Discovering Symmetries

Brennan Cottrell, Justice Roberts, Juan Rojas.

Abstract—**Deep Reinforcement Learning (DRL) is known to incur in high sample complexity often requiring millions of steps to optimize function representations. Recently, the use of group theory has enjoyed significant interest. The use of symmetry has been used both for data augmentation or to embed symmetry in neural net layers via equivariance. Nonetheless, both sets of approaches require specific symmetries to be set by hand. This work contributes a system that discovers symmetries for different environments and agents. The method consists of two steps: (i) an exploratory state that compiles state values as the agent spans the state and (ii) the identification of existing symmetries and their domain in the world.**

I. Introduction

How much can a single experience be leveraged in learning? This has been an important question in the field of deep learning. But in particular, even more so in the sample complex world of deep reinforcement learning where optimizing an agent can take hundreds of thousands or millions of steps [1]. Sample efficiency is particularly important with physical agents like robots where gathering interactions is even more costly.

Recently, works like [2]–[4] have all studied ways in which to use symmetry to speed up learning; however, their approaches assumed known *a priori* knowledge about these symmetries. In [2], given a transformation type, the method builds a basis of matrices to populate the layers of a neural network and lead to a weighted combination of basis components that can be trained. In [3], *a priori* $SE(2)$ transformations are used to build an equivariant Q-network architecture. In [5] rotational symmetries are used to data augment experiences from pick-and-place actions to speed up learning.

We propose an algorithm to discover symmetries across various environments and dimensions[1]. When such symmetries are discovered, they can later be used during training (either via data augmentation [1] or the conversion of neural network layers into equivariant systems [2]–[4], [6] to transform experienced trajectories allowing for increased sample efficiency. The algorithm works in a two step process: the acquisition of observations and symmetrical transformations identification. Observations that span the space are gathered in a buffer. Each of those observations is then transformed by a given transformation. Each transformation is given a confidence measure according to number of equivalences found between its transformed observation and the contents of the buffer.

† School of Engineering, Lipscomb University, Nashville, TN 37215. Corresponding author: juan.rojas@lipscomb.edu

[1]That is, worlds that contain different agents, workspaces, and bounds on state and action spaces. For example, one world could contain a 2D mobile agent, another could contain a 3D manipulator

II. Group Theory

The use of groups to encode symmetries is motivated by a need for a structure which exhibits the following properties: (i) the composition of symmetric operations is itself a symmetric operation such that the structure which contains the symmetries must be closed; (ii) there must be an action which represents taking no action; (iii) for every symmetric action, there must be a way to get back to the state in which no action has been taken. These properties of symmetry cleanly lead to the definition of a group.

More formally, a group is defined as a set G together with some operation $*$ such that the following four properties hold: (i) Closure: for some $a \in G$ and $b \in G$, $a * b \in G$; (ii) Associativity: for some $a \in G$, $b \in G$, and $c \in G$, $a*(b*c) = (a*b)*c$; (iii) Identity: there exists $e \in G$ which satisfies $a * e = a$ for all $a \in G$; and (iv) Inverse: for all $a \in G$ there exists an inverse a^{-1} which satisfies the equation $a * a^{-1} = e$ where e is the identity.

A subgroup of some group $(G, *)$ is a subset of G which is itself a group. For a group G and some subgroup H, a left coset is defined as gH for $g \in G$. For some group G and some set X which is acted on by G, an orbit of $x \in X$ is defined as $\{gx | g \in G\}$.

We then can define the set of $n \times n$ orthogonal matrices as a group under the operation of matrix multiplication. This group is called $O(n)$. This group contains a subgroup whose members are the orthogonal matrices of determinant 1. This group is called $SO(n)$.

III. Symmetry Discovery

We propose an algorithm to discover symmetries in an observation space. The algorithm works in a two step process: the acquisition of observations and symmetrical transformations identification.

