Symmetry and structure in deep reinforcement learning

van der Pol, E.

Publication date
2023
Document Version
Final published version

Symmetry and Structure in Deep Reinforcement Learning

Elise van der Pol

Elise van der Pol

Symmetry and Structure in Deep Reinforcement Learning

Universiteit van Amsterdam

This book was typeset by the author using LaTeX and the Tufte ebook template. The cover was designed by the author in Inkscape, adapted from a generated image.

Printing: www.proefschriftmaken.nl.

ISBN: 978-94-6469-413-0

Symmetry and Structure in Deep Reinforcement Learning

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. dr. ir. P.P.C.C. Verbeek
ten overstaan van een door het College voor Promoties ingestelde commissie,
in het openbaar te verdedigen in de Agnietenkapel
op woensdag 12 juli 2023, te 16.00 uur

door Elise Esmeralda van der Pol
geboren te Haarlem

Promotiecommissie

Promotor:	prof. dr. M. Welling	Universiteit van Amsterdam
Copromotores:	dr. H.C. van Hoof	Universiteit van Amsterdam
	dr. F.A. Oliehoek	Technische Universiteit Delft
Overige leden:	prof. dr. M. de Rijke	Universiteit van Amsterdam
	prof. dr. E. Kanoulas	Universiteit van Amsterdam
	dr. ir. E.J. Bekkers	Universiteit van Amsterdam
	dr. D. Precup	McGill University
	prof. dr. J. Peters	Technische Universität Darmstadt

Faculteit der Natuurwetenschappen, Wiskunde en Informatica

Voor Bibi

and we pour bag after bag
of leaves on the lawn,

waiting for them to leap
onto the bare branches.

— *Matt Rasmussen*

Contents

1

Introduction

Symmetry and structure are everywhere in the world. When we walk, the movement of our right leg mirrors that of our left leg. When molecules are rotated, their molecular properties are unchanged. When we navigate to a destination, we take the connectivity of different road segments into account. When we talk, we can string words together to form completely novel sentences. In every day life, we use information about the symmetry and structure of our tasks to guide our decision making.

In Artificial Intelligence, symmetries and structure are also ubiquitous. Consider a robot that mirrors its left and right leg movements during locomotion, automated chip design, a drone swarm tracking wildlife movement, a bot playing Atari Pong where the top and bottom part of the screen are reflections of each other, molecular design, a computer player considering rotated board states in the game of Go, and autonomous vehicles switching from the right side of the road in the Netherlands to the left side of the road in the UK. These are all examples of tasks within AI that exhibit some form of symmetry or structure. Leveraging knowledge of inherent symmetry and structure is an important step towards building systems that scale.

Reinforcement learning is a fundamental field of study in Artificial Intelligence that encourages artificial agents to learn from positive and negative feedback signals, which we call rewards. By trial-and-error, the agent can learn to associate situations, actions, and feedback in order to improve its decisions. For example, we can give a robot a positive reward for walking fast and a negative reward for falling over. Similarly, we can give a computer player a positive reward for winning a game, and a negative reward for losing a game, or give a positive reward to an agent that proposes a particularly efficient chip design. Using concepts from the field of reinforcement learning, we can formalize the examples above in order to propose approaches that lead to good decision making from an agent. In *deep reinforcement learning*, an agent uses neural networks to decide on which action to take, where the neural networks are adapted to the task using the received reward signals. However, even tasks that require intelligence far below human capabilities can present issues to artificial decision makers. Consider any vision-based control system acting in the real world. The agent receives observations as camera input, and has to learn the best action to take. The number of possible observations is prohibitively large, and it is unlikely that the agent will encounter two states that are exactly the same. As such, we would like to the agent to be able to re-use experience from earlier states to take good decisions in unseen states with similar characteristics. For example, when deciding how to move its

left leg, the agent should mirror the movements it learned for moving its right leg.

The examples above are a few of the cases where symmetry and structure appear in reinforcement learning problems. These can be formalized by considering when taking an action in a state is equivalent to taking another action in another state. In this thesis, we will study how we can use symmetry and structure in reinforcement learning when it is known, and how we can extract it if it is not.

An agent should not learn what is already known. Whether knowledge is provided by a system designer as prior knowledge or obtained by the agent itself through generalization should depend on the context of the problem. By properly re-using knowledge, we can reduce the number of times the agent needs to interact with the world, an essential part of scaling to real world settings. In this thesis, we will particularly look at symmetry and structure in reinforcement learning. As such, our main research question is:

Main Research Question: **How can we incorporate and extract symmetry and structure in reinforcement learning?**

We first consider the case where there is an obvious symmetry in the problem we are trying to solve. Many problems exhibit symmetry, since symmetry is a natural part of the physical world. To illustrate, consider the classic pole balancing task. In pole balancing, the agent controls a cart that can move left or right. Her goal is to move the cart left and right to ensure that a pole standing upright on the cart does not fall down. It does not matter if the pole is currently leaning to the left, and the car is moving right, or the pole is leaning right and the cart is moving left. Both cases are mirrored versions of the same underlying situation. An agent moving east to reach a goal on the east, or north to reach a goal in the north, are also two very similar situations. These problems and those mentioned above have something in common: they exhibit equivalences between different pairs of states and actions. The beauty of state-action equivalence is that it allows us to consider similarity between taking one action in one state and taking another action in another state. This is what allows us to express *symmetry* in reinforcement learning problems. In Chapter 2 we consider the problem of constraining the class of neural network policies to only those that are symmetric under certain transformations, and answer the first research question:

Research Question 1: How can we leverage knowledge of symmetries in deep reinforcement learning?

Our main contribution in Chapter 2 is proposing MDP Homomorphic Networks, a class of neural networks that incorporate reinforcement learning symmetries into neural networks. This approach bridges deep reinforcement learning and equivariant networks, and shows a substantial improvement in data efficiency compared to unstructured baselines. Additionally, we propose a new method for constructing equivariant neural network weights.

After considering the case of symmetric reinforcement learning tasks, we investigate the use of similar methods in a more complex setting: that of distributed cooperative multi-agent systems. In such settings, the task at hand must be solved by agents taking local actions and communicating locally with each other. In Chapter 3 we answer the second research question:

Research Question 2. How can we leverage knowledge of global symmetries in distributed cooperative multi-agent systems?

A straightforward approach to this problem would be a naive application of single agent approaches to symmetry in reinforcement learning. However, such a method would prohibit us from using distributed execution methods. Instead, we propose an equivariant distributed policy network that allows the policy to be distributed at execution time, requiring only local communication and computation to ensure global symmetries. This approach results in improved data efficiency in symmetric cooperative multi-agent problems.

In the second part of this thesis, we consider that there is usually some underlying structure in the unstructured information we receive. For example, objects of the same type tend to behave similarly when force is applied to them. Similarly, there are dynamics underlying a simple pole balancing system. An agent may observe only feature vectors, and not the equations governing the system. Finally, the space of possible images of $3 \times 48 \times 48$ pixels is significantly larger than the number of states in a simple grid world. However, an agent does not a priori know the number of possible actual states. It only knows that it receives images of a certain size. In the second part of this thesis, we consider how to extract structure from interaction data. First, in Chapter 4, we consider the problem of learning representations of decision making problems that are *plannable*. By plannable, we mean

that if we run standard planning algorithms on the graph of learned representations, we get good decision making strategies for the original problem. In practice, this means that we want the dynamics of the transition function in the original problem to be mirrored by the dynamics of the latent transition function. Our 3rd research question is therefore:

Research Question 3. How can we learn representations of the world that capture the structure of the environment?

In Chapter 4, we take cues from the equivariance literature [34, 193, 187] and contrastive learning [135, 94] to learn the abstract graph underlying a decision making problem. Noting that the effect of an action in the decision making problem should be matched by the effect on the abstract graph, we learn representations of the original problem and show that they are indeed plannable for a variety of problems, including continuous state spaces, and generalizing to unseen objects and goals. This action-equivariant planning approach results in improved data efficiency compared to model-free baselines and reconstruction baselines.

In Chapter 5, we consider a further structure in the problem: if we have a set of objects that can be acted upon individually, each state is itself structured. If we can recover the individual objects and a factored transition function from pixel observations, this can improve prediction and generalization. As such, our 4th research question is:

Research Question 4. How can we learn representations of the world that capture the structure in individual states?

In Chapter 5 we show that it is possible to find object-oriented representations from pixels based on factored actions and a fixed dataset of interactions. We propose an approach that uses contrastive learning on environment interaction samples, resulting in a structured representation that leverages a graph neural network transition function. The approach we propose improves prediction performance in latent space compared to unstructured and reconstruction baselines.

In this thesis, we explore symmetry and structure in deep reinforcement learning. We leverage prior knowledge of symmetries in single agent and multi-agent systems in Chapters 2- 3, and extract structure from interactions with the world in Chapters 4-5. We will conclude in

Chapter 6 and provide suggestions for future work.

1.1 List of Publications

The following publications form the basis of this thesis:

- **Elise van der Pol**, Thomas Kipf, Frans A. Oliehoek, Max Welling (2020). "Plannable Approximations to MDP Homomorphisms: Equivariance under Actions." In: International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS) [170]. Chapter 4.

- **Elise van der Pol**, Daniel E. Worrall, Herke van Hoof, Frans A. Oliehoek, Max Welling (2020). "MDP Homomorphic Networks: Group Symmetries in Reinforcement Learning." In: Advances in Neural Information Processing Systems (NeurIPS) [173]. Chapter 2.

- **Elise van der Pol**, Herke van Hoof, Frans A. Oliehoek, Max Welling (2021). "Multi-Agent MDP Homomorphic Networks." In: International Conference on Learning Representations (ICLR) [172]. Chapter 3.

- Thomas Kipf, **Elise van der Pol**, Max Welling (2019). "Contrastive Learning of Structured World Models." In: International Conference on Learning Representations (ICLR) [94]. Chapter 5.

- Johannes Brandstetter, Rob Hesselink, **Elise van der Pol**, Erik J. Bekkers, Max Welling (2021). "Geometric and Physical Quantities improve E(3) Equivariant Message Passing." In: International Conference on Learning Representations (ICLR) [23]. Chapter 3.

I have contributed in all aspects to all first author publications listed. Max Welling, Frans Oliehoek, and Herke van Hoof provided supervision, guidance, insight, and technical advice. In "MDP Homomorphic Networks: Group Symmetries in Reinforcement Learning." Daniel Worrall proposed the Symmetrizer, which was implemented by us jointly. In "Contrastive Learning of Structured World Models." Thomas Kipf contributed in all aspects. I provided reinforcement learning insights, proposed the use of action factorization and the evaluation method, and ran baseline experiments. Figures and tables reproduced with permission. In "Geometric and Physical Quantities improve E(3) Equivariant Message Passing.", Johannes Brandstetter, Rob Hesselink, and Erik Bekkers contributed in all aspects. I provided the original architectural

idea and took an advisory role. Figures reproduced with permission.

I have further contributed to:

- Tejaswi Kasarla, Gertjan J. Burghouts, Max van Spengler, **Elise van der Pol**, Rita Cucchiara, Pascal Mettes (2022). "Maximum Class Separation as Inductive Bias in One Matrix." Accepted to: Neural Information Processing Systems (NeurIPS) [87].

- Darius Muglich, Christian Schroeder de Witt, **Elise van der Pol**, Shimon Whiteson, Jakob Foerster (2022). "Equivariant Networks for Zero-Shot Coordination." Accepted to: Neural Information Processing Systems (NeurIPS).

- **Elise van der Pol**, Ian Gemp, Yoram Bachrach, Richard Everett (2022). "Stochastic Parallelizable Eigengap Dilation for Large Graph Clustering". In: ICML 2022 Workshop on Topology, Algebra, and Geometry in Machine Learning [169].

- Pascal Mettes, **Elise van der Pol**, Cees G.M. Snoek (2019). "Hyperspherical Prototype Networks." In: Advances in Neural Information Processing Systems (NeurIPS) [120].

- Ondrej Biza, **Elise van der Pol**, Thomas Kipf (2021). "The Impact of Negative Sampling on Contrastive Structured World Models." In: ICML Workshop on Self-Supervised Learning for Reasoning and Perception [17].

- Laurens Weitkamp, **Elise van der Pol**, Zeynep Akata (2018). "Visual Rationalizations in Deep Reinforcement Learning for Atari Games." In: Benelux Conference on Artificial intelligence (BNAIC) [190].

- Frans A. Oliehoek, Rahul Savani, Jose Gallego-Posada, **Elise van der Pol**, Roderich Groß (2018). "Beyond Local Nash Equilibria for Adversarial Networks." In: Annual Machine Learning Conference of Belgium and the Netherlands (Benelearn) [134].

Part I

Symmetry

2

MDP Homomorphic Networks: Group Symmetries in Reinforcement Learning

2.1 Introduction

This part of the dissertation leverages knowledge of symmetries in single and multi-agent reinforcement learning problems. In this Chapter, we propose MDP homomorphic networks, which enforce symmetries in reinforcement learning problems. In the following chapter we take this approach beyond single-agent symmetries.

This chapter introduces MDP homomorphic networks for deep reinforcement learning. MDP homomorphic networks are neural networks that are equivariant under *symmetries* in the joint state-action space of an MDP. Current approaches to deep reinforcement learning do not usually exploit knowledge about such structure. By building this prior knowledge into policy and value networks using an equivariance constraint, we can reduce the size of the solution space. We specifically focus on group-structured symmetries (invertible transformations). Additionally, we introduce an easy method for constructing equivariant network layers numerically, so the system designer need not solve the equivariance constraints by hand, as is typically done. We construct MDP homomorphic MLPs and CNNs that are equivariant under ei-

ther a group of reflections or rotations. We show that such networks converge faster than unstructured baselines on CartPole, a grid world and Pong.

This chapter considers learning decision-making systems that exploit symmetries in the structure of the world. Deep reinforcement learning (DRL) is concerned with learning neural function approximators for decision making strategies. While DRL algorithms have been shown to solve complex, high-dimensional problems [156, 153, 125, 124], they are often used in problems with large state-action spaces, and thus require many samples before convergence. Many tasks exhibit symmetries, easily recognized by a designer of a reinforcement learning system. Consider the classic control task of balancing a pole on a cart. Balancing a pole that falls to the right requires an *equivalent*, but mirrored, strategy to one that falls to the left. See Figure 2.1. In this chapter, we exploit knowledge of such symmetries in the state-action space of Markov decision processes (MDPs) to reduce the size of the solution space.

We use the notion of *MDP homomorphisms* [143, 141] to formalize these symmetries. Intuitively, an MDP homomorphism is a map between MDPs, preserving the essential structure of the original MDP, while removing redundancies in the problem description, i.e., equivalent state-action pairs. The removal of these redundancies results in a smaller state-action space, upon which we may more easily build a policy. While earlier work has been concerned with discovering an MDP homomorphism for a given MDP [143, 141, 127, 142, 16, 170], we are instead concerned with how to construct deep policies, satisfying the MDP homomorphism. We call these models *MDP homomorphic networks*.

MDP homomorphic networks use experience from one state-action pair to improve the policy for all 'equivalent' pairs. See Section 2.2 for a definition. They do this by tying the weights for two states if they are equivalent under a transformation chosen by the designer, such as s and $L[s]$ in Figure 2.1.

Such weight-tying follows a similar principle to the use of convolutional networks [109], which are equivariant to translations of the input [34]. In particular, when equivalent state-action pairs can be related by an invertible transformation, which we refer to as *group-structured*, we show that the policy network belongs to the class of *group-equivariant neural networks* [34, 193]. Equivariant neural networks are a class of neural network, which have built-in symmetries [34,

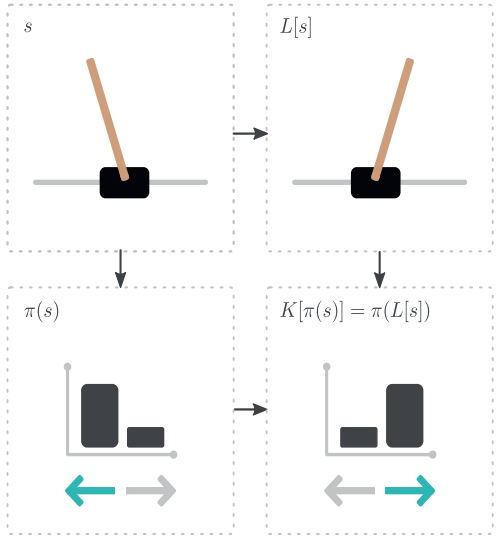


Figure 2.1: Example state-action space symmetry, where L is a horizontal reflection. The pairs $(s, \leftarrow)$ and $(L[s], \rightarrow)$ (and by extension $(s, \rightarrow)$ and $(L[s], \leftarrow)$) are symmetric under a horizontal flip. Constraining the set of policies to those where $\pi(s, \leftarrow) = \pi(L[s], \rightarrow)$ reduces the size of the solution space.

35, 193, 188, 186]. They are a generalization of convolutional neural networks—which exhibit translation symmetry—to transformation groups (group-structured equivariance) and transformation semigroups [194] (semigroup-structured equivariance). They have been shown to reduce sample complexity for classification tasks [193, 191] and also to be universal approximators of symmetric functions[1] [198]. We borrow from the literature on group equivariant networks to design policies that tie weights for state-action pairs given their equivalence classes, with the goal of reducing the number of samples needed to find good policies. Furthermore, we can use the MDP homomorphism property to design not just policy networks, but also value networks and even environment models. MDP homomorphic networks are agnostic to the type of model-free DRL algorithm, as long as an appropriate transformation on the output is given. In this chapter we focus on equivariant policy and invariant value networks. See Figure 2.1 for an example policy.

[1] Specifically group equivariant networks are universal approximators to functions symmetric under linear representations of compact groups.

An additional contribution of this chapter is a novel numerical way of finding equivariant layers for arbitrary transformation groups. The design of equivariant networks imposes a system of linear constraint equations on the linear/convolutional layers [35, 34, 193, 188]. Solving these equations has typically been done analytically by hand, which is a time-consuming and intricate process, barring rapid prototyping. Rather than requiring analytical derivation, our method only requires that the system designer specify input and output transformation groups of the form {state transformation, policy transformation}.

We provide Pytorch [138] implementations of our equivariant network layers, and implementations of the transformations used in this chapter. We also experimentally demonstrate that exploiting equivalences in MDPs leads to faster learning of policies for DRL.

Our contributions are two-fold:

- We draw a connection between MDP homomorphisms and group equivariant networks, proposing MDP homomorphic networks to exploit symmetries in decision-making problems;
- We introduce a numerical algorithm for the automated construction of equivariant layers.

2.2 Background

Here we outline the basics of the theory behind MDP homomorphisms and equivariance. We begin with a brief outline of the concepts of equivalence, invariance, and equivariance, followed by a review of the Markov decision process (MDP). We then review the MDP homomorphism, which builds a map between 'equivalent' MDPs.

Equivalence, Invariance, and Equivariance

Equivalence If a function $f : \mathcal{X} \to \mathcal{Y}$ maps two inputs $x, x' \in \mathcal{X}$ to the same value, that is $f(x) = f(x')$, then we say that x and x' are f-equivalent. For instance, two states s, s' leading to the same optimal value $V^*(s) = V^*(s')$ would be V^*-equivalent or *optimal value equivalent* [141]. An example of two optimal value equivalent states would be states s and $L[s]$ in the CartPole example of Figure 2.1. The set of all points f-equivalent to x is called the *equivalence class* of x.

Groups A group $(G, \cdot)$ is a set G, together with a binary operator $\cdot$, which is closed, i.e. $\forall g_1, g_2 \in G, g_1 \cdot g_2 \in G$. The group $(G, \cdot)$ obeys the group axioms:

- Associativity: For all $g_1, g_2, g_3 \in G$, $(g_1 \cdot g_2) \cdot g_3 = g_1 \cdot (g_2 \cdot g_3)$;
- Identity: There is an element $e \in G$ such that for all $g_1 \in G$, $e \cdot g_1 = g_1 \cdot e = g_1$;
- Invertibility: For all $g_1 \in G$, there exists an element $g_1^{-1} \in G$ such

that $g_1 \cdot g_1^{-1} = g_1^{-1} \cdot g_1 = e$.

