Online Prototype Alignment for Few-shot Policy Transfer

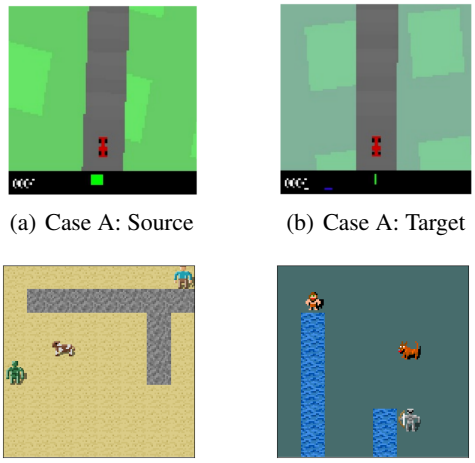
Qi Yi [1 2 3] **Rui Zhang** [2] **Shaohui Peng** [2 4 3] **Jiaming Guo** [2 4 3] **Yunkai Gao** [1 2 3] **Kaizhao Yuan** [2 4 3] **Ruizhi Chen** [5]
Siming Lan [1 2 3] **Xing Hu** [2] **Zidong Du** [2] **Xishan Zhang** [2 3] **Qi Guo** [2] **Yunji Chen** [2 4]

Abstract

Domain adaptation in reinforcement learning (RL) mainly deals with the changes of observation when transferring the policy to a new environment. Many traditional approaches of domain adaptation in RL manage to learn a mapping function between the source and target domain in explicit or implicit ways. However, they typically require access to abundant data from the target domain. Besides, they often rely on visual clues to learn the mapping function and may fail when the source domain looks quite different from the target domain. To address these problems, we propose a novel framework Online Prototype Alignment (OPA) to learn the mapping function based on the functional similarity of elements and is able to achieve the few-shot policy transfer within only several episodes. The key insight of OPA is to introduce an exploration mechanism that can interact with the unseen elements of the target domain in an efficient and purposeful manner, and then connect them with the seen elements in the source domain according to their functionalities (instead of visual clues). Experimental results show that when the target domain looks visually different from the source domain, OPA can achieve better transfer performance even with much fewer samples from the target domain, outperforming prior methods.

1. Introduction

Deep Reinforcement Learning has achieved impressive results in many domains, such as Atari (Mnih et al., 2013) and Mujoco (Lillicrap et al., 2015). However, traditional RL algorithms typically require many interactions with the environment (François-Lavet et al., 2018). Besides, the learned policy can easily be over-fitted to the source domain where it is trained and may collapse if faced with slight changes in the target domain (Cobbe et al., 2019; Peng et al., 2023). Therefore, it is essential to investigate how a policy can be transferred to a new environment.



(a) Case A: Source (b) Case A: Target

(c) Case B: Source (d) Case B: Target

Figure 1. The source and target domain for cases A (Xing et al., 2021) and B (considered in this work). Case B is more difficult than case A because we can not solely rely on visual clues to learn the mapping function between the source and target domain.

When trying to achieve such transfer, one of the most critical problems is dealing with the changes to the observation distribution, also known as domain adaptation in RL (Higgins et al., 2017; Li et al., 2021). Many previous works try to solve this problem by learning a mapping function between the target and source domain. For example, (Gamrian & Goldberg, 2018a; Tzeng et al., 2015; You et al., 2017) learn an image-to-image translation model that can map the observations from the target domain back into the source domain, and therefore the policy trained in the source domain is directly applicable when equipped with such translation. Some works (Xing et al., 2021; Higgins et al., 2017; Chen et al., 2021) also learn such a mapping indirectly, in which the observations from the source and target domain are

[1]University of Science and Technology of China [2]SKL of Processors, Institute of Computing Technology, CAS [3]Cambricon Technologies [4]University of Chinese Academy of Sciences, China [5]Institute of Software Chinese Academy of Sciences. Correspondence to: Yunji Chen <cyj@ict.ac.cn>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

mapped into aligned representations.

Although these works have achieved compelling performance in many tasks, they typically require access to abundant data from the target domain, which can be problematic when collecting these data is expensive. Besides, most works are applicable when the target domain *looks similar in appearances* to the source domain (e.g. case A in Figure 1). When faced with more challenging cases where the elements in the target domain have the same underlying functionalities but irrelevant appearances (e.g. case B in Figure 1), these methods are likely to fail. Therefore, how to quickly transfer between domains of irrelevant appearances still remains a problem.

On the other hand, it is possible for our human beings to achieve such a transfer. This is because we can utilize the *functional similarity* between elements to determine the mapping function between the source and target domain. For example, suppose we can get a score by eating an 'apple' and then learn to seek and eat the apples in a game. When faced with an unseen 'pear' in a new game, we find that we can also get a score by eating the 'pear', then we can quickly treat it as an 'apple' and also seek and eat the pears in the new game. In summary, the policy transfer from an 'apple' to a 'pear' is based on the fact that 'pear' and 'apple' have the same underlying functionality, i.e. both eating a 'pear' and an 'apple' can increase the score. However, learning the functional similarity of elements between source and target domains is difficult, because we have to interact *actively* with those unseen elements in the target domain. Thus, an efficient exploration mechanism is needed to discover the underlying functionalities.

Following the insight above, in this work, we propose a novel framework named **O**nline **P**rototype **A**lignment (OPA) to learn the mapping function based on the functional similarity of elements and achieve the few-shot policy transfer within only several episodes. To represent the underlying functionalities of elements, we assume the elements in the tasks can be divided into several kinds of prototypes such that elements of the same prototype share the same functionalities. To discover the prototypes of unseen elements quickly, OPA introduces an exploration policy. The exploration policy is trained by maximizing the mutual information between the trajectories it produces and the prototypes of unseen elements, therefore it can interact with these unseen elements in an efficient and purposeful manner to infer their prototypes. When deployed on the target domain, OPA first distinguishes unseen elements by novelty detection. Then the exploration policy interacts with these unseen elements so that OPA can infer their prototypes based on the produced trajectories. Finally, by building a mapping function based on the discovered prototypes, we can directly transfer the policy trained in the source domain to solve the task in the target domain. Compared with previous works, OPA introduces an exploration mechanism to learn the mapping function based on the functional similarity between elements in the source and target domain, and can efficiently achieve few-shot policy transfer even if there are no visual clues for transfer between the two domains.

The experiments are carried out on the task suite named Hunter (Yi et al., 2022). To reveal the strength of OPA, we use the original version of Hunter as the source domain and derive a new variant that looks significantly different from the original as the target domain. Compared with several baselines, OPA can achieve better transfer performance by only using a few data from the target domain, outperforming other baselines.

2. Related Work

Domain Adaptation in RL: The goal of domain adaptation is to address the domain shift between the source and target domain. Most domain adaptation approaches are designed to deal with the changes to the observation distribution. Current domain adaptation methods can be roughly divided into three categories: domain randomization (Tobin et al., 2017; Sadeghi & Levine, 2017; James et al., 2019), image-to-image translation (Gamrian & Goldberg, 2018a; Tzeng et al., 2015; You et al., 2017; Zhang et al., 2018), and adaptation via aligned representations (Xing et al., 2021; Higgins et al., 2017; Chen et al., 2021).

