Object-Category Aware Reinforcement Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

Object-oriented reinforcement learning (OORL) is a promising way to improve the sample efficiency and generalization ability over standard RL. Recent works that try to solve OORL tasks without additional feature engineering mainly focus on learning the object representations and then solving tasks via reasoning based on these object representations. However, none of these works tries to explicitly model the inherent similarity between different object instances of the same category. Objects of the same category should share similar functionalities; therefore, the category is the most critical property of an object. Following this insight, we propose a novel framework named Object-Category Aware Reinforcement Learning (OCARL), which utilizes the category information of objects to facilitate both perception and reasoning. OCARL consists of three parts: (1) Category-Aware Unsupervised Object Discovery (UOD), which discovers the objects as well as their corresponding categories; (2) Object-Category Aware Perception, which encodes the category information and is also robust to the incompleteness of (1) at the same time; (3) Object-Centric Modular Reasoning, which adopts multiple independent and object-category-specific networks when reasoning based on objects. Our experiments show that OCARL can improve both the sample efficiency and generalization in the OORL domain.

1 Introduction

Reinforcement Learning (RL) has achieved impressive progress in recent years, such as results in Atari [26] and Go [30] in which RL agents even perform better than human beings. Despite its successes, RL is also known to be of low sample efficiency [11] and poor generalization ability [4]. Object-oriented RL (OORL) [13; 9; 25; 18; 41] is a promising way to deal with these limitations. Inspired by studies in cognitive science [27] that objects are the basic units of recognizing the world, OORL focuses on learning the invariance of objects' functionalities in different scenarios to achieve better generalization ability.

In OORL, the agent's observation is a set of object representations, and the task can be solved via reasoning based on these objects. Most previous works [13; 18; 9] in OORL employ hand-crafted object features given in advance, which require human expertise and result in low generality, limiting the applications of these methods. Recent works [41; 40; 39] try to broaden the scope of OORL's applications by avoiding using additional feature engineering. Their works can be divided into either top-down approaches [41] which purely rely on reward signals to ground representations to objects, or bottom-up approaches [40; 39] which utilize unsupervised object discovery (UOD) technologies to provide structural object representations.

However, none of these works [41; 40; 39] tries to *explicitly* model the inherent similarity between different object instances of the same *category* when reasoning based on objects. In most cases, objects of the same category should share the same functionalities, such as obeying similar behaviour patterns or having the same effects when interacting with other objects. Therefore, an intelligent

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

agent should recognize an object's category to be aware of its functionalities. Since objects from different categories differ in their functionalities, the objects' functionalities can also be separately modelled (according to their categories), leading to modularity which is beneficial for generalization [6; 8; 38] in general.

Following the above insight, we propose a framework named **O**bject-**C**ategory **A**ware **R**einforcement **L**earning (OCARL), which utilizes the category information of objects to facilitate both perception and reasoning. OCARL consists of three parts: (1) category-aware UOD, which can automatically discover the objects *as well as their corresponding categories* via UOD and unsupervised clustering. (2) **O**bject-**C**ategory **A**ware **P**erception (OCAP), a perception module that takes the object category information obtained in (1) as an *additional* supervision signal. OCAP can encode the category information into representations and is also robust to the incompleteness of UOD at the same time. (3) **O**bject-**C**entric **M**odular **R**easoning (OCMR), a reasoning module that adopts multiple independent and object-category-specific networks, each of which processes the object features of the same corresponding category. Such a modular mechanism can enhance further the generalization ability of OCARL. Our experiments show that OCARL can improve the sample efficiency on several tasks and also enable the agent to generalize to unseen combinations of objects where other baselines fail.

2 Related Works

Unsupervised Object Discovery Unsupervised object discovery (UOD) methods try to automatically discover objects without additional supervision. Roughly speaking, there are two main categories in this area: spatial attention models and spatial mixture models. Spatial attention models such as SCALOR [17] and TBA [16] explicitly factorize the scene into a set of object properties such as position, scale and presence. Then these properties are utilized by a spatial transformer network to select small patches in the original image, which constitute a set of object proposals. These methods can deal with a flexible number of objects and thus are prominent in the UOD tasks. Spacial mixture models such as Slot Attention [23], IODINE [12] and MONet [3] decompose scenes via clustering pixels that belong to the same object (often in an iterative manner). However, they often assume a fixed maximum number of objects and thus cannot deal with a large number of objects.

SPACE [21], a UOD method used in this paper, combines both the spatial attention model and spatial mixture model. The spatial attention model extracts objects from the scene, whereas the spacial mixture model is responsible for decomposing the remaining background. Such a combination enables SPACE to distinguish salient objects from relative complex backgrounds.

Object-Oriented Reinforcement Learning Object-oriented reinforcement learning (OORL) has been widely studied in RL community. There are many different structural assumptions on the underlying MDP in OORL, such as relational MDP [13], propositional object-oriented MDPs [9], and deictic object-oriented MDP [25]. Although these assumptions differ in detail, they all share a common spirit that the state-space of MDPs can be represented in terms of objects. Following these assumptions, [5] proposes object-focused Q-learning to improve sample efficiency. Schema Networks [18] learn an object-based casual model for regression planning. [33] improve the generalization ability to novel objects by leveraging relation types between objects that are given in advance. Although these works have been demonstrated to be useful in their corresponding domain, they require explicit encodings of object representations, which limits their applications.

There are other lines of work in OORL such as COBRA [35], OODP [42], and OP3 [32] that try to solve OORL tasks in an end-to-end fashion. Most of these works fall into a model-based paradigm in which an object-centric model is trained and then utilized to plan. Although these methods are effective in their corresponding domains, in this paper, we restrict ourselves to the *model-free* setting, which can generally achieve better asymptotic performance than model-based methods [34; 14].

The most related works should be RRL [41] and SMORL [40]. RRL proposes an attention-based neural network to introduce a relational inductive bias into RL agents. However, since RRL relies purely on this bias, it may fail to capture the objects from the observations. On the other hand, SMORL utilizes an advanced UOD method to extract object representations from raw images, and then these representations are directly used as the whole observations. However, SMORL adopts a relatively simple reasoning module, which limits its applications in multi-object scenarios.

3 Method

OCARL consists of three parts: (1) category-aware unsupervised object discovery (category-aware UOD) module, (2) object-category aware perception (OCAP), and (3) object-centric modular reasoning (OCMR) module. The overall architecture is shown in Figure 1. We will introduce (1)(2)(3) in Section 3.1, Section 3.2, and Section 3.3 respectively.

3.1 Category-Aware Unsupervised Object Discovery

In this section, we explain how to automatically discover the objects as well as their corresponding categories. To put it simply, the objects are discovered by SPACE (a UOD method), and their categories are then predicted via unsupervised clustering.

Unsupervised Object Discovery In this work, SPACE [21] is utilized to discover objects from raw images. In SPACE, an image $\mathbf{x}$ is decomposed into two latent representations: foreground $\mathbf{z}^{fg}$ and background $\mathbf{z}^{bg}$. For simplicity, we only introduce the inference model of the foreground latent $\mathbf{z}^{fg}$, which is a set of object representations. We encourage the readers to refer to [21] for more details. To obtain $\mathbf{z}^{fg}$, $\mathbf{x}$ is treated as if it were divided into $H \times W$ cells and each cell is tasked with modelling at most one (nearby) object. Therefore, $\mathbf{z}^{fg}$ consists of a set of $H \times W$ object representations (i.e. $\mathbf{z}^{fg} = \{\mathbf{z}_{ij}^{fg}\}_{i=1}^{H}{}_{j=1}^{W}$), each of which is a 3-tuple[1] $\mathbf{z}_{ij}^{fg} = (z_{ij}^{pres}, \mathbf{z}_{ij}^{where}, \mathbf{z}_{ij}^{what})$. $z_{ij}^{pres} \in \{0, 1\}$ is a 1-D variable that indicates the presence of any object in cell (i, j), $\mathbf{z}_{ij}^{where}$ encodes $\mathbf{z}_{ij}^{fg}$'s size and location, and $\mathbf{z}_{ij}^{what}$ is a latent vector that identifies $\mathbf{z}_{ij}^{fg}$ itself. The whole inference model of $\mathbf{z}^{fg}$ can be written as:

$$q(\mathbf{z}^{fg}|\mathbf{x}) = \prod_{i=1}^{H}\prod_{j=1}^{W} q(z_{ij}^{pres}|\mathbf{x})(q(\mathbf{z}_{ij}^{where}|\mathbf{x})q(\mathbf{x}_{ij}|\mathbf{z}_{ij}^{where}, \mathbf{x})q(\mathbf{z}_{ij}^{what}|\mathbf{x}_{ij}))^{z_{ij}^{pres}}, \tag{1}$$

where $\mathbf{x}_{ij}$ is a small patch from the $\mathbf{x}$ that is obtained by the proposal bounding box identified by $\mathbf{z}_{ij}^{where}$, which should contain exactly one object if $z_{ij}^{pres} = 1$.

