Object-Oriented Decomposition of World Model in Reinforcement Learning

Leonid Ugadiarov[1] , **Aleksandr I. Panov**[2,3]
[1]Moscow Institute of Physics and Technology
[2]Federal Research Center "Computer Science and Control" of the Russian Academy of Sciences
[3]Autonomous Non-Profit Organization Artificial Intelligence Research Institute
ulaelfray@gmail.com, pan@isa.ru

Abstract

Object-oriented models are expected to have better generalization abilities and operate on a more compact state representation. Recent studies have shown that using pre-trained object-centric representation learning models for state factorization in model-free algorithms improves the efficiency of policy learning. Approaches using object-factored world models to predict the environment dynamics have also shown their effectiveness in object-based grid-world environments. Following those works, we propose a novel object-oriented model-based value-based reinforcement learning algorithm Object Oriented Q-network (OOQN) employing an object-oriented decomposition of the world and state-value models. The results of the experiments demonstrate that the developed algorithm outperforms state-of-the-art model-free policy gradient algorithms and model-based value-based algorithm with a monolithic world model in tasks where individual dynamics of the objects is similar.

1 Introduction

The Markov decision process (MDP) [Sutton and Barto, 2018] is considered as a mathematical model for reinforcement learning problems. Object-oriented MDP [Diuk *et al.*, 2008] is based on the relational MDP and describes the dynamics of the environment in terms of the objects it contains, their classes, and the relationships between them. One of the main challenges in visual-based reinforcement learning (RL) is determining how to effectively represent the state of the environment. The most common approach is to encode the entire input image which is then used as input for the policy network [Mnih *et al.*, 2015]. Previously, [Santoro *et al.*, 2017] has shown that such representations may fail to capture important relationships and interactions between objects in the state. It is expected that the use of object representations in reinforcement learning will lead to more compact models with better generalization ability [Keramati *et al.*, 2018]. The recent studies on the effect of state factorization on the performance of model-free algorithms [Stanić *et al.*, 2022] [Yoon *et al.*, 2023] show that the generalization ability of the algorithms does improve in this case. On the other hand, an environment model that uses object representations and is explicitly trained to model relationships between objects can further improve learning efficiency. A contrastively-trained transition model CSWM [Kipf *et al.*, 2020] that simultaneously learns to factorize the state and predict the change in the state of individual objects demonstrates a quality of prediction that is superior to the results of models with a monolithic state. In another work [Watters *et al.*, 2019], an object-oriented world model is employed during exploration phase, which helps to maintain the effectiveness of policy learning as the environment becomes more complex.

Models trained in an unsupervised learning approach which factorize visual input data into individual objects demonstrate high quality in relatively simple environments with strongly distinguishable objects. Also, in object-structured environments, an action performed in a step is often applied to a single object or a small number objects, rather than to all objects in the environment, makes it easier to predict the dynamics of individual objects. Despite recent progress in this problem [Biza *et al.*, 2022b] no fully-featured dynamics models have been proposed that takes into account sparsity of action-object relationships. These obstacles make it difficult to employ factored world models in reinforcement learning. For example, the CSWM model has never been used for policy learning in offline or online settings. In this work we demonstrate the successful application of a GNN-based object-oriented world model for model-based policy learning.

In model-based reinforcement learning (MBRL) the agent builds models for transition and reward functions using the experience of interaction with the environment. The agent performs multi-step planning to select the optimal action based on the predictions of the models. The model-based algorithms could be more efficient than model-free algorithms if the world model is sufficiently accurate. A notable example of model-based approach is Dreamer algorithm [Hafner *et al.*, 2023], whose latest version masters a wide range of environments with a fixed set of hyperparameters, outperforming specialized methods, and is acknowledged as the first algorithm to collect diamonds in Minecraft from scratch without human data or curricula. However, its world model is monolithic and recent attempts to factorize it have had limited success [Zholus *et al.*, 2022]. Our research is focused on value-based MBRL as object-based decomposition of value function could contribute to the training of object-oriented world model which is consistent with policy.