In domain randomization, a meta-simulator is required to generate many variants of the source domain. As a result, policies trained in these variants can learn to attend to the common features. However, these methods cannot work when the meta-simulator is not available, which is generally costly to attain in practice. In image-to-image translation approaches, a mapping function is learned to map the pixel observations from the target domain to the source domain. Such mapping is often learned via generative adversarial networks (GANs). In adaptation approaches via aligned representations, the source and target domain observations are mapped into a well-regularized latent space. Ideally, representations in this latent space can share consistent semantic meanings no matter which domain they come from. For example, (Xing et al., 2021) explicitly splits the latent representations into domain-specific and domain-general features and then builds policy on the domain-general features to ignore domain-specific variations.

Although these works have achieved compelling performance, they typically require access to abundant data from the target domain (or other domains that are different from the source domain). Besides, most of them rely on visual clues to learn the mapping function, which can be problematic when the elements in the target domain have irrelevant

appearances.

Object Oriented RL: The basic assumption of Object Oriented RL (OORL) is that the state space of MDPs can be represented in terms of objects, which is inspired by the fact that objects are the basic units of recognizing the world. In OORL, the agent's observations are a set of object representations, and the agent can solve the task by reasoning over these objects. By leveraging the invariance of objects' functionalities in different scenarios, policy trained in this way can often achieve better generalization ability (Yi et al., 2022; Zambaldi et al., 2019). Recent progress (Lin et al., 2020; Jiang et al., 2020) in Unsupervised Object Discovery also boosts the development of OORL. In our work, we follow the basic settings of OORL.

3. Preliminaries

3.1. Notation

We assume the underlying environment is a Markov decision process (MDP), described by the tuple $M = (S; A; P_T; R)$, where S is the state space, A the action space, $P_T : S_t \quad A_t \quad S_{t+1} \; ! \; [0; 1]$ the transition probability function which determines the distribution of next state given current state and action, and $R : S_t \quad A_t \quad S_{t+1} \; ! \; \mathbb{R}$ the reward function. Given the current state $s \; 2 \; S$, an agent chooses its action $a \; 2 \; A$ according to a policy function $a \quad (\,js)$. This action will update the system state to a new state s^0 according to the transition function P_T, and then a reward $r = R(s; a; s^0) \; 2 \; \mathbb{R}$ is given to the agent. The goal of the agent is to maximize the expected cumulative rewards by learning a policy :

$$J(\;) = \mathbb{E} \quad \sum_{t=0}^{\infty} R(s_t; a_t; s_{t+1}); \tag{1}$$

where := $(s_0; a_0; r_0; :::; s_T)$ is the trajectory generated by .

In this work, we also assume the state space S can be broken into a set of object representations: $S = \prod_{i=1}^{N} O$, where O is the space of object representations.

3.2. Problem Statement

We consider the domain adaptation problem in which a task policy $_{task}$ is first trained in the source domain $M_S = (S^{source}; A; P_T^{source}; R^{source})$ and then transferred to the target domain $M_T = (S^{target}; A; P_T^{target}; R^{target})$. We also assume that the M_S and M_T share the *same underlying dynamics and reward structures* such that there exists a mapping function $f : S^{target} \; ! \; S^{source}$ and $_{task}$ can achieve optimal transfer performance when equipped with f (i.e. $_{task} \quad f$).

Algorithm 1 The training procedure of OPA

Input: M_S
Output: $_{task}$, $_{exp}, q$, IsUnseen
/ Train $_{task}$ */*
Train $_{task}$ to solve M_S, and save the historic trajectories as D_{his}.
/ Train* IsUnseen **/*
Train $g_{enc}; g_{dec}$ on D_{his}, obtaining IsUnseen. (see Eq.(3))
*/*Pre-train q using D_{his}*/*
repeat
  Sample a batch of episodes $f \; _k g_k$ from D_{his}.
  Sample a subset of prototypes $I \quad P_{seen}$ and an injection $\quad : I \; ! \; P_{unseen}$.
  Update q using $f \; _k g_k; f_{I;}$ according to Eq.(6).
until convergence
/ Train $_{exp}$ using M_S and q */*
repeat
  Sample $I \quad P_{seen}$ and $\quad : I \; ! \; P_{unseen}$.
  Running the latest $_{exp}$ on M_S (with $f_{I;}$) to obtain trajectories $f \; _k g_k$
  Relabel the rewards of $f \; _k g_k$ using the intrinsic rewards generated by q . (see Eq.(7))
  Update $_{exp}$ with PPO using $f \; _k g_k$.
until certain steps

4. Method

As stated in Section 3 , we assume the observation space can be divided into the direct product of multiple object representation spaces: $S = \prod_{i=1}^{N} O$. We further assume that each object o has been assigned a category label o^c according to its appearance, which can be obtained by oracle or by unsupervised clustering on objects.

The goal of OPA is to learn a prototype mapping function $f_{proto} : O \; ! \; P_{seen} = f1; 2; :::; Cg$ that assigns a prototype o^p to each object o in M_S and M_T such that objects within the same prototype share the same functionalities. Intuitively, the prototype of an object can represent its functionality, therefore objects with the same prototype can be treated equally no matter which domain (M_S or M_T) they come from.

In M_S, we simply define f_{proto} as $f_{proto}|_S(o) = o^c$ (i.e. $o^p = o^c$) which means prototypes are exactly the category labels of objects. This is because objects with the same appearances share the same functionalities. However, it is not the case in M_T because we have to map objects into the prototype space *aligned* with M_S such that our task policy $_{task}$ is applicable. An object in M_T can be seen or unseen depending on whether it has shown in M_S. For the seen object, we can safely apply $f_{proto}|_S$ to obtain its prototype. For the unseen object, $f_{proto}|_S$ is not applicable,

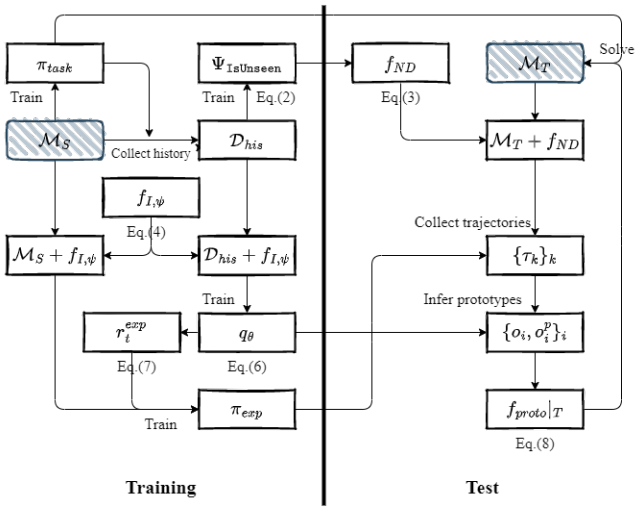


Figure 2. The training and test procedures of OPA.

therefore we have to explore its functionality to determine its prototype.

The overall procedures of OPA are presented in Algorithm 1 and Figure 2. In the training phase, we first train an indicator $\texttt{IsUnseen}$ to distinguish unseen objects (Section 4.1). Then, we train an exploration policy $_{exp}$ and an inference model q in M_S (Section 4.2), which aim to efficiently discover the prototypes of unseen objects. In the test phase, we obtain $f_{proto|T}$ for M_T by combining $\texttt{IsUnseen}$; $_{exp}$ and q together (Section 4.3), with which $_{task}$ can be transferred to M_T.

4.1. Novelty Detection

For an object in M_T, we want to classify whether it has shown in M_S. This problem is actually a task of novel detection, and many approaches in this field are able to solve it. For simplicity, in this work, we consider a native approach that relies on reconstruction loss.