Unsupervised Clustering After applying SPACE on the randomly collected dataset, we can obtain a set of object representations: $D = \{z_{nij}^{pres}, \mathbf{z}_{nij}^{where}, \mathbf{z}_{nij}^{what}, \mathbf{x}_{nij}\}_{n=1}^{N}{}_{i=1}^{H}{}_{j=1}^{W}$, where N is the size of the dataset, $z_{nij}^{pres}, \mathbf{z}_{nij}^{where}, \mathbf{z}_{nij}^{what}, \mathbf{x}_{nij}$ are defined in Eq.(1). We first select object patches from D that is of high object-presence probability by a threshold τ: $D_\tau = \{\mathbf{x}_{nij} \in D : p(z_{nij}^{pres} = 1) > \tau\}$. Roughly speaking, each patch $\mathbf{x}_{nij} \in D_\tau$ should contain exactly one object. Therefore, we first project $\mathbf{x}_{nij} \in D_\tau$ into a low-dimensional latent space $\mathbf{R}^d$ by running IncrementalPCA [28] on D_τ, and then adopt KMeans clustering with a given cluster number C upon the projected latent representations. The object-category predictor $q(\mathbf{z}_{ij}^{cat}|\mathbf{x}_{ij}) = \Psi_{\texttt{KMeans}} \circ \Psi_{\texttt{IncrementalPCA}}$ is the composition of IncrementalPCA and KMeans:

$$\mathbf{z}_{ij}^{cat} = \Psi_{\texttt{KMeans}} \circ \Psi_{\texttt{IcreamentalPCA}}(\mathbf{x}_{ij}) \tag{2}$$

where $\mathbf{z}_{ij}^{cat} \in \mathbf{R}^C$ is an one-hot latent which indicates the category of $\mathbf{z}_{ij}$.

For simplicity, we regard the background (i.e. $z_{ij}^{pres} = 0$) as special 'object', thus $\mathbf{z}_{ij}^{cat}$ can be written as $\hat{\mathbf{z}}_{ij}^{cat} = \Psi_{\texttt{concatenate}}([1 - z_{ij}^{pres}, z_{ij}^{pres} \cdot \mathbf{z}_{ij}^{cat}])$. Note that $\hat{\mathbf{z}}_{ij}^{cat}$ is an one-hot latent $\in \mathbf{R}^{C+1}$.

Although the object-category predictor Eq.(2) is very simple, we find it works well in our experiments. In practice, the cluster number C of KMeans can be set by leveraging prior knowledge about ground-truth category number, or by analysing some clustering quality metrics such as silhouette coefficients. In more complex environments, we can also rely on other advanced unsupervised clustering methods such as NVISA [20], CC [19], etc.

3.2 OCAP: Object-Category Aware Perception

The OCAP module is designed to robustly incorporate the object knowledge of the (category-aware) UOD model defined in Section 3.1. In a word, OCAP is a plain convolution encoder that accepts the

[1] we omit $\mathbf{z}_{ij}^{depth}$ for simplicity, because we only consider 2-D environment without object occlusions in this paper.

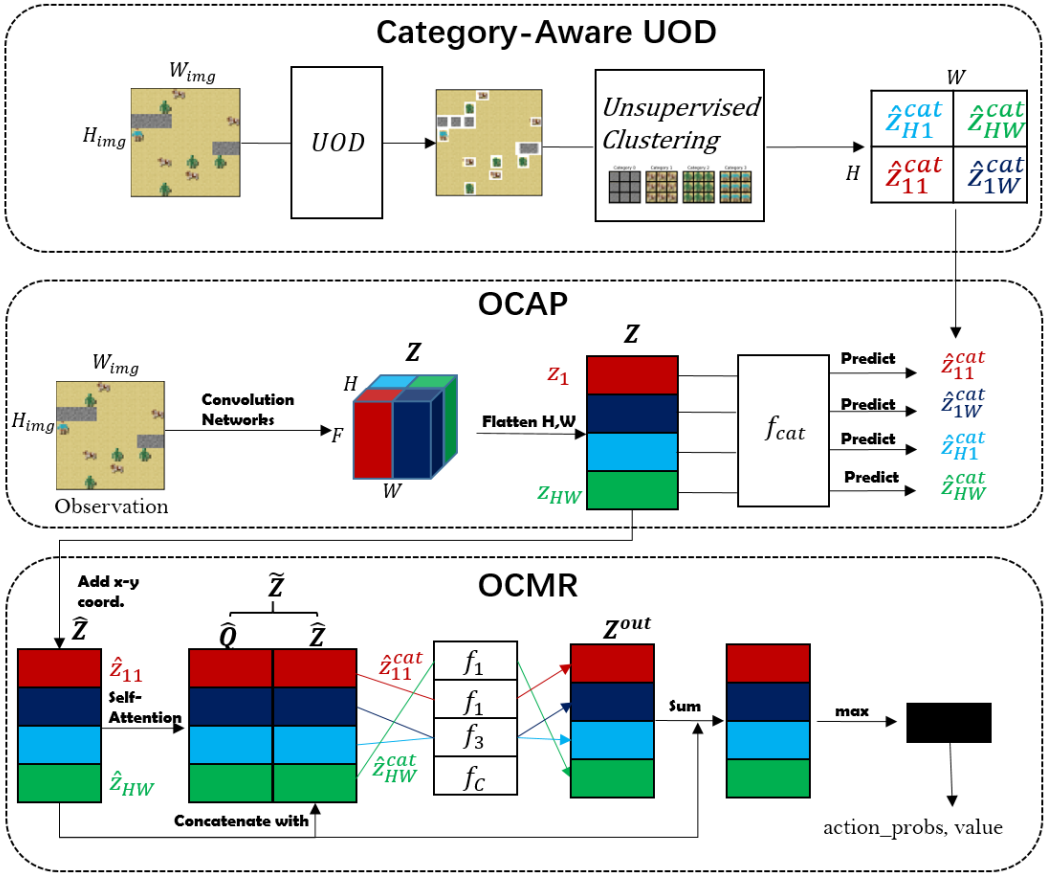


Figure 1: OCARL consists of 3 parts: (1) category-aware UOD, (2) OCAP, and (3) OCMR. (1) In category-aware UOD, the objects are firstly discovered by a UOD method (SPACE) and then assigned an (extended) category label by unsupervised clustering (IcreamentalPCA + KMeans). (2) In OCAP, the observation is first encoded into a set of features $\mathbf{Z}$ by convolution networks. Each vector in $\mathbf{z}_i = \mathbf{Z}_{:,i,j} \in \mathbf{Z}$ is given a task by the OCAP module to predict the corresponding (extended) object category, which is given by (1). $\mathbf{Z}$ is then fed to the OCMR module. (3) In OCMR, x-y coordinate information is added into $\mathbf{Z}$, which gives us $\widehat{\mathbf{Z}}$. Then we apply self-attention on $\widehat{\mathbf{Z}}$ to query related information $\widehat{\mathbf{Q}}$ for each object in $\widehat{\mathbf{Z}}$. $\widehat{\mathbf{Q}}$ and $\widehat{\mathbf{Z}}$ are concatenated to form $\widetilde{\mathbf{Z}}$. Each entity in $\widetilde{\mathbf{Z}}$ will be passed through different independent neural networks according to its corresponding object category, which will give us $\mathbf{Z}^{out}$. Finally, we perform a `max` operation along the H, W dimension of $\mathbf{Z}^{out} + \widehat{\mathbf{Z}}$, and use the resulting feature vector to predict the value function and action probabilities.

category information from the UOD model as additional supervision signals. Such a design makes OCARL robust to the incompleteness of the UOD model: in extreme cases in which the UOD model fails to discover any objects, OCAP degenerates into a plain convolution encoder which is still able to extract useful information from the raw images with the help of reward signals.