Invariance and Symmetries Typically there exist very intuitive relationships between the points in an equivalence class. In the Cart-Pole example of Figure 2.1 this relationship is a horizontal flip about the vertical axis. This is formalized with the transformation operator $L_g : \mathcal{X} \to \mathcal{X}$, where $g \in G$ and G is a mathematical group. If L_g satisfies

$$f(x) = f(L_g[x]), \qquad \text{for all } g \in G, x \in \mathcal{X}, \tag{2.1}$$

then we say that f is *invariant* or *symmetric* to L_g and that $\{L_g\}_{g \in G}$ is a set of *symmetries* of f . We can see that for the invariance equation to be satisfied, it must be that L_g can only map x to points in its equivalence class. Note that in abstract algebra for L_g to be a true transformation operator, G must contain an identity operation; that is $L_g[x] = x$ for some g and all x. An interesting property of transformation operators which leave f invariant, is that they can be composed and still leave f invariant, so $L_g \circ L_h$ is also a symmetry of f for all $g, h \in G$. In abstract algebra, this property is known as a *semigroup property*. If L_g is always invertible, this is called a *group property*. In this work, we experiment with group-structured transformation operators. For more information, see [42]. Another helpful concept is that of *orbits*. If f is invariant to L_g, then it is invariant along the orbits of G. The orbit $\mathcal{O}_x$ of point x is the set of points reachable from x via transformation operator L_g:

$$\mathcal{O}_x \triangleq \{L_g[x] \in \mathcal{X} | g \in G\}. \tag{2.2}$$

Equivariance A related notion to invariance is *equivariance*. Given a transformation operator $L_g : \mathcal{X} \to \mathcal{X}$ and a mapping $f : \mathcal{X} \to \mathcal{Y}$, we say that f is equivariant [34, 193] to the transformation if there exists a second transformation operator $K_g : \mathcal{Y} \to \mathcal{Y}$ in the output space of f such that

$$K_g[f(x)] = f(L_g[x]), \qquad \text{for all } g \in G, x \in \mathcal{X}. \tag{2.3}$$

The operators L_g and K_g can be seen to describe the same transformation, but in different spaces. In fact, an equivariant map can be seen to map orbits to orbits. We also see that invariance is a special case of equivariance, if we set K_g to the identity operator for all g. Given L_g and K_g, we can solve for the collection of equivariant functions f satisfying the equivariance constraint. Moreover, for linear transformation operators and linear f a rich theory already exists in which f is referred to as an *intertwiner* [35]. In the equivariant deep learning literature, neural networks are built from interleaving intertwiners

and equivariant nonlinearities. As far as we are aware, most of these methods are hand-designed per pair of transformation operators, with the exception of [40]. In this chapter, we introduce a computational method to solve for intertwiners given a pair of transformation operators.

Markov Decision Processes

A Markov decision process (MDP) is a tuple $(\mathcal{S}, \mathcal{A}, R, T, \gamma)$, with *state space* $\mathcal{S}$, *action space* $\mathcal{A}$, *immediate reward function* $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, *transition function* $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}_{\geq 0}$, and *discount factor* $\gamma \in [0, 1]$. The goal of solving an MDP is to find a policy $\pi \in \Pi$, $\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}_{\geq 0}$ (written $\pi(a|s)$), where π normalizes to unity over the action space, that maximizes the expected return $R_t = \mathbb{E}_\pi[\sum_{k=0}^{T} \gamma^k r_{t+k+1}]$. The expected return from a state s under a policy π is given by the *value function* V^π. A related object is the Q-value Q^π, the expected return from a state s after taking action a under π. V^π and Q^π are governed by the well-known Bellman equations [15] (see Supplementary). In an MDP, optimal policies π^* attain an optimal value V^* and corresponding Q-value given by $V^*(s) = \max_{\pi \in \Pi} V^\pi(s)$ and $Q^*(s) = \max_{\pi \in \Pi} Q^\pi(s)$.

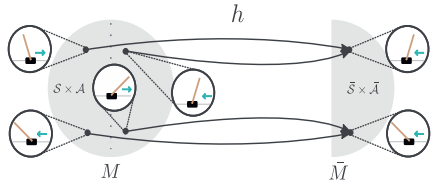


Figure 2.2: Example of a reduction in an MDP's state-action space under an MDP homomorphism h. Here 'equivalence' is represented by a reflection of the dynamics in the vertical axis. This equivalence class is encoded by h by mapping all equivalent state-action pairs to the same abstract state-actions.

MDP with Symmetries Symmetries can appear in MDPs. For instance, in Figure 2.2 CartPole has a reflection symmetry about the vertical axis. Here we define an *MDP with symmetries*. In an MDP with symmetries there is a set of transformations on the state-action space, which leaves the reward function and transition operator invariant. We define a state transformation and a state-dependent action transformation as $L_g : \mathcal{S} \to \mathcal{S}$ and $K_g^s : \mathcal{A} \to \mathcal{A}$ respectively. Invariance of the reward function and transition function is then characterized as

$$R(s, a) = R(L_g[s], K_g^s[a]) \qquad \text{for all } g \in G, s \in \mathcal{S}, a \in \mathcal{A} \tag{2.4}$$

$$T(s'|s, a) = T(L_g[s']|L_g[s], K_g^s[a]) \qquad \text{for all } g \in G, s \in \mathcal{S}, a \in \mathcal{A}. \tag{2.5}$$

Written like this, we see that in an MDP with symmetries the reward function and transition operator are invariant along orbits defined by the transformations (L_g, K_g^s).

MDP Homomorphisms MDPs with symmetries are closely related to MDP homomorphisms, as we explain below. First we define the lat-

ter. An *MDP homomorphism* h [143, 141] is a mapping from one MDP $M = (\mathcal{S}, \mathcal{A}, R, T, \gamma)$ to another $\bar{M} = (\bar{\mathcal{S}}, \bar{\mathcal{A}}, \bar{R}, \bar{T}, \gamma)$ defined by a surjective map from the state-action space $\mathcal{S} \times \mathcal{A}$ to an *abstract state-action space* $\bar{\mathcal{S}} \times \bar{\mathcal{A}}$. In particular, h consists of a tuple of surjective maps $(\sigma, \{\alpha_s | s \in \mathcal{S}\})$, where we have the state map $\sigma : \mathcal{S} \to \bar{\mathcal{S}}$ and the state-dependent action map $\alpha_s : \mathcal{A} \to \bar{\mathcal{A}}$. These maps are built to satisfy the following conditions

$$\bar{R}(\sigma(s), \alpha_s(a)) \triangleq R(s, a) \qquad \text{for all } s \in \mathcal{S}, a \in \mathcal{A}, \tag{2.6}$$

$$\bar{T}(\sigma(s') | \sigma(s), \alpha_s(a)) \triangleq \sum_{s'' \in \sigma^{-1}(s')} T(s'' | s, a) \qquad \text{for all } s, s' \in \mathcal{S}, a \in \mathcal{A}. \tag{2.7}$$

An exact MDP homomorphism provides a model equivalence abstraction [113]. Given an MDP homomorphism h, two state-action pairs (s, a) and (s', a') are called *h-equivalent* if $\sigma(s) = \sigma(s')$ and $\alpha_s(a) = \alpha_{s'}(a')$. Symmetries and MDP homomorphisms are connected in a natural way: If an MDP has symmetries L_g and K_g, the above equations 2.4 and 2.5 hold. This means that we can define a corresponding MDP homomorphism, which we define next.

Group-structured MDP Homomorphisms Specifically, for an MDP with symmetries, we can define an abstract state-action space, by mapping (s, a) pairs to (a representative point of) their equivalence class $(\sigma(s), \alpha_s(a))$. That is, state-action pairs and their transformed version are mapped to the same abstract state in the reduced MDP:

$$(\sigma(s), \alpha_s(a)) = \left(\sigma(L_g[s]), \alpha_{L_g[s]}(K_g^s[a])\right) \quad \forall g \in G, s \in \mathcal{S}, a \in \mathcal{A} \tag{2.8}$$

In this case, we call the resulting MDP homomorphism *group structured*. In other words, all the state-action pairs in an orbit defined by a group transformation are mapped to the same abstract state by a group-structured MDP homomorphism.

Optimal Value Equivalence and Lifted Policies h-equivalent state-action pairs share the same optimal Q-value and optimal value function [141]. There exists an abstract optimal Q-value $\bar{Q}^*$ and abstract optimal value function $\bar{V}^*$, such that $Q^*(s, a) = \bar{Q}^*(\sigma(s), \alpha_s(a))$ and $V^*(s) = \bar{V}^*(\sigma(s))$. This is known as *optimal value equivalence* [141]. Policies can thus be optimized in the simpler abstract MDP. The optimal abstract policy $\bar{\pi}(\bar{a} | \sigma(s))$ can then be pulled back to the original MDP using a procedure called *lifting* [2]. The lifted policy is given in Equation 2.9. A lifted optimal abstract policy is also an optimal policy

[2] Note that we use the terminology *lifting* to stay consistent with [141].

in the original MDP [141]. Note that while other lifted policies exist, we follow [141, 143] and choose the lifting that divides probability mass uniformly over the preimage:

$$\pi^{\uparrow}(a|s) \triangleq \frac{\bar{\pi}(\bar{a}|\sigma(s))}{|\{a \in \alpha_s^{-1}(\bar{a})\}|}, \qquad \text{for any } s \in \mathcal{S} \text{ and } a \in \alpha_s^{-1}(\bar{a}). \quad (2.9)$$

2.3 Method

The focus of the next section is on the design of *MDP homomorphic networks*—policy networks and value networks obeying the MDP homomorphism. In the first section of the method, we show that any policy network satisfying the MDP homomorphism property must be an equivariant neural network. In the second part of the method, we introduce a novel numerical technique for constructing group-equivariant networks, based on the transformation operators defining the equivalence state-action pairs under the MDP homomorphism.

Lifted Policies Are Invariant

Lifted policies in symmetric MDPs with group-structured symmetries are invariant under the group of symmetries. Consider the following: Take an MDP with symmetries defined by transformation operators (L_g, K_g^s) for $g \in G$. Now, if we take $s' = L_g[s]$ and $a' = K_g^s[a]$ for any $g \in G$, (s', a') and (s, a) are h-equivalent under the corresponding MDP homomorphism $h = (\sigma, \{\alpha_s | s \in \mathcal{S}\})$. So

$$\pi^{\uparrow}(a|s) = \frac{\bar{\pi}(\alpha_s(a)|\sigma(s))}{|\{a \in \alpha_s^{-1}(\bar{a})\}|} = \frac{\bar{\pi}(\alpha_{s'}(a')|\sigma(s'))}{|\{a' \in \alpha_{s'}^{-1}(\bar{a})\}|} = \pi^{\uparrow}(a'|s'), \quad (2.10)$$

for all $s \in \mathcal{S}, a \in \mathcal{A}$ and $g \in G$. In the first equality we have used the definition of the lifted policy. In the second equality, we have used the definition of h-equivalent state-action pairs, where $\sigma(s) = \sigma(L_g(s))$ and $\alpha_s(a) = \alpha_{s'}(a')$. In the third equality, we have reused the definition of the lifted policy. Thus we see that, written in this way, the lifted policy is invariant under state-action transformations (L_g, K_g^s). This equation is very general and applies for all group-structured state-action transformations. For a finite action space, this statement of invariance can be re-expressed as a statement of equivariance, by considering the vectorized policy.

Invariant Policies On Finite Action Spaces Are Equivariant Vectorized Policies For convenience we introduce a vector of probabilities

for each of the discrete actions under the policy

$$\boldsymbol{\pi}(s) \triangleq \Big[\pi(a_1|s), \quad \pi(a_2|s), \quad ..., \quad \pi(a_N|s)\Big]^\top, \tag{2.11}$$

where $a_1, ..., a_N$ are the N possible discrete actions in action space $\mathcal{A}$. The action transformation K_g^s maps actions to actions invertibly. Thus applying an action transformation to the vectorized policy permutes the elements. We write the corresponding permutation matrix as $\mathbf{K}_g$. Note that

$$\mathbf{K}_g^{-1}\boldsymbol{\pi}(s) \triangleq \Big[\pi(K_g^s[a_1]|s), \quad \pi(K_g^s[a_2]|s), \quad ..., \quad \pi(K_g^s[a_N]|s)\Big]^\top, \tag{2.12}$$

where writing the inverse $\mathbf{K}_g^{-1}$ instead of $\mathbf{K}_g$ is required to maintain the property $\mathbf{K}_g\mathbf{K}_h = \mathbf{K}_{gh}$. The invariance of the lifted policy can then be written as $\boldsymbol{\pi}^\uparrow(s) = \mathbf{K}_g^{-1}\boldsymbol{\pi}^\uparrow(L_g[s])$, which can be rearranged to the equivariance equation

$$\mathbf{K}_g\boldsymbol{\pi}^\uparrow(s) = \boldsymbol{\pi}^\uparrow(L_g[s]) \qquad \text{for all } g \in G, s \in \mathcal{S}, a \in \mathcal{A}. \tag{2.13}$$

This equation shows that the lifted policy must satisfy an equivariance constraint. In deep learning, this has already been well-explored in the context of supervised learning [34, 35, 193, 194, 188]. Next, we present a novel way to construct such networks.

Building MDP Homomorphic Networks

Our goal is to build neural networks that follow Eq. 2.13; that is, we wish to find neural networks that are *equivariant* under a set of state and policy transformations. Equivariant networks are common in supervised learning [34, 35, 193, 194, 188, 186]. For instance, in semantic segmentation shifts and rotations of the input image result in shifts and rotations in the segmentation. A neural network consisting of only equivariant layers and non-linearities is equivariant as a whole, too[3] [34]. Thus, once we know how to build a single equivariant layer, we can simply stack such layers together. Note that this is true regardless of the representation of the group, i.e. this works for spatial transformations of the input, feature map permutations in intermediate layers, and policy transformations in the output layer. For the experiments presented in this chapter, we use the same group representations for the intermediate layers as for the output, i.e. permutations. For finite groups, such as cyclic groups or permutations, pointwise nonlinearities preserve equivariance [34], allowing the use of e.g. rectified linear units without losing equivariance.

[3] See Appendix 2.B for more details.

In the past, learnable equivariant layers were designed by hand for each transformation group individually [34, 35, 193, 194, 191, 188, 186].

This is time-consuming and laborious. Here we present a novel way to build learnable linear layers that satisfy equivariance automatically.

Equivariant Layers We begin with a single linear layer $\mathbf{z}' = \mathbf{W}\mathbf{z} + \mathbf{b}$, where $\mathbf{W} \in \mathbb{R}^{D_{\text{out}} \times D_{\text{in}}}$ and $\mathbf{b} \in \mathbb{R}^{D_{\text{out}}}$ is a bias. To simplify the math, we merge the bias into the weights so $\mathbf{W} \mapsto [\mathbf{W}, \mathbf{b}]$ and $\mathbf{z} \mapsto [\mathbf{z}, 1]^\top$. We denote the space of the augmented weights as $\mathcal{W}_{\text{total}}$. For a given pair of linear group transformation operators in matrix form $(\mathbf{L}_g, \mathbf{K}_g)$, where $\mathbf{L}_g$ is the input transformation and $\mathbf{K}_g$ is the output transformation, we then have to solve the equation

$$\mathbf{K}_g \mathbf{W}\mathbf{z} = \mathbf{W}\mathbf{L}_g\mathbf{z}, \qquad \text{for all } g \in G, \mathbf{z} \in \mathbb{R}^{D_{\text{in}}+1}. \tag{2.14}$$

Since this equation is true for all $\mathbf{z}$ we can in fact drop $\mathbf{z}$ entirely. Our task now is to find all weights $\mathbf{W}$ which satisfy Equation 2.14. We label this space of equivariant weights as $\mathcal{W}$, defined as

$$\mathcal{W} \triangleq \{\mathbf{W} \in \mathcal{W}_{\text{total}} \mid \mathbf{K}_g \mathbf{W} = \mathbf{W}\mathbf{L}_g, \text{ for all } g \in G\}, \tag{2.15}$$

again noting that we have dropped $\mathbf{z}$. To find the space $\mathcal{W}$ notice that for each $g \in G$ the constraint $\mathbf{K}_g\mathbf{W} = \mathbf{W}\mathbf{L}_g$ is in fact linear in $\mathbf{W}$. Thus, to find $\mathcal{W}$ we need to solve a set of linear equations in $\mathbf{W}$. For this we introduce a construction, which we call a *symmetrizer* $S(\mathbf{W})$. The symmetrizer is

$$S(\mathbf{W}) \triangleq \frac{1}{|G|} \sum_{g \in G} \mathbf{K}_g^{-1} \mathbf{W} \mathbf{L}_g. \tag{2.16}$$

S has three important properties, of which proofs are provided in Appendix A. First, $S(\mathbf{W})$ is *symmetric* ($S(\mathbf{W}) \in \mathcal{W}$). Second, S *fixes* any symmetric $\mathbf{W}$: ($\mathbf{W} \in \mathcal{W} \implies S(\mathbf{W}) = \mathbf{W}$). Third, S is idempotent, $S(S(\mathbf{W})) = S(\mathbf{W})$. These properties show that S projects arbitrary $\mathbf{W} \in \mathcal{W}_{\text{total}}$ to the equivariant subspace $\mathcal{W}$.

Since $\mathcal{W}$ is the solution set for a set of simultaneous linear equations, $\mathcal{W}$ is a linear subspace of the space of all possible weights

---

Algorithm 1: Equivariant layer construction

1: Sample N weight matrices $\mathbf{W}_1, \mathbf{W}_2, ..., \mathbf{W}_N \sim \mathcal{N}(\mathbf{W}; \mathbf{0}, \mathbf{I})$ for $N \geq \dim(\mathcal{W}_{\text{total}})$
2: Symmetrize samples: $\bar{\mathbf{W}}_i = S(\mathbf{W}_i)$ for $i = 1, ..., N$
3: Vectorize samples and stack as $\bar{\mathbf{W}} = [\text{vec}(\bar{\mathbf{W}}_1), \text{vec}(\bar{\mathbf{W}}_2), ...]$
4: Apply SVD: $\bar{\mathbf{W}} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$
5: Keep first $r = \text{rank}(\bar{\mathbf{W}})$ right-singular vectors (columns of $\mathbf{V}$) and unvectorize to shape of $\mathbf{W}_i$

$\mathcal{W}_{\text{total}}$. Thus each $\mathbf{W} \in \mathcal{W}$ can be parametrized as a linear combination of basis weights $\{\mathbf{V}_i\}_{i=1}^r$, where r is the rank of the subspace and $\text{span}(\{\mathbf{V}_i\}_{i=1}^r) = \mathcal{W}$. To find as basis for $\mathbf{W}$, we take a Gram-Schmidt orthogonalization approach. We first sample weights in the total space $\mathcal{W}_{\text{total}}$ and then project them into the equivariant subspace with the symmetrizer. We do this for multiple weight matrices, which we then stack and feed through a singular value decomposition to find a basis for the equivariant space. This procedure is outlined in Algorithm 1. Any equivariant layer can then be written as a linear combination of bases

$$\mathbf{W} = \sum_{i=1}^{r} c_i \mathbf{V}_i, \tag{2.17}$$

where the c_i's are learnable scalar coefficients, r is the rank of the equivariant space, and the matrices $\mathbf{V}_i$ are the basis vectors, formed from the reshaped right-singular vectors in the SVD. An example is shown in Figure 2.3. To run this procedure, all that is needed are the

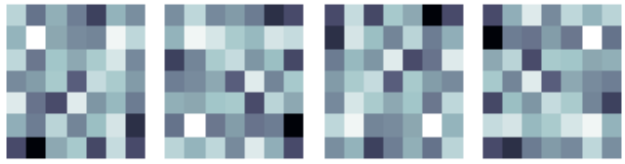


Figure 2.3: Example of 4-way rotationally symmetric filters.

transformation operators $\mathbf{L}_g$ and $\mathbf{K}_g$. Note we do not need to know the explicit transformation matrices, but just to be able to perform the mappings $\mathbf{W} \mapsto \mathbf{W}\mathbf{L}_g$ and $\mathbf{W} \mapsto \mathbf{K}_g^{-1}\mathbf{W}$. For instance, some matrix $\mathbf{L}_g$ rotates an image patch, but we could equally implement $\mathbf{W}\mathbf{L}_g$ using a built-in rotation function. Code is available [4].

[4] https://github.com/ElisevanderPol/symmetrizer/

2.4 Experiments

We evaluated three flavors of MDP homomorphic network—an MLP, a CNN, and an equivariant feature extractor—on three RL tasks that exhibit group symmetry: CartPole, a grid world, and Pong. We use RLPYT [159] for the algorithms. Hyperparameters (and the range considered), architectures, and group implementation details are in the Supplementary Material. Code is available [5].

[5] https://github.com/ElisevanderPol/mdp-homomorphic-networks

Environment		Space	Transformations
CartPole	$\mathcal{S}$	$(x, \theta, \dot{x}, \dot{\theta})$	$(x, \theta, \dot{x}, \dot{\theta}), (-x, -\theta, -\dot{x}, -\dot{\theta})$
	$\mathcal{A}$	$(\leftarrow, \rightarrow)$	$(\leftarrow, \rightarrow), (\rightarrow, \leftarrow)$
Grid World	$\mathcal{S}$	$\{0,1\}^{21\times 21}$	Identity, $\curvearrowright 90°$, $\curvearrowright 180°$, $\curvearrowright 270°$
	$\mathcal{A}$	$(\varnothing, \uparrow, \rightarrow, \downarrow, \leftarrow)$	$(\varnothing, \uparrow, \rightarrow, \downarrow, \leftarrow), (\varnothing, \rightarrow, \downarrow, \leftarrow, \uparrow), (\varnothing, \downarrow, \leftarrow, \uparrow, \rightarrow), (\varnothing, \leftarrow, \uparrow, \rightarrow, \downarrow)$
Pong	$\mathcal{S}$	$\{0, ..., 255\}^{4\times 80\times 80}$	Identity, reflect
	$\mathcal{A}$	$(\varnothing, \varnothing, \uparrow, \downarrow, \uparrow, \downarrow)$	$(\varnothing, \varnothing, \uparrow, \downarrow, \uparrow, \downarrow), (\varnothing, \varnothing, \downarrow, \uparrow, \downarrow, \uparrow)$

Table 2.1: Environments and Symmetries: We showcase a visual guide of the state and action spaces for each environment along with the effect of the transformations. Note, the symbols should not be taken to be hard mathematical statements, they are merely a visual guide for communication.

