defined

Algorithm 4 Training process

1: **Input:** pretrained backbone CNN, computation budget $\bar{B}_{com}$, storage budget $\bar{B}_{param}$
2: **Output:** CNN, importance predictor $f(\cdot)$, static importance u_s, runtime and static deep RL agents
3: $\mathbf{u_s} \leftarrow \mathbf{1}$, $a_t^s \leftarrow 1$, $a_t^r \leftarrow 1$
4: **while** not converge **do**
5: fix $\mathbf{u_s}$, update $f(\cdot)$ and CNN
6: **end while**
7: **while** not converge **do**
8: Update $h(\cdot)$, $\mathbf{v}$, $f(\cdot)$ and CNN {$\mathbf{u_s}$ is generated by $h(\cdot)$ and $\mathbf{v}$}
9: **end while**
10: Use deep RL agents to predict actions a_t^r and a_t^s at each layer t
11: **while** not converge **do**
12: **for** $i \in 1...N_1$ **do**
13: Compute rewards R_t^r and R_t^s using budget $\bar{B}_{com}$ and $\bar{B}_{param}$
14: Fix $\mathbf{u_s}$, $f(\cdot)$ and CNN, update deep RL agents
15: **end for**
16: **for** $i \in 1...N_2$ **do**
17: Fix deep RL agents, update $h(\cdot)$, $\mathbf{v}$, $f(\cdot)$ and CNN
18: **end for**
19: **end while**

as the same as that in runtime pruning, i.e., $R_{acc}^s = R_{acc}^{r'}$. To reduce the number of parameters of network to satisfy the parameters storage budget, the parameters relative reward is defined in an exponential form as $R_{param}^s = 1 - \exp(\alpha_2(B_{param} - \overline{B}_{param}))$ if $B_{param} > \overline{B}_{param}$, otherwise $R_{param}^s = 0$, where B_{param} is the number of preserved parameters after static pruning and $\overline{B}_{param}$ is the parameters storage budget.

Actor-Critic Agent This agent is similar to the one in runtime pruning. It has the same architecture as runtime pruning but differs in introducing a fully-connected layer as the encoder before RNN. This agent is also optimized by PPO.

6.3.4 Training Process

Alg. 4 illustrates the training process of our method. Given a pretrained backbone CNN, firstly we finetune the CNN and train runtime importance predictor jointly (line 5), with sparsity $a_t^r = 1$ and fixed all static pruning importance $\mathbf{u}_s$ to 1. We then remove the restriction on the static pruning importance $\mathbf{u}_s$ which is now generated by importance

predictor $h(\cdot)$ and layer-specific learnable vector $\mathbf{v}$. We train static pruning $h(\cdot)$, $\mathbf{v}$, the backbone CNN and the runtime importance predictor (line 8), with sparsity $a_t^s = 1$ and runtime pruning sparsity fixed as $a_t^r = 1$. After finetuning, we use the deep RL agents to predict the sparsity given computation and storage constraints. The deep RL agents and the CNN with runtime/static importance are trained in alternating manner: We first fix the CNN as well as runtime/static importance and train two deep RL agents, regarding the CNN as environments (line 12 to line 15). We then fix two agents and finetune the CNN and runtime/static importance (line 16 to line 18). We repeat these two steps until convergence.

For CIFAR-10, we train ($N_1 = 1560$) batches (4 epochs) for deep RL agents (line 12 to line 15) and ($N_2 = 780$) batches (2 epochs) for $h(\cdot)$, $\mathbf{v}$, $f(\cdot)$ and CNN (line 16 to 18) at each iteration. The total number of iterations is 40 (for line 11 to 19). For ImageNet, at each iteration of training, $N_1 = 1200$ and $N_2 = 600$ except last iteration. The total number of iterations is 64. At last iteration, $N_2 = 200{,}000$ for finetuning the CNN. We use Adam optimizer for both deep RL agent and CNN, and set learning rate to 10^{-6} for the deep RL agents. For CNN finetuning and runtime/static importance training, the learning rate is 10^{-3} on CIFAR-10. On ImageNet ILSVRC2012, the learning rate is 10^{-4}.