Suppose the convolution encoder in OCAP maps the raw image observation $\mathbf{X} \in \mathbf{R}^{3\times H_{img}\times W_{img}}$ to a latent representation $\mathbf{Z} \in \mathbf{R}^{F\times H\times W}$, where H, W are the same with those in the UOD model (see Eq.(1)). On the other hand, $\mathbf{X}$ is also fed into the UOD model to get the (extended) object category information $\hat{\mathbf{z}}^{cat} = \{\hat{\mathbf{z}}^{cat}_{ij}\}_{i=1}^{H}{}_{j=1}^{W}$ via Eq.(2).

The OCAP module forces the latent representation $\mathbf{Z}$ to encode the object category information, which is implemented by training a additional category predictor $f_{cat} : \mathbf{R}^F \to \texttt{Categorical}(C+1)$. f_{cat} predicts $\hat{\mathbf{z}}^{cat}_{ij}$ given $\mathbf{Z}_{:,i,j}$ which is a single channel in $\mathbf{Z}$. Therefore, the additional supervision signal that incorporates the object knowledge of the UOD model is given as:

$$L_{cat} = \sum_{i=1}^{H}\sum_{j=1}^{W} \texttt{CrossEntropyLoss}(f_{cat}(\mathbf{Z}_{:,i,j}); \hat{\mathbf{z}}^{cat}_{ij}).$$

L_{cat} is used for training the convolution encoder in OCAP. In practice, L_{cat} can be used as an *auxiliary loss* to any RL algorithm:

$$L_{total} = L_{RL} + \lambda_{cat} L_{cat}, \tag{3}$$

where $\lambda_{cat} \in R^+$ is a coefficient.

3.3 OCMR: Object-Centric Modular Reasoning

OCMR is a module that takes $\mathbf{Z} \in \mathbf{R}^{F \times H \times W}$ from the OCAP module as input and outputs a feature vector that summarizes $\mathbf{Z}$. The key design philosophy of OCMR is to adopt multiple independent and object-category-specific networks, each of which focuses on processing the object features of the same corresponding category. Compared with using a universal category-agnostic network, the processing logic of each independent network in OCMR is much simpler and therefore easier to master, which in turn allows for improved generalization as we will show in Section 4.3.2. This design philosophy also agrees with the recent discovery from [24; 6] which says that neural modules of specialization can lead to better generalization ability.

Given $\mathbf{Z} \in \mathbf{R}^{F \times H \times W}$ from OCAP, we first encode the x-y coordinates information (corresponding to the $H \times W$ grid) to each channel, and then map the resulting tensor into $\widehat{\mathbf{Z}} \in \mathbf{R}^{HW \times F}$. $\widehat{\mathbf{Z}}$ is treated as HW objects with x-y coordinate information encoded. To model the relations between objects, we apply a self-attention [31] module on $\widehat{\mathbf{Z}}$ to query information for each object from its related objects:

$$\widehat{\mathbf{Q}} = \texttt{softmax}(\widehat{\mathbf{Z}}\mathbf{W}_q(\widehat{\mathbf{Z}}\mathbf{W}_k)^T)\widehat{\mathbf{Z}}\mathbf{W}_v \in \mathbf{R}^{HW \times F}, \tag{4}$$

where $\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v \in \mathbf{R}^{F \times F}$ are trainable parameters. Modeling the relations is important, because one object's high-level semantic feature can be derived from other objects. For example, whether a door can be opened is determined by the existence of the key; therefore, the door should query other objects to check whether it is openable.

After the attention module, $\widehat{\mathbf{Q}}$ and $\widehat{\mathbf{Z}}$ are concatenated together to get $\widetilde{\mathbf{Z}} = \Psi_{\texttt{concatenate}}([\widehat{\mathbf{Q}}, \widehat{\mathbf{Z}}]) \in \mathbf{R}^{HW \times 2F}$. Each $\widetilde{\mathbf{Z}}_{i,j,:}$ in $\widetilde{\mathbf{Z}}$ is then fed into *different independent neural networks* according to its corresponding object category $\hat{\mathbf{z}}_{ij}^{cat}$. In practice, this can be implemented as:

$$\mathbf{Z}_{i,j,:}^{out} = \sum_{c=1}^{C+1} f_c(\widetilde{\mathbf{Z}}_{i,j,:}) \cdot \hat{z}_{ij;c}^{cat}, \tag{5}$$

where $f_c : \mathbf{R}^{2F} \to \mathbf{R}^F$, $[\hat{z}_{ij;1}^{cat}, ..., \hat{z}_{ij;C+1}^{cat}] = \hat{\mathbf{z}}_{ij}^{cat}$. Eq.(5) can be computed in parallel, resulting a tensor $\mathbf{Z}^{out} \in \mathbf{R}^{HW \times F}$.

Finally, the $\mathbf{Z}^{out}$ is added to $\widehat{\mathbf{Z}}$ (i.e. a residual connection). We perform a `max` operation along the H, W dimension, and using the resulting vector $\in \mathbf{R}^F$ to predict the value function and action probabilities:

$$\texttt{action_probs, value} = f_{ac}(\max_{H,W}(\mathbf{Z}^{out} + \widehat{\mathbf{Z}})). \tag{6}$$

4 Experiment

4.1 Task Description

In this work, we consider two task domains: `Crafter` and `Hunter`. The observations on both tasks are raw images of shape 64×64.

Crafter [15] is a complex 2-D MineCraft-style RL task, where complex behaviors are necessary for the agent' survival. The environment is procedurally generated by arranging various resources, terrain types, and objects (18 in total). The agent is rewarded if it can craft new items and accomplish achievements (22 possible achievements in total).

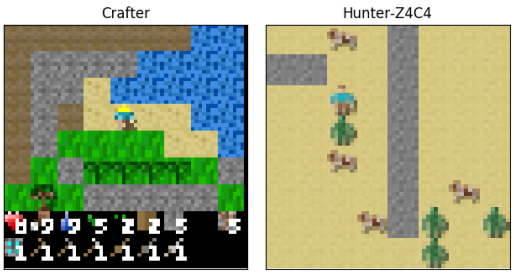


Figure 2: The `Crafter` (left) and `Hunter` (right) environment.

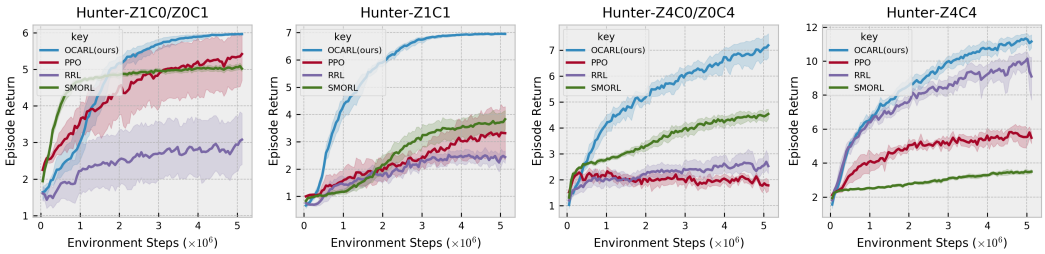


Figure 3: The results on `Hunter`. The mean returns over 12 seeds are plotted with 95% confidence interval. OCARL achieves better performance on *all* tasks; RRL is better than other baselines on `Hunter-Z4C4/Z4C4`, whereas SMORL is better on `Hunter-Z1C0/Z0C1`, `Hunter-Z1C1` and `Hunter-Z4C0`.

Alg.	Return	Score
RRL	3.58 ± 0.80	4.22 ± 1.24
SMORL	3.48 ± 0.26	3.94 ± 0.50
OCARL(ours)	$\mathbf{8.14 \pm 0.35}$	$\mathbf{12.31 \pm 0.99}$

Table 1: The results (averaged across 12 seeds) on `Crafter` after training for 25M environment steps. The 'Score' is a metric proposed in [15] which takes the difficultiy of each achievement into account, thus is more proper than 'Return' to benchmark agent's behaviours.

