Exchangeable Input Representations for Reinforcement Learning

John Mern, Dorsa Sadigh, and Mykel Kochenderfer

Abstract—Although deep reinforcement learning has advanced significantly over the past several years, poor sample efficiency remains a major obstacle to its application to many robotics problems. Careful choice of input representation can help improve efficiency depending on the structure present in the problem. In this work, we present an attention-based method to project inputs into an efficient representation space that is invariant under changes to input ordering. We show that our proposed representation results in a search space that is a factor of \(m!\) smaller for inputs of \(m\) objects. We also show that our method is able to represent inputs over variable numbers of objects. Our experiments demonstrate improvements in sample efficiency for policy gradient methods on a variety of tasks. We show that our representation allows us to solve problems that are otherwise intractable when using naïve approaches.

I. INTRODUCTION

Deep reinforcement learning (RL) has achieved state-of-the-art performance across a variety of tasks [1], [2]. However, successful deep RL training requires large amounts of sample data. Even relatively simple tasks can require millions to hundreds of millions of samples. While this problem can be overcome in simulated domains where gathering data is relatively easy, it is often insurmountable for robotics tasks where gathering samples is much more challenging. Various learning methods have been proposed to improve sample efficiency, such as model-based learning and incorporation of Bayesian priors [3], [4].

The way that the input to an RL problem is represented can also impact the sample efficiency of a given learning approach. It is common to represent input states as concatenations of sub-state vectors of the objects within the environment. For example, a state in a robotic manipulation task may be represented as a set of the position and orientation vectors for all work pieces in the work space. In this case, we can refer to the sub-states of each piece as an object in the factored state. For many problems, an optimal policy should provide the same action for any permutation of the input set. In these cases, the objects are exchangeable. The key insight of this paper is that we can significantly improve efficiency by leveraging the exchangeable structure inherent in many robotics problems.

When inputs to neural networks are ordered sets, permutation invariance must be learned during training. To avoid this additional learning requirement, methods have been proposed to represent inputs in an order-invariant form. The Object Oriented Markov Decision Process (OO-MDP) framework [5] proposes such a method for exchangeable objects, however the presented methods are limited to discrete spaces with tabular representations. Approximately Optimal State Abstractions [6] proposes continuous approximations of the OO-MDP framework which extend to Q-learning problems with continuous input spaces. Object-Focused Q-learning [7] uses object classes to decompose the Q-function output space by interaction types, though it does not address exchangeability in the input space.

This paper builds upon the insights presented in these works to propose a method to map any set of exchangeable objects to an order invariant representation. We show that applying such a mapping can reduce the input space by a factor of up to \(m!\), where \(m\) is the number of exchangeable objects. We further empirically demonstrate that this abstraction improves learning sample efficiency through a series of deep RL experiments.

Deep Sets [8] proposes a permutation invariant abstraction method similar to the one proposed in this paper. Additionally they provide necessary and sufficient conditions for permutation invariant input mappings. Unlike our method, the method proposed produces a static mapping. That is, each input object is weighted equally in the invariant space regardless of value during the mapping.

In contrast, our method proposes a permutation-invariant attention mechanism for the input mapping. Attention mechanisms are used in various deep learning tasks to dynamically filter the input to a down-stream neural network to emphasize the most important parts of the original input [9], [10], [11]. We adapt a dot-product neural mechanism to efficiently apply dynamic attention [12]. Use of such an attention mechanism to generate input abstractions allows an optimal mapping to be learned during policy training.

We also propose a method to apply our algorithm to environments with multiple object classes, leveraging the exchangeability of objects within each class to further reduce the search space.

An additional challenge facing deep RL comes from environments with varying numbers of objects. As most neural network architectures require a fixed input size, ad-hoc approaches such as input zero-padding are often adopted [13]. This can result in difficulty in training and does not efficiently scale to larger input sizes. We show that the proposed attention mechanism is robust to varying numbers of input objects.