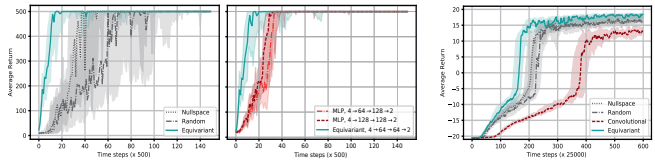
Environments

For each environment we show $\mathcal{S}$ and $\mathcal{A}$ with respective representations of the group transformations.

CartPole In the classic pole balancing task [10], we used a two-element group of reflections about the y-axis. We used OpenAI's Cartpole-v1 [24] implementation, which has a 4-dimensional observation vector: (cart position x, pole angle θ, cart velocity $\dot{x}$, pole velocity $\dot{\theta}$). The (discrete) action space consists of applying a force left and right $(\leftarrow, \rightarrow)$. We chose this example for its simple symmetries.

Grid world We evaluated on a toroidal 7-by-7 predator-prey grid world with agent-centered coordinates. The prey and predator are randomly placed at the start of each episode, lasting a maximum of 100 time steps. The agent's goal is to catch the prey, which takes a step in a random compass direction with probability 0.15 and stands still otherwise. Upon catching the prey, the agent receives a reward of +1, and -0.1 otherwise. The observation is a 21×21 binary image identifying the position of the agent in the center and the prey in relative coordinates. See Figure 2.6a. This environment was chosen due to its four-fold rotational symmetry.

Pong We evaluated on the RLPYT [159] implementation of Pong. In our experiments, the observation consisted of the 4 last observed frames, with upper and lower margins cut off and downscaled to an 80×80 grayscale image. In this setting, there is a flip symmetry over the horizontal axis: if we flip the observations, the up and down actions also flip. A curious artifact of Pong is that it has duplicate (up, down) actions, which means that to simplify matters, we mask out the policy values for the second pair of (up, down) actions. We chose Pong because of its higher dimensional state space. Finally, for Pong we additionally compare to two data augmentation baselines: stochastic data augmentation, where for each state, action pair we randomly transform them or not before feeding them to the network, and the second an equivariant version of [99] and similar to [156], where both state and transformed state are input to the network. The output of the transformed state is appropriately transformed, and both policies are averaged.



(a) Cartpole-v1: Bases (b) Cartpole-v1: MLPs (c) Pong

Figure 2.4: CARTPOLE: Trained with PPO, all networks fine-tuned over 7 learning rates. 25%, 50% and 75% quantiles over 25 random seeds shown. a) Equivariant, random, and nullspace bases. b) Equivariant basis, and two MLPs with different degrees of freedom. PONG: Trained with A2C, all networks tuned over 3 learning rates. 25%, 50% and 75% quantiles over 15 random seeds shown c) Equivariant, nullspace, and random bases, and regular CNN for Pong.

Models

We implemented MDP homomorphic networks on top of two base architectures: MLP and CNN (exact architectures in Supplementary). We further experimented with an equivariant feature extractor, appended by a non-equivariant network, to isolate where equivariance made the greatest impact.

Basis Networks We call networks whose weights are linear combinations of basis weights *basis networks*. As an ablation study on all equivariant networks, we sought to measure the effects of the basis training dynamics. We compared an *equivariant* basis against a pure *nullspace* basis, i.e. an explicitly non-symmetric basis using the right-null vectors from the equivariant layer construction, and a *random* basis, where we skip the symmetrization step in the layer construction and use the full rank basis. Unless stated otherwise, we reduce the number of 'channels' in the basis networks compared to the regular networks by dividing by the square root of the group size, ending up with a comparable number of trainable parameters.

Results and Discussion

We show training curves for CartPole in Figures 2.4a-b, Pong in Figure 2.4c and for the grid world in Figure 2.6. Across all experiments we observed that the MDP homomorphic network outperforms both the non-equivariant basis networks and the standard architectures, in terms of convergence speed.

This confirms our motivations that building symmetry-preserving policy networks leads to faster convergence. Additionally, when compared to the data augmentation baselines in Figure 2.5, using equivariant networks is more beneficial. This is consistent with other results in the equivariance literature [14, 187, 191, 193]. While data augmentation can be used to create a larger dataset by exploiting symmetries, it does not directly lead to effective parameter sharing (as our approach does). Note, in Pong we only train the first 15 million frames to high-

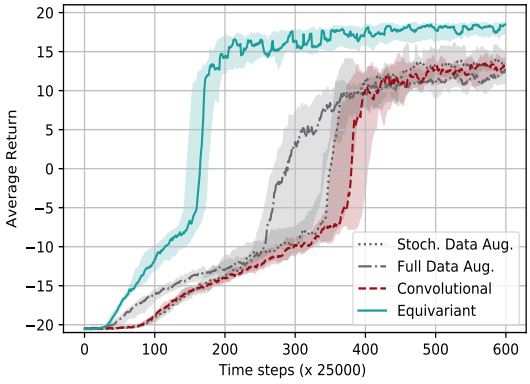
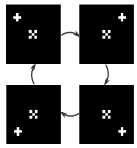


Figure 2.5: Data augmentation comparison on Pong.

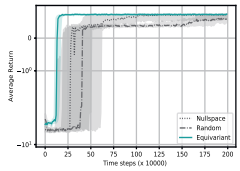
light the difference in the beginning; in constrast, a typical training duration is 50-200 million frames [124, 159].

For our ablation experiment, we wanted to control for the introduction of bases. It is not clear *a priori* that a network with a basis has the same gradient descent dynamics as an equivalent 'basisless' network. We compared equivariant, non-equivariant, and random bases, as mentioned above. We found the equivariant basis led to the fastest convergence. Figures 2.4a and 2.4c show that for CartPole and Pong the nullspace basis converged faster than the random basis. In the grid world there was no clear winner between the two. This is a curious result, requiring deeper investigation in a follow-up.

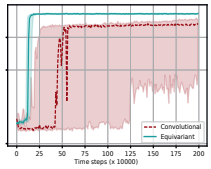
For a third experiment, we investigated what happens if we sacrifice strict equivariance of the policy. This is attractive because it removes the need to find a transformation operator for a flattened output feature map. Instead, we only maintained an equivariant feature extractor, compared against a basic CNN feature extractor. The networks built on top of these extractors were MLPs. The results, in Figure 2.4c, are two-fold: 1) Basis feature extractors converge faster than standard CNNs, and 2) the equivariant feature extractor has fastest convergence. We hypothesize the equivariant feature extractor is fastest as it is easiest to learn an equivariant policy from equivariant features.
We have additionally compared an equivariant feature extractor to a regular convolutional network on the Atari game Breakout, where the difference between the equivariant network and the regular network is much less pronounced. For details, see Appendix 2.C.



(a) Symmetries



(b) Grid World: Bases



(c) Grid World: CNNs

Figure 2.6: Grid World: Trained with A2C, all networks fine-tuned over 6 learning rates. 25%, 50% and 75% quantiles over 20 random seeds shown. a) showcase of symmetries, b) Equivariant, nullspace, and random bases c) plain CNN and equivariant CNN.

2.5 Related Work

Past work on MDP homomorphisms has often aimed at discovering the map itself based on knowledge of the transition and reward function, and under the assumption of enumerable state spaces [141, 142, 143, 165]. Other work relies on learning the map from sampled experience from the MDP [170, 16, 118]. Exactly computing symmetries in MDPs is graph isomorphism complete [127] even with full knowledge of the MDP dynamics. Rather than assuming knowledge of the transition and reward function, and small and enumerable state spaces, in this work we take the inverse view: we assume that we have an easily identifiable transformation of the joint state–action space and exploit this knowledge to learn more efficiently. Exploiting symmetries in deep RL has been previously explored in the game of Go, in the form of symmetric filter weights [151, 30] or data augmentation [156]. Other work on data augmentation increases sample efficiency and generalization on well-known benchmarks by augmenting existing data points state transformations such as random translations, cutout, color jitter and random convolutions [99, 31, 107, 111]. In contrast, we encode symmetries into the neural network weights, leading to more parameter sharing. Additionally, such data augmentation approaches tend to take the *invariance* view, augmenting existing data with state transformations that leave the state's Q-values intact [99, 31, 107, 111] (the exception being [115] and [119], who augment trajectories rather than just states). Similarly, permutation invariant networks are commonly used in approaches to multi-agent RL [160, 117, 81]. We instead take the *equivariance* view, which accommodates a much larger class of symmetries that includes transformations on the action space. Abdolhosseini et al. [1] have previously manually constructed an equivariant network for a single group of symmetries in a single RL problem, namely reflections in a bipedal locomotion task. Our MDP homomorphic networks allow for automated construction of networks that are equivariant under arbitrary discrete groups and are therefore applicable to a wide variety of problems.

From an equivariance point-of-view, the automatic construction of equivariant layers is new. [35] comes close to specifying a procedure, outlining the system of equations to solve, but does not specify an algorithm. The basic theory of group equivariant networks was outlined in [34, 35] and [33], with notable implementations to 2D roto-translations on grids [193, 188, 186] and 3D roto-translations on grids [192, 191, 187]. All of these works have relied on hand-constructed equivariant layers.

2.6 Conclusion

This chapter introduced MDP homomorphic networks, a family of deep architectures for reinforcement learning problems where symmetries have been identified. MDP homomorphic networks tie weights over symmetric state-action pairs. This weight-tying leads to fewer degrees-of-freedom and in our experiments we found that this translates into faster convergence. We used the established theory of MDP homomorphisms to motivate the use of equivariant networks in symmetric MDPs, thus formalizing the connection between equivariant networks and symmetries in reinforcement learning. As an innovation, we also introduced the first method to automatically construct equivariant network layers, given a specification of the symmetries in question, thus removing a significant implementational obstacle. For future work, we want to further understand the symmetrizer and its effect on learning dynamics, as well as generalizing to problems that are not fully symmetric.

2.7 Broader Impact Statement

The goal of this chapter is to make (deep) reinforcement learning techniques more efficient at solving Markov decision processes (MDPs) by making use of prior knowledge about symmetries. We do not expect the particular algorithm we develop to lead to immediate societal risks. However, Markov decision processes are very general, and can e.g. be used to model problems in autonomous driving, smart grids, and scheduling. Thus, solving such problems more efficiently can in the long run cause positive or negative societal impact.

For example, making transportation or power grids more efficient, thereby making better use of scarce resources, would be a significantly positive impact. Other potential applications, such as in autonomous

weapons, pose a societal risk [131]. Like many AI technologies, when used in automation, our technology can have a positive impact (increased productivity) and a negative impact (decreased demand) on labor markets.

More immediately, control strategies learned using RL techniques are hard to verify and validate. Without proper precaution (e.g. [177]), employing such control strategies on physical systems thus run the risk of causing accidents involving people, e.g. due to reward misspecification, unsafe exploration, or distributional shift [5].

Chapter Appendix

2.A The Symmetrizer

In this section we prove three properties of the symmetrizer: the symmetric property ($S(\mathbf{W}) \in \mathcal{W}$ for all $\mathbf{W} \in \mathcal{W}_{\text{total}}$), the fixing property ($\mathbf{W} \in \mathcal{W} \implies S(\mathbf{W}) = \mathbf{W}$) , and the idempotence property ($S(S(\mathbf{W})) = S(\mathbf{W})$ for all $\mathbf{W} \in \mathcal{W}_{\text{total}}$).

The Symmetric Property Here we show that the symmetrizer S maps matrices $\mathbf{W} \in \mathcal{W}_{\text{total}}$ to equivariant matrices $S(\mathbf{W}) \in \mathcal{W}$. For this, we show that a symmetrized weight matrix $S(\mathbf{W})$ from Equation 16 satisfies the equivariance constraint of Equation 14.

Proof 1 (The symmetric property) *We begin by recalling the equivariance constraint*

$$\boldsymbol{K}_g \boldsymbol{W} \boldsymbol{z} = \boldsymbol{W} \boldsymbol{L}_g \boldsymbol{z}, \qquad \textit{for all } g \in G, \boldsymbol{z} \in \mathbb{R}^{D_{in}+1}. \tag{2.18}$$

Now note that we can drop the dependence on $\boldsymbol{z}$, since this equation is true for all $\boldsymbol{z}$. At the same time, we left-multiply both sides of this equation by ${\boldsymbol{K}_g}^{-1}$, which is possible because group representations are invertible. This results in the following set of equations

$$\boldsymbol{W} = \boldsymbol{K}_g^{-1} \boldsymbol{W} \boldsymbol{L}_g, \qquad \textit{for all } g \in G. \tag{2.19}$$

Any $\boldsymbol{W}$ satisfying this equation satisfies Equation 2.18 and is thus a member of $\mathcal{W}$. To show that $S(\mathbf{W})$ is a member of $\mathcal{W}$, we thus would need show that $S(\boldsymbol{W}) = \boldsymbol{K}_g^{-1} S(\boldsymbol{W}) \boldsymbol{L}_g$ for all $\boldsymbol{W} \in \mathcal{W}_{total}$ and $g \in G$. This can be shown as

follows:

$$\boldsymbol{K}_g^{-1} S(\boldsymbol{W}) \boldsymbol{L}_g = \boldsymbol{K}_g^{-1} \left(\frac{1}{|G|} \sum_{h \in G} \boldsymbol{K}_h^{-1} \boldsymbol{W} \boldsymbol{L}_h \right) \boldsymbol{L}_g \quad \textit{substitute } S(\boldsymbol{W}) = \boldsymbol{K}_g^{-1} S(\boldsymbol{W}) \boldsymbol{L}_g \tag{2.20}$$

$$= \frac{1}{|G|} \sum_{h \in G} \boldsymbol{K}_g^{-1} \boldsymbol{K}_h^{-1} \boldsymbol{W} \boldsymbol{L}_h \boldsymbol{L}_g \tag{2.21}$$

$$= \frac{1}{|G|} \sum_{h \in G} \boldsymbol{K}_{hg}^{-1} \boldsymbol{W} \boldsymbol{L}_{hg} \qquad \textit{representation definition: } \boldsymbol{L}_h \boldsymbol{L}_g = \boldsymbol{L}_{hg} \tag{2.22}$$

$$= \frac{1}{|G|} \sum_{g' g^{-1} \in G} \boldsymbol{K}_{g'}^{-1} \boldsymbol{W} \boldsymbol{L}_{g'} \qquad \textit{change of variables } g' = hg, h = g' g^{-1} \tag{2.23}$$

$$= \frac{1}{|G|} \sum_{g' \in Gg} \boldsymbol{K}_{g'}^{-1} \boldsymbol{W} \boldsymbol{L}_{g'} \qquad g' g^{-1} \in G \iff g' \in Gg \tag{2.24}$$

$$= \frac{1}{|G|} \sum_{g' \in G} \boldsymbol{K}_{g'}^{-1} \boldsymbol{W} \boldsymbol{L}_{g'} \qquad G = Gg \tag{2.25}$$

$$= S(\boldsymbol{W}) \qquad \textit{definition of symmetrizer.} \tag{2.26}$$

Thus we see that $S(\mathbf{W})$ satisfies the equivariance constraint, which implies that $S(\mathbf{W}) \in \mathcal{W}$.

The Fixing Property For the symmetrizer to be useful, we need to make sure that its range covers the equivariant subspace $\mathcal{W}$, and not just a subset of it; that is, we need to show that

$$\mathcal{W} = \{ S(\mathbf{W}) \in \mathcal{W} | \mathbf{W} \in \mathcal{W}_{\text{total}} \}. \tag{2.27}$$

We show this by picking a matrix $\mathbf{W} \in \mathcal{W}$ and showing that $\mathbf{W} \in \mathcal{W} \implies S(\mathbf{W}) = \mathbf{W}$.

Proof 2 (The fixing property) *We begin by assuming that $\boldsymbol{W} \in \mathcal{W}$, then*

$$S(\boldsymbol{W}) = \frac{1}{|G|} \sum_{g \in G} \boldsymbol{K}_g^{-1} \boldsymbol{W} \boldsymbol{L}_g \quad \textit{definition} \tag{2.28}$$

$$= \frac{1}{|G|} \sum_{g \in G} \boldsymbol{K}_g^{-1} \boldsymbol{K}_g \boldsymbol{W} \quad \boldsymbol{W} \in \mathcal{W} \iff \boldsymbol{K}_g \boldsymbol{W} = \boldsymbol{W} \boldsymbol{L}_g, \forall g \in G \tag{2.29}$$

$$= \frac{1}{|G|} \sum_{g \in G} \boldsymbol{W} \tag{2.30}$$

$$= \boldsymbol{W} \tag{2.31}$$

This means that the symmetrizer leaves the equivariant subspace invariant. In fact, the statement we just showed is stronger in saying that each point in the equivariant subspace is unaltered by the symmetrizer. In the language of group theory we say that subspace $\mathcal{W}$ is fixed under *G. Since $S : \mathcal{W}_{total} \to \mathcal{W}$ and there exist matrices $\mathbf{W}$ such that for every $\mathbf{W} \in \mathcal{W}$, $S(\mathbf{W}) = \mathbf{W}$, we have shown that*

$$\mathcal{W} = \{S(\mathbf{W}) \in \mathcal{W} | \mathbf{W} \in \mathcal{W}_{total}\}. \tag{2.32}$$

The Idempotence Property Here we show that the symmetrizer $S(\mathbf{W})$ from Equation 16 is idempotent, $S(S(\mathbf{W})) = S(\mathbf{W})$.

Proof 3 (The idempotence property) *Recall the definition of the symmetrizer*

$$S(\mathbf{W}) = \frac{1}{|G|} \sum_{g \in G} \mathbf{K}_g^{-1} \mathbf{W} \mathbf{L}_g. \tag{2.33}$$

Now let's expand $S(S(\mathbf{W}))$:

$$S(S(\mathbf{W})) = S\left(\frac{1}{|G|} \sum_{h \in G} \mathbf{K}_h^{-1} \mathbf{W} \mathbf{L}_h\right) \tag{2.34}$$

$$= \frac{1}{|G|} \sum_{g \in G} \mathbf{K}_g^{-1} \left(\frac{1}{|G|} \sum_{h \in G} \mathbf{K}_h^{-1} \mathbf{W} \mathbf{L}_h\right) \mathbf{L}_g \tag{2.35}$$

$$= \frac{1}{|G|} \sum_{g \in G} \left(\frac{1}{|G|} \sum_{h \in G} \mathbf{K}_g^{-1} \mathbf{K}_h^{-1} \mathbf{W} \mathbf{L}_h \mathbf{L}_g\right) \quad \textit{linearity of sum} \tag{2.36}$$

$$= \frac{1}{|G|} \sum_{g \in G} \left(\frac{1}{|G|} \sum_{h \in G} \mathbf{K}_{hg}^{-1} \mathbf{W} \mathbf{L}_{hg}\right) \quad \textit{definition of group representations} \tag{2.37}$$

$$= \frac{1}{|G|} \sum_{g \in G} \left(\frac{1}{|G|} \sum_{g' g^{-1} \in G} \mathbf{K}_{g'}^{-1} \mathbf{W} \mathbf{L}_{g'}\right) \quad \textit{change of variables } g' = hg \tag{2.38}$$

$$= \frac{1}{|G|} \sum_{g \in G} \left(\frac{1}{|G|} \sum_{g' \in Gg} \mathbf{K}_{g'}^{-1} \mathbf{W} \mathbf{L}_{g'}\right) \quad g' g^{-1} \in G \iff g' \in Gg \tag{2.39}$$

$$= \frac{1}{|G|} \sum_{g \in G} \left(\frac{1}{|G|} \sum_{g' \in G} \mathbf{K}_{g'}^{-1} \mathbf{W} \mathbf{L}_{g'}\right) \quad Gg = G \tag{2.40}$$

$$= \frac{1}{|G|} \sum_{g' \in G} \mathbf{K}_{g'}^{-1} \mathbf{W} \mathbf{L}_{g'} \quad \textit{sum over constant} \tag{2.41}$$

$$= S(\mathbf{W}) \tag{2.42}$$

Thus we see that $S(\mathbf{W})$ satisfies the equivariance constraint, which implies that $S(\mathbf{W}) \in \mathcal{W}$.

2.B Experimental Settings

Designing representations

In the main text we presented a method to construct a space of intertwiners $\mathcal{W}$ using the symmetrizer. This relies on us already having chosen specific representations/transformation operators for the input, the output, and for every intermediate layer of the MDP homomorphic networks. While for the input space (state space) and output space (policy space), these transformation operators are easy to define, *it is an open question how to design a transformation operator for the intermediate layers* of our networks. Here we give some rules of thumb that we used, followed by the specific transformation operators we used in our experiments.

For each experiment we first identified the group G of transformations. In every case, this was a finite group of size $|G|$, where the size is the number of elements in the group (number of distinct transformation operators). For example, a simple flip group as in Pong has two elements, so $|G| = 2$. Note that the group size $|G|$ does not necessarily equal the size of the transformation operators, whose size is determined by the dimensionality of the input/activation layer/policy.