Hunter is much simpler than `Crafter`. It is also procedurally generated but only contains 4 types of objects: `Hunter`, `Cow`, `Zombie` and `Wall` [2]. The agent can control `Hunter` and can get positive reward (=1) if `Hunter` catches a `Cow` or kill a `Zombie`. The agent will be given a high reward (=5) if it can accomplish this task by catching/killing all `Cows` and `Zombies`. However, once `Hunter` is caught by a `Zombie`, the agent will receive a negative reward (=-1), and the episode ends. In a word, the agent in `Hunter` should master two different behavior patterns: chase & catch the `Cow`, and avoid & shoot at `Zombie`. Although `Hunter` is simple, it is a typical OORL task. We can derive different environment instances from `Hunter` by setting the number of `Zombies` and `Cows`. We use `Hunter-ZmCn` to denote an environment that spawns (`m` `Zombies` + `n` `Cows`) at the beginning of each episode, and `Hunter-ZmCn/ZnCm` an environment that spawns (`m` `Zombies` + `n` `Cows`) *or* (`n` `Zombies` + `m` `Cows`).

4.2 Evaluation

In this section, we evaluate OCARL's effect both on sample efficiency and generalization ability. We consider the following baselines: PPO [29], RRL [41], and SMORL [40]. PPO is a general-purpose on-policy RL algorithm with a plain neural network (i.e. Convolution + MLP). RRL proposes an attention-based neural network to introduce relational inductive biases and iterated relational reasoning into RL agent, which aims to solve object-oriented RL tasks without any other external supervision signals. SMORL utilizes the UOD model to decompose the observation into a set of object representations which are directly used as the agent's new observation. All algorithms (RRL, SMORL, and OCARL) are (re-)implemented upon PPO to ensure a fair comparison. For more implementation details, please refer to the Appendix.

4.2.1 Sample Efficiency

The results on `Hunter` domain are shown in Figure 3. We can conclude three facts from this figure: (1) OCARL performs better and is more stable than other baselines on *all* tasks. (2) RRL is competitive with OCARL on `Hunter-Z4C4` but significantly worse on other simper environments. This is because RRL relies solely on the reward signals to ground the visual images into objects; therefore, it requires enough (rewarded) interactions between objects, which simpler environments such as `Hunter-Z1C1` fail to supply. (3) SMORL performs better than other baselines (PPO, RRL) on simple environments such as `Hunter-Z1C0/Z0C1`, `Hunter-Z1C1` and `Hunter-Z4C0/Z0C4`, but unable to make progress in `Hunter-Z4C4` which consists of more objects. This is because SMORL adopts a simple reasoning module that may not be well-suited to multi-object reasoning.

[2] The images to render these objects come from `Crafter` [15], which is under MIT license.

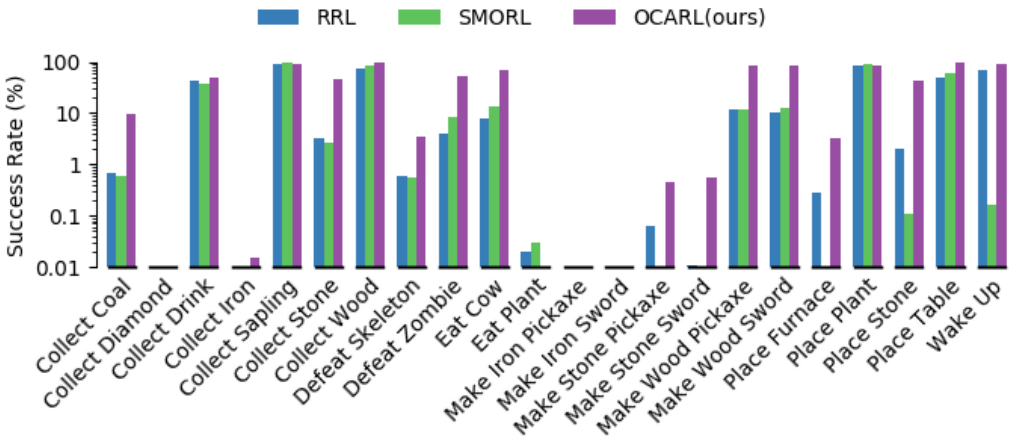


Figure 4: Success rates (averaged across 12 seeds) on 22 achievements in `Crafter`. These success rates are calculated using the final 1M environment steps during training. OCARL presents more meaningful behaviours than other baselines, such as collecting coal, defeating zombies, making wood pickaxes and so on.

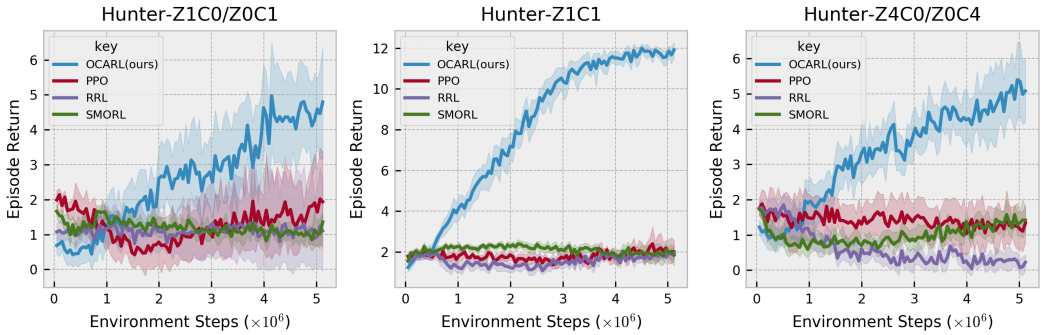


Figure 5: The generalization performance on `Hunter`. Agents are trained on different environments `Hunter-Z1C0/Z0C1`, `Hunter-Z1C1`, and `Hunter-Z4C0/Z0C4` respectively, but tested on `Hunter-Z4C4`. OCARL is the only algorithm that is able to generalize to the test environment. The mean returns over 12 seeds are plotted with 95% confidence interval.

The results on `Crafter` domain are shown in Table 1 and Figure 4. `Crafter` is a complex environment in that it features sparse (rewarded) interactions between objects and large object category numbers. Besides, some objects are omitted by the UOD model (see Appendix) because they almost do not appear in the training dataset due to the insufficient exploration of the random policy. Due to these features, both RRL and SMORL fail to provide noteworthy results (with scores of 4.22 and 3.94, respectively). On the other hand, OCARL is able to present more meaningful behaviours (with a score of 12.31), such as collecting coal, defeating zombies, making wood pickaxes and so on, as shown in Figure 4. The learning curves and detailed achievement success rates are provided in the Appendix.

4.2.2 Generalization

To evaluate the generalization ability, we train agents on `Hunter-Z1C0/Z0C1`, `Hunter-Z1C1` and `Hunter-Z4C0/Z0C4` separately, and then observe their test performance on `Hunter-Z4C4`. In `Hunter-Z1C1`, the agent needs to generalize from a few objects to more objects. In `Hunter-Z4C0/Z0C4`, the agent never observes the coincidence of `Cow` and `Zombie`. The generalization from `Hunter-Z1C0/Z0C1` to `Hunter-Z4C4` is the most difficult. Such a paradigm actually follows the *out-of-distribution* (OOD) setting, in that the object combination (`4 Zombies` + `4 Cows`) is never seen in the training environments. Therefore, although such generalization is possible for humans, it is much harder for RL algorithms.

As shown in Figure 5, OCARL is the only algorithm that make significant progress on the test environment. Although other baselines can improve their training performance (Figure 3), they are

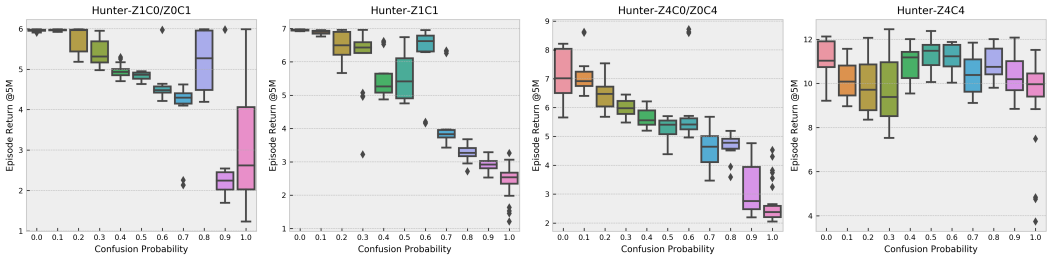


Figure 6: The Robustness of OCARL to the incompleteness of UOD methods. We randomly mask out the objects discovered by the UOD method with different confusion probability p (x-axis) and then record the final performance (y-axis) over 6 seeds after training for 5M environment steps. The performance of OCARL declines as p goes larger, but is still better than RRL (which exactly equals OCARL with $p = 1$)