We collect some object samples $O_S = \{o_j\}_{j=1}^{M}$ from M_S, and train an auto-encoder (consisting of g_{enc} and g_{dec}) that tries to map $o_j \in O_S$ into a latent space via g_{enc}, and then map the resulting latent back into o_j via g_{dec}. Since the g_{enc}, g_{dec} will be over-fitted to the O_S, it will present high reconstruction loss if faced with out-of-distribution samples, and therefore can be a hint for unseen objects:

$$\texttt{IsUnseen}(o) = \mathbb{I}\left[\|g_{dec}(g_{enc}(o)) - o\|_2^2 > \epsilon\right], \quad (2)$$

For a seen object in M_T, we can adopt $f_{proto|S}$ to obtain its prototype. For an unseen object, we want to remind the agent to explore its functionalities, therefore we also map it into a special prototype space P_{unseen} via an injection $(P_{seen} \cap P_{unseen} = \emptyset)$. Therefore, the overall mapping function of novelty detection is:

$$f_{ND}(o) = \begin{cases} f_{proto|S}(o), & \text{if } \texttt{IsUnseen}(o) = \texttt{False} \\ (o^c), & \text{if } \texttt{IsUnseen}(o) = \texttt{True} \end{cases}, \quad (3)$$

where o^c is the category label of o, and $$ is an injection that maps o^c to $P_{unseen} \subset \{C+1, \ldots, 2C\}$. Note that the exact value of (o^c) does not matter because the prototypes of objects in P_{unseen} are all unknown and require to be explored.

4.2. Online Prototype Alignment

In this section, we aim to train an exploration policy $_{exp}$ that can interact with unseen objects of M_T (i.e., $\{o : f_{ND}(o) \in P_{unseen}\}$) in a purposeful manner to discover their prototypes. However, we have no access to M_T in the training phase; Even though we do have it, we do not know the real prototypes of unseen objects which are needed for training $_{exp}$.

Fortunately, we can create some 'imaginary' environments from M_S to train $_{exp}$ in which we have access to ground-truth prototypes via $f_{proto|S}$. At the beginning of an episode, we randomly sample a subset of prototypes $I \subset P_{seen}$ and then map them into P_{unseen}:

$$f_{I,} (o) = \begin{cases} o^p, & \text{if } o^p \notin I \\ (o^p), & \text{if } o^p \in I \end{cases}; \quad (4)$$

where $o^p = f_{proto|S}(o)$ is the prototype of o and $: I \to P_{unseen}$ is a randomly sampled injection. Note that the randomness of $$ is essential, otherwise we can easily infer the prototypes by leveraging $$, which is actually a backdoor of non-sense. Both I and $$ keep fixed in the remaining part of the episode. Without loss of generality, we further assume the codomain of $$ is $P^I_{unseen} = \{C+1, C+2, \ldots, C+|I|\}$.

Compared Eq.(4) and Eq.(3), we can see that they induce the same prototype encodings (if we ignore the differences in P_{unseen}) when $I = \{o^c : \texttt{IsUnseen}(o) = \texttt{True}\}$, which means that we can learn $_{exp}$ in M_S with $f_{I,}$ and then apply it to M_T with f_{ND}.

The exploration policy $_{exp}$ is trained in M_S equipped with $f_{I,}$. The aim of $_{exp}$ is to interact with objects in the P^I_{unseen}, and $_{exp}$'s behaviour should be informative to infer the original prototypes. To this end, we propose to maximize the mutual information of the trajectory induced by $_{exp}$ (which is denoted as $_{exp}$) and the original prototypes of P^I_{unseen} (which are $I^0 = [\ ^{-1}(C+1), \ldots, \ ^{-1}(C+|I|)]$). Formally, $_{exp}$ is trained to maximize the following objec-

tive:

$$MI(\ _{exp}; I^0) = H(I^0) \quad H(I^0 j\ _{exp}) \\ H(I^0) + \mathsf{E}\ _{I;\ ;\ exp} \log\ q\ (I^0 j\ ) \\ = \mathsf{E}\ _{I;\ ;\ exp} \sum_{t=0} \log \frac{q\ (I^0 j\ _{:t+1})}{q\ (I^0 j\ _{:t})} + Const; \tag{5}$$

where q is an inference model that can predict I^0 given a trajectory, $_{:t} = [s_0; a_0; r_0; :::; s_t]$[1] is the sub-trajectory consisting of first t transitions in . The second line in Eq.(5) comes from the lower bound proposed in (Barber & Agakov, 2003), and the third line follows from the expansion along the time-step dimension and ignores the terms that are not related to $_{task}$. Note that $MI(\ ^{exp}; I^0)$ can be maximized by maximizing the lower bound in Eq.(5).

To predict I^0 as soon as possible in an episode, q is trained using all sub-trajectories $_{:t}$, and the loss function is given as:

$$L(\ ) = \mathsf{E}\ _{:t\ \ I';\ ;\ exp} \log q\ (I^0 j\ _{:t}): \tag{6}$$

To optimize $_{exp}$, we notice that the last line in Eq.(5) is quite similar to the objective of RL (see Eq.(1)). Therefore we can maximize Eq.(5) by giving $_{exp}$ an intrinsic reward as shown in Eq.(7) and training it using any RL algorithm such as PPO (Schulman et al., 2017):

$$r_t^{exp} = \log \frac{q\ (I^0 j\ _{:t+1})}{q\ (I^0 j\ _{:t})}: \tag{7}$$

Intuitively, Eq.(7) will assign a positive reward to $_{exp}$ if the environment transition at step t (i.e. $(s_t; a_t; r_t; s_{t+1})$) is useful to predict I^0, which will motivate $_{exp}$ to learn efficient exploration behaviours. These behaviours can reveal the underlying functionalities of unseen elements quickly, therefore are essential for few-shot transfer.

In practice, the modelling of q and $_{exp}$ is also important because a proper design can introduce useful inductive biases and facilitate the training of q and $_{exp}$. Please refer to Appendix for more details.

4.3. Policy Reuse

Our task policy $_{task}$ is built on the prototype space. Therefore, we wish to derive $f_{proto} j_T$ that can infer the prototypes in M_T such that our task policy is applicable when equipped with $f_{proto} j_T$ (i.e. $_{task}$ $f_{proto} j_T$).

In M_T, we first run $_{exp}$ (with f_{ND} to label unseen elements) for several episodes. For each episode, we utilize q to infer the probability distribution of prototypes. We average these distributions to combine them together and then obtain the final prototypes $f o_i^p g_i$ of objects $f o_i g_i$ based on the aggregate distribution. Given $f(o_i; o_i^p) g_i$, we train a classifier f_{cls} that can maps o_i to o_i^p. In practice, we use PCA and LinearSVC implemented in (Pedregosa et al., 2011) to realize this classifier because they are light-weighted and run fast. Together with the notations in Eq.(2), our $f_{proto} j_T$ can be formulated as:

$$f_{proto} j_T(o) = \begin{cases} f_{proto} j_S(o); & \text{if not } _{\texttt{IsUnseen}}(o) \\ f_{cls}(o); & \text{if } _{\texttt{IsUnseen}}(o) \end{cases}: \tag{8}$$

5. Experiment

5.1. Environment Setup

In this work, we mainly consider the task suite Hunter (Yi et al., 2022) (and also provide results on Crafter (Hafner, 2022) in the Appendix). Hunter is an environment that is designed to be object-centric, which is suitable for our method. It contains 5 kinds of objects in total: , , , and , as shown in Figure 1 (c). The goal is to train an agent that controls to interact with and . The same action may result in different rewards when interacting with different objects, e.g., the agent will get a positive reward (=1) if shoots at , but a negative reward (=-1) if at . Hunter also provides different variants (e.g., Hunter-Z1C1, Hunter-Z2C2,...), which differ in the number of objects.