2 Background and Related Work

We consider a simplified version of the object-oriented MDP: [Diuk *et al.*, 2008]:

$$\mathcal{U} = (\mathcal{S}, \mathcal{A}, T, R, \gamma, \mathcal{O}, \Omega), \tag{1}$$

$\mathcal{S} = \mathcal{S}_1 \times \cdots \times \mathcal{S}_K$ — a state space, $\mathcal{S}_i$ — an individual state space of the object i, $\mathcal{A}$ — an action space, $T = (T_1, \ldots, T_K)$ — a transition function, $T_i = T_i(T_{i1}(s_i, s_1, a), \ldots, T_{iK}(s_i, s_K, a))$ — an individual transition function of the object i, $R = \sum_{i=1}^{K} R_i$ — a reward function, $R_i = R_i(R_{i1}(s_i, s_1, a), \ldots, R_{iK}(s_i, s_K, a))$ — an individual reward function of the object i, $\gamma \in [0; 1]$ — a discount factor, $\mathcal{O}$ — an observation space, $\Omega : \mathcal{S} \to \mathcal{O}$ — an observation function. The goal of reinforcement learning is to find the optimal policy: $\pi^* = \operatorname{argmax}_\pi \mathbb{E}_{s_{t+1} \sim T(\cdot|s_t, a_t), a_{t+1} \sim \pi(\cdot|s_{t+1})} \left[\sum_{i=0}^{\tau} \gamma^i R(s_t, a_t)\right]$ for all s_0.

Based on the experience of interactions with the environment, the agent can build an world model that approximates the transition function $\hat{T} \approx T$ and the reward function $\hat{R} \approx R$, and use its predictions as an additional signal for policy learning. The accuracy of the world model is crucial for finding the optimal policy: an error-free model allows one to compute the optimal policy without interacting with the environment, but it requires a large number of steps for a deep exploration of the environment and data collection for the world model training. A less accurate model is easier to obtain, but erroneous predictions degrade the quality of the resulting policy, especially in the case of using multi-step forecasts when model errors tend to accumulate.

Of the many neural network architectures used as transition models, three categories can be distinguished: monolithic state models, structured models, and factorized models. The architecture of monolithic models does not take into account the nature of the environment. Examples of this type of model are autoencoders and variational autoencoders [Kingma and Welling, 2022] [Rezende *et al.*, 2014]. Structured models explicitly reflect the organization of the environment: an observation is transformed into a hidden state, divided into slots. The model is trained unsupervised to extract objects from the state and assign them to individual slots. An example of this type of model is a CSWM [Kipf *et al.*, 2020] model, which derives a factorized vector representation of a state using a convolutional encoder and predicts state transition using a graph neural network. Factorized models, unlike structured ones, take a factorized state as input. These models are trained to predict the transitions for individual factors without solving the problem of their extraction from the state. An example of a model in this category is a work [Biza *et al.*, 2022a] in which the transition model is also based on a graph neural network. Structured and factorized models perform better than monolithic models in environments where it is necessary to accurately predict the dynamics of individual objects.

In this paper, we propose an object-oriented model-based Q-learning algorithm inspired by TreeQN [Farquhar *et al.*, 2018] model. Following CSWM [Kipf *et al.*, 2020] we use a structured transition model. Our reward and state-value models are graph neural networks which should be consistent with the compositional structure of the environment. The proposed algorithm demonstrates high sample efficiency and outperforms TreeQN and model-free algorithm PPO [Schulman *et al.*, 2017] in tasks where individual dynamics of the objects is similar.

As related work in RL we consider [Watters *et al.*, 2019], where an object-oriented world model is used in a reinforcement learning setting, but only at the exploration phase. Another work in RL is [Ke *et al.*, 2021], where the authors used CSWM transition model in the task of approximating a reward function, but did not solve the problem of finding the optimal policy.

3 Object-Oriented Q-network

Figure 1 outline the high-level overview of the proposed framework (OOQN) predicting action values. A fully convolutional $Extractor$ takes an image-based observation s_t as input and produces feature maps $\{m_t^i\}_{i=1}^K$ (K - architectural parameter). We treat those feature maps as disjoint masks of individual objects presented in the image. A fully connected $Encoder$ shared across objects converts the masks into K lower-dimensional vector embeddings constituting a factored abstract state representation $\boldsymbol{z}_t$. An attention model is used to obtain a factored representation $\boldsymbol{a}_t$ of the given action a_t given the state $\boldsymbol{z}_t$. A GNN based transition model predicts shifts in object representations $\Delta\boldsymbol{z}$ after taking the action. The outputs of GNN based reward and state-values models are used to predict the value of the action a_t.

3.1 Action-attention model

An action-attention model $W : Z^K \times \mathcal{A} \to \mathbb{R}^K$ predicts action attention weights for each object for the given factored state $\boldsymbol{z}_t$ and action a_t. The weights $\boldsymbol{w} = (w^1, \ldots, w^K)$ are used to factorize the action: $\boldsymbol{a}_t = (a_t w^1, \ldots, a_t w^K)$, where a_t is the one-hot encoded action vector. The simplest approach, which we refer to as none-attention, is to use the original action a_t for all objects: $\boldsymbol{w} = (1, \ldots, 1)$. Another approach is to use GNN based attention model predicting attention weights: $W_{GNN} = GNN(\boldsymbol{z}_t, a_t)$. The other considered attention mechanisms are hard and soft attention.