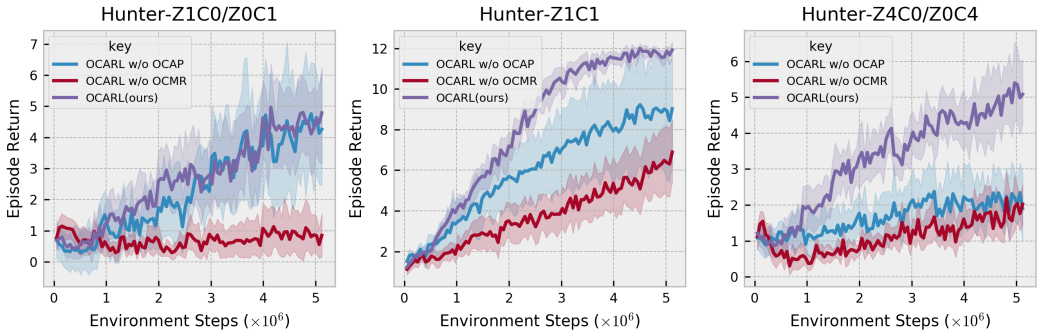


Figure 7: The generalization performance of OCARL, OCARL w/o OCMR, and OCARL w/o OCAP. The mean returns over 12 seeds are plotted with a 95% confidence interval. Without the help of OCMR, the generalization ability of OCARL will decline significantly. On the other hand, OCMR alone can bring benefits without the help of OCAP (although not as consistently as OCMR+OCAP).

unable to generalize to the test environment. The generalization performance of OCARL is most impressive on `Hunter-Z1C1`, because agent trained on `Hunter-Z1C1` even achieves slightly better than that on `Hunter-Z4C4` (i.e. directly trained on the test environment, see the last figure in Figure 3.).

4.3 Ablation study

4.3.1 Robustness to the Incompleteness of Object Discovery Methods

OCARL relies on UOD methods to discover different objects from given observations, which are not guaranteed to discover *all* objects, especially in complex environments [36]. Therefore, one may naturally ask to what extent OCARL's performance is affected by such incompleteness.

To answer this question, we randomly mask out each object (by setting z_{ij}^{pres} to 0) discovered by the UOD model with a given probability $p \in [0, 1]$, and evaluate OCARL's performance after training 5M environment steps on `Hunter`. Such random confusion of the UOD model will affect both the OCAP and OCMR modules of OCARL. Note that OCARL($p = 0$) is OCARL itself, and OCARL($p = 1$) is exactly RRL. The results are shown in Figure 6. Although increasing p does hinder OCARL's performance, OCARL can still benefit from the imperfect UOD model in that its performance is still better than RRL in general. Such robustness should be attributed to the fact that OCARL does not directly use the output of the UOD model as observation but instead treats it as an additional supervision signal that help the encoder to capture object-category information, which gives OCARL a second chance to find other useful information from the raw image.

4.3.2 OCMR Improves Generalization

One key advantage of OCARL is that it can generalize to unseen combinations of objects by leveraging the invariant relations between objects learnt during training. We argue that such an advantage is established via OCMR, which adopts object-category-specific networks to deal with different objects.

Test Environment	No Modification	Disable f_Z	Disable f_C
Hunter-Z4C0	8.27 ± 1.77	1.17 ± 1.16	8.32 ± 1.46
Hunter-Z0C4	8.29 ± 1.41	8.13 ± 1.78	4.90 ± 1.77

Table 2: The Modularity of OCMR. We first train agent on Hunter-Z4C4 and then disable the module f_Z (corresponding to Zombie) or f_C (corresponding to Cow) in OCMR (see Eq.(5)). Each agent is evaluated on both Hunter-Z4C0 and Hunter-Z0C4, and the mean and standard deviation of 12 seeds are reported . The results show that the behaviour patterns (chase & catch the Cow) and (avoid & shoot at Zombie) are encoded in f_C and f_Z respectively.

To confirm our argument, we consider a variant of OCARL (i.e. OCARL w/o OCMR) in which the OCMR module is removed and replaced by an RRL-like reasoning module which adopts a universal network (instead of multiple object-category-specific networks as OCARL does). The comparison between OCARL w/ and w/o OCMR is shown in Figure 7. OCARL w/o OCMR (almost completely) fails to generalize from Hunter-Z1C0/Z0C1, Hunter-Z4C0/Z0C4 to Hunter-Z4C4, while OCARL w/ OCMR does. In Figure 7, we also consider a variant of OCARL (i.e. OCARL w/o OCAP) in which the OCAP module is disabled by setting $\lambda_{cat} = 0$ in Eq.(3). In this paradigm, only OCMR is in effect. As shown in Figure 7, OCMR alone can still bring benefits without the help of OCAP (although not as consistently as OCMR+OCAP). According to these results, we can conclude that OCMR is crucial in OCARL's success.

4.3.3 The Modularity of OCMR

OCMR adopts different modules to deal with object features from different categories. In this section, we will show that each module in OCMR actually encodes an object-category-specific behaviour pattern.

As stated in Section 4.1, the agent should master two useful behaviour patterns in Hunter: (1) chase & catch the Cow, and (2) avoid & shoot at Zombie. The existence of these patterns can be observed by evaluating the agent in Hunter-Z0C4 and Hunter-Z4C0 respectively, because agents that master pattern (1) (pattern (2)) should achieve better performance on Hunter-Z0C4 (Hunter-Z4C0). In Table 2, we first train agent on Hunter-Z4C4 and then disable the module f_Z (corresponding to Zombie) or f_C (corresponding to Cow) by replacing f_C or f_E with f_{bg} (corresponding to the 'background') in Eq.(5). The results shows that disabling f_Z does not affect the performance on Hunter-Z0C4 ($8.29 \rightarrow 8.13$), which means that f_Z is not relative to behaviour pattern (1). Instead, f_Z is corresponding to behaviour pattern (2) because we can observe a significant performance decline ($8.27 \rightarrow 1.17$) on Hunter-Z4C0 when we disable f_Z. Similar phenomenon can also be observed on f_C.

5 Conclusion

In this paper, we propose OCARL, which utilize the category information of objects to facilitate both perception. For the perception, we propose the OCAP, which enables the encoder to distinguish the categories of objects. For the reasoning, we propose the OCMR, which adopts object-category-specific networks to deal with different objects. Our experiments are carried out on Crafter and Hunter. Experiments show that OCARL outperforms other baselines both on sample efficiency and generalization ability. We also perform several ablation studies to show that (1) OCARL is robust to the incompleteness of UOD methods, (2) OCMR is critical to improving generalization, and (3) each module in OCMR actually encodes an object-category-specific behaviour pattern in Hunter.

Limitation The main limitation is that we only test OCARL's generalization ability to unseen object combinations, but not novel object instances. In order to generalize to novel object instances that looks very different from old ones, the agent should interact with the novel objects first and then infer the their underlying categories according to these interactions. Such a problem is likely to be in the meta-RL domain and needs more efforts to solve, which we would like to leave for future work.

References

[1] Fabien Baradel, Natalia Neverova, Christian Wolf, Julien Mille, and Greg Mori. Object level visual reasoning in videos. In *ECCV*, 2018.

[2] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinícius Flores Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Çaglar Gülçehre, H. Francis Song, Andrew J. Ballard, Justin Gilmer, George E. Dahl, Ashish Vaswani, Kelsey R. Allen, Charlie Nash, Victoria Langston, Chris Dyer, Nicolas Manfred Otto Heess, Daan Wierstra, Pushmeet Kohli, Matthew M. Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. *ArXiv*, 2018.

[3] Christopher P. Burgess, Loïc Matthey, Nicholas Watters, Rishabh Kabra, Irina Higgins, Matthew M. Botvinick, and Alexander Lerchner. Monet: Unsupervised scene decomposition and representation. *ArXiv*, 2019.

[4] Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. *ArXiv*, 2020.

[5] Luis C. Cobo, Charles Lee Isbell, and Andrea Lockerd Thomaz. Object focused q-learning for autonomous agents. In *AAMAS*, 2013.

[6] Vanessa D'Amario, Tomotake Sasaki, and Xavier Boix. How modular should neural module networks be for systematic generalization? In *NIPS*, 2021.