Stacking Equivariant Layers If we stack equivariant layers, the resulting network is equivariant as a whole too [34]. To see that this is the case, consider the following example. Assume we have network f, consisting of layers f_1 and f_2, which satisfy the layer-wise equivariance constraints:

$$P_g[f_1(x)] = f_1(L_g[x]) \tag{2.43}$$
$$K_g[f_2(x)] = f_2(P_g[x]) \tag{2.44}$$

With K_g the output transformation of the network, L_g the input transformation, and P_g the intermediate transformation. Now,

$$\begin{aligned} K_g[f(x)] &= K_g[f_2(f_1(x))] && && (2.45)\\ &= f_2(P_g[f_1(x)] && (f_2 \text{ equivariance constraint}) && (2.46)\\ &= f_2(f_1(L_g[x])) && (f_1 \text{ equivariance constraint}) && (2.47)\\ &= f(L_g[x]) && && (2.48) \end{aligned}$$

and so the whole network f is equivariant with regards to the input transformation L_g and the output transformation K_g. Note that this depends on the intermediate representation P_g being shared between layers, i.e. f_1's output transformation is the same as f_2's input transformation.

MLP-structured networks For MLP-structured networks (CartPole), typically the activations have shape `[batch_size, nc]`, with `nc` the number of channels. Instead we used a shape of `[batch_size, nc, representation_size]`, where for the intermediate layers `representation_size=|G|+1` (we have a +1 because of the bias). The transformation operators we then apply to the activations is the set of permutations for group size $|G|$ appended with a 1 on the diagonal for the bias, acting on this last 'representation dimension'. Thus a forward pass of a layer is computed as

$$\mathbf{y}_{b,c_{\text{out}},r_{\text{out}}} = \sum_{c_{\text{in}}=1}^{\texttt{nc}} \sum_{r_{\text{in}}=1}^{\texttt{|G|+1}} \mathbf{z}_{b,c_{\text{in}},r_{\text{in}}} \mathbf{W}_{c_{\text{out}},r_{\text{out}},c_{\text{in}},r_{\text{in}}} \tag{2.49}$$

where

$$\mathbf{W}_{c_{\text{out}},r_{\text{out}},c_{\text{in}},r_{\text{in}}} = \sum_{i=1}^{\text{rank}(\mathcal{W})} c_{i,c_{\text{out}},c_{\text{in}}} \mathbf{V}_{i,r_{\text{out}},r_{\text{in}}}. \tag{2.50}$$

CNN-structured networks For CNN-structured networks (Pong and Grid World), typically the activations have shape `[batch_size, nc, height, width]`. Instead we used a shape of `[batch_size, nc, representation_size, height, width]`, where for the intermediate layers `representation_size=|G|+1`. The transformation operators we apply to the input of the layer is a spatial transformation on the `height, width` dimensions and a permutation on the `representation` dimension. This is because in the intermediate layers of the network the activations do not only transform in space, but also along the representation dimensions of the tensor. The transformation operators we apply to the output of the layer is just a permutation on the `representation` dimension. Thus a forward pass of a layer is computed as

$$\mathbf{y}_{b,c_{\text{out}},r_{\text{out}},h_{\text{out}},w_{\text{out}}} = \sum_{c_{\text{in}}=1}^{\texttt{nc}} \sum_{r_{\text{in}}=1}^{\texttt{|G|+1}} \sum_{h_{\text{in}},w_{\text{in}}} \mathbf{z}_{b,c_{\text{in}},r_{\text{in}},h_{\text{out}}+h_{\text{in}},w_{\text{out}}+w_{\text{in}}} \mathbf{W}_{c_{\text{out}},r_{\text{out}},c_{\text{in}},r_{\text{in}},h_{\text{in}},w_{\text{in}}} \tag{2.51}$$

where

$$\mathbf{W}_{c_{\text{out}},r_{\text{out}},c_{\text{in}},r_{\text{in}},h_{\text{in}},w_{\text{in}}} = \sum_{i=1}^{\text{rank}(\mathcal{W})} c_{i,c_{\text{out}},c_{\text{in}}} \mathbf{V}_{i,r_{\text{out}},r_{\text{in}},h_{\text{in}},w_{\text{in}}}. \tag{2.52}$$

Equivariant	Nullspace	Random	MLP
0.01	0.005	0.001	0.001

Table 2.B.1: Final learning rates used in CartPole-v1 experiments.

Cartpole-v1

Group Representations For states:

$$\mathbf{L}_{g_e} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \mathbf{L}_{g_1} = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

For intermediate layers and policies:

$$\mathbf{K}^{\pi}_{g_e} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \mathbf{K}^{\pi}_{g_1} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

For values we require an invariant rather than equivariant output. This invariance is implemented by defining the output representations to be $|G|$ identity matrices of the desired output dimensionality. For predicting state values we required a 1-dimensional output, and we thus used $|G|$ 1-dimensional identity matrices, i.e. for value output V:

$$\mathbf{K}^{V}_{g_e} = \begin{pmatrix} 1 \end{pmatrix}, \mathbf{K}^{V}_{g_1} = \begin{pmatrix} 1 \end{pmatrix}$$

Hyperparameters For both the basis networks and the MLP, we used Xavier initialization. We trained PPO using ADAM on 16 parallel environments and fine-tuned over the learning rates $\{0.01, 0.05, 0.001, 0.005, 0.0001, 0.0003, 0.0005\}$ by running 25 random seeds for each setting, and report the best curve. The final learning rates used are shown in Table 2.B.1. Other hyperparameters were defaults in RLPYT [159], except that we turn off learning rate decay.

Architecture
Basis networks:

Listing 2.1: Basis Networks Architecture for CartPole-v1

```
BasisLinear(repr_in=4, channels_in=1, repr_out=2, channels_out=64)
ReLU()
BasisLinear(repr_in=2, channels_in=64, repr_out=2, channels_out=64)
ReLU()
BasisLinear(repr_in=2, channels_in=64, repr_out=2, channels_out=1)
BasisLinear(repr_in=2, channels_in=64, repr_out=1, channels_out=1)
```

First MLP variant:

Listing 2.2: First MLP Architecture for CartPole-v1

```
Linear(channels_in=1, channels_out=64)
ReLU()
Linear(channels_in=64, channels_out=128)
ReLU()
Linear(channels_in=128,  channels_out=1)
Linear(channels_in=128, channels_out=1)
```

Second MLP variant:

Listing 2.3: Second MLP Architecture for CartPole-v1

```
Linear(channels_in=1, channels_out=128)
ReLU()
Linear(channels_in=128, channels_out=128)
ReLU()
Linear(channels_in=128,  channels_out=1)
Linear(channels_in=128, channels_out=1)
```

GridWorld

Group Representations For states we use `numpy.rot90`. The stack of weights is rolled.

For the intermediate representations:

$$\mathbf{L}_{g_e} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \mathbf{L}_{g_1} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix},$$

$$\mathbf{L}_{g_2} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}, \mathbf{L}_{g_3} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

For the policies:

$$\mathbf{K}^{\pi}_{g_e} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \mathbf{K}^{\pi}_{g_1} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix},$$

$$\mathbf{K}^{\pi}_{g_2} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}, \mathbf{K}^{\pi}_{g_3} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

For the values:

$$\mathbf{K}^V_{g_e} = \left(1\right), \mathbf{K}^V_{g_1} = \left(1\right), \mathbf{K}^V_{g_2} = \left(1\right), \mathbf{K}^V_{g_3} = \left(1\right)$$

Hyperparameters For both the basis networks and the CNN, we used He initialization. We trained A2C using ADAM on 16 parallel environments and fine-tuned over the learning rates $\{0.00001, 0.00003, 0.0001, 0.0003, 0.001, 0.003\}$ on 20 random seeds for each setting, and reporting the best curve. The final learning rates used are shown in Table 2.B.2. Other hyperparameters were defaults in RLPYT [159].

Equivariant	Nullspace	Random	CNN
0.001	0.003	0.001	0.003

Table 2.B.2: Final learning rates used in grid world experiments.

Architecture
Basis networks:

Listing 2.4: Basis Networks Architecture for GridWorld

```
BasisConv2d(repr_in=1, channels_in=1, repr_out=4, channels_out=⌊16/√4⌋,
            filter_size=(7, 7), stride=2, padding=0)
ReLU()
BasisConv2d(repr_in=4, channels_in=⌊16/√4⌋, repr_out=4, channels_out=⌊32/√4⌋,
            filter_size=(5, 5), stride=1, padding=0)
ReLU()
GlobalMaxPool()
BasisLinear(repr_in=4, channels_in=⌊32/√4⌋, repr_out=4, channels_out=⌊512/√4⌋)
ReLU()
BasisLinear(repr_in=4, channels_in=⌊512/√4⌋, repr_out=5, channels_out=1)
BasisLinear(repr_in=4, channels_in=⌊512/√4⌋, repr_out=1, channels_out=1)
```

CNN:

Listing 2.5: CNN Architecture for GridWorld

```
Conv2d(channels_in=1, channels_out=16,
            filter_size=(7, 7), stride=2, padding=0)
ReLU()
Conv2d(channels_in=16,channels_out=32,
            filter_size=(5, 5), stride=1, padding=0)
ReLU()
GlobalMaxPool()
Linear(channels_in=32, channels_out=512)
ReLU()
Linear(channels_in=512, channels_out=5)
Linear(channels_in=512, channels_out=1)
```

Pong

Group Representations For the states we use `numpy`'s indexing to flip the input, i.e.
`w = w[..., ::-1, :]`, then the permutation on the `representation` dimension of the weights is a `numpy.roll`, since the group is cyclic.

For the intermediate layers:

$$\mathbf{L}_{g_e} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \mathbf{L}_{g_1} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

Hyperparameters For both the basis networks and the CNN, we used He initialization. We trained A2C using ADAM on 4 parallel environments and fine-tuned over the learning rates $\{0.0001, 0.0002, 0.0003\}$ on 15 random seeds for each setting, and reporting the best curve. The learning rates to fine-tune over were selected to be close to where the baseline performed well in preliminary experiments. The final learning rates used are shown in Table 2.B.3. Other hyperparameters were defaults in RLPYT [159].

Equivariant	Nullspace	Random	CNN
0.0002	0.0002	0.0002	0.0001

Table 2.B.3: Learning rates used in Pong experiments.

Architecture
Basis Networks:

Listing 2.6: Basis Networks Architecture for Pong

```
BasisConv2d(repr_in=1, channels_in=4, repr_out=2, channels_out=⌊16/√2⌋,
            filter_size=(8, 8), stride=4, padding=0)
ReLU()
BasisConv2d(repr_in=2, channels_in=⌊16/√2⌋, repr_out=2, channels_out=⌊32/√2⌋,
            filter_size=(5, 5), stride=2, padding=0)
ReLU()
Linear(channels_in=2816, channels_out=⌊512/√2⌋)
ReLU()
Linear(channels_in=⌊512/√2⌋, channels_out=6)
Linear(channels_in=⌊512/√2⌋, channels_out=1)
```

CNN:

Listing 2.7: CNN Architecture for Pong

```
Conv2d(channels_in=4, channels_out=16, filter_size=(8, 8), stride=4, padding=0)
ReLU()
```

```
Conv2d(channels_in=16,channels_out=32, filter_size=(5, 5), stride=2, padding=0)
ReLU()
Linear(channels_in=2048, channels_out=512)
ReLU()
Linear(channels_in=512, channels_out=6)
Linear(channels_in=512, channels_out=1)
```

2.C Breakout Experiments

We evaluated the effect of an equivariant basis extractor on Breakout, compared to a baseline convolutional network. The hyperparameter settings and architecture were largely the same as those of Pong, except for the input group representation, a longer training time, and that we considered a larger range of learning rates. To ensure symmetric states, we remove the two small decorative blocks in the bottom corners.

Group Representations For the states we use `numpy`'s indexing to flip the input, i.e.
`w = w[..., :, ::-1]` (note the different axis than in Pong), then the permutation on the `representation` dimension of the weights is a `numpy.roll`, since the group is cyclic.

For the intermediate layers:

$$\mathbf{L}_{g_e} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \mathbf{L}_{g_1} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

Hyperparameters We used He initialization. We trained A2C using ADAM on 4 parallel environments and fine-tuned over the learning rates $\{0.001, 0.005, 0.0001, 0.0002, 0.0003, 0.0004, 0.0005, 0.00001, 0.00005\}$ on 15 random seeds for each setting, and reporting the best curve. The final learning rates used are shown in Table 2.C.1. Other hyperparameters were defaults in RLPYT [159].

Equivariant	CNN
0.0002	0.0002

Table 2.C.1: Learning rates used in Breakout experiments.

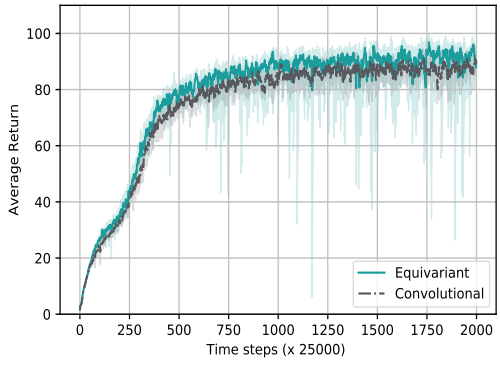
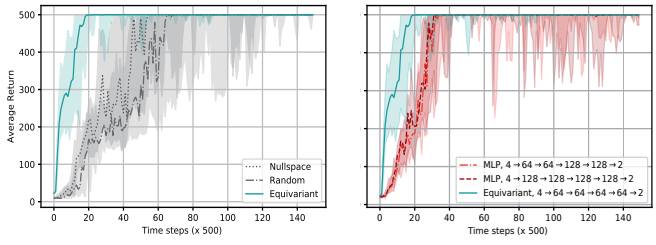


Figure 2.C.1: Breakout: Trained with A2C, all networks fine-tuned over 9 learning rates. 25%, 50% and 75% quantiles over 14 random seeds shown.

Results Figure 2.C.1 shows the result of the equivariant feature extractor versus the convolutional baseline. While we again see an improvement over the standard convolutional approach, the difference is much less pronounced than in CartPole, Pong or the grid world. It is not straightforward why. One factor could be that the equivariant feature extractor is not end-to-end MDP homomorphic. It instead outputs a type of MDP homomorphic state representations and learns a regular policy on top. As a result, the unconstrained final layers may negate some of the advantages of the equivariant feature extractor. This may be more of an issue for Breakout than Pong, since Breakout is a more complex game.

2.D Cartpole-v1 Deeper Network Results

We show the effect of training a deeper network – 4 layers instead of 2 – for CartPole-v1 in Figure 2.D.1. The performance of the regular depth networks in Figure 4b and the deeper networks in Figure 2.D.1 is comparable, except that for the regular MLP, the variance is much higher when using deeper networks.



(a) Cartpole-v1: Bases

(b) Cartpole-v1: MLPs

Figure 2.D.1: CARTPOLE: Trained with PPO, all networks fine-tuned over 7 learning rates. 25%, 50% and 75% quantiles over 25 random seeds shown. a) Equivariant, random, and nullspace bases. b) Equivariant basis, and two MLPs with different degrees of freedom.

2.E Bellman Equations

$$V^{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(s,a) \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V^{\pi}(s') \right] \tag{2.53}$$

$$Q^{\pi}(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V^{\pi}(s'). \tag{2.54}$$

3

Multi-Agent MDP Homomorphic Networks

3.1 Introduction

In the previous Chapter, we have shown that enforcing group symmetries in single agent reinforcement learning improves data efficiency. In this Chapter, we will go beyond the single agent case and consider global symmetries in distributed cooperative multi-agent learning.

Equivariant and geometric deep learning have gained traction in recent years, showing promising results in supervised learning [34, 191, 187, 186, 193, 52, 166], unsupervised learning [39] and reinforcement learning [173, 157]. In single agent reinforcement learning, incorporating symmetry information has been a successful approach, for example using MDP homomorphic networks [173], trajectory augmentation [115, 119], or symmetric locomotion policies [1]. Equivariance enables an agent to learn policies more efficiently within its environment by sharing weights between state-action pairs that are equivalent under a transformation. As a result of this weight sharing, the agent implicitly learns a policy in a reduced version of the MDP. We are interested in cooperative multi-agent reinforcement learning, where symmetries exist both in the global environment, and between individual agents in the larger multi-agent system. Existing work on symmetries in single agent reinforcement learning can only be generalized to the fully centralized multi-agent setting, because such approaches rely on

the global symmetry in the full state-action spaces and these can result in correspondences *across* agents, as shown in Figure 3.1.1.

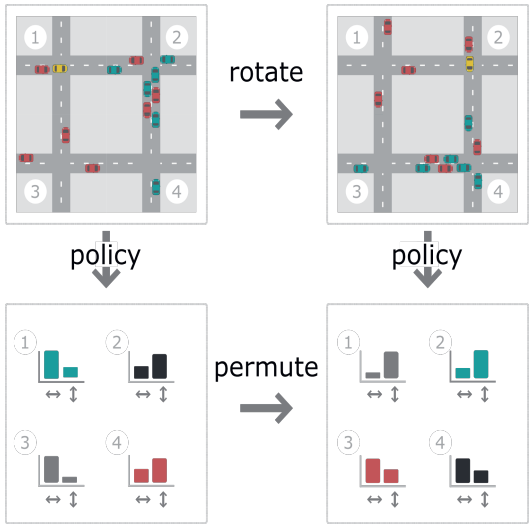


Figure 3.1.1: Example of a global symmetry in a traffic light control problem, where each agent, locally controlling the traffic lights of a single intersection, has to decide to give a green light to either the horizontal or vertical lanes. When a rotated state occurs in the environment the optimal policy is permuted, both between and within agents. Local policies are color coded. Multi-Agent MDP Homomorphic Networks are equivariant to global symmetries while still allowing distributed execution based on local observations and local communications only.

This means such approaches cannot be used in *distributed* multi-agent systems with communication constraints. Here, we seek to be equivariant to global symmetries of cooperative multi-agent systems while still being able to execute policies in a distributed manner.

Existing work in deep multi-agent reinforcement learning has shown the potential of using permutation symmetries and invariance between agents [117, 81, 163, 144, 18, 160, 171]. Such work takes an anonymity view of homogeneous agents, where the agent's observations matter for the policy but not its identity. Using permutation symmetries ensures extensive weight sharing between agents, resulting in improved data efficiency. Here, we go beyond such permutation symmetries, and consider more general symmetries of global multi-agent systems, such as rotational symmetries.

In this chapter, we propose Multi-Agent MDP Homomorphic Networks, a class of distributed policy networks which are equivariant to global symmetries of the multi-agent system, as well as to standard permutation symmetries. Our contributions are as follows. (i) We propose a factorization of global symmetries in the joint state-action space of cooperative multi-agent systems. (ii) We introduce a multi-agent equivariant policy network based on this factorization. (iii) Our main contribution is an approach to cooperative multi-agent reinforce-

ment learning that is globally equivariant while requiring only local agent computation and local communication between agents at execution time. We evaluate Multi-Agent MDP Homomorphic Networks on symmetric multi-agent problems and show improved data efficiency compared to non-equivariant baselines.

3.2 Related Work

Symmetries in single agent reinforcement learning Symmetries in Markov Decision Processes have been formalized by [204, 141]. Recent work on symmetries in single agent deep reinforcement learning has shown improvements in terms of data efficiency. Such work revolves around symmetries in policy networks [173, 157], symmetric filters [30], invariant data augmentation [107, 99] or equivariant trajectory augmentation [115, 119, 122] These approaches are only suitable for single agent problems or centralized multi-agent controllers. Here, we solve the problem of enforcing global equivariance while still allowing distributed execution.

Graphs and permutation symmetries in multi agent reinforcement learning Graph-based methods in cooperative multiagent reinforcement learning are well-explored. Much work is based around coordination graphs [64, 62, 97], including approaches that approximate local Q-functions with neural networks and use max-plus to find a joint policy [171, 18], and approaches that use graph-structured networks to find joint policies or value functions [81, 160]. In deep learning for multi-agent systems, the use of permutation symmetries is common, either through explicit formulations [163, 18] or through the use of graph or message passing networks [117, 81, 160]. Policies in multi-agent systems with permutation symmetries between agents are also known as functionally homogeneous policies [204] or policies with agent anonymity [144, 175]. Here, we move beyond permutation symmetries to a broader group of symmetries in multiagent reinforcement learning.

Symmetries in multi agent reinforcement learning Recently, [74] used knowledge of symmetries to improve zero-shot coordination in games which require symmetry-breaking. Here, we instead use symmetries in cooperative multi-agent systems to improve data efficiency by parameter sharing between different transformations of the global system.

3.3 Background

In this section we introduce the necessary definitions and notation used in the rest of the chapter.

Multi-Agent MDPs

We will start from the definition of Multi-Agent MDPs, a class of fully observable cooperative multi-agent systems. Full observability implies that each agent can execute the same centralized policy. Later on we will define a distributed variant of this type of decision making problem.

Definition 1 *A* Multi-Agent Markov Decision Process *(MMDP) [21] is a tuple* $(\mathcal{N}, S, \mathbf{A}, T, R)$ *where* $\mathcal{N}$ *is a set of m agents,* S *is the state space,* $\mathbf{A} = A_1 \times \cdots \times A_m$ *is the joint action space of the MMDP,* $T : S \times A_1 \times \cdots \times A_m \times S \to [0, 1]$ *is the transition function, and* $R : S \times A_1 \times \cdots \times A_m \to \mathbb{R}$ *is the immediate reward function.*

The goal of an MMDP, as in the single agent case, is to find a *joint policy* that maps states to probability distributions over joint actions, $\pi : S \to \Delta(A)$, (with $\Delta(A)$ the space of such distributions) to maximize the expected discounted return of the system, $R_t = \mathbb{E}[\sum_{k=0}^{T} \gamma^k r_{t+k+1}]$ with $\gamma \in [0, 1]$ a discount factor. An MMDP can be viewed as a single-agent MDP where the agent takes joint actions.