Hard / Soft attention

In this approach intermediate attention weights $\tilde{\boldsymbol{w}}$ are obtained as the product of the outputs of MLPs $key : Z \to \mathbb{R}^D$ and $query : \mathcal{A} \to \mathbb{R}^D$: $\tilde{w}^i = key(z_t^i)^T query(a_t)$. Soft attention weights are obtained by applying $softmax$ function to intermediate weights: $\boldsymbol{w}_{soft} = softmax(\tilde{\boldsymbol{w}})$. In order to obtain hard attention weights we construct a one-hot vector corresponding to the maximum weight in $\tilde{\boldsymbol{w}}$: $\boldsymbol{w}_{hard} = one\_hot(argmax\ \tilde{\boldsymbol{w}})$. In hard attention approach we always associate an action only with one object.

3.2 Transition model

We approximate transition function using a graph neural network [Biza *et al.*, 2022b] with an edge model $edge_T$ and an node model $node_T$ which takes a factored state $\boldsymbol{z}_t =$

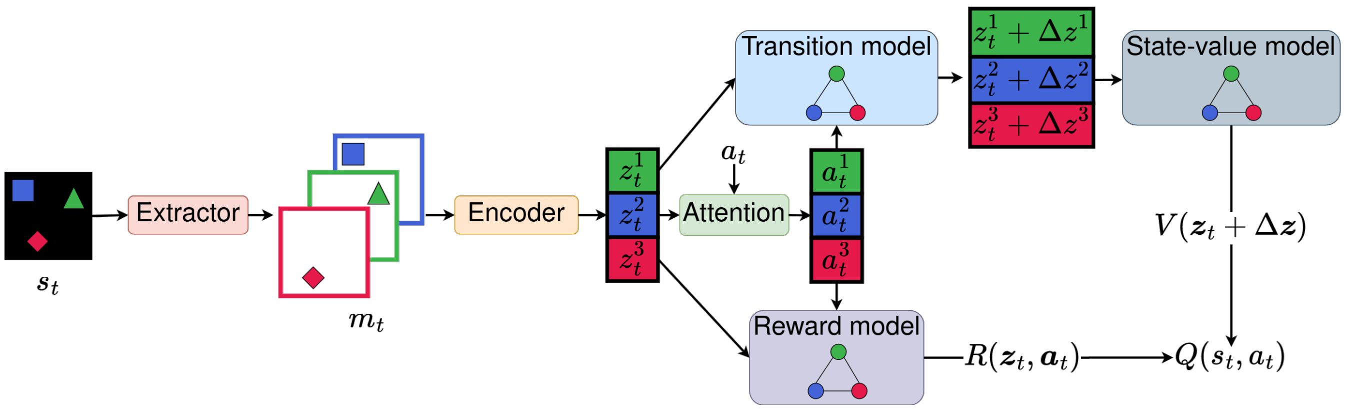

Figure 1: OOQN overview. Framework consists of the following blocks: a CNN-based *Extractor* module, a MLP-based *Encoder*, an action attention module and a GNN-based transition, reward and state-value models. The attention module, transition and reward models together form a world model. This diagram illustrates prediction of Q-values with one-step planning with the world model. In the case of multi-step planning on every step the output of the transition model is redirected to the input of the world model.

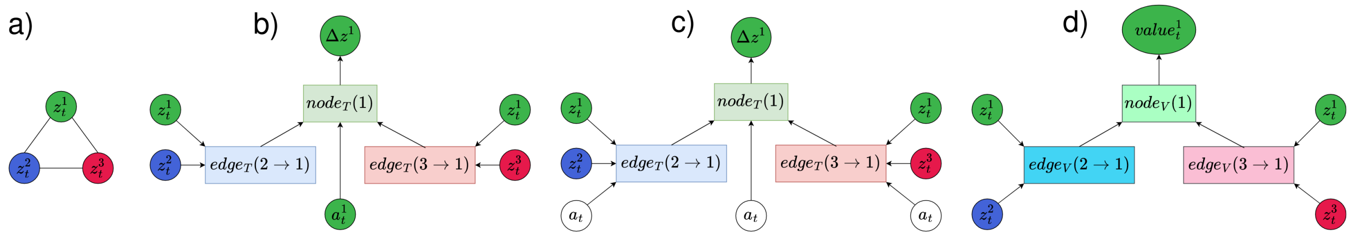

Figure 2: Overview of gnn-based transition model and state-value model. **a)** Representation of the state as a complete graph. **b)** Transition model: message-passing update scheme for the embedding of object 1 for hard / soft / GGN attention mechanisms. **c)** Transition model: message-passing update scheme for the embedding of object 1 for none-attention mechanisms. **d)** State-value model: message-passing update scheme for the state-value prediction for the object 1.