A. Exploratory Data Acquisition

Before identifying symmetries, one must be aware of the range of plausible observations in a given environment. Such range is not usually obvious or trivial. The range of plausible observations is affected by both the constraints of the agent and the environment. A particular agent might be composed by a series of connected links, all of which have a limited range of motion. The environment might have a support surface (*i.e.* a table, the floor, etc.) that determines not only the space over which the agent can function but also the entities with which it interacts (i.e. objects in the world). There might also be other difficulties in collecting observation data, as in the case of obstructing elements in the environment. Put simply, the range of plausible observations needs to be extracted either deliberately from the start or progressively as an agent interacts with the environment.

Upon extracting observations we face the question of how to store such observations. Should each and every observation be stored? For small discrete spaces that might be a consideration, however, not so for the majority of cases. We hold that a well crafted subset can be collected and stored in a buffer instead. To enable symmetry discovery with this data, it must have a wide coverage of observations. Narrowly concentrated observations will fail to find many symmetries. Consider an environment with actions *Left, Right*, where *Left* steps an agent in $-x$ direction and *Right* steps the agent in the $+x$ direction by a finite distance x. If all stored observations originated from the *Right* action, the reflection defined by the bisecting axis would not be found[2].

In discussing "wide-coverage", we need to specify the coverage of what. Not all dimensions in an observation space consist of the same data type. Observation vectors are often multimodal and include diverse information like position, velocity, force, etc. We currently limit our buffer to spatial (Cartesian) point information ascribed to an agent's end-effector (*e.g.* the inverted pendulum's tip, a robot's finger, etc).

To have a buffer with a wide coverage we implement an *exploration actor* that attempts to uniformly cover the agent's continuous *action space*. The generic exploration actor can be applied to any environment if the number of action dimensions and lower and upper action value bounds are known. The actor partitions each dimension in an action space from the lower bound to the upper bound via a predetermined number of discrete steps and stores them in a buffer. Then the actor traverses through the the possible combinations of admissible action movements in each dimension via a nested loop. Namely, given a set of action dimensions (e.g. a_x, a_y, a_z), loop through the possible range of actions $[a_{i_{min}}, a_{i_{max}}]$. Every time the environment is reset, the actor selects the succeeding action command in the buffer. On time steps at which the environment was not reset, it randomly picks any of the one of the permutations.

B. Symmetry Identification

Using the data collected by the exploration actor, we then seek to identify valid environment-wide symmetry transformations in the observation space. We first generate a set of candidate transformations that span the space from a lower bound to an upper bound. Subsequently, to test if the provided transformation belongs to the built-in symmetry of the environment, we sample a random observation from the buffer, apply the transformation and check if the result exists within our data buffer. We repeat a finite number of times and return the number of success as a confidence measure.

More concretely, let D be the set of all valid observations, and $f(x)$ be some transformation function where $x \in \mathbb{R}^n$ and $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$. Then a symmetrical transformation is one that meets the requirement $f(x \in D), \forall x \in D$. Intuitively, this means the transformation does not leave the set; it maps valid observations to valid observations. In practice, we do not deal directly with D because of its size. When the state space is continuous, it would be infinite. Instead we work with a buffer B, where $B \subset D$. As stated earlier, it is important that the buffer B is uniformly distributed over the observation set D. Note that the incomplete nature of the buffer implies that if a transformation maps an observation outside the buffer, we cannot conclude that the transformation is not symmetrical. It is certainly possible the transformed observation is a valid observation but not present in the buffer. To this end, a confidence value is then generated based on the number of successful observation pairs that a transformation yielded. The method keeps all transformations that have a confidence greater than an empirically set parameter $\#\Delta \in [0, 1]$. In theory, the maximum confidence number would be attained according to: $(range - increment)/range$. For example, if you consider a 90 degree rotation across a span of 180 degrees, observations that lied in the first quadrant, could only find matches in the second quadrant, attaining at most success for half of the observations.

Equivalence checks across vector observations use an error tolerance ϵ. The value of ϵ has a significant impact on what transformations are identified as symmetrical. We elaborate further in Sec. IV-B.