Groups and Transformations

In this chapter we will refer extensively to group symmetries. Here, we will briefly introduce these concepts and explain their significance to discussing equivalences of decision making problems.

A group G is a set with a binary operator $\cdot$ that obeys the group axioms: identity, inverse, closure, and associativity. Consider as a running example the set of 90 degree rotations $\{0°, 90°, 180°, 270°\}$, which we can write as rotation matrices:

$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{3.1}$$

with $\theta \in \{0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}\}$. Composing any two matrices in this set results in another matrix in the set, meaning the set is closed under composition. For example, composing $R(\frac{\pi}{2})$ and $R(\pi)$ results in another member of the set, in this case $R(\frac{3\pi}{2})$. Similarly, each member of the set has an inverse that is also in the set, and $R(0)$ is an identity ele-

ment. Since matrix multiplication is associative, the group axioms are satisfied and the set is a group under composition.

A group action is a function $G \times X \to X$ that satisfies $ex = x$ (where e is the identity element) and $(g \cdot h)x = g \cdot (hx)$. For example, the group of 90 degree rotation matrices acts on vectors to rotate them. Similar to the action of this group on vectors, we can define an action of the same group on image space: e.g., the NumPy [68] function `np.rot90` acts on the set of images. We will consider group actions on the set of states represented as image observations. We match these with group actions on policies. Since we consider discrete action spaces, a group element g acting on a policy π will be represented as a matrix multiplication of the policy with a permutation matrix.

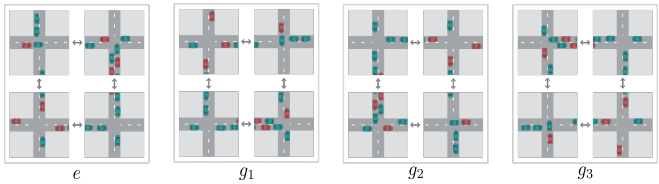


Figure 3.3.1: *The orbit of a traffic light state under the group of 90 degree rotations.*

When discussing symmetries in decision making problems, we identify sets of state-action pairs that are equivalent: if the state is transformed, the policy should be transformed as well, but potentially with a different representation of the transformation. See Figure 3.1.1. We are interested in the case where the reward and transition functions are invariant in the orbit of state-action pairs under a symmetry group. The *orbit* of a point $v \in V$, with V a vector space, is the set of all its transformations (e.g. all rotations of the point), defined as $\mathcal{O}(v) = \{gv | \forall g \in G\}$. The orbit of a point under a group forms an equivalence class. See Figure 3.3.1 for an example of an orbit of a traffic light state.

3.4 Distributing Symmetries over Multiple Agents

Consider the cooperative traffic light control system in Figure 3.1.1 that contains transformation-equivalent global state-action pairs. We first formalize global symmetries of the system similarly to symmetries in a single agent MDP. Then we will discuss how we can formulate distributed symmetries in a distributed MMDP. Finally, we introduce Multi-Agent MDP Homomorphic Networks.

Symmetries in MMDPs

We define symmetries in an MMDP similar to an MDP with symmetries [173].

Definition 2 *An MMDP is an MMDP with symmetries if reward and transition functions are invariant under a transformation group G. That is, the MMDP has symmetries if there is at least one non-trivial group G of transformations $L_g : S \to S$ and for every s, $K^s_g : A \to A$ such that*

$$R(s,a) = R(L_g[s], K^s_g[a]) \qquad \forall g \in G, s \in S, a \in A, \tag{3.2}$$

$$T(s,a,s') = T(L_g[s], K^s_g[a], L_g[s']) \qquad \forall g \in G, s, s' \in S, a \in A. \tag{3.3}$$

If two state-action pairs s, a and $L_g[s], K^s_g[a]$ obey Eq. 3.2 and 3.3, then they are equivalent [173]. Consider as an example the symmetries in Figure 3.1.1. These symmetries can result in correspondences across agents, for example when the observation of agent i is mapped by the symmetry to another agent j that is arbitrarily far away and with which there is no communication channel. In the next section, we will resolve this problem by defining distributed symmetries in terms of local observations and the communication graph defined by the state.

If we have an MMDP with symmetries, that means that there are symmetric optimal policies, i.e. if the state of the MMDP transforms, the policy transforms accordingly. The above definition of an MMDP with symmetries is only applicable to the centralized setting. If we want to be able to execute policies in a distributed manner, we will need to enforce equivariance in a distributed manner.

Distributed Multi-Agent Symmetries

In a distributed MMDP, agents make decisions based on local information only, i.e. the local states they observe, and the communications they receive from neighbors, defined as follows:

Definition 3 *A* Distributed Multiagent Markov Decision Process *(Distributed MMDP) $(\mathcal{N}, \mathbf{S}, \mathbf{A}, T, R)$ is an MMDP where agents can communicate as specified by a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with one node $v_i \in \mathcal{V}$ per agent and an edge $(i,j) \in \mathcal{E}$ if agents i and j can communicate. Thus, $\mathbf{S} = (\{S_i\}_{i\in\mathcal{N}}, \{E_{ij}\}_{(i,j)\in\mathcal{E}})$, with S_i the set of state features observable by agent i, which may include shared global features, and E_{ij} the set of edge features between i and j. In a distributed MMDP, each agent's action can only depend on the local state and the communications it receives*[1].

[1] Communication is constrained, i.e. agents cannot simply share their full observations with each other.

Here, we focus on Distributed MMDPs which have a spatial component, i.e. each agent has a coordinate in some space, and the attributes of the edges between the agents in the communication graph contain spatial information as well. For instance, the attributes $e_{ij} \in E$ for edge (i, j) might be the difference vector between agent i and agent j's coordinates. Since both agent observations and interaction edges have spatial features, a global symmetry will affect both the agent observations, the agent locations, and the features on the interaction edges. See Figure 3.4.1.

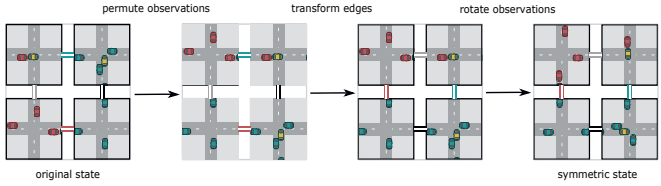


Figure 3.4.1: *Example of how a global transformation of a distributed traffic light control state can be viewed as 1) a permutation of the observations over the agents, 2) a permutation of the interaction edges, 3) a transformation of the local observations.*

To allow a globally equivariant policy network with distributed execution, we might naively decide to restrict each agent's local policy network to be equivariant to local transformations. However, this does not give us the correct global transformation, as joining the local transformations does not give us the same as the global transformations, as illustrated in Figure 3.4.2.

Instead, to get the correct transformation as shown in the left side of Figure 3.4.2, the local state is transformed, but also its position is changed, which can be seen as a permutation of the agents and their neighbors. To give an example of the equivariance constraint that we want to impose: the lower left agent (before transformation) should select an action based on its local state and communication received from its northern and eastern neighbor, while the top left agent (after transformation) should select the transformed version of the action based on its rotated local state and communication from its eastern and southern neighbor.

Since the agent has no other information about the system, if the local observations are transformed (e.g. rotated), and the messages it receives are transformed similarly, then from a local perspective the agent is in an equivalent state and should execute the same policy, but with an equivalently transformed action.

From the perspective of our agent and all its neighbors, the equivalence holds for this local subgraph as well: if the observations and

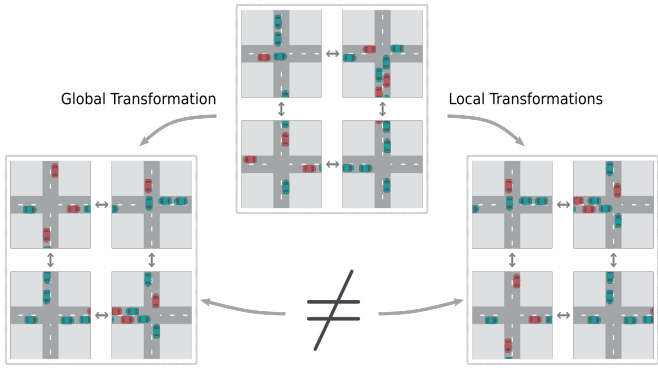


Figure 3.4.2: *Example of the difference between a global transformation on the global state, and a set of local transformations on local states. On the left we rotate the entire world by 90 degrees clockwise, which involves rotating crossings and streets. On the right we perform local uncoordinated transformations only at the street level. The latter is not a symmetry of the problem.*

local interactions rotate *relative to each other*, then the whole subgraph rotates. See Figure 3.4.1. Thus, as long as the transformations are applied to the full set of observations and the full set of communications, we have a global symmetry. We therefore propose the following definition of a Distributed MMDP with Symmetries.

Definition 4 *Let* $\mathbf{s} = (\{s_i\}_{i\in\mathcal{N}}, \{e_{ij}\}_{(i,j)\in\mathcal{E}})$. *Then a* Distributed MMDP with symmetries *is a Distributed MMDP for which the following equations hold for at least one non-trivial set of group transformations* $L_g : S \to S$ *and for every* s, $K^s_g : A \to A$ *such that*

$$R(\mathbf{s}, \mathbf{a}) = R(L_g[\mathbf{s}], K^{\mathbf{s}}_g[\mathbf{a}]) \qquad \forall g \in G, \mathbf{s} \in \mathbf{S}, \mathbf{a} \in \mathbf{A} \quad (3.4)$$

$$T(\mathbf{s}, \mathbf{a}, \mathbf{s}') = T(L_g[\mathbf{s}], K^{\mathbf{s}}_g[\mathbf{a}], L_g[\mathbf{s}']) \qquad \forall g \in G, \mathbf{s}, \mathbf{s}' \in \mathbf{S}, \mathbf{a} \in \mathbf{A} \quad (3.5)$$

where equivalently to acting on $\mathbf{s}$ *with* L_g, *we can act on the interaction and agent features separately with* $\tilde{L}_g$ *and* U_g, *to end up in the same global state:*

$$L_g[\mathbf{s}] = (P^s_g[\{\tilde{L}_g[s_i]\}_{i\in\mathcal{N}}], P^e_g[\{U_g[e_{ij}]\}_{(i,j)\in\mathcal{E}}]) \quad (3.6)$$

for $\tilde{L}_g : S_i \to S_i$, $U_g : E \to E$, *and* P^s_g *and* P^e_g *the state and edge permutations.*

Here, E is the set of edge features. The symmetries acting on the agents and agent interactions in the Distributed MMDP are a class of symmetries we call *distributable symmetries*. We have now defined a class of Distributed MMDPs with symmetries for which we can distribute a global symmetry into a set of symmetries on agents and agent interactions. This distribution allows us to define distributed policies that respect the global symmetry.

Multi-Agent MDP Homomorphic Networks

We have shown above how distributable global symmetries can be decomposed into local symmetries on agent observations and agent inter-

actions. Here, we discuss how to implement distributable symmetries in multi-agent systems in practice.

General Formulation

We want to build a neural network that 1) allows distributed execution, so that we can compute policies without a centralized controller 2) allows us to pass communications between agents (agent interactions), to enable coordination and 3) exhibits the following global equivariance constraint:

$$\vec{\pi}_\theta(L_g[\mathbf{s}]) = H_g^s[\vec{\pi}_\theta(\mathbf{s})] \tag{3.7}$$

with $H_g^s : \Pi \to \Pi$. Thus, the policy network $\vec{\pi}$ that outputs the joint policy must be *equivariant* under group transformations of the global state. To satisfy 1) and 2), i.e. allowing distributed execution and agent-to-agent communication, as well as permutation equivariance, we formulate the network as a message passing network (MPN), but with global equivariance constraints.

$$H_g^s[\vec{\pi}] = \mathrm{MPN}_\theta(L_g[s]) \tag{3.8}$$

Since a network is end-to-end equivariant if all its layers are equivariant with matching representations [34], we require layer-wise equivariance constraints on the layers. A single layer in an MPN is given by a set of node updates, i.e. $f_i^{(l+1)} = \phi_u\left(f_i^{(l)}, \sum_{j=1}^{|\mathcal{N}_i|} \phi_m\left(e_{ij}, f_j^{(l)}\right)\right)$, with $f_j^{(l)}$ the current encoding of agent j in layer l, ϕ_m the message function that computes $m_{j\to i}$ based on the edge e_{ij} and the current encoding of agent j, and ϕ_u the node update function that updates agent i's current encoding based on $f_i^{(l)}$ and the aggregated received message $m_i^{(l)}$. Since the layer is given by a set of node updates, the equivariance constraint is on the node updates. In other words, ϕ_u must be an equivariant function of local encoding $f_i^{(l)}$ and aggregated message $m_i^{(l)}$:

$$P_g\left[\phi_u(f_i^{(l)}, m_i^{(l)})\right] = \phi_u\left(L_g[f_i^{(l)}], L_g[m_i^{(l)}]\right) \tag{3.9}$$

Thus, the node update function ϕ_u is constrained to be equivariant to transformations of inputs $f_i^{(l)}$ and $m_i^{(l)}$. Therefore, to conclude that the outputs of ϕ_u transform according to P_g, we only need to enforce that its inputs f_i and m_i transform according to L_g. Thus, the subfunction ϕ_m that computes the messages $m_i^{(l)}$ must be constrained to be equivariant as well. Since ϕ_m takes as input the previous layer's encodings as well as the edges e_{ij}, this means that 1) the encodings must contain geometric information about the state, e.g. which rotation the local state is in and 2) the edge attributes contain geometric

information as well, i.e. they transform when the global state transforms (Appendix 3.B).

$$L_g\left[m_i^{(l)}\right] = \sum_{j=1}^{|\mathcal{N}_i|} \phi_m\left(U_g\left[e_{ij}\right], L_g[f_j^{(l)}]\right) \tag{3.10}$$

Note that this constraint is satisfied when ϕ_m is equivariant, since linear combinations of equivariant functions are also equivariant [35]. Putting this all together, the local encoding $f_i^{(l)}$ for each agent is equivariant to the set of edge rotations and the set of rotations of encodings in the previous layer. For more details, see Appendix 3.B. Thus, we now have the general formulation of Multi-Agent MDP Homomorphic Networks. At execution time, the distributed nature of Multi-Agent MDP Homomorphic Networks allows them to be copied onto different devices and messages exchanged between agents only locally, while still enforcing the global symmetries.

Multi-Agent MDP Homomorphic Network Architecture

Multi-Agent MDP Homomorphic Networks consist of equivariant local observation encoders $\phi_e : S_i \to \mathbb{R}^{|G|\times D}$, where G is the group, $|G|$ is its size, and D the dimension of the encoding, equivariant local message functions $\phi_m : \mathcal{E} \times \mathbb{R}^{|G|\times D} \to \mathbb{R}^{|G|\times F}$ where F is dimension of the message encoding, equivariant local update functions $\phi_u : \mathbb{R}^{|G|\times D} \times \mathbb{R}^{|G|\times F} \to \mathbb{R}^{|G|\times D}$, and equivariant local policy predictors $\phi_\pi : \mathbb{R}^{|G|\times D} \to \Pi(A_i)$. Take the example of multi-agent traffic light control with 90 degree rotation symmetries, which we evaluate on in Section 3.5. In this setting, we wish to constrain ϕ_e to be equivariant to rotations R_g of the local observations. We will require the outputs of ϕ_e to permute according to L_g^{-1} whenever the input rotates by R_g.

$$L_g\left[\phi_e(s_i)\right] = \phi_e(R_g\left[s_i\right]) \quad \forall g \in G \tag{3.11}$$

This has the form of a standard equivariance constraint, which allows conventional approaches to enforcing group equivariance, e.g. [34]. In this chapter, we will enforce group equivariance using the Symmetrizer [173]. Before training, the Symmetrizer enforces group (e.g. rotational) symmetries by projecting neural network weights onto an equivariant subspace, and then uses SVD to find a basis for the equivariant subspace. Then, during and after training, the weight matrix of the neural network is realised as a linear combination of equivariant basis weights, and the coefficients of the linear combination are updated during training with PPO [153]. We use ReLU non-linearities and regular representations. For more details on the Symmetrizer, we refer the reader to [173].

After encoding the input observations with the equivariant encoding function ϕ_e, we have an equivariant encoding of the local states that has a compositional form: the rotation of the state is represented by the ordering of *group channels* (see [173]) and the other state features are represented by the information in those channels. Similarly, we constrain the message update functions to be equivariant to the permutation L_g in the group channels of the state encodings and the rotation U_g of a difference vector e_{ij}, representing the edge (i, j):

$$L_g \left[\phi_m \left(e_{ij}, f_j^{(l)}\right)\right] = \phi_m \left(U_g \left[e_{ij}\right], L_g \left[f_j^{(l)}\right]\right) \tag{3.12}$$

Since linear combinations of equivariant functions are equivariant as well [35], the aggregated message $m_i^{(l)} = \sum_j^{|\mathcal{N}_i|} m_{j \to i}$ is equivariant too. While e_{ij} and $f_j^{(l)}$ transform under the same group G, they do not transform under the same group *action*: e_{ij} is a vector that transforms with a rotation matrix, whereas $f_j^{(l)}$ transforms with a permutation of group channels. The question arises how to build group equivariant layers that transform both the edge and the agent features appropriately. The method we use is to build equivariant layers using direct sum representations, where the representations U_g and L_g are combined as follows:

$$T_g = U_g \oplus L_g = \begin{bmatrix} U_g & 0 \\ 0 & L_g \end{bmatrix} \tag{3.13}$$

where 0 represents a zero-matrix of the appropriate size. Consider a weight matrix W^l acting on $\begin{bmatrix} e_{ij} \\ f_j^{(l)} \end{bmatrix}$. The equivariance constraint then becomes $W^{(l)} T_g = L_g W^{(l)}$.

To preserve the geometric information coming from the messages, the node update function is similarly constrained to be equivariant. Importantly, the permutation on the outputs of ϕ_u must match the permutation on the inputs of the next layer's ϕ_m (i.e. the output of one layer must use the same group representation as the input of the next layer). This ensures that we can add multiple layers together while preserving the geometric information. In practice, it is convenient to use a single representation L_g for all permutation representations.

$$L_g \left[\phi_u(f_i^{(l)}, m_i^{(l)})\right] = \phi_u \left(L_g[f_i^{(l)}], L_g[m_i^{(l)}]\right) \tag{3.14}$$

Finally, after M layers (M message passing rounds), we output local equivariant policies based on the state encodings at layer M using local policy network π:

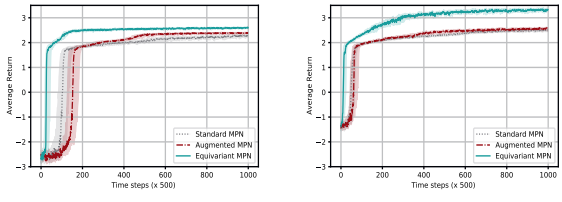
$$\pi_i \left(L_g \left[f_i^{(M)}\right]\right) = P_g \left[\pi_i \left(f_i^{(M)}\right)\right] \tag{3.15}$$

Here, P_g is the permutation representation on the actions of the individual agent, e.g. if in a grid world the state is flipped, P_g is the matrix that permutes the left and right actions accordingly.

3.5 Experiments

The evaluation of Multi Agent MDP Homomorphic networks has a singular goal: to investigate and quantify the effect of distributed versions of global equivariance in symmetric cooperative multi-agent reinforcement learning. We compare to three baselines. The first is a non-homomorphic variant of our network. This is a standard MPN, which is a permutation equivariant multi-agent graph network but not equivariant to global rotations. The other two are variants with symmetric data augmentation, in the spirit of [107, 99]. For a stochastic data augmentation baseline, on each forward pass one of the group elements is sampled and used to transform the input, and appropriately transform the output as well. For a full augmentation baseline, every state and policy is augmented with all its rotations in the group. For evaluation, we use the common centralized training, decentralized execution paradigm [100, 133] (see Appendix 3.A for more details). We train in a centralized fashion, with PPO [153], which will adjust the coefficients of the weight matrices in the network. The information available to the actors and critics is their local information and the information received from neighbors. We first evaluate on a wildlife monitoring task, a variant of predator-prey type problems with pixel-based inputs where agents can have information none of the other agents have. Additionally, we evaluate the networks on the more complex coordination problem of traffic light control, with pixel-based inputs. We focus on C_4 as the discrete group to investigate whether equivariance improves multi-agent systems, as C_4 has been shown to be effective in supervised learning and single-agent settings.

Wildlife Monitoring



(a) 3 agents.

(b) 4 agents.