$(z_t^1, \ldots, z_t^K)$ and action $\boldsymbol{a}_t$ as input and predicts changes in factored states $\Delta\boldsymbol{z}$. When we do not use attention mechanisms to factorize actions the action is provided not only to the node model $node_T$ but also to the edge model $edge_T$ as shown in Figure 2. The factored representation of the next state is obtained via $\hat{\boldsymbol{z}}_{t+1} = \boldsymbol{z}_t + \Delta\boldsymbol{z}$.

$$\Delta z^i = node_T(z_t^i, a_t^i, \sum_{i \neq j} edge_T(z_t^i, z_t^j, a_t^i)) \tag{2}$$

The transition model and action-attention model are trained using the joint loss function $\mathcal{L}$ (3). Transition loss $\mathcal{L}_T$ depends on the attention type. The second term in $\mathcal{L}$ is a contrastive objective, which prevents the transition model from converging to a trivial solution and scores the current state against a state $\boldsymbol{z}^-$ sampled at random from the experience buffer. To improve the extracted masks and evaluate the impact of their quality on the efficiency of the policy learning algorithm, additional loss functions with coefficients α_{bce} and α_{min} are applied to the produced feature maps.

$$\begin{cases} \mathcal{L}_T^{hard/soft} = \sum_{j=1}^K w^j \|\hat{z}_{t+1}^j - z_{t+1}^j\|^2 \\ \mathcal{L}_T^{gnn/none} = \|\hat{\boldsymbol{z}}_{t+1} - \boldsymbol{z}_{t+1}\|^2 \\ \mathcal{L} \qquad = \mathcal{L}_T + \alpha_{contr} \max\left[0, \xi - \|\boldsymbol{z}_t^- - \boldsymbol{z}_t\|^2\right] \\ \qquad\quad + \alpha_{bce} BCE(\sum_{j=1}^K m_t^j, thresh(o_t)) \\ \qquad\quad + \alpha_{min} \sum_{i=1}^K \sum_{i<j}^K (\min[m_t^i, m_t^j])^2 \end{cases} \tag{3}$$

3.3 Reward model

The reward model is implemented using the same architecture as the transition model, but we use a sum for the reduction in the readout to obtain the full reward. The reward model is trained using the mean squared error loss function with environmental rewards r_t as target (4).

$$\begin{cases} \hat{R}^i(\boldsymbol{z}_t, \boldsymbol{a}_t) = node_R(z_t^i, a_t^i, \sum_{i \neq j} edge_R(z_t^i, z_t^j, a_t^i)) \\ \hat{R}(\boldsymbol{z}_t, \boldsymbol{a}_t) = \sum_{i=1}^K \hat{R}^i(\boldsymbol{z}_t, \boldsymbol{a}_t) \\ \mathcal{L}_R \qquad = (\hat{R}(\boldsymbol{z}_t, \boldsymbol{a}_t) - r_t)^2 \end{cases} \tag{4}$$

3.4 State-Value model

The state-value function is approximated using a graph neural network $\hat{V}$, which does not depend on actions in either the edge model $edge_V$ or the node model $node_V$. The action-value function $\hat{Q}$ estimated with Bellman equation is used to select the optimal action when the agent interacts with the environment. $\hat{V}$ is trained with an n-step Q-loss function $\mathcal{L}_{V-nstep}$ computed via $\hat{Q}$ values using a frozen version of