For review, the contributions presented in this paper are:

- An attention mechanism to map exchangeable object sets to permutation-invariant space
- Bounds on the sample efficiency improvement gained from permutation-invariant representation

The authors are with Stanford University, Stanford, California, 94305. Email: \{jmr91,dorsa,mykel\}@stanford.edu
is defined to be permutation invariant if and only if:

Definition II.2. If \( S \) is a set of objects \( \{s_1, \ldots, s_m\} \) of state \( S \), then a function \( f \) is defined to be permutation invariant if and only if:

\[
f(S) = f(\hat{S}) \quad \forall \hat{S} \in \Pi
\]

The problem this work seeks to address is to specify a permutation invariant function that can map ordered sets of exchangeable objects into an abstract state-space that

1) Retains all information necessary to solve the RL problem
2) Can map sets of varying size
3) Can be applied to problems with multiple object classes

II. PROBLEM STATEMENT

Deep RL is a class of methods to solve Markov decision processes (MDPs) using deep neural networks. An MDP is a way to frame a sequential decision problem that makes the Markovian assumption that the evolution of state depends only on the current state and action taken. An MDP is defined by the state-space \( \mathcal{S} \), action space \( \mathcal{A} \), a reward function \( r \), and transition model \( T \). A solution to an MDP is a policy \( \pi \) that maps states \( S_t \) to actions \( a_t \). Policies are closed-loop plans, in that they are reactive to the state of the environment.

It is common in MDPs to represent the state space as a set of objects, but the sample complexity of the solution can grow exponentially with the number of objects [14]. This growth can be reduced by treating as exchangeable the sub-state vectors corresponding to individual objects.

Definition II.1. We define a set of random variables \( \mathcal{X} \) to be exchangeable if and only if for any finite subset of \( \mathcal{X} \), and any of the permutations of this subset, i.e., \( \mathcal{X}, \mathcal{X}_\xi \subset \mathcal{X} \):

\[
P(\mathcal{X}) = P(\mathcal{X}_\xi),
\]

where \( \mathcal{X} \) is a finite set of random variables and \( \mathcal{X}_\xi \) is an arbitrary permutation of \( \mathcal{X} \).

Objects that may be treated as exchangeable are often referred to as being from the same class. Related to exchangeable variables are permutation invariant functions. Permutation invariant functions are functions that operate appropriately over sets of exchangeable variables. In other words, the output of the function on a set of exchangeable variables is invariant under permutations to the set ordering.

Definition II.2. If \( S \) is a state defined by a set of objects and \( \Pi \) is the set of all permutations on \( S \), then a function \( f \) is defined to be permutation invariant if and only if:

\[
f(S) = f(\hat{S}) \quad \forall \hat{S} \in \Pi
\]

The parameters of the networks in the attention mechanism are not known a priori and must be learned along with the parameters of the policy network. They may be learned by propagating the gradients defined by the main learning algorithm through the attention sub-graph. No additional loss-term or gradient definition is required.

Sample Efficiency. We can now define bounds on the sample efficiency benefits of an invariant mapping. Define a state space \( \mathcal{S} \) such that \( S = \{s_1, \ldots, s_m\} \) for \( S \in \mathcal{S} \), where \( m \) is...
the number of objects. Let each object \( s_i \) take on \( n \) unique values. If we represent the states as sets of objects in the RL algorithm, then the state-space size \( |S| \) can be calculated from the expression for \( m \) permutations of \( n \) values.

If all objects are exchangeable, there exists an abstraction that is permutation invariant. Since the order does not matter, the size of this abstract state \( |S| \) can then be calculated from the expression for \( m \) combinations of \( n \) values:

\[
|S| = \frac{n!}{(n-m)!}, \quad |\mathcal{S}| = \frac{n!}{m!(n-m)!} \tag{1}
\]

Using a permutation invariant representation allows us to reduce the input space that the RL algorithm may be required to search by a factor of \( \frac{S}{|\mathcal{S}|} = \frac{1}{m^m} \) compared to a naïve ordered set representation. This factor should be viewed as a bound on sample efficiency improvement, as learning the parameters of the attention mechanism may increase the number of steps required.