Figure 3.5.1: *Results for the distributed drone wildlife monitoring task. 25%, 50% and 75% quantiles shown over 15 random seeds. All approaches tuned over 6 learning rates.*

Setup We evaluate on a distributed wildlife monitoring setup, where a set of drones has to coordinate to trap poachers. To trap a poacher, one drone has to hover above them while the other assists from the side, and for each drone that assists the team receives +1 reward. Two drones cannot be in the same location at the same time. Since the drones have only cameras mounted at the bottom, they cannot see each other. The episode ends when the poacher is trapped by at least 2 drones, or 100 time steps have passed. On each time step the team gets -0.05 reward. All agents (including the poacher) can stand still or move in the compass directions. The poacher samples actions uniformly. We train for 500k time steps. The drones can send communications to drones within a 3 by 3 radius around their current location, meaning that the problem is a distributed MMDP. Due to changing agent locations and the limited communication radius, the communication graph is dynamic and can change between time steps. The observations are 21 by 21 images representing a agent-centric view of a 7 by 7 toroidal grid environment that shows where the target is relative to the drone. While the grid is toroidal, the communication distance is not: at the edges of the grid, communication is blocked. This problem exhibits 90 degree rotations: when the global state rotates, the agents' local policies should permute, and so should the probabilities assigned to the actions in the local policies.

Results Results for this task are shown in Figure 3.5.1, with on the y-axis the average return and on the x-axis the number of time steps. In both the 3-agent and 4-agent case, using a Multi-Agent MDP Homomorphic Network improves compared to using MPNs without symmetry information, and compared to using symmetric data augmentation. We conclude that in the proposed task, our approach learns effective joint policies in fewer environment interactions compared to the baselines.

Traffic Light Control

For a second experiment, we focus on a more complex coordination problem: reducing vehicle wait times in traffic light control. Traffic light control constitutes a longstanding and open problem (see [185] for an overview): not only is the optimal coordination strategy non-obvious, traffic light control is a problem where wrong decisions can quickly lead to highly suboptimal states due to congestion. We use this setting to answer the following question: does enforcing symmetries help in complex coordination problems?

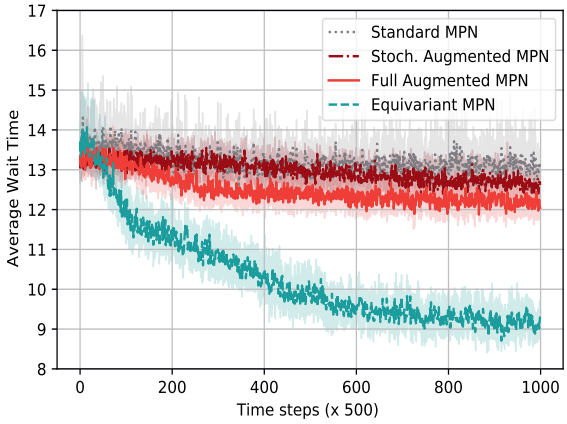


Figure 3.5.2: *Average vehicle wait times for distributed settings of the traffic light control task. Graphs show 25%, 50% and 75% quantiles over 20 independent training runs. All approaches tuned over 6 learning rates.*

Setup We use a traffic simulator with four traffic lights. On each of eight entry roads, for 100 time steps, a vehicle enters the simulation on each step with probability 0.1. Each agent controls the lights of a single intersection and has a local action space of (`grgr`, `rgrg`), indicating which two of its four lanes get a red or green light. Vehicles move at a rate of one unit per step, unless they are blocked by a red light or a vehicle. If blocked, the vehicle needs one step to restart. The goal is reducing the average vehicle waiting time. The simulation ends after all vehicles have exited the system, or after 500 steps. The team reward is $-\frac{1}{1000}\frac{1}{C}\sum_{c\in C} w(c)$, with C the vehicles in the system and $w(c)$ vehicle c's cumulative waiting time.

Results We show results in Figure 3.5.2. While the standard MPN architecture had reasonable performance on the toy problem, it takes many environment interactions to improve the policy in the more complex coordination problem presented by traffic light control. Adding data augmentation helps slightly. However, we see that enforcing the global symmetry helps the network find an effective policy much faster. In this setting, the coordination problem is hard to solve: in experiments with centralized controllers, the standard baseline performs better, though it is still slower to converge than the equivariant centralized controller. Overall, enforcing global symmetries in distributed traffic light control leads to effective policies in fewer environment interactions.

3.6 E(3) Equivariance

Many tasks can benefit from making use of their inherent symmetries and structure. We saw in this Chapter that we can construct graph networks that are equivariant to global transformations while still allowing for distributed execution. Such networks make use of both the structure and the symmetry of cooperative multi-agent problems.

In this Chapter and Chapter 2, we consider symmetries in decision making problems, looking in particular at discrete groups of 2 or 4 elements using regular representations. However, many real world problems exhibit symmetries that are continuous, for example as is the case in robotics [203, 179, 136] or molecular design [157]. For that reason, later work has looked at a broader class of symmetries in single agent learning, such as the SE(2) [203], SO(2) [179], SO(3) [136, 157] groups. In this Section, we discuss Steerable E(3) Equivariant graph networks (SEGNNs), which take the ideas presented in this Chapter a step further: we consider the E(3) group, the group of all isometries of 3d Euclidean space (rotations, reflections, and translations). SEGNNs are a class of E(3) equivariant graph neural networks that use non-linear convolutions. SEGNNs show promising results on physics and chemistry tasks, where for example molecular data tends to exhibit both graph structure and E(3) symmetry. See Figure 3.6.1.

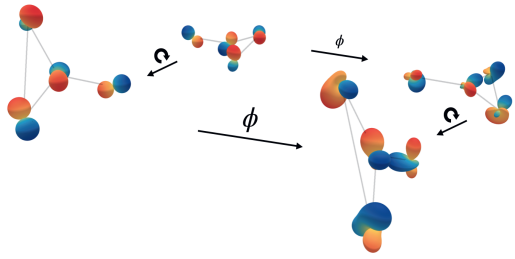


Figure 3.6.1: Equivariance diagram for equivariant operator ϕ for 3D molecular graph with steerable node features. If the molecule rotates, the node features do as well.

Earlier work considered equivariant convolutional neural networks [34, 35, 193, 32, 98, 187, 14, 13, 186]. Similarly, there is earlier work on equivariant GNN architectures [195, 101, 166, 7, 52, 50]. It turns out that where early work tends to linearly transform the node and edge features[2] followed by a non-linearity, Multi-Agent MDP Homomorphic Networks and SEGNNs use *non-linear* message aggregation. SEGNNs in particular also use *steerable messages* (rather than invariant or discrete equivariant messages). SEGNNs use non-linear group convolutions, and can be seen as a generalization of EGNNs [147], which sends *invariant* messages, where SEGNNs send *equivariant* messages. SEGNNs are able to perform non-linear convolutions with rotation-

[2] Pseudo-linearly for methods using attention [52, 7]

ally invariant scalars or covariant vectors or tensors. Since molecular data sets can include node information such as velocity, force or atomic spin, SEGNNs are evaluated on n-Body toy datasets, QM9, and OC20, where they are able to use the geometric and physical attributes present in the data. SEGNNs set a new state of the art on n-Body toy datasets. Additionally, SEGNNs are a new state of the art in ISRE of OC20 and competitive on QM9. For more details, see [23]. Seeing the potential of SEGNNs on molecular prediction tasks, it is likely they can provide possible benefits when adapted into Multi-agent MDP Homomorphic Networks for E(3) Equivariant multi-agent policy networks following the ideas in this Chapter, for example for applications in robotics or molecular design.

3.7 Conclusion

We consider distributed cooperative multi-agent systems that exhibit global symmetries. In particular, we propose a factorization of global symmetries into symmetries on local observations and local interactions. On this basis, we propose Multi-Agent MDP Homomorphic Networks, a class of policy networks that allows distributed execution while being equivariant to global symmetries. We compare to non-equivariant distributed networks, and show that global equivariance improves data efficiency on both a predator-prey variant, and on the complex coordination problem of traffic light control.

Scope We focus on discrete groups. For future work, this could be generalized by using steerable representations, at the cost of not being able to use pointwise nonlinearities. We also focus on discrete actions. This might be generalized by going beyond action permutations, e.g., for 2D continuous worlds a continuous rotation of the actions. Furthermore, our approach uses group channels for each layer (regular representations). For small groups this is not an issue, but for much larger groups this would require infeasible computational resources. Finally, this work has focused on exact symmetries and considers imperfect symmetries and violations of symmetry constraints a promising future topic.

3.8 Ethics Statement

Our work has several potential future applications, e.g. in autonomous driving, decentralized smart grids or robotics. Such applications hopefully have a positive societal impact, but there are also risks of negative societal impact: through the application itself (e.g. military), labor market impact, or by use in safety-critical applications without proper verification and validation. These factors should be taken into account when developing such applications.

3.9 Reproducibility Statement

To ensure reproducibility, we describe our setup in the Experiments section and we include hyperparameters, group actions, and architecture details in the Appendix. We will also make our code publicly available.

Chapter Appendix

3.A Message Passing Networks, Communication, and Distribution

Message passing algorithms are commonly used to coordinate between agents while allowing for factorization of the global decision making function [64, 105, 171, 18]. Message passing networks approximate such message passing algorithms [199, 148]. Thus, we can view message passing networks as a type of learned communication between coordinating agents. Message passing networks can be executed using only local communication and computation. To see that this is the case, consider the equations that determine the message passing networks in this paper:

$$f_i^{(0)} = \phi_e(s_i) \tag{3.16}$$

$$m_{j\to i}^{(l)} = \phi_m(e_{ij}, f_j^{(l)}) \tag{3.17}$$

$$m_i^{(l)} = \sum_j^{|\mathcal{N}_i|} m_{j\to i} \tag{3.18}$$

$$f_i^{(l+1)} = \phi_u(f_i^{(l)}, m_i^{(l)}) \tag{3.19}$$

for all agents i and layers (message passing rounds) l. In Eq. 3.16, each agent encodes its local observation s_i into a local feature vector $f_i^{(0)}$. In Eq. 3.17, each agent j computes its message to agent i using its own local feature vector $f_j^{(l)}$ and its shared edge features $e_{ij} = x_i - x_j$. After agent i receives its neighbors' messages, it aggregates them in Eq. 3.18. Finally, in Eq. 3.19 agent i updates its local feature vector using the aggregated message and its local feature vector. Clearly, every step in this network requires only local computation and local communication, therefore allowing the network to be distributed over agents at execution time.

3.B Equivariance of Proposed Message Passing Layers

Recall that $e_{ij} = x_i - x_j$ (the difference between the locations of agent i and agent j). Therefore,

$$U_g[e_{ij}] = U_g[x_i - x_j] = U_g[x_i] - U_g[x_j] \tag{3.20}$$

So, when both agent i and agent j are moved to a location transformed by U_g, the edge features e_{ij} are transformed by U_g as well. If all agent positions and observations rotate by the same underlying group, this means the full system has rotated. In this paper, we place equivariance constraints on the message function:

$$K_g[\sum_j \phi_m(e_{ij}, f_j)] = \sum_j \phi_m(U_g[e_{ij}], L_g[f_j]) \tag{3.21}$$

This means that the messages are only permuted by K_g if both the local features f_j *and* the edge features e_{ij} are transformed (by L_g and U_g respectively). To see that the proposed message passing layers are equivariant to transformations on agent features and edge features, consider the following example. Assume we have a message passing layer ϕ consisting of node update function ϕ_u and message update function ϕ_m, such that with agent features $\{f_i\}$, and edge features $\{e_{ij}\}$, the output for agent i is given by

$$\phi^{(i)}\left(\{e_{ij}\}, \{f_j\}\right) = \phi_u\left(f_i, m_i\right) \tag{3.22}$$

$$= \phi_u\left(f_i, \sum_j \phi_m\left(e_{ij}, f_j\right)\right) \tag{3.23}$$

Assume ϕ_u and ϕ_m are constrained to be equivariant in the following way:

$$P_g[\phi_u(f_i, m_i)] = \phi_u(L_g[f_i], K_g[m_i]) \tag{3.24}$$

$$K_g[\sum_j \phi_m(e_{ij}, f_j)] = \sum_j \phi_m(U_g[e_{ij}], L_g[f_j]) \tag{3.25}$$

for group elements $g \in G$. Then, for layer ϕ:

$$P_g\left[\phi^{(i)}\left(\{e_{ij}\},\{f_j\}\right)\right] = P_g\left[\phi_u\left(f_i, m_i\right)\right] \tag{3.26}$$

$$= P_g\left[\phi_u\left(f_i, \sum_j \phi_m\left(e_{ij}, f_j\right)\right)\right] \tag{3.27}$$

$$= \phi_u\left(L_g\left[f_i\right], K_g\left[\sum_j \phi_m\left(e_{ij}, f_j\right)\right]\right) \tag{3.28}$$

(using Eq. 3.24) (3.29)

$$= \phi_u\left(L_g\left[f_i\right], \sum_j \phi_m\left(U_g\left[e_{ij}\right], L_g\left[f_j\right]\right)\right) \tag{3.30}$$

(using Eq. 3.25) (3.31)

$$= \phi^{(i)}\left(\{U_g\left[e_{ij}\right]\},\{L_g\left[f_j\right]\}\right) \tag{3.32}$$

3.C Discrete Rotations of Continuous Vectors

Here we outline how to build weight matrices equivariant to discrete rotations of continuous vectors. Let $e_{ij} = x_j - x_i$ be an arbitrary, continuous difference vector between the coordinates of agent j and the coordinates of agent i. Then this difference vector transforms under 90 degree rotations using the group of standard 2D rotation matrices of the form

$$R_g = R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{3.33}$$

for $\theta \in [0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}]$. A weight matrix W is now equivariant to 90 degree rotations of e_{ij} if

$$K_g W e_{ij} = W R_g e_{ij} \tag{3.34}$$

with $\{K_g\}_{g\in G}$ e.g. a permutation matrix representation of the same group. So,

$$W = K_g^{-1} W R_g \tag{3.35}$$

which we can solve using standard approaches.

3.D Experimental Details

For all approaches, including baselines, we run at least 15 random seeds for 6 different learning rates, $\{0.001, 0.003, 0.0001, 0.0003, 0.00001,$

0.00003}, and report the best learning rate for each. Other hyperparameters are taken as default in the codebase [159, 173].

Learning Rates Reported

After tuning the learning rate, we report the best one for each approach. See Table 3.D.1.

Distributed Settings	Standard	Augmented	Equivariant
Drones, 3 agents	0.001	0.0003	0.001
Drones, 4 agents	0.0003	0.001	0.001
Traffic, 4 agents	0.0001	0.0001	0.0001

Table 3.D.1: Best learning rates for distributed settings for MPNs.

Assets Used

- Numpy [68] 1.19.2: BSD 3-Clause "New" or "Revised" License;
- PyTorch [137] 1.2.0: Modified BSD license;
- RLPYT [159]: MIT license;
- MDP Homomorphic Networks & Symmetrizer [173]: MIT license.

3.E Architectural details

Architectures are given below and were chosen to be as similar as possible between different approaches, keeping the number of trainable parameters comparable between approaches. We chose 2 message passing layers to allow for 2 message passing hops. For the message passing networks, we use L1-normalization of the adjacency matrix.

Architectural Overview

The global structure of our network is given in Figure 3.E.1.

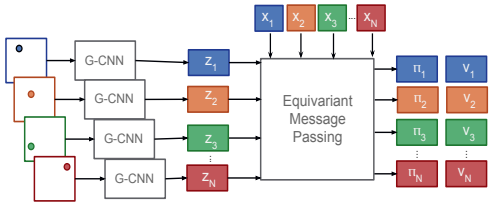


Figure 3.E.1: General overview of Multi-Agent MDP Homomorphic Networks. G-CNN refers to a group-equivariant CNN encoder. Equivariant message passing refers to the proposed equivariant message passing networks. Encoding local states with group-CNNs ensure the state encodings z_i are group-equivariant. The locations x_i are input to the equivariant message passing network.

Architectures

Wildlife Monitoring

Listing 3.1: Equivariant Network Architecture for Centralized Drones

```
EqConv2d(repr_in=1, channels_in=m, repr_out=4, channels_out=⌊16/√4⌋,
            filter_size=(7, 7), stride=2, padding=0)
ReLU()
EqConv2d(repr_in=4, channels_in=⌊16/√4⌋, repr_out=4, channels_out=⌊32/√4⌋,
            filter_size=(5, 5), stride=1, padding=0)
ReLU()
GlobalMaxPool()
EqLinear(repr_in=4, channels_in=⌊32/√4⌋, repr_out=4, channels_out=⌊128/√4⌋)
ReLU()
EqLinear(repr_in=4, channels_in=⌊128/√4⌋, repr_out=5, channels_out=⌊64/√4⌋)
ReLU()
ModuleList([EqLinear(repr_in=4, channels_in=⌊64/√4⌋, repr_out=5,
  channels_out=1) for i in range(m)])
EqLinear(repr_in=4, channels_in=⌊64/√4⌋, repr_out=1, channels_out=1)
```

Listing 3.2: CNN Architecture for Centralized Drones

```
Conv2d(channels_in=m, channels_out=16,
            filter_size=(7, 7), stride=2, padding=0)
ReLU()
Conv2d(channels_in=16,channels_out=32,
            filter_size=(5, 5), stride=1, padding=0)
ReLU()
GlobalMaxPool()
Linear(channels_in=32, channels_out=256)
ReLU()
ModuleList([Linear(channels_in=256,
  channels_out=5) for i in range(m)])
Linear(channels_in=256, channels_out=1)
```

Listing 3.3: Equivariant Network Architecture for Distributed Drones

```
EqConv2d(repr_in=1, channels_in=1, repr_out=4, channels_out=⌊16/√4⌋,
            filter_size=(7, 7), stride=2, padding=0)
ReLU()
EqConv2d(repr_in=4, channels_in=⌊16/√4⌋, repr_out=4, channels_out=⌊32/√4⌋,
            filter_size=(5, 5), stride=1, padding=0)
ReLU()
GlobalMaxPool()
EqMessagePassingLayer(repr_in=4+4+2, channels_in=⌊32/√4⌋, repr_out=4,
  channels_out=⌊64/√4⌋)
ReLU()
EqMessagePassingLayer(repr_in=4+4+2, channels_in=⌊64/√4⌋, repr_out=4,
  channels_out=⌊64/√4⌋)
```

```
ReLU()
EqMessagePassingLayer(repr_in=4, channels_in=⌊64/√4⌋, repr_out=5,
  channels_out=1)
EqMessagePassingLayer(repr_in=4, channels_in=⌊64/√4⌋, repr_out=1,
  channels_out=1)
```

Listing 3.4: MPN Architecture for Distributed Drones

```
Conv2d(channels_in=1, channels_out=16,
            filter_size=(7, 7), stride=2, padding=0)
ReLU()
Conv2d(channels_in=16, channels_out=32,
            filter_size=(5, 5), stride=1, padding=0)
ReLU()
GlobalMaxPool()
MessagePassingLayer(channels_in=32+32+2, channels_out=64)
ReLU()
MessagePassingLayer(channels_in=64+64+2, channels_out=64)
ReLU()
Linear(channels_in=64, channels_out=5)
Linear(channels_in=64, channels_out=1)
```

Traffic Light Control

Listing 3.5: Equivariant Network Architecture for Centralized Traffic

```
EqConv2d(repr_in=1, channels_in=3, repr_out=4, channels_out=⌊16/√4⌋,
            filter_size=(7, 7), stride=2, padding=0)
ReLU()
EqConv2d(repr_in=4, channels_in=⌊16/√4⌋, repr_out=4, channels_out=⌊32/√4⌋,
            filter_size=(5, 5), stride=1, padding=0)
ReLU()
GlobalMaxPool()
EqLinear(repr_in=4, channels_in=⌊32/√4⌋, repr_out=4, channels_out=⌊128/√4⌋)
ReLU()
EqLinear(repr_in=4, channels_in=⌊128/√4⌋, repr_out=5, channels_out=⌊64/√4⌋)
ReLU()
EqLinear(repr_in=4, channels_in=⌊64/√4⌋, repr_out=8, channels_out=1)
EqLinear(repr_in=4, channels_in=⌊64/√4⌋, repr_out=1, channels_out=1)
```

Listing 3.6: CNN Architecture for Centralized Traffic

```
Conv2d(channels_in=3, channels_out=16,
            filter_size=(7, 7), stride=2, padding=0)
ReLU()
Conv2d(channels_in=16,channels_out=32,
            filter_size=(5, 5), stride=1, padding=0)
ReLU()
GlobalMaxPool()
Linear(channels_in=32, channels_out=256)
```

```
ReLU()
Linear(channels_in=256, channels_out=8)
Linear(channels_in=256, channels_out=1)
```

Listing 3.7: Equivariant Network Architecture for Distributed Traffic

```
EqConv2d(repr_in=1, channels_in=3, repr_out=4, channels_out=⌊16/√4⌋,
            filter_size=(7, 7), stride=2, padding=0)
ReLU()
EqConv2d(repr_in=4, channels_in=⌊16/√4⌋, repr_out=4, channels_out=⌊32/√4⌋,
            filter_size=(5, 5), stride=1, padding=0)
ReLU()
GlobalMaxPool()
EqMessagePassingLayer(repr_in=4+4+2, channels_in=⌊32/√4⌋, repr_out=4,
  channels_out=⌊64/√4⌋)
ReLU()
EqMessagePassingLayer(repr_in=4+4+2, channels_in=⌊64/√4⌋, repr_out=4,
  channels_out=⌊64/√4⌋)
ReLU()
EqMessagePassingLayer(repr_in=4, channels_in=⌊64/√4⌋, repr_out=2,
  channels_out=1)
EqMessagePassingLayer(repr_in=4, channels_in=⌊64/√4⌋, repr_out=1,
  channels_out=1)
```

Listing 3.8: MPN Architecture for Distributed Traffic

```
Conv2d(channels_in=3, channels_out=16,
            filter_size=(7, 7), stride=2, padding=0)
ReLU()
Conv2d(channels_in=16, channels_out=32,
            filter_size=(5, 5), stride=1, padding=0)
ReLU()
GlobalMaxPool()
MessagePassingLayer(channels_in=32+32+2, channels_out=64)
ReLU()
MessagePassingLayer(channels_in=64+64+2, channels_out=64)
ReLU()
Linear(channels_in=64, channels_out=2)
Linear(channels_in=64, channels_out=1)
```

Group Actions

Here we list the group actions used in different equivariant layers throughout our experiments. For all equivariant layers, we use the Symmetrizer [173] to find equivariant weight bases.