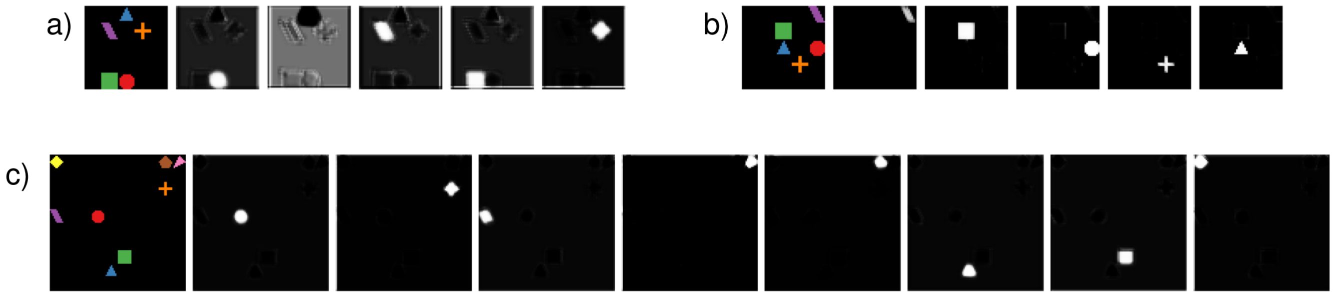

Figure 3: **a)** An example of observation in Shapes2D 5x5 with 5 objects and masks obtained with $Extractor$ module trained without auxiliary loss ($\alpha_{bce} = 0$, $\alpha_{min} = 0$). **b)** An example of observation in Shapes2D 5x5 with 5 objects and masks obtained with $Extractor$ module trained with auxiliary loss ($\alpha_{bce} = 0.025$, $\alpha_{min} = 0.1$). **c)** An example of observation in Shapes2D 10x10 and masks obtained with $Extractor$ module trained with auxiliary loss ($\alpha_{bce} = 0.000001$, $\alpha_{min} = 0$)

$\hat{V}$ as a target model.

$$\begin{cases} \hat{V}^i(\boldsymbol{z}_t, \boldsymbol{a}_t) = node_V(z_t^i, a_t^i, \sum_{i \neq j} edge_V(z_t^i, z_t^j)) \\ \hat{Q}^i(\boldsymbol{z}_t, \boldsymbol{a}_t) = \hat{R}^i(\boldsymbol{z}_t, \boldsymbol{a}_t) + \gamma \hat{V}^i(\hat{\boldsymbol{z}}_{t+1}) \\ \hat{Q}(\boldsymbol{z}_t, \boldsymbol{a}_t) = \sum_{i=1}^{K} \hat{Q}^i(\boldsymbol{z}_t, \boldsymbol{a}_t) \\ \mathcal{L}_{V-nstep} = \sum_{j=1}^{n} \Big(\sum_{k=1}^{j} \gamma^{j-k} r_{t+n-k} + \\ \qquad + \gamma^j \hat{Q}\big(\boldsymbol{z}_{t+n}, argmax_{\boldsymbol{a}} \hat{Q}(\boldsymbol{z}_{t+n}, \boldsymbol{a}, \theta^-), \theta^-\big) - \\ \qquad - \hat{Q}^i(\boldsymbol{z}_{t+n-j}, \boldsymbol{a}_{t+n-j}, \theta) \Big)^2 \end{cases} \tag{5}$$

4 Experiments

The efficiency of the proposed OOQN was evaluated in the modified Shapes2D environment [Kipf *et al.*, 2020]. We compare OOQN with a model-based TreeQN algorithms and the implementation [Raffin *et al.*, 2021] of the model-free PPO algorithm.

4.1 Environment

Shapes2D environment is a four-connected grid world, where objects are represented as figures of simple shapes. Examples of observations in the considered versions of the Shapes2D environment are shown in the Figure 3. One object — the cross is selected as a stationary target, the other objects are movable. We implemented three tasks in Shapes2D.

Navigation

The agent controls all movable objects. In one step, the agent can move an object to any adjacent cell that is free. The goal of the agent is to collide the controlled objects with the target object. Upon collision, the object disappears and the agent receives a reward $+1$. When an object collides with another movable object or field boundaries, the agent receives a reward -0.1 and positions of objects are not changed. For each step in the environment, the agent receives a reward -0.01. The episode ends if only the target object remains on the field. In the experiments, we use 5x5-sized environment with five objects and 10x10-sized environment with eight objects.

Pushing

The agent controls only one object — the red circle. The task is to push the other movable objects into the target. Upon collision, the object disappears and the agent receives a reward $+1$. When the red circle collides with the target object or field boundaries, the agent receives a reward -0.1. When the agent pushes a movable object into the field boundaries, the agent receives a reward -0.1. For each step in the environment, the agent receives a reward -0.01. The episode ends if only the target object and the controlled object remain on the field. In the experiments, we use 7x7-sized environment with five objects and 8x8-sized environment with six objects.

Pushing (no-agent)

The agent controls all movable objects as in Navigation task, but collisions between two movable objects are permitted: both objects move in the direction of motion. The agent is tasked to push another movable object into the target while controlling the current object. The pushed object disappears and the agent receives a reward $+1$ for such an action. When the currently controlled object collides with the target object or field boundaries, the agent receives a reward -0.1. When the agent pushes a movable object into the field boundaries, the agent receives a reward -0.1. For each step in the environment, the agent receives a reward -0.01. The episode ends if only the target object and one movable object remain on the field. In the experiments, we use 5x5-sized environment with five objects.