The procedure described above was implemented as the $testTransformation$ function. The function takes any transformation as input and outputs a confidence value. It applies the transformation $symmetry\_num\_test$ number of times on randomly selected observations from B. We opted for random selection of observations to cover different regions of the data without having to transform every observation in the buffer. We test rotations at discrete steps. The step size is an adjustable parameter called $angle\_increment$. In 3D problems, each axis is tested separately. Reflections over axes are tested next. Any transformation with a confidence value less than or equal to Δ are discarded. The pseudo-code is presented below:

IV. Experiments and Results

A. Experiments

For our experiments, we need to specify 3 aspects: (i) the types of transformations we seek to discover in our environments, (ii) the step size of the transformations we test, and (iii) the confidence threshold to accept a symmetry. In this section, we also introduce our environments, the action bounds used during the exploratory phase,

1) Transformations: In this work, we consider transformations that belong to the orthogonal group $(O(2), O(3))$ and include rotations and reflections. We used a right handed coordinate system for every environment. Rotations are tested at discrete steps with distance $angle\_increment$ between them. We tested with $angle\_increment$ equal to 0.034 radians which is approximately 2 degrees. In 3D environments, rotation is attempted over every axis. In 2D environments,

[2]In this work, we assume we care to find symmetries across the entire workspace. This may not always be desirable. It is possible, given some information, we only care about some subspace of the environment. Though, we do not consider this case in our work.

Algorithm Symmetry Discovery

```
1: function TESTTRANSFORMATION ($T$)
2:     $works \leftarrow 0$
3:     for $iteration = 1, 2, \ldots symmetry\_num\_test$ do
4:         $a \leftarrow a$ random state from the buffer
5:         $b \leftarrow$ transformation of $a$ by $T$
6:         if $b$ in buffer then
7:             $works \leftarrow works + 1$
8:         end if
9:     end for
10:    return $works/symmetry\_num\_test$
11: end function
12: Fill buffer with exploration actor
13: $angle \leftarrow 0$
14: while $angle < 2\pi$ do
15:     $T \leftarrow$ transformation that is rotation by $angle$
16:     $T.confidence \leftarrow testTransformation(T)$
17:     $angle \leftarrow angle + angle\_increment$
18: end while
19: In 3D, repeat while $\forall$ rotational axes
20: call $testTransformation(T)$ for $x, y, z$ reflections
21: remove transformation with $T.confidence <= \Delta$
```

reflection over a given axis is identified as equivalence between the other value and its negative. For the y-axis, the reflection of (x, y) would be $(-x, y)$. In 3D environments, reflection is thought of as a mirroring over a plane. Mathematically, this is done by negating the dimension that is not utilized in forming the plane. The reflection of the point (x, y, z) over the plane xoz would be $(x, -y, z)$. As for the confidence threshold Δ we used a value of 0.5 since all kept transformation work on more observations than they do not.

2) Environments: In this work we used three Mujoco environments: Inverted Pendulum, Reacher, and FetchPush [7], [8] as shown in Fig. 1. All environments have continuous action and observation spaces. The Inverted Pendulum moves a cart in 2D and balances an attached pole. The Reacher is a two degrees of freedom (DoF) robot arm that is rewarded when its fingertip touches the goal. Note the z position is always 0 and is thus a 2D problem. The FetchPush consists of a 7 DoF robot manipulator that tries to push a block to a target position in 3D space.

As for observations abstractions, the Inverted Pendulum consists of an (x, y) point where x is the position of the cart and y is the cosine of the vertical angle of the pole. The Reacher, uses the fingertip position, which was derived by subtracting the fingertip distance to target from the target position; and for the FetchPush environment, given the gripper does not have an origin at $(0, 0, 0)$, we reset the gripper position to the origin to facilitate reflection and rotational transformations.

B. Results

In this section we quantify and illustrate identified symmetries. The algorithm identified symmetries in all three environment with properly set parameters.