6.3.5 Inference

While runtime pruning performs for individual instance dynamically, static pruning strategy does not depend on individual input data points. Therefore, the action a_t^s predicted by static agent and the static importance vector $\mathbf{u_s}$ generated by static importance predictor $h(\cdot)$ are fixed, regardless of input instances. We can consequently remove static agent and $h(\cdot)$ in inference, and eliminate the storage and computation cost of them. With the action a_t^s and hyper-parameter R_r, we can decide which filters can be pruned permanently. Specifically, channels with $((1 - a_t^s)(1 - R_r))$-smallest static importance values are pruned permanently.

6.4 Experiment

We evaluate our deep RL pruning framework on two benchmark datasets: CIFAR-10 [44] and ImageNet ILSVRC2012 [70]. Our goal is to verify our framework is able to achieve

comparable or even better performance in terms of prediction accuracy as other state-of-the-art channel pruning methods, but save much more storage space. We further analyze the effect of hyper-parameters and different sparsity settings on CIFAR-10. For CIFAR-10, we use M-CifarNet [96] as the backbone CNN. On ImageNet ILSVRC2012, ResNet-18 [29] and MobileNet [36] are used as the backbone CNN.

6.4.1 Experimental results on CIFAR-10

We compare our proposed method with the following state-of-the-art runtime pruning methods: FBS [21], RNP [55] on CIFAR-10. The comparison results at sparsity 0.5 and 0.7 are shown in Table 6.1 and Table 6.2 respectively. The sparsity is defined as the ratio of preserved output channels after pruning at every layer.

Method	Baseline acc.	Acc.	$\Delta acc.$	Speed-up	#Params	GPU Time	CPU Time
FBS [21]	91.37	89.88	-1.49	**3.93**×	1.11×	10.9 ms	172.0ms
RNP [55]	92.07	84.93	-7.14	3.56×	1.00×	11.1 ms	175.3ms
ours ($R_r = 1$)	92.07	91.425	**-0.645**	3.92×	1.31×	11.2 ms	178.3ms
ours ($R_r = 0.5$)	92.07	91.228	**-0.842**	3.92×	**0.78**×	**9.8 ms**	**110.7ms**

Table 6.1: Comparison with state-of-the-art runtime pruning methods on CIFAR-10 at sparsity 0.5. Speed-up is calculated on MACs.

Method	Baseline acc.	Acc.	$\Delta acc.$	Speed-up	#Params
FBS [21]	91.37	91.23	-0.14	**2**×	1.11×
ours ($R_r = 1.0$)	92.07	93.184	**1.114**	1.99×	**1.31**×
ours ($R_r = 0.5$)	92.07	92.714	**0.629**	1.99×	**0.97**×

Table 6.2: Comparison with state-of-the-art runtime pruning methods on CIFAR-10 at sparsity 0.7. Speed-up is calculated on MACs.

Note that for a fair comparison with other methods, the computation and storage budget constraints in our method is calculated according to the sparsity of other methods. Under these constraints, our method does not necessarily lead to the same sparsity as other methods in each layer. RNP cannot set exact sparsity ratio. Instead, its average sparsity ratio is accessible only during testing, which is 0.537 in Table 6.1. The result of FBS is reproduced using the released code[2]. The column *#Params* represents the number

[2]https://github.com/deep-fry/mayo

of parameters compared to the backbone CNN. Speed-up is calculated on MACs. The column of GPU time is the time of forwarding a batch of 256 images on an NVIDIA P100 GPU and CPU time is on CPU.

Table 6.1 shows that our method outperforms other state-of-the-art methods, achieving highest accuracy at an overall sparsity ratio of 0.5. Our method has very close computation speed-up compared to FBS, but outperforms FBS around 0.65% to 0.86%. When the runtime pruning strategy is solely considered by setting $R_r = 1$, our method surpasses other comparison methods, indicating that our deep RL-based framework improves the performance of channel runtime pruning. By balancing runtime and static pruning via setting $R_r = 0.5$, our method reduces the number of the overall stored parameters and achieves lower accuracy drop than other methods. Table 6.2 shows that our method outperforms FBS at sparsity of 0.7. When $R_r = 0.5$, our method achieves better performance than the baseline CNN with 2× speed-up and contains less parameters.

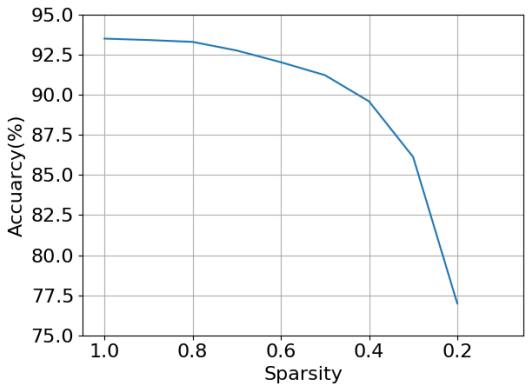


Fig. 6.2: Accuracy drop for M-CifarNet on CIFAR-10 with computational budget.