To test the transfer ability of OPA, we derive a new environment from Hunter by changing the appearances of objects (, , , , ! , , , , [2]), as shown in Figure 1 (d). The original and the new environments serve as the source domain and target domain, respectively. To obtain object representations, we divide the 64 64 3 image (the observation space in Hunter) into 8 8 tiles, and each tile is of shape 8 8 3. By the design of Hunter, each tile contains exactly one object. Therefore, these 64 tiles can be used as object representations for OPA.

5.1.1. Baseline Settings

We compare OPA with other approaches designed for domain adaptation, including DARLA(Higgins et al., 2017), LUSR(Xing et al., 2021), UNIT4RL (Gamrian & Goldberg, 2018b) and LTMBR (Sun et al., 2022). DARLA relies on learning disentangled representations to achieve transfer. It utilizes a special -VAE in which the reconstruction loss is replaced with a perceptual similarity loss. LUSR explicitly splits the latent into domain-specific and domain-general features and only relies on domain-general features to build task policy. UNIT4RL utilizes an image-to-image translation approach named UNIT (Liu et al., 2017) that can translate images between domains with unpaired samples. When deployed in the target domain, UNIT4RL translates

[1] $\tau_{:0} := \emptyset$

[2] These textures come from https://nethackwiki.com/

Table 1. The mean and standard deviation of episode returns across 4 seeds, both in the source and target domain. The UNIT4RL(LTMBR)@nM (n=0,3,5) means the UNIT4RL(LTMBR) fine-tuned for n million environment steps in the target domain.

	Hunter-Z1C1		Hunter-Z2C2		Hunter-Z3C3		Hunter-Z4C4	
	Source	Target	Source	Target	Source	Target	Source	Target
PPO	1.73 0.02	-0.01 0.09	3.04 0.28	-0.03 0.13	4.15 0.62	-0.04 0.14	5.12 0.21	-0.03 0.15
DARLA	1.25 0.07	-0.01 0.14	1.76 0.06	-0.02 0.13	1.91 0.09	0.01 0.17	2.23 0.09	0.02 0.2
LUSR	1.14 0.02	-0.33 0.23	1.19 0.06	-0.03 0.15	0.89 0.23	-0.05 0.19	0.90 0.16	-0.03 0.20
UNIT4RL@0M	1.73 0.02	0.22 1.10	3.04 0.28	0.81 2.12	4.15 0.62	-0.87 0.40	5.12 0.21	1.34 3.31
LTMBR@0M	1.50 0.02	0.00 0.02	2.73 0.03	-0.01 0.04	3.89 0.08	-0.01 0.04	4.68 0.06	-0.03 0.04
OPA(ours)	1.65 0.05	**1.71 0.05**	3.22 0.07	**3.03 0.31**	4.40 0.11	**4.47 0.18**	5.61 0.06	**5.68 0.30**
UNIT4RL@3M	-	1.35 0.33	-	3.13 0.26	-	3.84 0.05	-	4.67 0.77
UNIT4RL@5M	-	1.68 0.05	-	3.24 0.14	-	4.35 0.03	-	5.40 0.4
LTMBR@3M	-	1.27 0.05	-	2.14 0.07	-	2.83 0.07	-	3.30 0.12
LTMBR@5M	-	1.38 0.04	-	2.51 0.10	-	3.64 0.11	-	4.63 0.16

Table 2. The performance ratio of the target and source domain (higher is better) averaged across all environments. Both UNIT4RL and LTMBR need more than 3M adaptation steps in the target domain to match up with OPA.

PPO	DARLA	LUSR	UNIT4RL@0M	LTMBR@0M	OPA(ours)	UNIT4RL@3M	LTMBR@3M
-0.01	-0.00	-0.1	0.11	0.00	**1.00**	0.91	0.84

the observations back into the source domain and further fine-tunes the task policy using the translated observations. LTMBR introduces an auxiliary task to help the learning of representations in the target domain, which also includes a fine-tuning stage.

All approaches are trained with PPO using the same hyper-parameters. For OPA, UNIT4RL and LTMBR, we train the task policy $_{task}$ for 25M steps in the source domain. OPA uses additional 10M steps in the source domain to train $_{exp}$, and four episodes in the target domain to infer prototypes. Since the source domain and target domain are totally different, we also set $I = P_{seen}$ to facilitate the training of $_{exp}$. For LUSR and DARLA, we find $_{task}$ improves much more slowly, therefore we train $_{task}$ for 100M steps. Since UNIT4RL needs observations from the target domain, we collect 0.5M steps in the target domain via a random policy. This dataset is also granted to LUSR [3]. For other details, please refer to our Appendix.

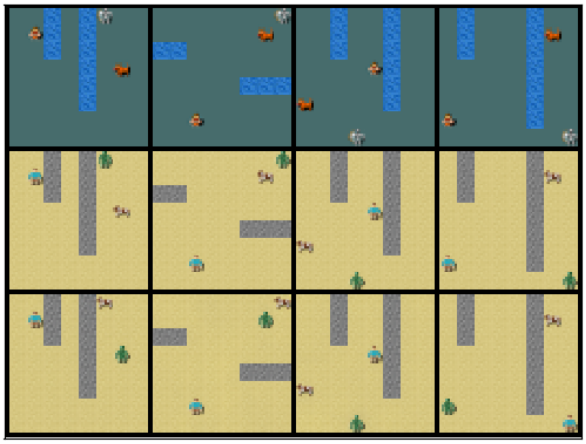


Figure 3. The observations (first row) in the target domain, (second row) generated from the first row using a ground truth mapping function, and (third row) generated using UNIT4RL trained with 4 different seeds.

5.1.2. Results

In Table 1, we present the performance results for all baselines. Because we are interested in the transfer performance, therefore we also calculate the ratio of performance between the target and source domain ($\texttt{ratio} = \frac{\texttt{performance}(M_T)}{\texttt{performance}(M_S)}$) in Table 2. From the results reported in Table 1 and 2, we can conclude that OPA achieves best performance in all tasks.

In DARLA and LUSR, we find that the $_{task}$ improves much more slowly than other baselines, therefore we train $_{task}$ for 100M steps in the source domain, as we described

[3] According to the original paper of LUSR, the data from the target domain is not essential in LUSR if we have access to other variants of environments that are different from the source domain.