4.2 Training

Extractor

During the experiments in the Shapes2D environment, it was found that $Extractor$ module extracts only movable objects. To get around this limitation and obtain masks for all objects including the stationary cross, in all experiments the $Extractor$ module is pretrained on a set of trajectories collected by a uniform random policy in the version of Shapes2D where agent can move all objects. Examples of masks extracted by $Extractor$ are shown in the Figure (3). It is important to note that the use of additional loss functions during $Extractor$ training significantly improved the quality of the resulting masks.

World model

We consider offline setting for world model training and pretrain the attention model, reward model and transition model on the data set of trajectories collected with suboptimal policy. Namely, we use ϵ-greedy version of the trained PPO with $\epsilon = 0.5$ to collect the data. The training data sets sizes:

- Navigation 5x5: 100000 transitions
- Navigation 10x10: 200000 transitions
- Pushing (no-agent) 5x5: 150000 transitions

For Pushing tasks we collect trajectories with $\epsilon = 0.8$ and add to the data set manually generated examples of transitions between states, where three object <controlled object, movable object, target object > or <controlled object, movable object, movable object> are closely spaced in an area of size 3x3. Ratio between suboptimal policy data and manually generated data is 1:1. The training data sets sizes for Pushing tasks is 150000 transitions. The transition model, attention model and $Encoder$ are trained jointly using loss $\mathcal{L}$ (3) with frozen $Extractor$. The reward model is trained using loss $\mathcal{L}_{\mathcal{R}}$ (4) with frozen $Extractor$ and $Encoder$.

State-Value model

The state-value model is trained in online setting using loss $\mathcal{L}_{V-nstep}$ (5).

4.3 Results

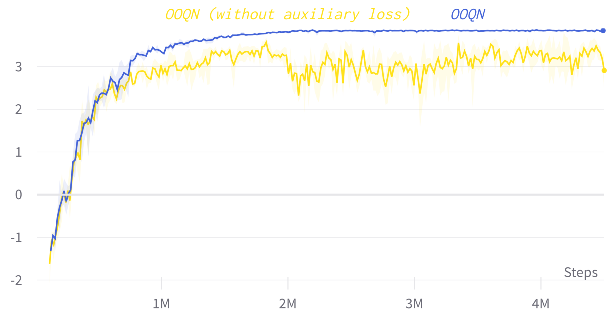

Figure 4: Moving average of episode return for OOQN models with hard attention in Navigation 5x5. The model utilizing $Extractor$ trained with auxiliary terms in loss function demonstrates better final performance. The curves are averaged over three seeds. Shaded areas indicate min-max ranges.

The Figure 4 demonstrates graphs of the dependence of the episode return on the number of steps in the Navigation 5x5 task for two variants of OOQN algorithm with hard attention. The variant of OOQN which employs $Extractor$ trained with auxiliary terms in loss function $\mathcal{L}$ (3) outperforms the variant without auxiliary terms. We concluded that the effectiveness OOQN correlates directly with the quality of the feature maps extracted by $Extractor$. Thus, all the reported results for OOQN are obtained with $Extractor$ trained with auxiliary terms in loss function.

We run experiments for OOQN with a number of variants of attention types: hard attention (OOQN-Hard), sofr attention (OOQN-Soft), GNN attention (OOQN-GNN) and none-attention (OOQN-None). Also we implement OOQN that uses ground-truth attention (OOQN-GT). Ground-truth attention weights are equal to one for objects which will be moved after the agent takes an action. Ground-truth attention weights corresponding to the other objects equals to zeros. OOQN variants are compared with the implementation of model-free algorithm PPO [Raffin *et al.*, 2021] and the model-based algorithm TreeQN [Farquhar *et al.*, 2018] with monolithic world model. The reported results are episode returns averaged by 100 episodes at two stages of training. We use mean and standard error over 3 runs. The results are summarized in Table 1 and Table 2. We report results of OOQN with two-step planning with world model, but did not notice significant differences compared to one-step planning.

We find that hard attention and ground-truth attention variants of OOQN consistently outperforms baselines in Navigation 5x5 task. The variant without attention outperforms PPO only at the final stage of training. In Pushing 7x7 the best baseline outperforms all OOQN models, but the results of the ground-truth variant is close to the PPO's result after 10 million steps. In Navigation 10x10 task all OOQN variants are more sample efficient than the best baseline at the beginning of the training, but only hard attention and ground-truth attention variants outperform PPO at the final stage of the training. In Pushing 8x8 task the only OOQN model which converges and outperforms one of the baselines after 20 million steps is the variant without attention. In Pushing (no-agent) 5x5 task the OOQN-GT and OOQN-None outperforms the best baseline at 5 million steps. At the final stage of training only the variant without attention is better than the baselines.