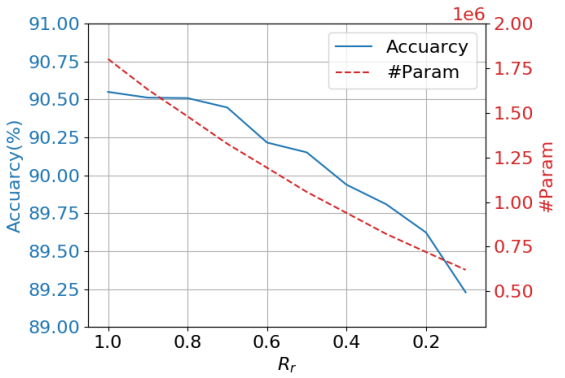


Fig. 6.3: Trade-off between runtime and static pruning at sparsity 0.45.

6.4.2 Trade-off between Runtime and Static Pruning

Fig. 6.2 shows the performance of various sparsity ratios in our method. Again, our method does not prune with one single sparsity ratio for all layers, but uses the sparsity ratio to calculate computation and storage constraints, with which the sparsity ratio is learned for each layer. Fig. 6.2 demonstrates that our method holds the accuracy when sparsity is larger than about 0.5, which corresponds to about 4× computational acceleration.

We also study the relation between R_r and network compactness in our framework. Fig. 6.3 demonstrates the impact of R_r when sparsity is 0.45. The hyper-parameter R_r determines how much we trust about runtime pruning. With R_r close to 1, the accuracy becomes higher due to the more dynamic network flexibility but the space of the parameters storage also increases. When R_r diminishes, the network accuracy decreases but the parameter storage is reduced.

Method	Baseline acc.	Acc.	$\Delta acc.$
ours (runtime only $R_r = 1$)	92.07	91.425	**-0.645**
ours ($R_r = 0.5$)	92.07	91.228	**-0.842**
ours (separate static and runtime)	92.07	90.03	-2.04
FPGM+FBS	92.07	90.456	-1.614

Table 6.3: Comparison to methods with separately static and runtime pruning on CIFAR-10 at sparsity 0.5.

6.4.3 Comparison to Separately Static and Runtime Pruning

In this section, we compare our method with two additional baseline methods. One is a variation of our method by separately training static pruning and runtime pruning. In this method, we start from a pretrained backbone CNN, $f(\cdot)$ and u_s. Then we add the static deep RL agent to prune channels statically by learning the static policy and u_s. Finally, we add the runtime deep RL agent to prune channels dynamically, by fixing the static deep RL agent and u_s, and updating the runtime deep RL agent and $f(\cdot)$ only. Another method is to combine state-of-the-art static and runtime pruning methods. We start from a pretrain backbone CNN, and then prune channels with the static pruning method FPGM [31], and finally prune channels with the runtime method FBS [21]. The experimental results are shown in Table 6.3.

6.4.4 Hyper-parameter choosing: learning rate

In this section, we set up an experiment with comparing various learning rates for training procedure of our proposed method. Table 6.4 shows the accuracy results of various learning rates settings. The first column indicates the learning rate for deep RL agents

Deep RL agents lr	CNN & $f(\cdot)$ lr	Acc.
10^{-6}	10^{-3}	**91.23**
10^{-6}	10^{-2}	10.00
10^{-6}	10^{-4}	90.67
10^{-5}	10^{-3}	89.95
10^{-7}	10^{-3}	90.79

Table 6.4: Comparison of various learning rates on CIFAR-10.

and second column indicates the learning rate for training $f(\cdot)$ and CNN. It shows that our learning rates setting, 10^{-6} for deep RL agents and 10^{-3} for $f(\cdot)$ and CNN, achieves the best accuracy performance.

6.4.5 Experimental results on ImageNet ILSVRC2012

We compare our method with state-of-the-art channel pruning methods on ImageNet ILSVRC2012. In the first experiment as shown in Table 6.5, we use ResNet-18 as the backbone CNN. Among the methods for comparison, FBS [21] and CGNN [38] are runtime pruning methods and AMC [33] is deep RL-based static pruning. The overall sparsity ratio of our method is 0.7, which is under the same setting of FBS. Our method with $R_r = 0.5$ achieves the smallest top-1 accuracy drop compared with other methods, and also achieves the highest top-1 accuracy after pruning. Our method has very close MACs to FBS, while the number of preserved parameters is reduced to 81.2% of the baseline.