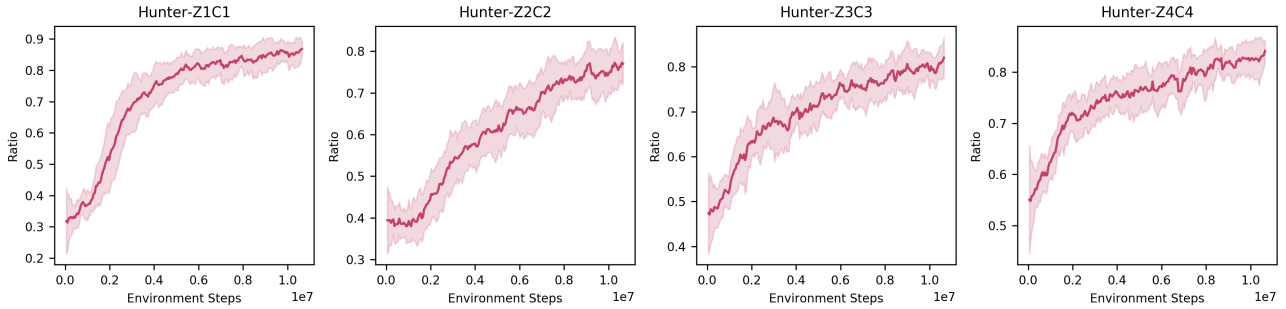


Figure 4. The ratio of episodes that OPA can successfully find the ground truth prototype alignment along the training procedure of π_{exp}. After training, OPA can find the ground truth prototypes in a single episode with a probability of more than 0.8.

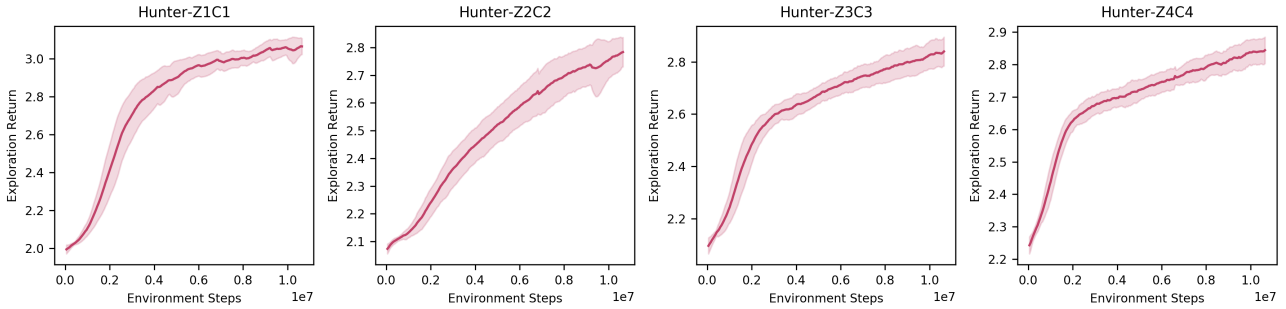


Figure 5. The exploration return produced by the inference model along the training procedure of π_{exp}. There is an obvious positive correlation between this return and the ratio reported in Figure 4.

in the baseline settings. However, even with 4x steps, we can still find $_{task}$ can not match up with others. We argue that this is because both DARLA and LUSR pre-train an encoder to extract a vectorized latent from observations (and keep frozen in the training of $_{task}$), which ignores the fact that the environments are object-oriented and therefore results in poor performance.

Despite the inferior task performance of DARLA and LUSR in the source domain, they also totally fail to transfer $_{task}$ to the target domain. For DARLA, this is not surprising because it does not use any additional data from the target domain and is solely trained in the source domain. For LUSR, we find that the domain-general and domain-specific features are not well-regularized (see Appendix), in that the domain-specific features can also contain important features such as the position of objects. Therefore, the domain-general features may lose important information, which can also explain its inferior task performance compared with DARLA in the source domain.

For UNIT4RL and LTMBR, we further fine-tune $_{task}$ for 3M and 5M steps in the target domain. As shown in Table 1, both UNIT4RL and LTMBR accelerate the fine-tuning process and only spend less than 5M steps to match up the PPO policy trained for 25M steps. Compared with these approaches, OPA only needs about 100 steps in the target domain to achieve almost optimal transfer performance, which is significantly less than the need of UNIT4RL and LTMBR (3M 5M).

To further expose the failure mode of UNIT4RL@0M and other image-to-images approaches for domain adaptation, we present the translation results of UNIT4RL in Figure 3. We can see that UNIT4RL can discover the mapping between ! and ! because these objects have unique existence distribution compared with others. However, UNIT4RL fails to reliably learn the mappings between , , and , , in that each different trial can result in a different mapping. A similar phenomenon should also appear in LUSR, although not explicitly. As we have argued before, this is because these objects can not be distinguished solely via visual clues, and therefore we have to rely on their functionalities to learn the mapping, which is one of the main motivations of our work.

Table 3. The adaptation performance of OPA with different number of exploration episodes. OPA can achieve high performance (0.8) even with only 1 episode.

	Hunter-Z1C1	Hunter-Z2C2	Hunter-Z3C3	Hunter-Z4C4	Aggregate Performance Ratio
OPA@1episode	1.41	2.21	3.48	4.64	0.80
OPA@2episodes	1.67	2.92	4.23	5.44	0.96
OPA@4episodes	1.71	3.05	4.47	5.68	1.00
OPA@16episodes	1.68	3.12	4.45	5.63	1.00

Table 4. The necessity of π_{exp}. We equip OPA with different exploration policies (i.e. $\pi_{exp}, \pi_{random}, \pi_{task}$), and run OPA for a single episode in the target domain of Hunter-Z1C1. π_{exp} is much more efficient for exploration than π_{random} and π_{task}.

	exp	*random*	*task*
Ratio of Correct Mapping	**0.86**	0.32	0.28
Ratio of Adaptation Performance	**0.85**	0.21	0.23
Average Number of Informative Interactions	**1.62**	0.12	0.08

5.2. Ablation Study

5.2.1. The quality of prototype alignment

To better evaluate the quality of prototypes discovered by OPA, in Figure 4 we plot the ratio of episodes that OPA can successfully match with the ground truth prototypes of unseen objects in the target domain. We can see that this ratio continues to increase during the training process of *exp* and eventually reaches 0.8+ for all environments. This means OPA can find the ground truth prototypes in a single episode with a probability of more than 0.8, which can be further improved by multi-episode exploration.

In Figure 5, we plot the exploration return (produced by q) of *exp*. We can notice that there is an obvious positive correlation between this return and the ratio plotted in Figure 4. This means that the intrinsic reward generated by q is informative and instructive because when following this reward *exp* can improve its ability to find the ground truth prototypes.

In our experiment setting, OPA takes four episodes in the target domain for exploration. In Table 3, we report the performance of OPA with other numbers of episodes. We can see that OPA achieves a performance ratio of 0.8 even only has access to a single episode in the target domain, and two episodes can quickly improve this ratio to 0.96. This means OPA can still obtain prototype assignments of relatively high quality in the absence of enough exploration chances.

5.2.2. The necessity of *exp*

In OPA, we put effort into training *exp*, and one may ask whether *exp* can pay back. To answer this question, we compare *exp* with other easy-to-obtain exploration policies in Hunter-Z1C1, which includes a random policy *random* and the task policy *task*.

The results are shown in Table 4. We can see that *exp* is much more efficient for exploration than *random* and *task*. Note that the performance of *task* is almost the same with *random*, which means that *task* can not present meaningful behaviours in the target domain to facilitate the inference of q .

To further investigate the difference between *exp* and *random* & *task*, in Table 4 we also report the average number of informative interactions in an episode, which includes meaningful interactions between objects that are useful for distinguishing the prototypes and therefore informative for the functionalities of objects. As shown in Table 4, we can see that *exp* will manage to find these informative interactions, whereas *random* and *task* do not present such a purposeful behaviour.