The hard attention of OOQN fails to converge in Pushing and Pushing (no-agent) tasks. We associate this behavior with the fact that hard attention always binds an action only to one object, which is insufficient in Pushing and Pushing (no-agent) tasks, where an action can affect several objects.

It could be seen that OOQN outperforms baselines in Navigation and Pushing (no-agent) tasks, which are symmetrical in terms of movable objects. Every object has the same transition function and the same reward is generated in states that differ only in the permutation of movable objects. We hypothesize that GNN based architecture of OOQN facilitates faster generalization and transfer of the learned object's dynamics to the other objects. It increases sample efficiency in tasks where this generalization is correct, but hinder policy learning in Pushing task, where the dynamics of the object controlled by agent differs from the dynamics of all the other movable objects.

The none-attention version OOQN-None demonstrates the most stable results across all test tasks, so we consider it as our best model at the current stage of our research.

5 Limitations

The current version of $Extractor$ module extracts only movable objects from the input image. This is an obstacle to the application of OOQN in arbitrary environments. However, as our experiments have shown, we can replace the $Extractor$ with any model that produces good quality object masks. For example, we can use a pre-trained segmentation model to get masks. As a further study, we consider the use of object-centric models like SLATE [Singh *et al.*, 2022].

Model	Episode return			
	Navigation 5x5		Pushing 7x7	
	3M	10M	3M	10M
OOQN-Hard	3.81 ± 0.04	3.81 ± 0.04	-0.92 ± 0.09	-0.91 ± 0.11
OOQN-Soft	3.30 ± 0.04	3.75 ± 0.03	0.16 ± 1.68	2.62 ± 0.12
OOQN-GT	3.84 ± 0.01	3.85 ± 0.00	2.31 ± 0.02	2.68 ± 0.00
OOQN-GNN	3.22 ± 0.05	3.76 ± 0.02	0.86 ± 1.40	2.64 ± 0.05
OOQN-None	3.74 ± 0.01	3.84 ± 0.00	1.85 ± 0.22	2.61 ± 0.05
PPO	$\mathbf{3.81 \pm 0.03}$	$\mathbf{3.81 \pm 0.03}$	$\mathbf{2.57 \pm 0.04}$	$\mathbf{2.69 \pm 0.00}$
TreeQN	3.28 ± 0.09	3.77 ± 0.04	1.14 ± 0.11	2.64 ± 0.02

Table 1: Episode return in Navigation 5x5 and Pushing 7x7 tasks at 3M and 10M environment steps of the state-value model training. The results of the best baseline are in bold. The results of our models, which are not worse than the best baseline, are underlined

Model	Episode return					
	Navigation 10x10		Pushing 8x8		Pushing (no-agent) 5x5	
	5M	20M	5M	20M	5M	20M
OOQN-Hard	6.39 ± 0.08	6.44 ± 0.06	-0.92 ± 0.09	-0.92 ± 0.70	-0.93 ± 0.11	-0.92 ± 0.13
OOQN-Soft	5.49 ± 0.35	6.16 ± 0.13	-0.91 ± 0.03	-0.13 ± 1.14	-1.43 ± 0.25	-0.53 ± 0.29
OOQN-GT	6.37 ± 0.08	6.45 ± 0.04	-1.08 ± 0.05	-1.04 ± 0.05	2.32 ± 0.16	2.57 ± 0.06
OOQN-GNN	2.58 ± 0.23	4.83 ± 0.22	-0.91 ± 0.08	1.66 ± 2.11	-1.05 ± 0.42	-0.09 ± 0.36
OOQN-None	4.78 ± 1.79	6.23 ± 0.28	-0.93 ± 0.03	3.30 ± 0.19	2.70 ± 0.02	2.78 ± 0.04
PPO	$\mathbf{0.76 \pm 0.15}$	$\mathbf{6.39 \pm 0.01}$	$\mathbf{2.66 \pm 0.83}$	$\mathbf{3.54 \pm 0.01}$	1.63 ± 0.16	2.63 ± 0.08
TreeQN	-0.27 ± 0.08	-0.23 ± 0.06	-0.38 ± 0.49	3.15 ± 0.18	$\mathbf{1.92 \pm 0.10}$	$\mathbf{2.77 \pm 0.02}$

Table 2: Episode return in Navigation 10x10, Pushing 8x8 and Pushing (no-agent) 5x5 tasks at 5M and 20M environment steps of the state-value model training. The results of the best baseline are in bold. The results of our models, which are not worse than the best baseline, are underlined

6 Conclusion and future work

In this paper we present the preliminary results of our investigation of object-oriented model based RL. We demonstrate that CSWM-like world models can be successfully used in model-based RL setting for policy learning in compositional environments. The results of experiments with OOQN in Shapes2D show that the GNN based object-oriented world model improves the efficiency of policy learning in tasks, where objects have the same dynamics. In this case variants of OOQN outperform the state-of-the-art model-free algorithm PPO and the model-based value-based algorithm TreeQN and scales better to more complex tasks (more objects, increased field size).