We also implement the our method on MobileNet which is a more compact than ResNet-18. Table 6.6 shows the comparison results on MobileNet. AMC [33] is deep RL-based static pruning and NetAdapt [88] is heuristics-based static pruning. 75% MobileNet [36] is a variant of MobileNet with width multiplier 75% and input size of 224. Table 6.6 shows that our method outperforms the comparison methods on both top-1 and top-5 accuracy.

6.5 Conclusion

This chapter successfully extends the scope of deep RL applications to neural network compressions. In this chapter, we present a deep reinforcement learning based framework

Method	Baseline top-1 acc.	Top-1 acc.	Δ top-1 acc.	Baseline top-5 acc.	Top-5 acc.	Δ top-5 acc.	Speed-up	#Params
DCP [100]	69.64	67.35	-2.29	88.98	**88.86**	**-0.12**	1.71×	0.71×
FPGM [31]	70.28	68.41	-1.87	89.63	88.48	-1.15	1.71×	0.72×
Dynamic Sparse Graph [57]	69.48	64.8	-4.68	-	-	-	1.4 ×	-
CGNN [38]	69.02	67.95	-1.07	88.84	88.21	-0.63	1.63×	-
FBS [21]	70.71	68.17	-2.54	89.68	88.22	-1.46	1.98×	1.12×
AMC [33]	69.76	66.63	-3.13	89.08	87.20	-1.88	2.00×	0.76×
Ours ($R_r = 0.5$)	69.76	**68.73**	**-1.03**	89.08	88.65	-0.43	1.94×	0.81×

Table 6.5: Comparison with the state-of-the-art channel pruning with ResNet-18 on ImageNet. Speed-up is calculated on MACs.

Method	Baseline top-1 acc.	Top-1 acc.	Δ top-1 acc.	Baseline top-5 acc.	Top-5 acc.	Δ top-5 acc.	Speed-up
AMC [33]	70.9	70.5	-0.4	89.5	89.3	-0.2	2.00×
NetAdapt [88]	-	69.1	-	-	-	-	2.00×
75% MobileNet [36]	70.9	68.4	-2.5	89.9	88.2	-1.7	1.79×
Ours ($R_r = 0.5$)	70.9	**70.6**	**-0.3**	89.5	**89.6**	**+0.1**	2.00×

Table 6.6: Comparison with the state-of-the-art channel pruning with MobileNet on ImageNet. Speed-up is calculated on MACs.

for deep neural network channel pruning in both runtime and static schemes. Specifically, channels are pruned according to input feature as runtime pruning, and based on entire training dataset as static pruning, with 2 reinforcement agents to determine the corresponding sparsity. Our method combines the merits of runtime and static pruning, and provides trade-off between storage and dynamic flexibility. Extensive experiments demonstrate the effectiveness of our proposed method.

Chapter 7

Summary

In this thesis, I present studies on deep reinforcement learning applications on control from pixels and neural network pruning. I discuss the challenges to real-world deep RL applications in Chapter 1, including generalization and sample efficiency issues for pixels controls, and the lack of extending deep RL to new scopes of applications. In Chapter 2, I describe the preliminaries of deep reinforcement learning, such as the MDPs assumption and Bellman equation of the value functions, and several fundamental algorithms for deep RL. Chapter 3 surveys the recent research progress of RL agents for control from pixels, including 4 classes of approaches: reconstruct-based methods, data augmentation, contrastive learning, and behavioral metric. Besides, I also introduce the previous research on neural network pruning using deep RL.

Since previous approaches of control from pixels have shortages of generalization or rarely consider the properties, e.g. reward or transition probability, of an MDP, two approaches based on behavioral metrics to address the generalization and efficiency problems are proposed in Chapter 4 and Chapter 5.

Chapter 4 presents a framework with meta-learners that learn behavioral metrics for capturing task-relevant and generalizable state representation in deep RL. In this framework, the pixel observations are encoded into rewards and dynamics representations that are measured with behavioral metrics. The meta-learners are designed to self-learn by measuring behavioral metrics for rewards and dynamics representations, and a learnable strategy is adopted to balance these two representations. The state representations that are learned by this method show their generalization and transfer ability in various experiments and achieve new state-of-the-art results.