Rotation-equivariant Filters For all equivariant encoder networks, we create 90 degree rotation-equivariant filters using `np.rot90`.

Linear layers Permutation matrices representing the following permutations:
$e = [0, 1, 2, 3]$
$g_1 = [3, 0, 1, 2]$
$g_2 = [2, 3, 0, 1]$
$g_3 = [1, 2, 3, 0]$

Policy layers, centralized Permutation matrices representing the following permutations:
$e = [0, 1, 2, 3, 4]$
$g_1 = [0, 2, 3, 4, 1]$
$g_2 = [0, 3, 4, 1, 2]$
$g_3 = [0, 4, 1, 2, 3]$

Value layers, centralized Permutation matrices representing the following permutations:
$e = [1]$
$g_1 = [1]$
$g_2 = [1]$
$g_3 = [1]$

Message Passing Layers Acting on state features, permutation matrices representing the following permutations:
$e = [0, 1, 2, 3]$
$g_1 = [3, 0, 1, 2]$
$g_2 = [2, 3, 0, 1]$
$g_3 = [1, 2, 3, 0]$
Acting on edge features, the following rotation matrices:
e=`np.eye(2)`
g_1=`np.array([[0, -1], [1, 0]])`
g_2=`np.array([[-1, 0], [0, -1]])`
g_3=`np.array([[0, 1], [-1, 0]])`

Policy layers, distributed Permutation matrices representing the following permutations:
$e = [0, 1, 2, 3, 4]$
$g_1 = [0, 2, 3, 4, 1]$
$g_2 = [0, 3, 4, 1, 2]$

$g_3 = [0, 4, 1, 2, 3]$

Value layers, distributed Permutation matrices representing the following permutations:
$e = [1]$
$g_1 = [1]$
$g_2 = [1]$
$g_3 = [1]$

Group Actions for Traffic Light Control

Linear layers Permutation matrices representing the following permutations:
$e = [0, 1, 2, 3]$
$g_1 = [3, 0, 1, 2]$
$g_2 = [2, 3, 0, 1]$
$g_3 = [1, 2, 3, 0]$

Policy layers, centralized Permutation matrices representing the following permutations:
$e = [0, 1, 2, 3, 4, 5, 6, 7]$
$g_1 = [5, 4, 1, 0, 7, 6, 3, 2]$
$g_2 = [6, 7, 4, 5, 2, 3, 0, 1]$
$g_3 = [3, 2, 7, 6, 1, 0, 5, 4]$

Value layers, centralized Permutation matrices representing the following permutations:
$e = [1]$
$g_1 = [1]$
$g_2 = [1]$
$g_3 = [1]$

Message Passing Layers Acting on state features, permutation matrices representing the following permutations:
$e = [0, 1, 2, 3]$
$g_1 = [3, 0, 1, 2]$
$g_2 = [2, 3, 0, 1]$
$g_3 = [1, 2, 3, 0]$
Acting on edge features, the following rotation matrices:

e =`np.eye(2)`
g_1 =`np.array([[0, -1], [1, 0]])`
g_2 =`np.array([[-1, 0], [0, -1]])`
g_3 =`np.array([[0, 1], [-1, 0]])`

Policy layers, distributed Permutation matrices representing the following permutations:
$e = [0, 1]$
$g_1 = [1, 0]$
$g_2 = [0, 1]$
$g_3 = [1, 0]$

Value layers, distributed Permutation matrices representing the following permutations:
$e = [1]$
$g_1 = [1]$
$g_2 = [1]$
$g_3 = [1]$

Part II

Structure

4

Plannable Approximations to MDP Homomorphisms

4.1 Introduction

For this part of the dissertation, we turn our attention to the problem of learning representations for decision making problems. In this Chapter, we learn the structure of the environment, and propose the concept of *action-equivariance* as a generalization of group-equivariance. In the following Chapter we will capture the structure in individual states.

Dealing with high dimensional state spaces and unknown environmental dynamics presents an open problem in decision making [70]. Classical dynamic programming approaches require knowledge of environmental dynamics and low dimensional, tabular state spaces [140]. Recent deep reinforcement learning methods on the other hand offer good performance, but often at the cost of being unstable and sample-hungry [70, 124, 77, 125]. The deep model-based reinforcement learning literature aims to fill this gap, for example by finding policies after learning models based on input reconstruction [103, 66, 202, 36], by using environmental models in auxiliary losses [45, 77], or by forcing network architectures to resemble planning algorithms [164, 132]. While effective in learning end-to-end policies, these types of approaches are not forced to learn good representations and may thus not build proper environmental models. In this work, we focus on learning representations of the world that are suitable for exact planning methods.

To combine dynamic programming with the representational power of deep networks, we factorize the online decision-making problem into a self-supervised model learning stage and a dynamic programming stage.

We do this under the assumption that good representations minimize MDP metrics [57, 46, 113, 165]. While such metrics have desirable theoretical guarantees, they require an enumerable state space and knowledge of the environmental dynamics, and are thus not usable in many problems. To resolve this issue, we propose to learn representations using the more flexible notion of action equivariant mappings, where the effects of actions in input space are matched by equivalent action effects in the latent space. See Figure 4.1.1.

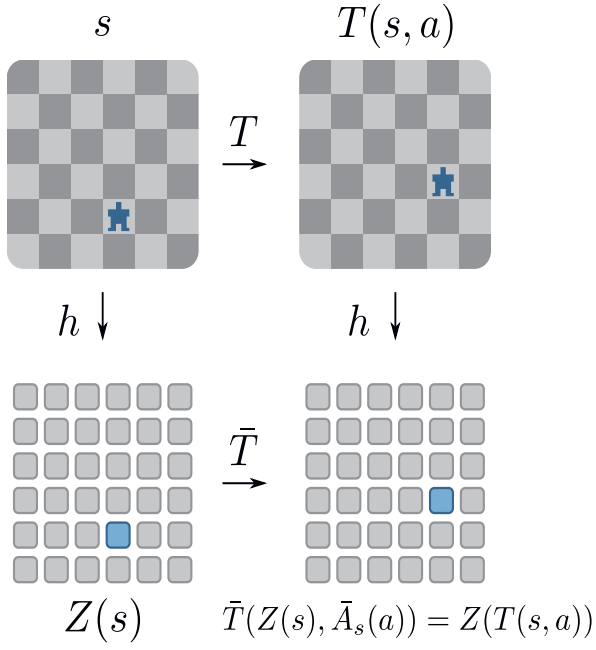


Figure 4.1.1: Visualization of the notion of equivariance under actions. We say Z is an action equivariant mapping if $Z(T(s,a)) = K_g^s(Z(s), \bar{A}_s(a))$.

We make the following contributions. First, we propose learning an equivariant map and corresponding action embeddings. This corresponds to using MDP homomorphism metrics [165] of deterministic MDPs, enabling planning in the homomorphic image of the original MDP. Second, we prove that for deterministic MDPs, when our loss is zero, we have an MDP homomorphism [143]. This means that the resulting policy can be lifted to the original MDP. Third, we provide experimental evaluation in a variety of settings to show 1) that we can recover the graph structure of the input MDP, 2) that planning in this

abstract space results in good policies for the original space, 3) that we can change to arbitrary new goal states without further gradient descent updates and 4) that this works even when the input states are continuous, or when generalizing to new instances with the same dynamics.

4.2 Background

Markov Decision Processes An infinite horizon Markov Decision Process (MDP) is a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, R, T, \gamma)$, where $s \in \mathcal{S}$ is a Markov state, $a \in \mathcal{A}$ is an action that an agent can take, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function that returns a scalar signal r defining the desirability of some observed transition, $0 \leq \gamma \leq 1$ is a discount factor that discounts future rewards exponentially and $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is a transition function, that for a pair of states and an action assigns a probability of transitioning from the first to the second state. The goal of an agent in an MDP is to find a policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, a function assigning probabilities to actions in states, that maximizes the *return* $G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$. The expected return of a state, action pair under a policy π is given by a Q-value function $Q_\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ where $Q_\pi(s, a) = \mathbb{E}_\pi [G_t | s_t = s, a_t = a]$. The value of a state under an optimal policy π^* is given by the value function $V^* : \mathcal{S} \to \mathbb{R}$, defined as $V^* = \max_a Q^*(s, a)$ under the Bellman optimality equation.

Value Iteration Value Iteration (VI) is a dynamic programming algorithm that finds Q-values in MDPs, by iteratively applying the Bellman optimality operator. This can be viewed as a graph diffusion where each state is a vertex and transition probabilities define weighted edges. VI is guaranteed to find the optimal policy in an MDP. For more details, see [140].

Bisimulation Metrics To enable computing optimal policies in MDPs with very large or continuous state spaces, one approach is aggregating states based on their similarity in terms of environmental dynamics [38, 113]. A key concept is the notion of *stochastic bisimulations* for MDPs, which was first introduced by [38]. Stochastic bisimulation defines an equivalence relation on MDP states based on matching reward and transition functions, allowing states to be compared to each other. Later work [46] observes that the notion of stochastic bisimulation is too stringent (the dynamics must match exactly) and proposes using a

more general *bisimulation metric* instead, with the general form

$$d(s,s') = \max_a \Big(c_R |R(s,a) - R(s',a)| + c_T d_P(T(s,a), T(s',a)) \Big) \quad (4.1)$$

where c_R and c_T are weighting constants, $T(\cdot, a)$ is a distribution over next states and d_P is some probability metric, such as the Kantorovich (Wasserstein) metric. Such probability metrics are recursively computed. For more details, see [46]. The bisimulation metric provides a distance between states that is not based on input features but on environmental dynamics.

MDP Homomorphism A generalization of the mapping induced by bisimulations is the notion of MDP homomorphisms [143]. MDP homomorphisms were introduced by [141] as an extension of [38]. An MDP homomorphism $(\sigma, \{\alpha_s | s \in \mathcal{S}\})$ is a tuple of functions $\langle Z, \{\bar{A}_s\} \rangle$ with $Z : \mathcal{S} \to \mathcal{Z}$ a function that maps states to abstract states, and each $\bar{A}_s : \mathcal{A} \to \bar{\mathcal{A}}$ a state-dependent function that maps actions to abstract actions, that preserves the structure of the input MDP. We use the definition given by [143]:

Definition 5 (Stochastic MDP Homomorphism) *A* Stochastic MDP homomorphism *from a stochastic MDP* $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, T, R \rangle$ *to an MDP* $\bar{\mathcal{M}} = \left\langle \mathcal{Z}, \bar{\mathcal{A}}, K^s_g, \bar{R} \right\rangle$ *is a tuple* $(\sigma, \{\alpha_s | s \in \mathcal{S}\}) = \langle Z, \{\bar{A}_s\} \rangle$ *, with*

- $Z : \mathcal{S} \to \mathcal{Z}$ *the state embedding function, and*
- $\bar{A}_s : \mathcal{A} \to \bar{\mathcal{A}}$ *the action embedding functions,*

such that the following identities hold:

$$\forall_{s,s' \in \mathcal{S}, a \in \mathcal{A}} \quad K^s_g(Z(s')|Z(s), \bar{A}_s(a)) = \sum_{s'' \in [s']_Z} T(s''|s,a) \quad (4.2)$$

$$\forall_{s \in \mathcal{S}, a \in \mathcal{A}} \quad \bar{R}(Z(s), \bar{A}_s(a)) = R(s,a) \quad (4.3)$$

Here, $[s']_Z = Z^{-1}(Z(s'))$ *is the equivalence class of* s' *under* Z*.*

We specifically consider deterministic MDPs. In that case:

Definition 6 (Deterministic MDP Homomorphism) *A* Deterministic MDP homomorphism *from a deterministic MDP* $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, T, R \rangle$ *to an MDP* $\bar{\mathcal{M}} = \left\langle \mathcal{Z}, \bar{\mathcal{A}}, K^s_g, \bar{R} \right\rangle$ *is a tuple* $(\sigma, \{\alpha_s | s \in \mathcal{S}\}) = \langle Z, \{\bar{A}_s\} \rangle$ *, with*

- $Z : \mathcal{S} \to \mathcal{Z}$ *the state embedding function, and*
- $\bar{A}_s : \mathcal{A} \to \bar{\mathcal{A}}$ *the action embedding functions,*

such that the following identities hold:

$$\forall_{s,s' \in \mathcal{S}, a \in \mathcal{A}} \quad T(s,a) = s' \implies K_g^s(Z(s), \bar{A}_s(a)) = Z(s') \tag{4.4}$$

$$\forall_{s \in \mathcal{S}, a \in \mathcal{A}} \quad \bar{R}(Z(s), \bar{A}_s(a)) = R(s,a) \tag{4.5}$$

The states s are organized into equivalence classes under Z if they follow the same dynamics in z-space. The MDP $\bar{\mathcal{M}}$ is referred to as the *homomorphic image* of $\mathcal{M}$ under h [143]. An important property of MDP homomorphisms is that a policy optimal in homomorphic image $\bar{\mathcal{M}}$ can be *lifted* to an optimal policy in $\mathcal{M}$ [143, 69]. Looking at these definitions, it may be clear that MDP homomorphisms and bisimulation metrics are closely related. The difference is that the latter measures distances between two MDP states, while the former is a map from one MDP to another. However, the idea of forming a distance metric by taking a sum of the distances can be extended to homomorphisms, as proposed by [165]:

$$\begin{aligned} d((s,a),(Z(s),\bar{A}_s(a))) = c_R |R(s,a) - \bar{R}(Z(s),\bar{A}_s(a))| \\ + c_T d_P(ZT(s,a), K_g^s(Z(s),\bar{A}_s(a))), \end{aligned} \tag{4.6}$$

with d_P a suitable measure of the difference between distributions (e.g., Kantorovich metric), and $ZT(s,a)$ shorthand for projecting the distribution over next states into the space of $\mathcal{Z}$ (see [54] for details). We refer to this as the *MDP homomorphism metric*.

Action-Equivariance We define a mapping $Z : \mathcal{S} \to \mathcal{Z}$ to be action-equivariant if $Z(T(s,a)) = \bar{T}(Z(s), \bar{A}_s(a))$ and $R(s,a) = \bar{R}(Z(s), \bar{A}_s(a))$, i.e. when the constraints in Eq. 4.4 and Eq. 4.5 hold.

4.3 Learning MDP Homomorphisms

We are interested in learning compact, plannable representations of MDPs. We call MDP representations *plannable* if the optimal policy found by planning algorithms such as VI can be lifted to the original MDP and still be close to optimal. This is the case when the representation respects the original MDP's dynamics, such as when the equivariance constraints in Eq. 4.4 and Eq. 4.5 hold. In this chapter we leverage MDP homomorphism metrics to find such representations. In particular, we introduce a loss function that enforces these equivari-

ance constraints, then construct an abstract MDP in the learned representation space. We compute a policy in the abstract MDP $\bar{\mathcal{M}}$ using VI, and *lift* the abstract policy to the original space. To keep things simple, we focus on deterministic MDPs, but in preliminary experiments our method performed well out of the box on stochastic MDPs. Additionally, the framework we outline here can be extended to the stochastic case, as [54] does for bisimulation metrics.

Learning State Representations

Here we show how to learn state representations that respect action-equivariance. We embed the states in $\mathcal{S}$ into Euclidean space using a contrastive loss based on MDP homomorphism metrics. Similar losses have often been used in related work [54, 94, 6, 135, 51], which we compare in Section 4.5. We represent the mapping Z using a neural network parameterized by θ, whose output will be denoted Z_θ. This function maps a state $s \in \mathcal{S}$ to a latent representation $z \in \mathcal{Z} \subseteq \mathbb{R}^D$. We additionally approximate the abstract transition K^s_g by a function $\bar{T}_\phi : \mathcal{Z} \times \bar{\mathcal{A}} \to \mathcal{Z}$ parameterized by ϕ, and the abstract rewards $\bar{R}$ by a neural network $\bar{R}_\zeta : \mathcal{Z} \to \mathbb{R}$, parameterized by ζ, that predicts the reward for an abstract state. From Eq. 4.5 we simplify to a state-dependent reward using $R(s) = \bar{R}(Z(s))$ where $R(s)$ is the reward function that outputs a scalar value for an $s \in \mathcal{S}$, and $\bar{R}$ is its equivalent in $\bar{\mathcal{M}}$. During training, we first sample a set of *experience tuples* $\mathcal{D} = \{(s_t, a_t, r_t, s_{t+1})\}_{n=1}^N$ by rolling out an exploration policy π_e for K trajectories. To learn representations that respect Eq. 4.4 and 4.5, we minimize the distance between the result of transitioning in observation space, and then mapping to $\mathcal{Z}$, or first mapping to $\mathcal{Z}$ and then transitioning in latent space (see Figure 4.1.1). Additionally, the distance between the observed reward $R(s)$ and the predicted reward $\bar{R}_\zeta(Z_\theta(s))$ is minimized. We thus include a general reward loss term. We write $s'_n = T(s_n, a_n)$, $z_n = Z_\theta(s_n)$, and minimize

$$\mathcal{L}(\theta, \phi, \zeta) = \frac{1}{N} \sum_{n=1}^{N} \Big[d\left(Z_\theta(s'_n), \bar{T}_\phi(z_n, \bar{A}_\phi(z_n, a_n))\right) + d\left(R(s_n), \bar{R}_\zeta(z_n)\right) \Big] \tag{4.7}$$

by randomly sampling batches of experience tuples from $\mathcal{D}$. In this chapter, we use $d(z, z') = \frac{1}{2}(z - z')^2$ to model distances in $\mathcal{Z} \subseteq \mathbb{R}^D$. Here, $\bar{T}_\phi$ is a function that maps a point in latent space $z \in \mathcal{Z}$ to a new state $z' \in \mathcal{Z}$ by predicting an *action-effect* that acts on z. We adopt earlier approaches of letting $\bar{T}_\phi$ be of the form $\bar{T}_\phi(z, \bar{a}) = z + \bar{A}_\phi(z, a)$, where $\bar{A}_\phi(z, a)$ is a simple feedforward network [94, 51]. Thus $\bar{A}_\phi : \mathcal{Z} \times \mathcal{A} \to \bar{\mathcal{A}}$ is a function mapping from the original action space to

an abstract action space, and $\bar{A}_\phi(z, a)$ approximates $\bar{A}_s(a)$ (Eq. 4.4). The resulting transition loss is a variant of the loss proposed in [94]. The function $\bar{R}_\zeta : \mathcal{Z} \to \mathbb{R}$ predicts the reward from z. Since Z, K_g^s and $\bar{R}$ are neural networks optimized with SGD, Eq. 4.7 has a trivial solution where all states are mapped to the same point, especially in the sparse reward case. When the reward function is informative, minimizing Eq. 4.7 can suffice, as is empirically demonstrated in [54]. However, when rewards are sparse, the representations may collapse to the trivial embedding, and for more complex tasks [54] requires a pixel reconstruction term. In practice, earlier works use a variety of solutions to prevent the trivial map. Approaches based on pixel reconstructions are common [182, 183, 36, 104, 67, 85, 66, 202, 54], but there are also approaches based on self-supervision that use alternatives to reconstruction of input states [6, 94, 51, 135, 3, 9, 200].

To prevent trivial solutions, we use a contrastive loss, maximizing the distance between the latent next state and the embeddings of a set of random other states, $\mathcal{S}_\neg = \{s_j\}_{j=1}^J$ sampled from the same trajectory on every epoch. Thus, the complete loss is

$$\begin{aligned}\mathcal{L}(\theta, \phi, \zeta) = \frac{1}{N} \sum_{n=1}^{N} \Bigg[& d\left(Z_\theta(s'_n), \bar{T}_\phi(z_n, \bar{A}_\phi(z_n, a_n))\right) \\ & + d\left(R(s_n), \bar{R}_\zeta(z_n)\right) \\ & + \sum_{s_\neg \in \mathcal{S}_\neg} d_\neg\left(Z_\theta(s_\neg), \bar{T}_\phi(z_n, \bar{A}_\phi(z_n, a_n))\right) \Bigg] \end{aligned} \tag{4.8}$$

where $d_\neg$ is a negative distance function. Similar to [94], we use the hinge loss $d_\neg(z, z') = \max(0, \epsilon - d(z, z'))$ to prevent the negative distance from growing indefinitely. Here, ϵ is a parameter that controls the scale of the embeddings. To limit the scope of this chapter, we consider domains where we can find a reasonable data set of transitions without considering exploration. Changing the sampling policy will introduce bias in the data set, influencing the representations. Here we evaluate if we can find plannable MDP homomorphisms and leave the exploration problem to future work.