6. Conclusion

In this paper, we propose a novel framework named OPA that aims to transfer a policy to an unfamiliar environment in a few-shot manner. The key of OPA is to introduce an exploration mechanism that can purposefully interact with the unseen elements in the target domain. By doing so, we can build a mapping function between these unseen elements to seen elements according to their functionalities, and then transfer the policy trained in the source domain to the target domain. Our experiments show that OPA can not only achieve better transfer performance on tasks in which other baselines fail but also consume much fewer samples from the target domain.

Acknowledgements

This work is partially supported by the NSF of China(under Grants 62102399, 61925208, 62002338, 62222214, U22A2028, U19B2019), Beijing Academy of Artificial Intelligence (BAAI), CAS Project for Young Scientists in Basic Research(YSBR-029), Youth Innovation Promotion Association CAS and Xplore Prize.

References

Barber, D. and Agakov, F. V. The im algorithm: a variational approach to information maximization. In *NeurIPS*, 2003.

Chen, X.-H., Jiang, S., Xu, F., Zhang, Z., and Yu, Y. Cross-modal domain adaptation for cost-efficient visual reinforcement learning. In *NeurIPS*, 2021.

Cho, K., van Merrienboer, B., Bahdanau, D., and Bengio, Y. On the properties of neural machine translation: Encoder–decoder approaches. In *SSST@EMNLP*, 2014.

Cobbe, K., Klimov, O., Hesse, C., Kim, T., and Schulman, J. Quantifying generalization in reinforcement learning. In *ICML*, 2019.

François-Lavet, V., Henderson, P., Islam, R., Bellemare, M. G., and Pineau, J. An introduction to deep reinforcement learning. *Found. Trends Mach. Learn.*, 2018.

Gamrian, S. and Goldberg, Y. Transfer learning for related reinforcement learning tasks via image-to-image translation. In *ICML*, 2018a.

Gamrian, S. and Goldberg, Y. Transfer learning for related reinforcement learning tasks via image-to-image translation. In *ICML*, 2018b.

Hafner, D. Benchmarking the spectrum of agent capabilities. In *ICLR*, 2022.

Higgins, I., Pal, A., Rusu, A. A., Matthey, L., Burgess, C. P., Pritzel, A., Botvinick, M. M., Blundell, C., and Lerchner, A. Darla: Improving zero-shot transfer in reinforcement learning. In *ICML*, 2017.

James, S., Wohlhart, P., Kalakrishnan, M., Kalashnikov, D., Irpan, A., Ibarz, J., Levine, S., Hadsell, R., and Bousmalis, K. Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. *CVPR*, 2019.

Jiang, J., Janghorbani, S., de Melo, G., and Ahn, S. SCALOR: generative world models with scalable object representations. In *ICLR*, 2020.

Li, B., Franccois-Lavet, V., Doan, T. V., and Pineau, J. Domain adversarial reinforcement learning. *ArXiv*, 2021.

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

Lin, Z., Wu, Y., Peri, S. V., Sun, W., Singh, G., Deng, F., Jiang, J., and Ahn, S. SPACE: unsupervised object-oriented scene representation via spatial attention and decomposition. In *ICLR*, 2020.

Liu, M.-Y., Breuel, T. M., and Kautz, J. Unsupervised image-to-image translation networks. *ArXiv*, 2017.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. A. Playing atari with deep reinforcement learning. *ArXiv*, 2013.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Louppe, G., Prettenhofer, P., Weiss, R., Weiss, R. J., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.*, 2011.

Peng, S., Hu, X., Zhang, R., Guo, J., Yi, Q., Chen, R., Du, Z., Li, L., Guo, Q., and Chen, Y. Conceptual reinforcement learning for language-conditioned tasks. *ArXiv*, 2023.

Sadeghi, F. and Levine, S. Cad2rl: Real single-image flight without a single real image. *RSS*, 2017.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *ArXiv*, 2017.

Sun, Y., Zheng, R., Wang, X., Cohen, A. E., and Huang, F. Transfer RL across observation feature spaces via model-based regularization. In *ICLR*, 2022.

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. Domain randomization for transferring deep neural networks from simulation to the real world. *IROS*, 2017.

Tzeng, E., Devin, C., Hoffman, J., Finn, C., Abbeel, P., Levine, S., Saenko, K., and Darrell, T. Adapting deep visuomotor representations with weak pairwise constraints. In *Workshop on the Algorithmic Foundations of Robotics*, 2015.

Weng, J., Chen, H., Yan, D., You, K., Duburcq, A., Zhang, M., Su, H., and Zhu, J. Tianshou: A highly modularized deep reinforcement learning library. *ArXiv*, 2021.

Xing, J., Nagata, T., Chen, K., Zou, X., Neftci, E. O., and Krichmar, J. L. Domain adaptation in reinforcement learning via latent unified state representation. *ArXiv*, 2021.

Yi, Q., Zhang, R., Peng, S., Guo, J., Hu, X., Du, Z., Zhang, X., Guo, Q., and Chen, Y. Object-category aware reinforcement learning. *CoRR*, 2022.

You, Y., Pan, X., Wang, Z., and Lu, C. Virtual to real reinforcement learning for autonomous driving. *ArXiv*, 2017.

Zambaldi, V. F., Raposo, D., Santoro, A., Bapst, V., Li, Y., Babuschkin, I., Tuyls, K., Reichert, D. P., Lillicrap, T. P., Lockhart, E., Shanahan, M., Langston, V., Pascanu, R., Botvinick, M. M., Vinyals, O., and Battaglia, P. W. Deep reinforcement learning with relational inductive biases. In *ICLR*, 2019.

Zhang, J., Tai, L., Yun, P., Xiong, Y., Liu, M., Boedecker, J., and Burgard, W. Vr-goggles for robots: Real-to-sim domain adaptation for visual control. *IEEE Robotics and Automation Letters*, 2018.

Table 5. The data consumption for OPA and other baselines, both in the source and target domain. 'Enc.' and 'Expl.' are corresponding to 'Encoder' and 'Exploration' respectively.

	Source Domain			Target Domain		
	$task$	exp	Enc.	Fine-tuning	Enc.	Expl.
OPA(ours)	25M	10M	-	-	-	100
DARLA	100M	-	0.5M	-	-	-
LUSR	100M	-	0.5M	-	0.5M	-
UNIT4RL	25M	-	-	0-5M	0.5M	-
LTMBR	25M	-	-	0-5M	-	-

A. Implementation for Baselines

Implementation for PPO The task policies for all approaches included in this work are trained via PPO. Our PPO implementation is based on Tianshou (Weng et al., 2021) which is purely based on PyTorch. We adopt the hyper-parameters which are shown in Table 6.

Implementation for DARLA For DARLA, we first collect 0.5M samples in the source domain via a random policy. Using these samples, we train a -VAE with a grid search over $= 0.1; 0.5; 1; 2; 5; 10$. We set $= 2$ because it achieves the best results In the original paper of DARLA, the reconstruction loss of -VAE is replaced by a perceptual similarity loss produced by a denoising autoencoder (DAE). However, we find the reconstruction loss works better in our case, therefore is used in practice.

After pre-training the encoder, we then train a task policy $_{task}$ for 100M steps in the source domain based on this encoder. The encoder is frozen during the training of $_{task}$, and will encode the pixel observation into a latent of size 128. The task policy is a 3-layer MLP with hidden sizes 64, and outputs the action probability and value function.