Chapter 5 illustrates a study that looks deeper into behavioral metric-based representation learning and discovers the issues in approximating the metrics. To address such issues, a new form of behavioral metric is proposed and meanwhile, a practical algorithm for approximating the metic is developed. Theoretical analysis of the proposed method is given and empirical results verify the robustness, effectiveness, and generalization of the proposed method.

Besides, I also propose to use RL algorithms to help neural network compression. Chapter 6 introduces a deep RL-based framework for deep neural network channel pruning in both runtime and static schemes. This framework combines the benefit of runtime and static pruning and provides a trade-off between dynamic flexibility and storage efficiency. Unlike previous pruning methods that specify the same sparsity ratios for every layer, this approach utilizes RL for sequential decision making so that the sparsity ratio is learned by deep RL agents in a layer-by-layer manner. The experimental results demonstrate the effectiveness of the proposed method compared to state-of-the-art methods.

7.1 Discussion

Although recent research on deep RL has increased the generalization in control tasks with pixel input, there are still a lot of challenges that remain to be solved in real-world scenarios. Current research is mostly addressing the generalization in visual features, resulting in agents robust in noisy backgrounds, view-points, texture, etc. However, learning in real-world scenarios may require generalization in the semantic level and transfer ability among different tasks. Recent research of zero-shot policy transfer links to generalize to different tasks but it is limited to build upon domain-specific solutions. Therefore, generalization problems in real-world scenarios still need to be further studied.

Besides, there are likely to be more exciting areas in which deep RL could be applied to NLP and CV, such as dialog systems, machine translation, and image and video generation. In these applications, human feedback can be an important factor influencing the learning process. Defining and formulating the reward model based on human feedback may be crucial to the success of these applications. Moreover, using DRL in such large models in NLP and CV also poses significant data efficiency challenges. These problems could be explored in future research.

References

[1] M. Mehdi Afsar, Trafford Crump, and Behrouz Far. Reinforcement learning based recommender systems: A survey. *ACM Comput. Surv.*, jun 2022. ISSN 0360-0300. doi: 10.1145/3543846. URL https://doi.org/10.1145/3543846. Just Accepted.

[2] Rishabh Agarwal, Marlos C. Machado, Pablo Samuel Castro, and Marc G Bellemare. Contrastive behavioral similarity embeddings for generalization in reinforcement learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=qda7-sVg84.

[3] Tolga Bolukbasi, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama. Adaptive neural networks for efficient inference. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 527–536. JMLR. org.

[4] Pablo Samuel Castro. Scalable methods for computing state similarity in deterministic markov decision processes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 10069–10076, 2020.

[5] Pablo Samuel Castro, Tyler Kastner, Prakash Panangaden, and Mark Rowland. MICo: Improved representations via sampling-based state similarity for markov decision processes. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=wFp6kmQELgu.

[6] Jianda Chen and Sinno Pan. Learning generalizable representations for reinforcement learning via adaptive meta-learner of behavioral similarities. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=zBOI9LFpESK.

[7] Jianda Chen, Shangyu Chen, and Sinno Jialin Pan. Storage efficient and dynamic flexible runtime channel pruning via deep reinforcement learning. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 14747–14758. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/a914ecef9c12ffdb9bede64bb703d877-Paper.pdf.

[8] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1597–1607. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/chen20j.html.

[9] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1282–1289. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/cobbe19a.html.

[10] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1.

[11] Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando de Freitas. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems 26*, pages 2148–2156. Curran Associates, Inc.

[12] Terrance Devries and Graham W. Taylor. Improved regularization of convolutional neural networks with cutout. *CoRR*, abs/1708.04552, 2017. URL http://arxiv.org/abs/1708.04552.

[13] Henghui Ding, Xudong Jiang, Bing Shuai, Ai Qun Liu, and Gang Wang. Semantic correlation promoted shape-variant context for segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8885–8894.

[14] Xin Dong, Shangyu Chen, and Sinno Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *Advances in Neural Information Processing Systems*, pages 4857–4867.

[15] Alexey Dosovitskiy, Jost Tobias Springenberg, Martin Riedmiller, and Thomas Brox. Discriminative unsupervised feature learning with convolutional neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL `https://proceedings.neurips.cc/paper/2014/file/07563a3fe3bbe7e3ba84431ad9d055af-Paper.pdf`.