For the Inverted Pendulum we set the buffer to 1000 observations, $symmetry\_num\_test$ to 100, and $epsilon$ to 0.1. The algorithm finds reflections over the y-axis as symmetrical. Additionally, it finds rotations with angles near 0 or 2π as these are trivial identity rotations. These results are displayed in Fig. 2. We repeated the experiment with a 20-fold larger error tolerance $\epsilon = 2$ and kept all other parameters the same. This tolerance yielded reflections over the x- and y-axis as symmetrical as well as continuous rotation symmetry. This result is incorrect since there is no x-axis reflection nor continuous rotation in this environment.

For the Reacher we set the buffer to 1000 observations, $symmetry\_num\_test$ to 100, and an ϵ of 0.1. Here, the algorithm identifies infinite rotational symmetries and orthogonal reflections as shown in Fig. 3. The method identifies the continuous range of the rotational symmetries inherent in the environment. As for reflections, we only consider what we define as orthogonal reflections; those that align with the vertical and horizontal axes. Otherwise, we would deal with an infinite count of them. We ran a another trial with a tolerance a tenth of the original $\epsilon = 0.01$ and keeping all other parameters the same.Here, the method finds no symmetrical transformations other than the identity rotation.

For the more complex FetchPush environment, we increased the observations in the buffer to 5000 and $symmetry\_num\_test$ to 250. We still used an ϵ of 0.1. The results for this run are shown in Fig. 4 and 5. Noise in the exploration actor and simulator means the algorithm can report different symmetries for the same parameters. For these parameters, the algorithm can reliably find x- and y-axis reflection symmetries but did not reliably find any continuous rotational symmetry. However, most of the tested rotations over the z-axis are symmetrical. There are only a few rotations by an angle over the z-axis that do not reach the confidence threshold. If the buffer size and $symmetry\_num\_test$ are increased to 7500 and 500 respectively, the algorithm can reliably report continuous rotations over the z-axis since all tested z-axis rotations were found to be symmetrical. The algorithm still finds the previously mentioned reflection symmetries for the new parameters. Reflection over the z-axis does not exist in this in this environment due to obstruction by the table. Our algorithm correctly does not label z-axis reflection as a symmetry when using either of the previously described parameters. If the algorithm is run with a lower ϵ of 0.01, the algorithm does not find any symmetries other than the identity rotation for both values for the buffer size and $symmetry\_num\_test$.

One of the most important parameters used by the algorithm is ϵ which controls the margin of error allowed for determining equality. We saw that if ϵ was set to a large enough value, the algorithm will report all transformations are symmetrical even if they are not. On the other end, too small of an ϵ means that no symmetries will be found due to the continuous nature of the environment. Careful tuning of ϵ is vital in obtaining correct results. Ablation studies for all our experimentation

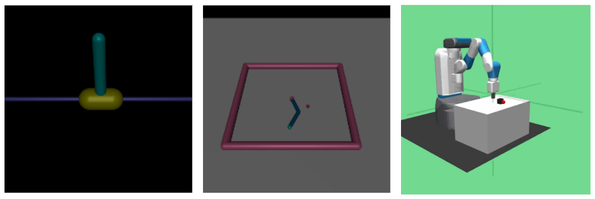

Fig. 1: Three Mujoco environments: (i) Inverted Pendulum, (ii) Reacher, and (iii) Fetch-Push.

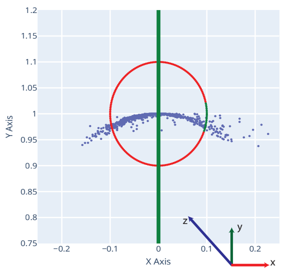

Fig. 2: Inverted Pendulum results with 1000 observations and $\epsilon = 0.1$. Blue points are observations. The green line is the discovered reflectional axis. The circle represents valid rotations. Green and red points indicate angles over which you have symmetrical and non symmetrical rotations.