Implementation for LUSR LUSR needs a set of different domains to train the encoder. In our case, we simply collect 0.5M samples from the source domain and 0.5M from the target domain to train LUSR. The coefficient of the reverse loss in LUSR is grid-searched for 0.1, 0.5, 1, 2, 5, and we find 0.5 works best in our case. LUSR splits the latent representations into domain-specific features z_s and domain-general z_g features. We also search for the dimensions of both features (including $(|z_s|; |z_g|) = (8; 32); (8; 64); (16; 64); (16; 128); (32; 128)$), and choose $(16; 128)$ in practice.

After pre-training the encoder, we then train a task policy $_{task}$ for 100M steps in the source domain based on the domain-general features provided by LUSR. The encoder is frozen during the training of $_{task}$, and will encode the pixel observation into a latent of size 128. The task policy is a 3-layer MLP with a hidden size of 64 and outputs the action probability and value function.

Implementation for UNIT4RL First, we collect 0.5M samples from the source domain and 0.5M from the target. This data is used to train an image-to-image translation model T. All hyper-parameters of UNIT4RL are the same as in the original paper.

We train a task policy $_{task}$ for 25M steps in the source domain. When deploying $_{task}$ in the target domain, we first translate the observations into the source domain via the translation model T, then calculate the action probability and value function using $_{task}$. $_{task}$ is further fine-tuned for 0-5M steps in the target domain via PPO, with T kept fixed.

Implementation for LTMBR LTMBR (Sun et al., 2022) introduces an auxiliary task to help the learning of representations in the target domain. We conduct a grid search over the coefficients (1,2,4,8,16) of the auxiliary loss in the Hunter-Z2C2 and then apply the optimal coefficient (=4) to other tasks. We train the task policy $_{task}$ for 25M steps with this auxiliary loss in the source domain.

B. Implementation for OPA

The implementation is available at https://github.com/albertcity/OPA.

B.1. The modelling of $_{exp}$ and q

In practice, the modelling of q and $_{exp}$ is also important because a proper design can introduce useful inductive biases and facilitate the training of q and $_{exp}$. For simplicity, in the following we assume the prototype o^p of an object o has already been mapped into $P \ [\ P^I_{unseen}$ via Eq.(4).

For $_{exp}$, we use the predicted prototypes of q as encodings of objects if $o^p \ 2 \ P_{unseen}$ and o^p if $o^p \ \not\in \ P_{unseen}$. The predicted results of q can directly inform $_{task}$ which objects are still unfamiliar to q , and therefore $_{task}$ can learn to interact with them.

For q , we maintain a hidden state $h_p \ 2 \ R^F$ for each prototype p in P^I_{unseen} ($p = 1; :::; jIj$), which summarizes the history of interactions related to p. h_p is also tasked to predict the ground truth prototype (i.e. $^1(p)$) via a learnable classifier. All h_ps are initialized to the *same* hidden states at the beginning of an episode. At each transition $(s_t; a_t; r_t; s_{t+1})$, h_p is updated by the following steps:

Broadcasting h_p. For each objects o in $s_t; s_{t+1}$, we embed o^p into R^F if $o^p \not\in P^I_{unseen}$, which serves as o's encoding . For $o^p \ 2 \ P^I_{unseen}$, we use h_p instead, because it summarizes the history of interactions related to p. This will give us new representations of s_t and s_{t+1}, which are denoted as $[o^1_t; :::; o^N_t] \ 2 \ R^{N \ \ F}$ and $[o^1_{t+1}; :::; o^N_{t+1}] \ 2 \ R^{N \ \ F}$

Processing the transition information. In our settings, each object o actually represents a tile in the original observation, therefore we can re-arrange the $[o^1_t; :::; o^N_t] \ 2 \ R^{N \ \ F}$ into $[o^{i;j}_t]^{H \ \ W}_{i=1 j=1} \ 2 \ R^{H \ \ W \ \ F}$ ($N = H \ \ W$). $o^{i;j}_t$ is corresponding to the $(i; j)$'th tile of location $((i \ \ 1) \ \ size_h; (j \ \ 1) \ \ size_w)$ ($size_h; size_w$ is the size of each tile).

In order to process the transition information, we first concatenate $o^{i;j}_t; o^{i;j}_{t+1}; a_t; r_t$ together. This will give us $[o^{i;j}_t]_{i;j} \ 2 \ R^{H \ \ W \ \ (2F+A+R)}$, where A is corresponding to the one-hot embedding of a_t and R corresponding to the embedding of r_t. Then we process the resulting features using several convolution layers of (kernel size=3, stride=1, padding=1), which will give us $O \ 2 \ R^{H \ \ W \ \ F}$. O can be seen as a latent that summarizes the information of transition $(s_t; a_t; r_t; s_{t+1})$.

Updating h_p. In order to extract relative information related to h_p from O, we adopt an attention mechanism to get a latent z_p from O. The query vector of this attention is h_p, and both the key and value vectors are $fO_{i;j;:} : o^{i;j}_t \ = p \ or \ o^{i;j}_{t+1} = pg$. In other words, z_p only extracts information from the objects related to p. After obtaining z_p, h_p is then updated via GRU (Cho et al., 2014) by taking z_p as the current input.

By the design of $_{exp}$ and q , we can see that the choice of does not influence the inference results of q and $_{exp}$ (i.e.

Table 6. PPO hyper-parameters.

Hyper-parameter	Value
Discount factor	0.9
Lambda for GAE	0.95
Epsilon clip (clip range)	0.2
Coefficient for value function loss	0.5
Normalize Advantage	True
Learning rate	5e-4
Optimizer	Adam
Max gradient norm	0.5
Steps per collect	4096
Repeat per collect	3
Batch size	256

any will give the same results), which means we can choose a fixed to simplify the training process.

B.2. Other implementation details

In OPA, we first train $_{task}$ for 25M steps in the source domain. The $_{task}$ uses the $f_{proto}j_S$ as the encoder of objects, and also adopts a self-attention mechanism to model the relations between objects, which is a common practice in OORL (Yi et al., 2022; Zambaldi et al., 2019).

During the training of $_{task}$, we save its trajectories as D_{his}. D_{his} is then used to train the indicator $_{\texttt{IsUnseen}}$ and the inference model q . Thanks to the special design of $_{exp}; q$, can be fixed for simplicity as explained in the Appendix B.1.

After pre-training q , we train an exploration policy $_{exp}$ for 10M steps in the source domain using the intrinsic rewards generated by q . The network architecture in $_{exp}$ is the same as $_{task}$.