[16] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[17] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1407–1416. PMLR, 10–15 Jul 2018. URL `https://proceedings.mlr.press/v80/espeholt18a.html`.

[18] Norm Ferns, Prakash Panangaden, and Doina Precup. Metrics for finite markov decision processes. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, UAI '04, page 162–169, Arlington, Virginia, USA, 2004. AUAI Press. ISBN 0974903906.

[19] Norm Ferns, Prakash Panangaden, and Doina Precup. Bisimulation metrics for continuous markov decision processes. *SIAM J. Comput.*, 40(6):1662–1714, December 2011. ISSN 0097-5397. doi: 10.1137/10080484X. URL `https://doi.org/10.1137/10080484X`.

[20] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1587–1596. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/fujimoto18a.html.

[21] Xitong Gao, Yiren Zhao, Łukasz Dudziak, Robert Mullins, and Cheng zhong Xu. Dynamic channel pruning: Feature boosting and suppression. In *International Conference on Learning Representations*. URL https://openreview.net/forum?id=BJxh2j0qYm.

[22] Carles Gelada, Saurabh Kumar, Jacob Buckman, Ofir Nachum, and Marc G. Bellemare. DeepMDP: Learning continuous latent space models for representation learning. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2170–2179. PMLR, 09–15 Jun 2019. URL http://proceedings.mlr.press/v97/gelada19a.html.

[23] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1861–1870. PMLR, 10–15 Jul 2018. URL http://proceedings.mlr.press/v80/haarnoja18b.html.

[24] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2555–2565. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/hafner19a.html.

[25] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference*

on Learning Representations, 2020. URL https://openreview.net/forum?id=S1lOTC4tDS.

[26] Danijar Hafner, Timothy P Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=0oabwyZbOu.

[27] Nicklas Hansen, Rishabh Jangir, Yu Sun, Guillem Alenyà, Pieter Abbeel, Alexei A Efros, Lerrel Pinto, and Xiaolong Wang. Self-supervised policy adaptation during deployment. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=o_V-MjyyGV_.

[28] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, . doi: 10.1109/ICCV.2017.322.

[29] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, . doi: 10.1109/CVPR.2016.90.

[30] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[31] Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4340–4349, .

[32] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *The IEEE International Conference on Computer Vision (ICCV)*, .

[33] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–800, 2018.

[34] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. 2016.

[35] Irina Higgins, Arka Pal, Andrei Rusu, Loic Matthey, Christopher Burgess, Alexander Pritzel, Matthew Botvinick, Charles Blundell, and Alexander Lerchner. DARLA: Improving zero-shot transfer in reinforcement learning. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1480–1490. PMLR, 06–11 Aug 2017. URL `https://proceedings.mlr.press/v70/higgins17a.html`.

[36] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications.

[37] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[38] Weizhe Hua, Christopher De Sa, Zhiru Zhang, and G Edward Suh. Channel gating neural networks.

[39] Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, Mustafa Suleyman, and Andrew Zisserman. The kinetics human action video dataset, 2017.

[40] Mete Kemertas and Tristan Ty Aumentado-Armstrong. Towards robust bisimulation metric learning. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL `https://openreview.net/forum?id=ySFGlFjgIfN`.

[41] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. 2014.

[42] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 971–980. Curran Associates, Inc. URL `http://papers.nips.cc/paper/6698-self-normalizing-neural-networks.pdf`.

[43] Petar Kormushev, Sylvain Calinon, and Darwin G. Caldwell. Reinforcement learning in robotics: Applications and real-world challenges. *Robotics*, 2(3): 122–148, 2013. ISSN 2218-6581. doi: 10.3390/robotics2030122. URL `https://www.mdpi.com/2218-6581/2/3/122`.

[44] Alex Krizhevsky. Learning multiple layers of features from tiny images.

[45] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL `https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf`.

[46] Sascha Lange and Martin Riedmiller. Deep auto-encoder neural networks in reinforcement learning. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2010.

[47] Michael Laskin, Aravind Srinivas, and Pieter Abbeel. CURL: Contrastive unsupervised representations for reinforcement learning. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 5639–5650. PMLR, 13–18 Jul 2020. URL `https://proceedings.mlr.press/v119/laskin20a.html`.

[48] Misha Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. In H. Larochelle,

M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 19884–19895. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/e615c82aba461681ade82da2da38004a-Paper.pdf.