[7] Long Hoang Dang, Thao Minh Le, Vuong Le, and Truyen Tran. Hierarchical object-oriented spatio-temporal reasoning for video question answering. In *IJCAI*, 2021.

[8] Harm de Vries, Dzmitry Bahdanau, Shikhar Murty, Aaron C. Courville, and Philippe Beaudoin. Closure: Assessing systematic generalization of clevr models. In *NeurIPS Workshop*, 2019.

[9] Carlos Diuk, Andre Cohen, and Michael L. Littman. An object-oriented representation for efficient reinforcement learning. In *ICML*, 2008.

[10] Lasse Espeholt, Hubert Soyer, Rémi Munos, Karen Simonyan, Volodymyr 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. *ICML*, 2018.

[11] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *Found. Trends Mach. Learn.*, 2018.

[12] Klaus Greff, Raphael Lopez Kaufman, Rishabh Kabra, Nicholas Watters, Christopher P. Burgess, Daniel Zoran, Loïc Matthey, Matthew M. Botvinick, and Alexander Lerchner. Multi-object representation learning with iterative variational inference. In *ICML*, 2019.

[13] Carlos Guestrin, Daphne Koller, Chris Gearhart, and Neal Kanodia. Generalizing plans to new environments in relational mdps. In *IJCAI*, 2003.

[14] Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors. *NIPS*, 2017.

[15] Danijar Hafner. Benchmarking the spectrum of agent capabilities. *ICLR*, 2022.

[16] Zhen He, Jian Li, Daxue Liu, Hangen He, and David Barber. Tracking by animation: Unsupervised learning of multi-object attentive trackers. *CVPR*, 2019.

[17] Jindong Jiang, Sepehr Janghorbani, Gerard de Melo, and Sungjin Ahn. Scalor: Generative world models with scalable object representations. In *ICLR*, 2020.

[18] Ken Kansky, Tom Silver, David A. Mély, Mohamed Eldawy, Miguel Lázaro-Gredilla, Xinghua Lou, Nimrod Dorfman, Szymon Sidor, D. Scott Phoenix, and Dileep George. Schema networks: Zero-shot transfer with a generative causal model of intuitive physics. *ArXiv*, 2017.

[19] Yunfan Li, Peng Hu, Zitao Liu, Dezhong Peng, Joey Tianyi Zhou, and Xi Peng. Contrastive clustering. *AAAI*, 2021.

[20] Zhihan Li, Youjian Zhao, Haowen Xu, Wenxiao Chen, Shangqing Xu, Yilin Li, and Dan Pei. Unsupervised clustering through gaussian mixture variational autoencoder with non-reparameterized variational inference and std annealing. In *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020.

[21] Zhixuan Lin, Yi-Fu Wu, Skand Vishwanath Peri, Weihao Sun, Gautam Singh, Fei Deng, Jindong Jiang, and Sungjin Ahn. Space: Unsupervised object-oriented scene representation via spatial attention and decomposition. *ArXiv*, 2020.

[22] Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. Object-centric learning with slot attention. *ArXiv*, abs/2006.15055, 2020.

[23] Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. Object-centric learning with slot attention. In *NeurIPS*, 2020.

[24] Spandan Madan, Timothy Henry, Jamell Dozier, Helen Ho, Nishchal Bhandari, Tomotake Sasaki, Frédo Durand, Hanspeter Pfister, and Xavier Boix. When and how convolutional neural networks generalize to out-of-distribution category–viewpoint combinations. *Nature Machine Intelligence*, 2022.

[25] Ofir Marom and Benjamin Rosman. Zero-shot transfer with deictic object-oriented representation in reinforcement learning. In *NeurIPS*, 2018.

[26] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charlie Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 2015.

[27] Jean Piaget. Piaget's theory. 1976.

[28] David A. Ross, Jongwoo Lim, Ruei-Sung Lin, and Ming-Hsuan Yang. Incremental learning for robust visual tracking. *International Journal of Computer Vision*, 2007.

[29] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *ArXiv*, 2017.

[30] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, L. Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 2018.

[31] Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *NIPS*, 2017.

[32] Rishi Veerapaneni, John D. Co-Reyes, Michael Chang, Michael Janner, Chelsea Finn, Jiajun Wu, Joshua B. Tenenbaum, and Sergey Levine. Entity abstraction in visual model-based reinforcement learning. In *CoRL*, 2019.

[33] V. Vijay, Abhinav Ganesh, Hanlin Tang, and Arjun K. Bansal. Generalization to novel objects using prior relational knowledge. *ArXiv*, 2019.

[34] Haonan Wang, Ning Liu, Yi yun Zhang, Dawei Feng, Feng Huang, Dongsheng Li, and Yiming Zhang. Deep reinforcement learning: a survey. *Frontiers Inf. Technol. Electron. Eng.*, 2020.

[35] Nicholas Watters, Loïc Matthey, Matko Bosnjak, Christopher P. Burgess, and Alexander Lerchner. Cobra: Data-efficient model-based rl through unsupervised object discovery and curiosity-driven exploration. *ArXiv*, abs/1905.09275, 2019.

[36] Marissa A. Weis, Kashyap Chitta, Yash Sharma, Wieland Brendel, Matthias Bethge, Andreas Geiger, and Alexander S. Ecker. Benchmarking unsupervised object representations for video sequences. *J. Mach. Learn. Res.*, 2021.

[37] Jiayi Weng, Huayu Chen, Dong Yan, Kaichao You, Alexis Duburcq, Minghao Zhang, Hang Su, and Jun Zhu. Tianshou: A highly modularized deep reinforcement learning library. *arXiv preprint arXiv:2107.14171*, 2021.

[38] Guangyu Robert Yang, Madhura R. Joglekar, H. Francis Song, William T. Newsome, and Xiao-Jing Wang. Task representations in neural networks trained to perform many cognitive tasks. *Nature Neuroscience*, 2019.

[39] Andrii Zadaianchuk, Georg Martius, and Fanny Yang. Self-supervised reinforcement learning with independently controllable subgoals. *ArXiv*, 2021.

[40] Andrii Zadaianchuk, Maximilian Seitzer, and Georg Martius. Self-supervised visual reinforcement learning with object-centric representations. In *ICLR*, 2021.

[41] Vinícius Flores Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David P. Reichert, Timothy P. Lillicrap, Edward Lockhart, Murray Shanahan, Victoria Langston, Razvan Pascanu, Matthew M. Botvinick, Oriol Vinyals, and Peter W. Battaglia. Deep reinforcement learning with relational inductive biases. In *ICLR 2019*, 2019.

[42] Guangxiang Zhu and Chongjie Zhang. Object-oriented dynamics predictor. In *NeurIPS*, 2018.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [No]
   (c) Did you discuss any potential negative societal impacts of your work? [No]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] see Appendix.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [No]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes] Partially.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No]
5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Implementation Details

The implementation of OCARL is available at `https://anonymous.4open.science/r/OCARL-51BF`.

Hyper-parameters for SPACE In this paper, we use SPACE for object discovery. Most hyper-parameters are the same as the default parameters for Atari in SPACE, except that (1) we use a smaller network to deal with 64×64 images; (2) For `Hunter`, we disable the background module in SPACE because the observations in `Hunter` is quite simple. (3) We set the dimension of object representations $\mathbf{z}_{ij}^{what}$ to be 16, and (in `Crafter`) $\mathbf{z}^{fg}$ to be 8. The training data for SPACE is obtained via running a random policy on `Hunter-Z4C4` and `Crafter` for 100000 environment steps.

Hyper-parameters for PPO Our PPO implementation is based on Tianshou [37] which is purely based on PyTorch. We adopt the default hyper-parameters in Tianshou, which are shown in Table 3.

Hyper-parameters for OCARL OCARL only introduces 2 hyper-parameter λ_{cat} (the coefficient for L_{cat} defined in Eq.(3.2)) and C (the number of object categories). We set $\lambda_{cat} = 0.01$ for all experiments, $C = 4, 8$ for `Hunter` and `Crafter` respectively.

Network Architecture In OCAP, we use the convolution encoder from IMPALA [10], which consists of 3 residual blocks with 16, 32, and 32 channels, respectively. By the encoder, the $3 \times 64 \times 64$ image observation is encoded into $32 \times 8 \times 8$. The category predictor f_{cat} is a simple 2-layer MLP with hidden size 32. The object-category-specific networks $f_1, ..., f_{C+1}$ in the OCMR module consists of $C + 1$ MLPs, each of which is of 2-layer with hidden size 64.