**Permutation Invariance.** It can be shown that it is necessary and sufficient for a mapping \( f \) to be invariant on all countable sets \( \mathcal{X} \) if and only if it can be decomposed using transformations \( \phi \) and \( \rho \), where \( \phi \) and \( \rho \) are any vector valued functions, to the form [8]:

\[
f(X) = \rho \left( \sum_{x \in \mathcal{X}} \phi(x) \right) \tag{2}
\]

We will now demonstrate that the proposed mechanism can be factored into the permutation invariant form of eq. (2). The variable names below correspond to the variable names in fig. 1:

\[
x_i^* = \sum_{j=1}^{m} z_j^{(j)} y_i^{(j)}
\]

\[
x_i^* = (\sum_{j=1}^{m} \rho y_i^{(j)})^{-1} \sum_{j=1}^{m} z_j^{(j)} e_i^{(j)}
\]

\[\rho \leftarrow (\sum_{j=1}^{m} \rho y_i^{(j)})^{-1}\]

\[x_i^* = \rho \sum_{j=1}^{m} \pi_{input}(x_i^{(j)}) e_i^{\phi(x_i^{(j)})}\]

\[\phi(x_i^{(j)}) \leftarrow \pi_{input}(x_i^{(j)}) e_i^{\phi(x_i^{(j)})}\]

\[x_i^* = \rho \sum_{j=1}^{m} \phi(x_i^{(j)})\]

Hence, the proposed mechanism is indeed permutation invariant. The neural networks can be treated as functions operating on single objects because the parameters are shared for all objects. Additionally, it can be seen that the projection also offers an effective method to handle variable numbers of input objects. As the number of individual state vectors \( m \) changes, the dimension of the output vector \( x_i^* \) remains constant as a result of the final summation operation.

**Multi-Class Attention.** In environments for which all objects are of the same class, the attention mechanism described can be applied directly. However, a slight extension is required for tasks with multiple object classes. Before defining that extension, we will provide a formal definition of class.

**Definition III.1.** If \( S \) is a state defined by a set of objects and \( S^0 \) and \( S^1 \) are disjoint subsets of \( S \), \( S^0 \) and \( S^1 \) define object classes if all objects \( s_i^0 \in S^0 \) are exchangeable and \( s_i^1 \in S^1 \) are exchangeable.

In our robotic manipulator example, object classes could be defined by work piece type such as nuts and bolts.

In problems with multiple object classes, only objects within a given class are exchangeable. Applying a single attention mechanism to all objects would not be appropriate. To address this, we can implement a separate attention sub-graph for each object class. The multi-class architecture is shown in fig. 2. In this we have a an attention sub-graph for each object class. Each sub-graph will maintain parameters that are shared for objects in that class only. The outputs from each sub-graph are concatenated into a final abstracted input vector. Note that this input vector will be ordered; however, this is appropriate as the abstract class vectors are not exchangeable.

**IV. Experiments**

We conducted a series of experiments to validate the effectiveness of our proposed abstraction. The first two tasks are simple MDPs in which a scavenger agent navigates a continuous two-dimensional world to find food particles. The third task is a convoy protection task with variable numbers of objects.

Scavenger Task 1 is designed to illustrate the effect of abstraction with a single class of objects. The goal of Scavenger Task 2 is to validate the effect of multi-class abstraction through the introduction of poison particles to
the environment. Renderings of both of these environments are shown in fig. 3.