[49] Hei Law and Jia Deng. Cornernet: Detecting objects as paired keypoints. In *The European Conference on Computer Vision (ECCV)*.

[50] Nevena Lazic, Craig Boutilier, Tyler Lu, Eehern Wong, Binz Roy, MK Ryu, and Greg Imwalle. Data center cooling using model-predictive control. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/059fdcd96baeb75112f09fa1dcc740cc-Paper.pdf.

[51] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989. doi: 10.1162/neco.1989.1.4.541.

[52] Alex X. Lee, Anusha Nagabandi, Pieter Abbeel, and Sergey Levine. Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 741–752. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/08058bf500242562c0d031ff830ad094-Paper.pdf.

[53] Kimin Lee, Kibok Lee, Jinwoo Shin, and Honglak Lee. Network randomization: A simple technique for generalization in deep reinforcement learning. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HJgcvJBFvB.

[54] 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. In *ICLR (Poster)*, 2016. URL http://arxiv.org/abs/1509.02971.

[55] Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. Runtime neural pruning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 2181–2191. Curran Associates, Inc., 2017. URL `http://papers.nips.cc/paper/6813-runtime-neural-pruning.pdf`.

[56] Lanlan Liu and Jia Deng. Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[57] Liu Liu, Lei Deng, Xing Hu, Maohua Zhu, Guoqi Li, Yufei Ding, and Yuan Xie. Dynamic sparse graph for efficient deep learning. In *International Conference on Learning Representations*, . URL `https://openreview.net/forum?id=H1goBoR9F7`.

[58] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2736–2744, .

[59] Szymon Łukaszyk. A new concept of probability metric and its applications in approximation of scattered data sets. *Computational Mechanics*, 33(4):299–304, 2004.

[60] Jian-Hao Luo and Jianxin Wu. Autopruner: An end-to-end trainable filter pruning method for efficient deep model inference. abs/1805.08941. URL `http://arxiv.org/abs/1805.08941`.

[61] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *The IEEE International Conference on Computer Vision (ICCV)*.

[62] Marc Masana, Joost van de Weijer, Luis Herranz, Andrew D. Bagdanov, and Jose M. Alvarez. Domain-adaptive deep network compression. In *The IEEE International Conference on Computer Vision (ICCV)*.

[63] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533, February 2015. ISSN 00280836. URL `http://dx.doi.org/10.1038/nature14236`.

[64] 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 Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937, New York, New York, USA, 20–22 Jun 2016. PMLR. URL `https://proceedings.mlr.press/v48/mniha16.html`.

[65] Hanyu Peng, Jiaxiang Wu, Shifeng Chen, and Junzhou Huang. Collaborative channel pruning for deep networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5113–5122. PMLR.

[66] Silviu Pitis, Elliot Creager, and Animesh Garg. Counterfactual data augmentation using locally factored dynamics. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 3976–3990. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/294e09f267683c7ddc6cc5134a7e68a8-Paper.pdf`.

[67] Jordi Pont-Tuset, Federico Perazzi, Sergi Caelles, Pablo Arbeláez, Alex Sorkine-Hornung, and Luc Van Gool. The 2017 davis challenge on video object segmentation, 2018.

[68] Roberta Raileanu, Max Goldstein, Denis Yarats, Ilya Kostrikov, and Rob Fergus. Automatic data augmentation for generalization in deep reinforcement learning, 2021.

[69] Clemens Rosenbaum, Tim Klinger, and Matthew Riemer. Routing networks: Adaptive selection of non-linear functions for multi-task learning. In *International Conference on Learning Representations*.

[70] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. 115(3):211–252. ISSN 1573-1405. doi: 10.1007/s11263-015-0816-y. URL `https://doi.org/10.1007/s11263-015-0816-y`.

[71] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1889–1897, Lille, France, 07–09 Jul 2015. PMLR. URL `http://proceedings.mlr.press/v37/schulman15.html`.

[72] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL `http://arxiv.org/abs/1707.06347`.

[73] Max Schwarzer, Ankesh Anand, Rishab Goel, R Devon Hjelm, Aaron Courville, and Philip Bachman. Data-efficient reinforcement learning with self-predictive representations. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=uCQfPZwRaUu`.

[74] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, Jan 2016. ISSN 1476-4687. doi: 10.1038/nature16961. URL `https://doi.org/10.1038/nature16961`.

[75] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*.