Implementation for RRL The main implementation differences between RRL and OCARL are that (1) RRL does not use the supervision signals from the UOD model (i.e. $\lambda_{cat} = 0$ in Eq.(3)); (2) RRL adopts a different (although very similar) network architecture. In RRL, we use the same convolution encoder as OCARL. The reasoning module in RRL is similar to OCARL, except that it utilizes a non-linear universe transformation instead of multiple object-category-specific networks. We also use only one relational block in RRL as OCARL does, because we found more blocks may hinder the performance in our early experiments.

Implementation for SMORL We use SPACE [21] instead of SCALOR [17] (as suggested in the SMORL paper) to obtain object representations for SMORL. Since SMORL is originally designed for goal-based RL, it uses an attention model to gather information from the set of object representations, in which the goal serves as a query vector. In our settings, we do not consider goal-based RL; therefore, we use L learnable vectors as queries, which are also used in the original SMORL. We search over $L = [1, 2, 4, 8]$ and find that $L = 4$ works best in our experiments. We also find that it is better to apply an MLP to the object representations before they are fed into the attention module. To be more specific, we first run SPACE to get the object representations, sharing the same procedure as OCARL. The information of these objects is first processed by a 2-layer MLP with hidden size 32 and then gathered through an attention module in which the query consists of $L = 4$ learnable vectors. The resulting tensor (of shape $L \times 32$) is flattened and then fed into a 2-layer MLP with hidden size 64, which finally output the value function and action probabilities.

B Details of SPACE

OCARL utilizes SPACE to discovery objects from raw objects. In Section 3.1, we have introduced the inference model of $\mathbf{z}^{fg}$. For completeness, we would like to introduce the remaining parts of SPACE in this section: (1) inference model of $\mathbf{z}^{bg}$; (2) generative model of the image $\mathbf{x}$; and (3) the ELBO to train the model.

Inference model of $\mathbf{z}^{bg}$ $\mathbf{z}^{bg}$ consists of several components $\mathbf{z}_k^{bg}$, and these components are inferred from the image $\mathbf{x}$ in an iterative manner: $q(\mathbf{z}_k^{bg}|\mathbf{x}) = \prod_{k=1}^{K} q(\mathbf{z}_k^{bg}|\mathbf{z}_{<k}^{bg}, \mathbf{x})$. Each component $\mathbf{z}_k^{bg}$ is futher divided into two parts: $\mathbf{z}_k^{bg} = (\mathbf{z}_k^m, \mathbf{z}_k^c)$, where $\mathbf{z}_k^m$ models the mixing weight assigned to the background component (see π_k in the generative model of $\mathbf{x}$), and $\mathbf{z}_k^c$ models the RGB distribution ($p(\mathbf{x}|\mathbf{z}_k^{bg})$) of the background component.

Hyper-parameter	Value
Discount factor	0.99
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	1024
Repeat per collect	3
Batch size	256

Table 3: PPO hyper-parameters.

Generative model of $\mathbf{x}$ The generative model consists of two parts: $p(\mathbf{x}|\mathbf{z}^{fg})$ and $p(\mathbf{x}|\mathbf{z}^{bg})$. For $p(\mathbf{x}|\mathbf{z}^{fg})$, each $\mathbf{z}_{ij}^{fg}$ is passed through a decoder to reconstruct the image patch determined by $\mathbf{z}_{ij}^{where}$. For $p(\mathbf{x}|\mathbf{z}^{bg})$, each $\mathbf{z}_{k}^{bg}$ is decoded into a background component and all components are mixed together to get the background. Foreground and background are combined with a pixel-wise mixture model to reconstruct the original image $\mathbf{x}$. The whole generative model of $\mathbf{x}$ is:

$$p(\mathbf{x}) = \int\int p(\mathbf{x}|\mathbf{z}^{fg}, \mathbf{z}^{bg}) p(\mathbf{z}^{fg}) p(\mathbf{z}^{bg}) d\mathbf{z}^{fg} d\mathbf{z}^{bg}$$

$$p(\mathbf{x}|\mathbf{z}^{fg}, \mathbf{z}^{bg}) = \alpha p(\mathbf{x}|\mathbf{z}^{fg}) + (1-\alpha)\sum_{k=1}^{K} \pi_k p(\mathbf{x}|\mathbf{z}_k^{bg}), \alpha = f_\alpha(\mathbf{z}^{fg}), \pi_k = f_{\pi_k}(\mathbf{z}_{1:k}^{bg})$$

$$p(\mathbf{z}^{fg}) = \prod_{i=1}^{H}\prod_{j=1}^{W} p(z_{ij}^{pres})(p(\mathbf{z}_{ij}^{where})p(\mathbf{z}_{ij}^{what}))^{z_{ij}^{pres}}$$

$$p(\mathbf{z}^{bg}) = \prod_{k=1}^{K} p(\mathbf{z}_k^c|\mathbf{z}_k^m) p(\mathbf{z}_k^m|\mathbf{z}_{<k}^m)$$

In above equations, $z_{ij}^{pres}, \mathbf{z}_{ij}^{where}, \mathbf{z}_{ij}^{what}$ have been discussed in Section 3.1, α is foreground mixing probability, and π_k is the mixing weight assigned to the background component $\mathbf{z}_k^{bg}$.

The ELBO to train the model SPACE is trained using the following ELBO via reparameterization tricks:

$$\mathcal{L}(\mathbf{x}) = \mathbb{E}_{q(\mathbf{z}^{fg}, \mathbf{z}^{bg}|\mathbf{x})}\left[p\left(\mathbf{x} \mid \mathbf{z}^{fg}, \mathbf{z}^{bg}\right)\right] - D_{\mathrm{KL}}\left(q\left(\mathbf{z}^{fg} \mid \mathbf{x}\right) \| p\left(\mathbf{z}^{fg}\right)\right) - D_{\mathrm{KL}}\left(q\left(\mathbf{z}^{bg} \mid \mathbf{x}\right) \| p\left(\mathbf{z}^{bg}\right)\right)$$

Achievement	**OCARL(ours)**	**SMORL**	**RRL**
Collect Coal	**9.5%**	0.7%	0.7%
Collect Diamond	0.0%	0.0%	0.0%
Collect Drink	**49.0%**	41.6%	42.3%
Collect Iron	**0.0%**	0.0%	0.0%
Collect Sapling	**89.7%**	**93.4%**	**89.9%**
Collect Stone	**46.1%**	2.9%	3.2%
Collect Wood	**98.5%**	85.7%	71.7%
Defeat Skeleton	**3.5%**	0.5%	0.6%
Defeat Zombie	**51.4%**	11.3%	3.8%
Eat Cow	**67.2%**	11.6%	7.6%
Eat Plant	0.0%	0.0%	0.0%
Make Iron Pickaxe	0.0%	0.0%	0.0%
Make Iron Sword	0.0%	0.0%	0.0%
Make Stone Pickaxe	**0.4%**	0.0%	0.1%
Make Stone Sword	**0.6%**	0.0%	0.0%
Make Wood Pickaxe	**86.1%**	15.2%	12.0%
Make Wood Sword	**83.5%**	17.8%	10.6%
Place Furnace	**3.2%**	0.0%	0.3%
Place Plant	**86.9%**	**88.4%**	83.0%
Place Stone	**44.1%**	1.2%	2.1%
Place Table	**95.6%**	67.5%	49.8%
Wake Up	**88.1%**	0.5%	70.2%
Score	**12.3%**	3.9%	4.2%

Table 4: Success rates on 22 achievements in `Crafter`.

C More Results on `Crafter`

In Figure 8, we plot the learning curves on the `Crafter` domain. Only OCARL is able to make progress continuously. Both RRL and SMORL get stuck in local minimal after training for about 1M environment steps.

In Table 4, we give the detailed success rates for 22 achievements on `Crafter`, which is also reported in Figure 4. As shown in Table 4, OCARL presents more meaningful behaviours such as collecting coal, defeating zombies, making wood pickaxe and so on.