**Task 1: Food Scavenger.** The food scavenger task can be defined by the standard MDP tuple. The state space contains vectors $s \in \mathbb{R}^{2m+2}$, where $m$ is the number of target objects. The vector contains the relative position of each food particle as well as the ego position of the agent. The action space contains velocity vectors in two dimensions $a \in \mathbb{R}^2$, that are limited to a maximum velocity magnitude such that $||a|| \leq d_{\text{max}}$. The agent receives a reward of +1.0 when reaching a food particle, and a reward of −0.05 for every time-step otherwise.

The state transition model is deterministic with agent position updated according to a constant-velocity Euler approximation at each time-step as shown below, where $\delta$ is the simulated time-step interval:

$$s_{t+1} \leftarrow s_t + \delta a_t$$

The agent is initialized at the center of the world at each episode and the food positions are sampled from a uniform distribution. The episode terminates upon reaching a food particle or when the number of time-steps exceeds a limit.

We trained a stochastic policy to solve this task with and without the proposed attention mechanism. The baseline policy trained without our attention mechanism received a vector concatenation of the object state set, with each object’s position in the vector remaining fixed through training. This is considered the standard RL approach. This same vector was used as an input to the attention mechanism, the output of which was used by the policy. All other training parameters were shared between the two experiments.

The policies were simple feed-forward neural networks, with four hidden layers of 64 units each. Leaky rectified linear units (ReLU) were used for all hidden layer activation functions. The network output parameters for a multivariate Gaussian distribution with diagonal covariance. The policy was trained using Proximal Policy Optimization (PPO) [15], with epoch batch sizes of 1,000 time steps and update batch-size of 256 steps. The policy ratio clipping parameter was set to 0.1 and no entropy bonus was provided. The reward signal used was advantage as calculated by the Generalized Advantage Estimation Lambda (GAE-\(\lambda\)) [16], with $\lambda = 0.9$ and $\gamma = 0.99$. All training was done using Tensorflow. Policies were trained for cases with varying numbers of objects, from two to five particles.

**Task 2: Food Scavenger with Poisons.** Scavenger Task 2 introduced poison particles in addition to the food particles (one poison for each food particle). If an agent reaches a poison particle, a reward of −1.0 is given and the episode terminates. As with the food particles, the initial positions of the poison particles are sampled from a uniform distribution. The remainder of the MDP tuple is identical to the base task. As before, we train our policy with a baseline and abstracted representation.

This extended task was developed to test the effect of abstraction over multiple classes. In our framework, poison objects are in a separate class from food objects. We introduce them to implement and test our multi-class attention method.

**Task 3: Convoy Protection.** A final experiment was conducted on a more difficult task in which the number of objects varies across episodes. The task requires a defender agent to protect a convoy that follows a predetermined path through a 2D environment. Attackers are spawned at the periphery of the environment during the episode, and the defender must block them while they attempt to approach the convoy. The environment was simulated in Anvel, a high-fidelity ground-vehicle simulation engine. We decomposed the solving of this problem using a hierarchical learning approach in which a primitive policy was trained a priori and used with fixed parameters over the training of a high-level policy. The primitive policy was trained to map the desired vehicle location change to vehicle commands (wheel orientation and throttle). Each time-step for the primitive policy MDP was set to 0.1 second of simulation world time. Our hierarchical approach resulted in a semi-MDP [17] problem, which would normally require adaptation of the reward function for the variable time-step lengths. In order to avoid this, we fixed execution of the primitive policy calls to ten time-steps.

Because the training of the high-level policy is of interest to this work, we will define the MDP in terms of its inputs and outputs. The state space is the space of vectors representing the state of each non-ego object in the environment $\{(x,y)\}$, where the status is a binary flag of whether or not the object is currently active. The full state also contains the state of the ego-vehicle $(x,y,\theta)$, where $\theta$ is the z-axis orientation of the vehicle. The action space contains vectors of the reachable changes in position. The three vehicle convoy was generated at a fixed point at the left side of the environment for each episode and traveled at a constant rate toward the right. The attackers were spawned at random times from one of eight spawn points at the periphery of the environment. The attackers approached the closest
convoy member and with maximum speed equal to twice the convoy speed.