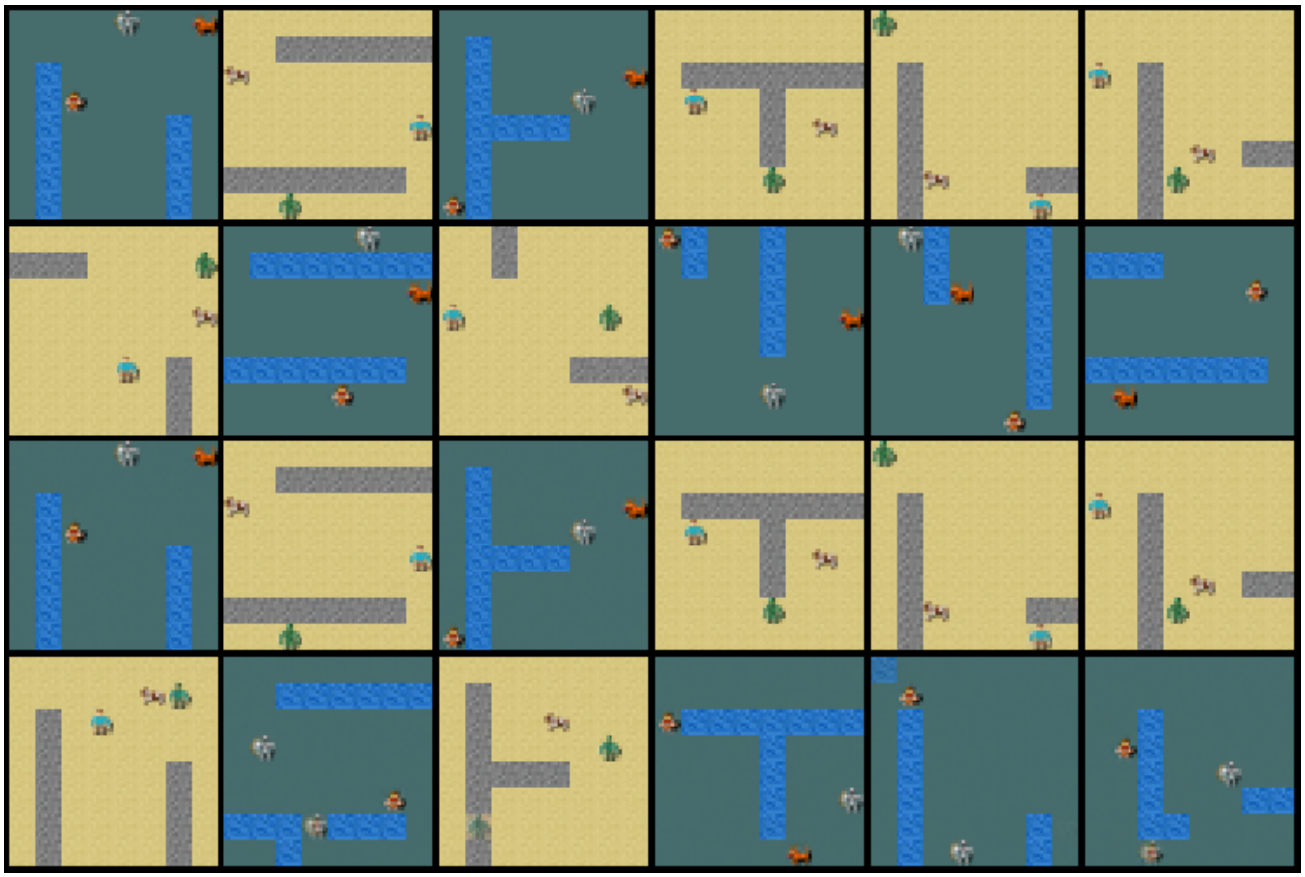


Figure 6. The reconstruction results of LUSR. First and second row: two sampled observations a, b; Third row: the reconstruction results of LUSR using $z_g(a)$ and $z_s(a)$; Fourth row: reconstruction using $z_g(a)$ and $z_s(b)$. Although the $z_s(a)$ and $z_s(b)$ are able to distinguish the source and target domain, it also contains important information such as the positions of objects.

C. Analysis of LUSR

LUSR splits the latent embedding of an observation o into domain-general and domain-specific features, which are denoted as $z_g(o)$ and $z_s(o)$ respectively. Intuitively, $z_g(o)$ should contain crucial information such as the position of each object, and $z_s(o)$ should contain non-important information such as the image style of the observation that is different in different domains.

In Figure 6, we show the reconstruction results of LUSR using different combinations of z_g and z_s. We can see that the positions of objects not only depend on z_g but also on z_s. Although the z_s can be used to distinguish the source and target domain, it also contains important information such as the positions of objects. This is problematic because z_g may lose important information.

Table 7. Results on Crafter.

Algorithm	Source Domain	Target Domain	Ratio
PPO	11.62 0.37	3.00 0.57	0.26
DARLA	7.50 0.29	4.30 0.37	0.57
LUSR	7.97 0.23	2.15 0.27	0.27
UNIT4RL@0M	11.62 0.37	3.50 0.23	0.30
OPA(ours)	11.57 0.52	10.69 0.41	1.01
UNIT4RL@5M	-	7.88 0.45	0.68
UNIT4RL@20M	-	9.17 0.21	0.79

D. Results on Crafter

In this section, we provide the transfer results on the Crafter (Hafner, 2022), which is a complicated 2-D Minecraft-like environment. We use the original version of Crafter as the source domain. The target domain is a modified version in which we select several objects (i.e. 'stone', 'tree', 'coal', 'cow', 'zombie', 'skeleton') and replace their textures using icons from the Nethack (https://nethackwiki.com/).

The $_{task}$ are trained for 20M for all algorithms in M_S. For LUSR and UNIT4RL, we take 0.5M from M_T to train the encoder. For OPA, we take 4 episodes to run $_{exp}$ in M_T to transfer $_{task}$. When training $_{exp}$, we set I to the chosen objects that are different in M_S and M_T, which can accelerate the training process of $_{exp}$. The observation of Crafter can be rendered into an image of size 72 72 3, which consists of two parts: (part A) the 7 9 region around the agent (of size 56 72 3 in the image), and (part B) the status of the agent and items in its backpack (of size 16 72 3). When building the encoder for $_{task}; _{exp}; q$, we separate parts A and B, and use the numerical closed-form of part B (instead of pixels).

The overall results are as shown in Table 7. According to this table, OPA still achieves the best transfer performance in the Crafter environment. Interestingly, we find that UNIT4RL@20M can not recover the performance in the source domain even after training for 20M in the target domain, which means that the learned mapping function has lost some important information.

Table 8. The reconstruction errors in the source and target domain.

	obj_1	obj_2	obj_3	obj_4
Source Domain (Seen objects)	0.0112	0.0142	0.0179	0.0410
Target Domain (Unseen objects)	4.920	8.034	9.998	14.327

E. Other Discussions

OPA introduce several stages which may bring cumulative errors. In this section, we analyse this problem.

Formally, there are four parts in OPA that may bring errors:

(1) $_{unseen} : O \; ! \; 0;1$ in Eq.(2),

(2) $q \; : \; ! \; I^0$ in Eq.(6),

(3) $_{exp} : S \; ! \; A$,

(4) $f_{cls} : O \; ! \; P_{seen}$ in Eq.(8).

However, the approximation errors brought by (1) and (4) should be very small because the domain of both (1) and (4) is the space of objects O, which is simple and small in many cases. In our environment, O is actually a set of size 4. Even in the Crafter (a complicated 2D Mincraft-like environment), it is just 19. Therefore, the error brought by (1) and (4) can be ignored in many cases. For example, $_{unseen}$ can correctly identify all unseen objects in our cases as we will show later. The training of (3) relies on (2), therefore the quality of (2) does affect the training of (3). However, this is unavoidable, because (2) and (3) are designed to work together.

The accuracy of $_{unseen}$. The binary classifier $_{unseen}$ is built upon $g_{dec} \; g_{enc}$(i.e. $_{unseen} = jjg_{dec} \; g_{dec}(o) \quad ojj$). Therefore, we can use the reconstruction error $jjg_{dec} \; g_{enc}(o) \quad ojj$ to measure the accuracy of $_{unseen}$. We expect a small error for objects in the source domain and a large error for objects in the target domain. We list the reconstruction error for all objects in Table 8. As shown in Table 8, the reconstruction error is significantly different in the source and target domain, which means that $_{unseen}$ can correctly identify the unseen objects.