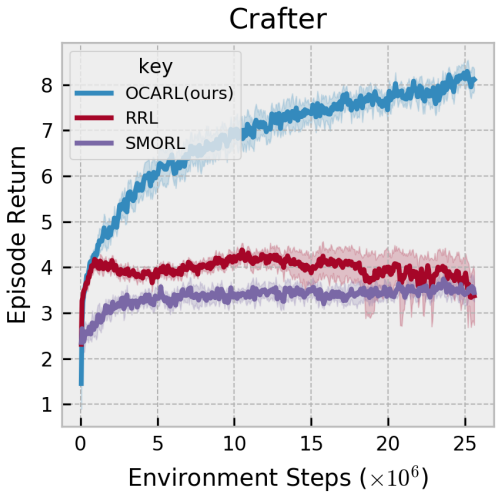


Figure 8: The learning curves on `Crafter`. The mean returns over 12 seeds are plotted with a 95% confidence interval. Both RRL and SMORL fail to make progress after the initial 5M environment steps, while OCARL does.

water	grass	stone	path	sand	tree	lava	coal	iron
2.76	46.5	1.09	0.36	0.94	2.24	0.018	0.094	0.0294
diamond	table	furnace	player	cow	zombie	skeleton	arrow	plant
0.0025	0.0108	0	1	0.495	0.14	0.0947	0.0973	0.165

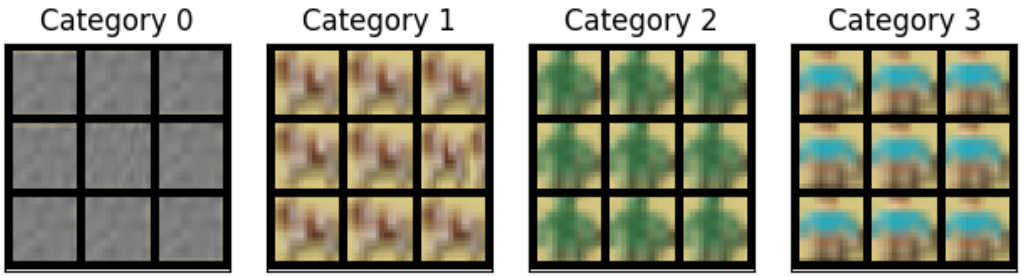
Table 5: The average number of objects in a single image. These numbers are obtained on the training dataset of SPACE via analysing the ground truth object category label provided by `Crafter`.

D Analysis of Unsupervised Object Discovery

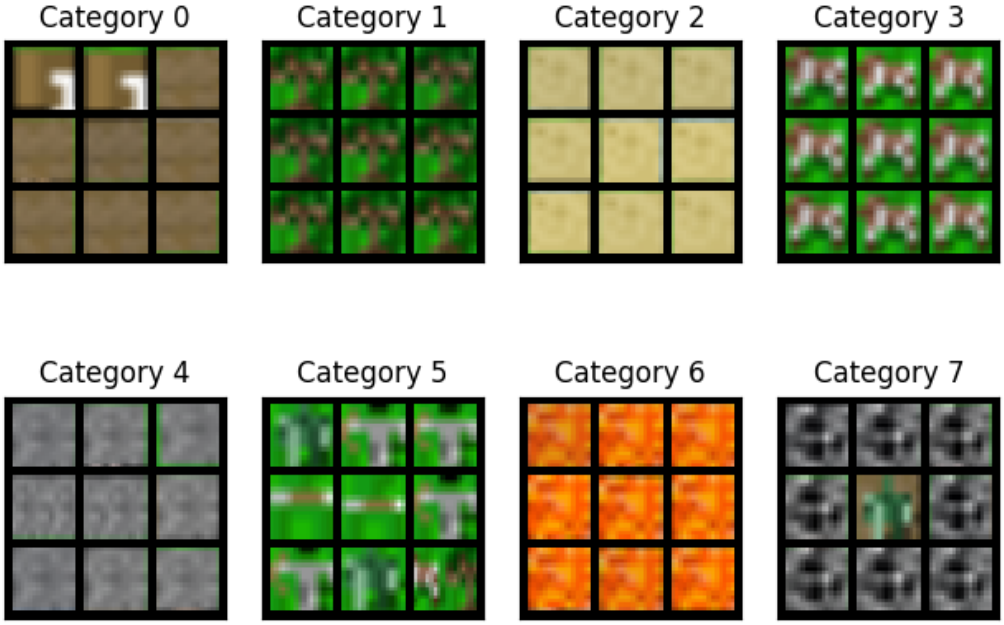
In Figure 9, we plot the discovered object categories in `Crafter` and `Hunter`. Since `Hunter` is quite simple, our algorithm can assign correct categories for all objects. In `Crafter`, there exist 18 kinds of objects in total, making it much harder than `Hunter`. Since we use only 8 categories, there are some cases that multiple objects with different ground-truth categories are predicted into the same category (such as `Category` 5 and 7 in Figure 9). Although there exist assignment mistakes in `Crafter`, OCARL is still much better than other baselines.

Figure 9 also shows that some objects in `Crafter` is omitted by the UOD model, such as , , , , and . This is probably because the some objects almost does not appear in in training dataset (Table 5) or are treated as background by SPACE.

In Figure 10, we report the object discovered by SPACE. The combination of SPACE and unsupervised clustering on the discovered objects (i.e. Eq.(2)) can tell us not only where one object locates, but also what category it is.

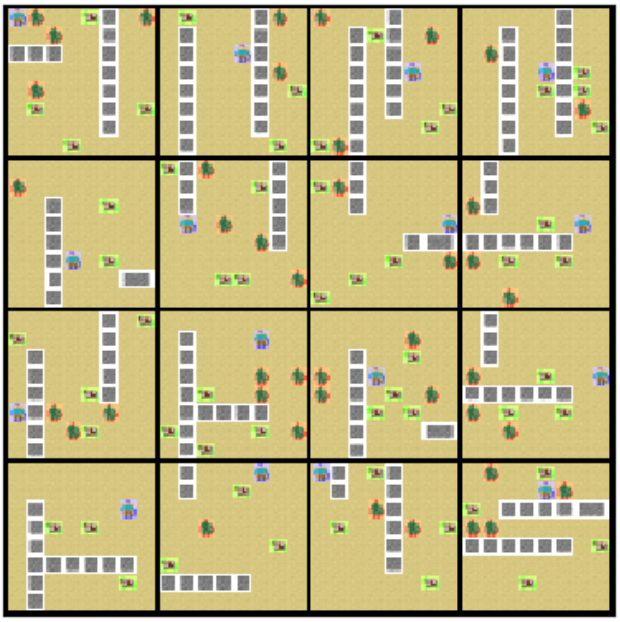


(a) The discovered object categories on `Hunter`.

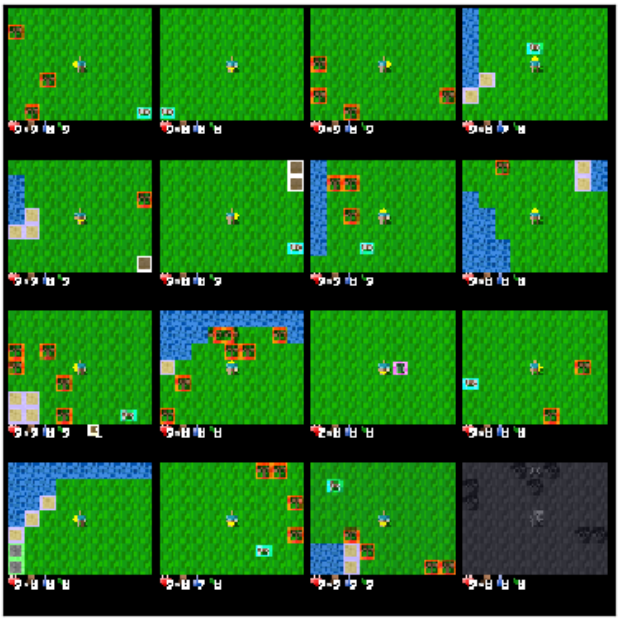


(b) The discovered object categories on on `Crafter`

Figure 9: The objects discovered by Eq.(2). In `Hunter`, object categories are perfectly predicted, whereas in `Crafter` there exist some mistakes.



(a) The objects discovered on `Hunter`.



(b) The objects discovered on `Crafter`

Figure 10: The objects discovered by SPACE. The objects are bound by boxes whose colour indicates its corresponding category that is predicted by Eq.(2). In `Crafter`, the object centred in each image is ignored by the SPACE because its location never changes and thus is distinguished as background.