The episode terminates when all convoy members either reach the goal position or are reached by an attacker. The agent receives a reward of $-1.0$ for each convoy member that is attacked and a reward of $+0.1$ for each attacker that is successfully blocked. As with previous experiments, we trained this policy with a baseline representation in which all object states were concatenated in fixed-order vectors and with a representation generated with our proposed methods.

V. RESULTS

![Fig. 4. Robotic Convoy Task Environment - Defender agent must protect the convoy as it travels across the environment by blocking the attackers from approaching](image)

![Fig. 5. Convoy Task Training Curve - “Abstracted Score” shows performance of agent with attention mechanism. “Baseline Score” shows performance of agent without attention mechanism](image)

The Scavenger 1 training curves for the baseline and abstracted policies are shown in fig. 6. A simple optimal policy was defined for the task (travel toward to closest particle) and the performance of this policy is also shown on the graphs.

The introduction of the permutation invariant representation allowed the RL algorithm to efficiently scale to tasks with more objects. While PPO failed to solve the problem in 1,000 training epochs using the naive representation for more than 2 targets, it was able to solve the problem for tasks up to 5 targets with the abstraction. A slight slowdown in the initial improvement can be seen in the abstracted cases likely due to the need to learn the parameters of the abstraction sub-graph.

For Scavenger 2, the same outcomes can be observed as in Scavenger 1. PPO fails to scale beyond tasks of 2 targets (four total objects) while learning on the naive representation. While using our proposed permutation invariant representation, the algorithm effectively scales to cases of up to 10 total objects in just 1,000 training epochs.

The abstraction was also tested for the more difficult convoy protection problem. This problem presented the additional challenge of accommodating an input space with a variable number of objects and a long time horizon. In this case, PPO was completely unable to learn using the naive representation over 10,000 training epochs containing 20M sample time steps. With the invariant abstraction, the algorithm was able to successfully learn a policy that achieved average performance within the $1 \sigma$ bound of the optimal policy in only 3,000 epochs (6M time steps). This demonstrates that even in complex tasks, a significant improvement in sample efficiency is gained with the abstraction.

VI. CONCLUSION

**Summary.** We presented an attention-based method to project sets of object state vectors into representations that leverage the exchangeability of objects. We showed that this attention mechanism is permutation invariant. In order to apply this mechanism across multiple object classes, we presented a simple extension, allowing the leveraging of class-dependent object exchangeability. The proposed mechanism was also shown to accommodate varying numbers of objects. We demonstrated the effectiveness of the approach to enhance the scalability of the PPO policy gradient learning algorithm on a set of demonstration problems. Our simple scavenger tasks highlighted the effect of ignoring exchangeability, as PPO was unable to scale with increasing numbers of objects. In addition, we demonstrated the effect of using the method in a more difficult hierarchical learning problem.

**Limitations and Future Work.** Though the abstraction improved the ability to learn with higher numbers of objects, it was observed to slow down learning on tasks with fewer objects. This is likely due to the need to learn the additional parameters of the attention network. For tasks with fewer objects, a static mapping such as the one proposed in Deep Sets may be more appropriate.
Another limitation of our work is the reliance on dot-product attention. While this method does enable dynamic weighting of objects during mapping, the importance is determined for each object without considering the other objects present. An attention mechanism that also considers interactions between objects in weighting could provide better performance.

Our work only addressed the effects of exchangeability in state representation, though similar investigation should be made into other parts of the MDP. In particular, extensions of the concept of interaction classes should be developed to leverage exchangeability in the action space and transition function. In our approach, we fixed several elements of the attention graph, such as the abstracted state dimension, as hyperparameters. The effect of these on learning rate and converged policy performance should be further investigated.

REFERENCES