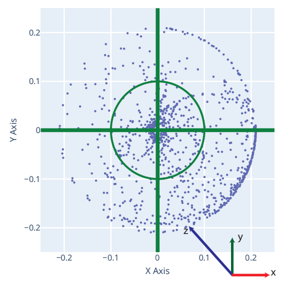

Fig. 3: Reacher environment results with 1000 observations, $symmetry\_num\_test = 100$ and $\epsilon = 0.1$. Notation explained in Fig. 2

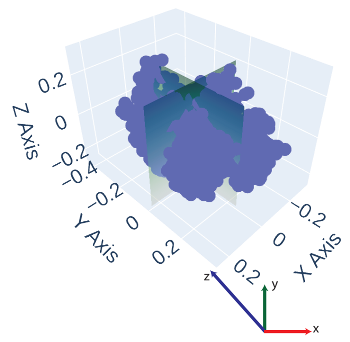

Fig. 4: Visualization of reflection symmetries from the Fetch-Push environment. A blue point represents one of 5000 observations. The green planes represent the found symmetry by reflecting over the corresponding axis.

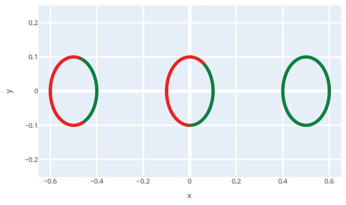

Fig. 5: Visualizations of rotation symmetry in the FetchPush environment. Each circle represents one rotation axis. They are ordered left to right: x, y, z. A green point means that a rotation over that axis by that angle is symmetrical. A red point means it is not symmetrical.

can be found at the following link.

V. Discussion

This method is as an initial step towards allowing systems to dynamically learn the natural symmetry of their environments as they interact with them. Such symmetries can then be used during policy learning steps to speed up optimization.

The method's ability to determine the correct symmetries with proper parameter setting has been demonstrated, though applying these learned symmetries to learning has yet to be demonstrated. Additionally, the algorithm provides a framework for testing if any transformation is symmetrical. Our formulation allows for easy expansion and testing of new arbitrary transformations.

The need for empirically set values is a major limitation of our work as discussed in Sec. IV-B. The improper setting of the parameters can lead to incorrect symmetries. Additionally, the current work struggles to identify symmetries at the edges of the stored observations given that transformations there lead to observations outside the range.

In the near future, we hope to consider additional transformations like translations and reflection across arbitrary axes or planes; include theoretical guarantees of finding existing symmetries, and applying it to robots in real-time to incrementally discover symmetries in probabilistic manners. Finally, we would like to extend the approach to image space.

VI. Conclusion

This paper presented a symmetry discover algorithm that worked across environments and dimensions. The algorithm discovers reflectional axes or planes and rotational ranges in 2D and 3D showing its ability to generalize across settings. Such algorithm has important potential to learn symmetries on the go and increase learning efficiency in DRL settings.

References

[1] Y. Lin, J. Huang, M. Zimmer, Y. Guan, J. Rojas, and P. Weng, "Invariant transform experience replay: Data augmentation for deep reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 5, pp. 6615–6622, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9158366/

[2] E. van der Pol, D. E. Worrall, H. van Hoof, F. A. Oliehoek, and M. Welling, "MDP homomorphic networks: Group symmetries in reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 2020-Decem, pp. 4199–4210, 2020.

[3] H. Huang, D. Wang, R. Walters, and R. Platt, "Equivariant Transporter Network," *Robotics: Science and Systems Foundation*, 2 2022. [Online]. Available: https://arxiv.org/abs/2202.09400v5

[4] D. Wang, R. Walters, X. Zhu, and R. Platt, "Equivariant Q Learning in Spatial Action Spaces," pp. 1713–1723, 1 2021. [Online]. Available: http://arxiv.org/abs/2110.15443

[5] D. Wang, R. Walters, and R. Platt, "SO(2)-EQUIVARIANT REINFORCEMENT LEARNING." [Online]. Available: https://pointw.github.io/equi_rl_page/.

[6] A. Zhou, T. Knowles, and C. Finn, "Meta-Learning Symmetries by Reparameterization," *arXiv*, 2020. [Online]. Available: http://arxiv.org/abs/2007.02933

[7] Gymnasium, "Mujoco Environments," 2023. [Online]. Available: https://gymnasium.farama.org/

[8] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," 2012, pp. 5026–5033.