[76] Austin Stone, Oscar Ramirez, Kurt Konolige, and Rico Jonschkowski. The distracting control suite – a challenging benchmark for reinforcement learning from pixels, 2021.

[77] Adam Stooke, Kimin Lee, Pieter Abbeel, and Michael Laskin. Decoupling representation learning from reinforcement learning. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 9870–9879. PMLR, 18–24 Jul 2021. URL `https://proceedings.mlr.press/v139/stooke21a.html`.

[78] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, 1st edition. ISBN 0262193981.

[79] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018. ISBN 0262039249.

[80] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. Deepmind control suite, 2018.

[81] Víctor Uc-Cetina, Nicolás Navarro-Guerrero, Anabel Martin-Gonzalez, Cornelius Weber, and Stefan Wermter. Survey on reinforcement learning for language processing. *Artificial Intelligence Review*, Jun 2022. ISSN 1573-7462. doi: 10.1007/s10462-022-10205-5. URL `https://doi.org/10.1007/s10462-022-10205-5`.

[82] Aäron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *CoRR*, abs/1807.03748, 2018. URL `http://arxiv.org/abs/1807.03748`.

[83] Niklas Wahlström, Thomas B. Schön, and Marc Peter Deisenroth. From pixels to torques: Policy learning with deep dynamical models, 2015.

[84] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 409–424.

[85] Manuel Watter, Jost Springenberg, Joschka Boedecker, and Martin Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper/2015/file/a1afc58c6ca9540d057299ec3016d726-Paper.pdf.

[86] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in neural information processing systems*, pages 2074–2082.

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

[88] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. In *ECCV 2018*.

[89] Denis Yarats, Ilya Kostrikov, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=GY6-6sTvGaf.

[90] Denis Yarats, Amy Zhang, Ilya Kostrikov, Brandon Amos, Joelle Pineau, and Rob Fergus. Improving sample efficiency in model-free reinforcement learning from images. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(12):10674–10681, May 2021. URL https://ojs.aaai.org/index.php/AAAI/article/view/17276.

[91] Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Mastering visual continuous control: Improved data-augmented reinforcement learning. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=_SJ-_yyes8`.

[92] Chang Ye, Ahmed Khalifa, Philip Bontrager, and Julian Togelius. Rotation, translation, and cropping for zero-shot generalization. In *2020 IEEE Conference on Games (CoG)*, pages 57–64, 2020. doi: 10.1109/CoG47356.2020.9231907.

[93] Amy Zhang, Yuxin Wu, and Joelle Pineau. Natural environment benchmarks for reinforcement learning, 2018.

[94] Amy Zhang, Rowan Thomas McAllister, Roberto Calandra, Yarin Gal, and Sergey Levine. Learning invariant representations for reinforcement learning without reconstruction. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=-2FCwDKRREu`.

[95] Marvin Zhang, Sharad Vikram, Laura Smith, Pieter Abbeel, Matthew Johnson, and Sergey Levine. SOLAR: Deep structured representations for model-based reinforcement learning. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 7444–7453. PMLR, 09–15 Jun 2019. URL `http://proceedings.mlr.press/v97/zhang19m.html`.

[96] Yiren Zhao, Xitong Gao, Robert Mullins, and Chengzhong Xu. Mayo: A framework for auto-generating hardware friendly deep neural networks. In *Proceedings of the 2nd International Workshop on Embedded and Mobile Deep Learning*, EMDL'18, pages 25–30, New York, NY, USA. ACM. ISBN 978-1-4503-5844-6. doi: 10.1145/3212725.3212726. URL `http://doi.acm.org/10.1145/3212725.3212726`.

[97] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. In *arXiv preprint arXiv:1904.07850*.

[98] Yi Zhu, Karan Sapra, Fitsum A. Reda, Kevin J. Shih, Shawn Newsam, Andrew Tao, and Bryan Catanzaro. Improving semantic segmentation via video propagation and label relaxation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[99] Yuke Zhu, Josiah Wong, Ajay Mandlekar, and Roberto Martín-Martín. robosuite: A modular simulation framework and benchmark for robot learning. In *arXiv preprint arXiv:2009.12293*, 2020.

[100] Zhuangwei Zhuang, Mingkui Tan, Bohan Zhuang, Jing Liu, Yong Guo, Qingyao Wu, Junzhou Huang, and Jinhui Zhu. Discrimination-aware channel pruning for deep neural networks. In *Advances in Neural Information Processing Systems*, pages 875–886.