Our plans for future research include use of object-centric models instead of our fully convolutional $Extractor$. Currently our GNN-based models treat an environment state as a complete graph of object representations, while object interactions are essentially sparse. Taking this observation into account, we consider to implement state-graph sparsification as it was done in concurrent work [Goyal *et al.*, 2022].

References

[Biza *et al.*, 2022a] Ondrej Biza, Thomas Kipf, David Klee, Robert Platt, Jan-Willem van de Meent, and Lawson L.S. Wong. Factored world models for zero-shot generalization in robotic manipulation, 2022.

[Biza *et al.*, 2022b] Ondrej Biza, Robert Platt, Jan-Willem van de Meent, Lawson L.S. Wong, and Thomas Kipf. Binding actions to objects in world models. In *ICLR2022 Workshop on the Elements of Reasoning: Objects, Structure and Causality*, 2022.

[Diuk *et al.*, 2008] Carlos Diuk, Andre Cohen, and Michael L. Littman. An object-oriented representation for efficient reinforcement learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, page 240–247, New York, NY, USA, 2008. Association for Computing Machinery.

[Farquhar *et al.*, 2018] Gregory Farquhar, Tim Rocktaeschel, Maximilian Igl, and Shimon Whiteson. TreeQN and ATreec: Differentiable tree planning for deep reinforcement learning. In *International Conference on Learning Representations*, 2018.

[Goyal *et al.*, 2022] Anirudh Goyal, Aniket Didolkar, Nan Rosemary Ke, Charles Blundell, Philippe Beaudoin, Nicolas Heess, Michael Mozer, and Yoshua Bengio. Neural production systems: Learning rule-governed visual dynamics, 2022.

[Hafner *et al.*, 2023] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models, 2023.

[Ke *et al.*, 2021] Nan Rosemary Ke, Aniket Rajiv Didolkar, Sarthak Mittal, Anirudh Goyal, Guillaume Lajoie, Stefan Bauer, Danilo Jimenez Rezende, Michael Curtis Mozer, Yoshua Bengio, and Christopher Pal. Systematic evaluation of causal discovery in visual model based reinforcement learning, 2021.

[Keramati *et al.*, 2018] Ramtin Keramati, Jay Whang, Patrick Cho, and Emma Brunskill. Fast exploration with simplified models and approximately optimistic planning in model based reinforcement learning, 2018.

[Kingma and Welling, 2022] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022.

[Kipf *et al.*, 2020] Thomas Kipf, Elise van der Pol, and Max Welling. Contrastive learning of structured world models. In *International Conference on Learning Representations*, 2020.

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

[Raffin *et al.*, 2021] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[Rezende *et al.*, 2014] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 1278–1286, Bejing, China, 22–24 Jun 2014. PMLR.

[Santoro *et al.*, 2017] Adam Santoro, David Raposo, David G. T. Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A simple neural network module for relational reasoning, 2017.

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

[Singh *et al.*, 2022] Gautam Singh, Fei Deng, and Sungjin Ahn. Illiterate dall-e learns to compose, 2022.

[Stanić *et al.*, 2022] Aleksandar Stanić, Yujin Tang, David Ha, and Jürgen Schmidhuber. Learning to generalize with object-centric agents in the open world survival game crafter, 2022.

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

[Watters *et al.*, 2019] Nicholas Watters, Loic Matthey, Matko Bosnjak, Christopher P. Burgess, and Alexander Lerchner. Cobra: Data-efficient model-based rl through unsupervised object discovery and curiosity-driven exploration, 2019.

[Yoon *et al.*, 2023] Jaesik Yoon, Yi-Fu Wu, Heechul Bae, and Sungjin Ahn. An investigation into pre-training object-centric representations for reinforcement learning, 2023.

[Zholus *et al.*, 2022] Artem Zholus, Yaroslav Ivchenkov, and Aleksandr Panov. Factorized world models for learning causal relationships. In *ICLR2022 Workshop on the Elements of Reasoning: Objects, Structure and Causality*, 2022.