Constructing the Abstract MDP

After learning a structured latent space, we find abstract MDP $\bar{\mathcal{M}}$ by constructing reward and transition functions from Z_θ, $\bar{T}_\phi$, $\bar{R}_\zeta$.

Abstract States

Core to our approach is the idea that exploiting action-equivariance constraints leads to nicely structured abstract spaces that can be planned

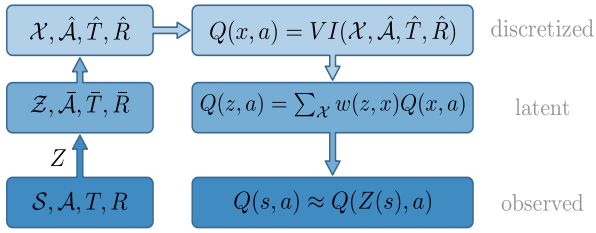


Figure 4.3.1: Schematic overview of our method. We learn the map Z from $\mathcal{S}$ to $\mathcal{Z}$ and discretize $\mathcal{Z}$ to obtain $\mathcal{X}$. We plan in $\mathcal{X}$ and use interpolated Q-values to obtain a policy in $\mathcal{S}$.

in. Of course the space $\mathcal{Z}$ is still continuous, which requires either more complex planning methods, or state discretization. In this chapter we aim for the latter, simpler, option, by constructing a discrete set $\mathcal{X}$ of ('prototype') latent states in $\mathcal{Z}$ over which we can perform standard dynamic programming techniques. We will denote such prototype states as $x \in \mathcal{X}$, cf. Figure 4.3.1. Of course, we then also need to construct discrete transition $\hat{T}_\phi$ and reward $\hat{R}_\zeta$ functions. The next sub-sections will outline methods to obtain these from $\mathcal{Z}$, $\bar{T}_\phi$ and $\bar{R}_\zeta$. To find a 'plannable' set of states, the abstract state space should be sufficiently covered. To construct the set, we sample L states from the replay memory and encode them, i.e. $\mathcal{X} = \{Z_\theta(s_l) | s_l \sim \mathcal{D}\}_{l=1}^{L}$, pruning duplicates.

Reward Function

In Eq. 4.8 we use a reward prediction loss to encourage the latent states to contain information about the rewards. This helps separate distinct states with comparable transition functions. During planning, we can use this predicted reward $\bar{R}_\zeta$. When the reward depends on a changing goal state, such as in the goal-conditioned tasks in Section 3.5, $\hat{R}_\zeta(Z_\theta(s)) = 1$ if $Z_\theta(s) = Z_\theta(s_g)$ and 0 otherwise. We use this reward function in planning, i.e. $\hat{R}_\zeta(x) = 1$ if $x = Z_\theta(s_g)$ and 0 otherwise.

Transition Function

We model the transitions on the basis of similarity in the abstract space. We follow earlier work [53, 92] and assume that if two states are connected by an action in the state space, they should be close after applying the latent action transition. The transition function is a distribution over next latent states. Therefore, we use a temperature softmax to model transition probabilities between representations of abstract states in $\mathcal{X}$:

$$\hat{T}'_\phi(z_j | z_i, a) = \frac{e^{-d(z_j, z_i + \bar{A}_\phi(z_i, a))/\tau}}{\sum_{k \in \mathcal{X}} e^{-d(z_k, z_i + \bar{A}_\phi(z_i, a))/\tau}} \tag{4.9}$$

Thus, for the transitions between abstract states:

$$\hat{T}_\phi(x = j|x' = i, \hat{a} = a) = \hat{T}'_\phi(z_j|z_i, a) \tag{4.10}$$

where τ is a temperature parameter that determines how 'soft' the edges are, and z_j is the representation of abstract state j. Intuitively, this means that if an action moves two states closer together, the weight of their connection increases, and if it moves two states away from each other, the weight of their connection decreases. For very small τ, the transitions are deterministic.

Convergence to an MDP homomorphism

We now show that when combining optimization of our proposed loss fuction equation 4.8 with the construction of an abstract MDP as detailed in this subsection, we can approximate an MDP homomorphism. Specifically, for deterministic MDPs, we show that when the loss function in Eq. 4.8 reaches zero, we have an MDP homomorphism of $\mathcal{M}$.

Theorem 1 *In a deterministic MDP $\mathcal{M}$, assuming a training set that contains all state, action pairs, and an exhaustively sampled set of abstract states $\mathcal{X}$ we consider a sequence of losses in a successful training run, i.e. the losses converge to 0. In the limit of the loss $\mathcal{L}$ in Eq. 4.8 approaching 0, i.e. $\mathcal{L} \to 0$ and $0 < \tau \ll 1$, $\tau \ll \epsilon$, $(\sigma, \{\alpha_s | s \in \mathcal{S}\}) = (Z_\theta, \bar{A}_\phi)$ is an MDP homomorphism of $\mathcal{M}$.*

Proof 4 *Fix $0 < \tau \ll 1$ and write $z = Z_\theta(s)$ and $\bar{a} = \bar{A}_\phi(z, a)$. Consider that learning converges, i.e. $\mathcal{L} \to 0$. This implies that the individual loss terms $d(\bar{T}_\phi(z, \bar{a}), z')$, $d_\neg(\bar{T}_\phi(z, \bar{a}), z_\neg)$ and $d(R(s), \bar{R}_\zeta(z))$ also go to zero for all $(s, a, r, s', s_\neg) \sim \mathcal{D}$.*
Positive samples: *As the distance for positive samples $d_+ = d(\bar{T}_\phi(z, \bar{a}), z') \to 0$, then $d_+ \ll \tau$. Since $d_+ \ll \tau$, then $e^{-d_+/\tau} \approx 1$.*
Negative samples: *Because the negative distance $d_\neg(\bar{T}_\phi(z, \bar{a}), z_\neg) \to 0$, $d_\neg \leq \epsilon$. This, in turn, implies that the distance to all negative samples $d_- = d(\bar{T}_\phi(z, \bar{a}), z_\neg) \geq \epsilon$ and thus $\tau \ll \epsilon \leq d_-$, meaning that $1 \ll \frac{d_-}{\tau}$ and thus $e^{-d_-/\tau} \approx 0$.*
This means that when the loss approaches 0, $\hat{T}'_\phi(z'|z, \bar{a}) = 1$ where $T(s'|s, a) = 1$ and $\hat{T}'_\phi(z_\neg|z, \bar{a}) = 0$ when $T(s_\neg|s, a) = 0$. Since $\mathcal{M}$ is deterministic, $T(s'|s, a)$ transitions to one state with probability 1, and probability 0 for the others. Therefore, $\hat{T}'_\phi(Z_\theta(s')|Z_\theta(s), \bar{A}_\phi(Z_\theta(s), a)) = \sum_{s'' \in [s']_Z} T(s''|s, a)$ and Eq. 4.4 holds. As the distance for rewards $d(R(s), \bar{R}_\zeta(z)) \to 0$, we have that $\bar{R}_\zeta(z) = R(s)$ and Eq. 4.5 holds. Therefore, when the loss reaches zero we have an MDP homomorphism of $\mathcal{M}$.

Note that Eq. 4.8 will not completely reach zero: negative samples are drawn uniformly. Thus, a positive sample may occasionally be treated

as a negative sample. Refining the negative sampling can further improve this approach.

Planning and Acting

After constructing the abstract MDP we plan with VI [140] and lift the found policy to the original space by interpolating between Q-value embeddings. Given $\hat{\mathcal{M}} = (\mathcal{X}, \hat{\mathcal{A}}, \hat{T}_\phi, \hat{R}_\zeta, \gamma)$, VI finds a policy $\hat{\pi}$ that is optimal in $\hat{\mathcal{M}}$. For a new state $s^* \in \mathcal{S}$, we embed it in the representation space $\mathcal{Z}$ as $z^* = Z_\theta(s^*)$ and use a softmax over its distance to each $x \in \mathcal{X}$ to interpolate between their Q-values, i.e.

$$Q(z^*, a) = \sum_{x \in \mathcal{X}} w(z^*, x) Q(x, a) \tag{4.11}$$

$$w(z^*, x) = \frac{e^{-d(z_x, z^*)/\eta}}{\sum_{k \in \mathcal{X}} e^{-d(z_k, z^*)/\eta}} \tag{4.12}$$

where η is a temperature parameter that sets the 'softness' of the interpolation. We use the interpolated Q-values for greedy action selection for s^*, transition to s^{**} and iterate until the episode ends.

4.4 Experiments

Here we show that in simple domains, our approach 1) succeeds at finding plannable MDP homomorphisms for discrete and continuous problems 2) requires less data than model-free approaches, 3) generalizes to new reward functions and data and 4) trains faster than approaches based on reconstructions. We focus on deterministic MDPs. While preliminary results on stochastic domains were promising, an in-depth discussion is beyond the scope of this chapter.

Baselines

To evaluate our approach, we compare to a number of baselines:

1. WM-AE: An auto-encoder approach inspired by World Models [66]. We follow their approach of training representations using a reconstruction loss, then learning latent dynamics on fixed representations. We experimented with a VAE [91], which did not perform well (see [94] for similar results). We thus use an auto-encoder to learn an embedding, then train an MLP to predict the next state from embedding and action.

2. LD-AE: An auto-encoder with latent dynamics. We train an auto-

encoder to reconstruct the input, and predict the next latent state. We experimented with reconstructing the next state, but this resulted in the model placing the next state embeddings in a different location than the latent transitions.

3. DMDP-H: We evaluate the effectiveness of training without negative sampling. This is similar to DeepMDP [54]. However, unlike DeepMDP, DMDP-H uses action-embeddings, for a fairer comparison.

4. GC-Model-Free: Finally, we compare to a goal-conditioned model-free baseline (REINFORCE with state-value baseline), to contrast our approach with directly optimizing the policy[1]. We include the goal state as input for a fair comparison.

[1] Deep reinforcement learning algorithms such as our baseline may fail catastrophically depending on the random seed [70]. For a fair comparison, we train the baseline on 6 random seeds, then remove those seeds where the method fails to converge for the train setting.

To fairly compare the planning approaches, we perform a grid search over the softness of the transition function by evaluating performance on the train goals in $\tau \in [1, 0.1, 0.001, 0.0001, 0.00001, 1e-20]$. Unless otherwise stated, the planning approaches are all trained on datasets of 1000 trajectories, sampled with a random policy. The learning rate is set to 0.001 and we use Adam [90]. For the hinge loss, we use $\epsilon = 1$. The latent dimensionality is set to 50 everywhere. Our approach is trained for 100 epochs. WM-AE is trained for 1000 epochs in total: 500 for the auto-encoder and 500 for the dynamics. LD-AE is trained for 1000 epochs. For constructing the abstract MDP we sample 1024 states from $\mathcal{D}$, project unto $\mathcal{Z}$ and prune duplicates. For planning we use VI with discount factor $\gamma = 0.9$, 500 backups and interpolation parameter (Eq. 4.12) $\eta = 1e-20$. The learning rate for the model-free baseline was chosen by fine-tuning on the training goals. For the model-free baseline, we use a learning rate of $5e-4$ and we train for 500k steps (more than five times the number of samples the planning approaches use). Network Z_θ has 2 convolutional layers (both 16 channels, 3×3 filters) and 3 fully connected layers (input$\rightarrow 64 \rightarrow 32 \rightarrow |z|$). Networks T_ϕ and R_ξ each have 2 fully connected layers. We use ReLU non-linearities between layers.

Object Collection

We test our approach on an object collection task inspired by the key task in [51], with major differences: rather than searching for three keys in a labyrinth, the agent is placed in a room with some objects. Its task is to collect the key. On every time step, the agent receives a state—a $3 \times 48 \times 48$ pixel image (a channel per object, including the agent), as shown in Figure 4.4.1—and a goal state of the same size.

At train time, the agent receives reward of 1 on collection of the *key* object, and a reward of -1 if it grabs the wrong object, and a reward of -0.1 on every time step. The episode ends if the agent picks up one (or more) of the objects and delivers it to one of the four corners (randomly sampled at episode start), receiving an additional delivery reward of 1. At test time, the agent is tasked with retrieving one of the objects chosen at random, and delivering to a randomly chosen location, encoded as a desired goal state. This task will evaluate how easily the trained agent adapts to new goals/reward functions. The agent can interact with the environment until it reaches the goal or 100 steps have passed. For both tasks, we compare to the model-free baseline. We also compare to the DMDP-H, WD-AE and LD-AE baselines. We additionally perform a grid search over the hinge, number of

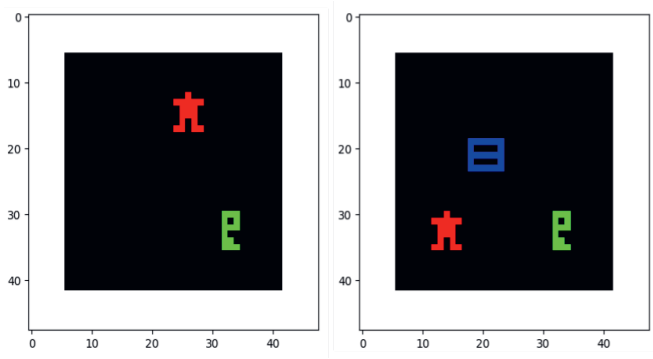


Figure 4.4.1: Example states in the object collection domain for the single object and double object tasks.

state samples for discretization and η hyperparameters for insight in how these influence the performance. This showed that our approach is robust with respect to the hinge parameter, but it influences the scale of the embeddings. The results decrease only when using 256 or fewer state samples. Lastly, η is robust for values lower than 1. We opt for a low value of η, to assign most weight to the Q-value of the closest state.

Single Object Task

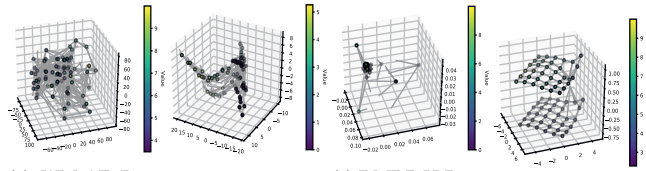
We first evaluate a simple task with only one object (a key). The agent's task is to retrieve the key, and move to one of four delivery locations in the corners of the room. The delivery location is knowledge supplied to the agent in the form of a goal state that places the agent in the correct corner and shows that there is no key. These goal states are also supplied to the baseline, during training and testing. Additionally, we perform an ablation study on the effect of the reward loss. The average episode lengths are shown in Table 4.4.1. Our approach outperforms

Avg. ep. length ↓	Single Object		Double Object	
Task				
Goal Set	Train	Test	Train	Test
GC-Model-free	10.00 ± 0.11	67.25 ± 6.81	10.10 ± 0.69	38.25 ± 15.30
WM-AE	12.96 ± 8.93	10.03 ± 5.56	29.61 ± 19.42	22.53 ± 22.12
LD-AE	23.46 ± 27.10	21.04 ± 21.71	60.26 ± 29.14	52.72 ± 27.32
DMDP-H ($J = 0$)	82.88 ± 11.62	85.69 ± 7.98	81.24 ± 2.45	81.17 ± 2.69
Ours, $J = 1$,	8.61 ± 0.35	**7.53** ± 0.24	8.53 ± 0.36	**8.38** ± 0.07
Ours, $J = 3$	8.68 ± 0.27	7.63 ± 0.19	8.61 ± 0.38	8.95 ± 0.63
Ours, $J = 5$	**8.57** ± 0.48	7.74 ± 0.22	**8.26** ± 0.84	8.96 ± 1.15

Table 4.4.1: Comparing average episode length of 100 episodes on the object collection domain. Reporting mean and standard deviation over 5 random seeds for the planning approaches. The model free approach is averaged over 4 random seeds for the single object domain, 3 random seeds for the double object domain.

all baselines, both at train and at test time. There is no clear preference in terms of the number of negative samples — as long as $J > 0$ — the result for all values of J are quite close together. The DMDP-H approach fails to find a reasonable policy, possibly due to the sparse rewards in this task providing little pull against state collapse. Out of the planning baselines, WM-AE performs best, probably because visually salient features are aligned with decision making features in this task. Finally, the model-free approach is the best performing baseline on the training goals, but does not generalize to test goals.

The results of the reward ablation are shown in Table 4.4.2. While



(a) WM-AE Baseline (b) LD-AE Baseline (c) DMDP-H Baseline (d) This chapter

Figure 4.4.2: Abstract MDP for three approaches in the single object room domain. Nodes are PCA projections of abstract states, edges are predicted $\bar{T}_\phi$, colors are predicted values.

removing the reward loss does not influence performance much for $J = 0$, $J = 3$ and $J = 5$, when $J = 1$ the reward prediction is needed to separate the states. Without the reward, the single negative sample does not provide enough pull for complete separation.

Avg. ep. length ↓	Reward Loss		No Reward Loss	
Goal Set	Train	Test	Train	Test
DMDP-H ($J = 0$)	82.88 ± 11.62	85.69 ± 7.98	87.03 ± 3.08	84.08 ± 3.02
Ours, $J = 1$	8.61 ± 0.35	7.53 ± 0.24	74.32 ± 19.90	68.54 ± 17.29
Ours, $J = 3$	8.68 ± 0.27	7.63 ± 0.19	8.54 ± 0.36	7.44 ± 0.21
Ours, $J = 5$	8.57 ± 0.48	7.74 ± 0.22	8.52 ± 0.19	7.53 ± 0.20

Table 4.4.2: Ablation study of the effect of the reward loss. Comparing average episode length of 100 episodes for the single object room domain. Reporting mean and standard deviation over 5 random seeds.

We show the latent spaces found for the baselines and our approach in Figure 4.4.2. Our approach has found a double grid structure - repre-

senting the grid world before, and after picking up the key. The baselines are reasonably plannable after training for long enough, but the latent spaces aren't as nicely structured as our approach. This mirrors results in earlier work [94]. Thus, while pixel reconstruction losses may be able to find reasonable representations for certain problems, these rely on arbitrarily complex transition functions. Moreover, due to their need to train a pixel reconstruction loss they take much longer to find useable representations. This is shown in Figure 4.4.3b, where

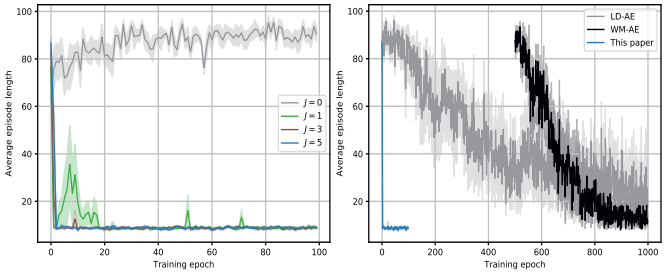


Figure 4.4.3: Average episode length per training epoch for the single object domain. Reported mean and standard error over 5 random seeds.

(a) Comparison of different values of J.

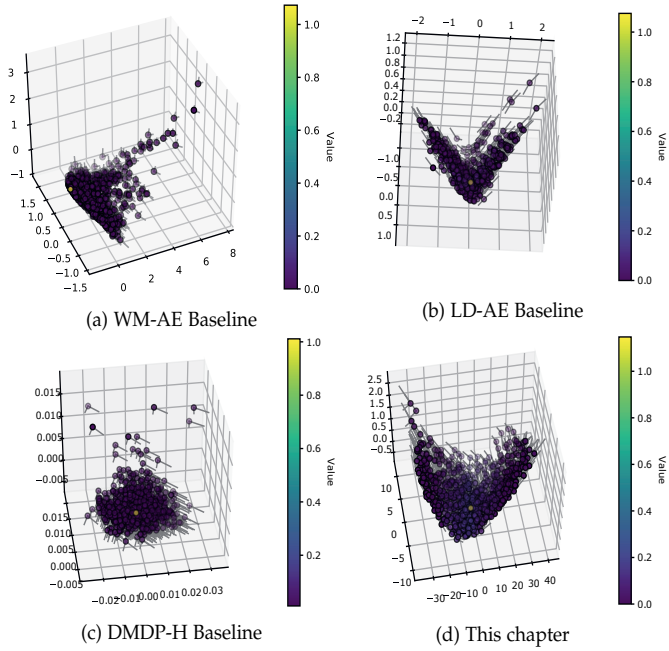
(b) Comparison of this chapter and the WM-AE and LD-AE baselines. WM-AE can not be evaluated until the auto-encoder has finished training and training of the dynamics model begins.

the performance after planning for each training epoch is plotted and compared. Additionally, we observe state collapse for DMDP-H in Figure 4.4.2c, and this is reflected in a high average episode length after planning.

Double Object Task

We now extend the task to two objects: a key and an envelope. The agent's task at train time is still to retrieve the key. At test time, the agent has to pick up the key or the envelope (randomly chosen) and deliver it to one of the corners. We show results in Table 4.4.1. Again, our method performs well on both train and test set, having clearly learned a useful abstract representation, that generalizes to new goals. The WM-AE baseline again fares better than the LD-AE baseline, and DMDP-H fails to find a plannable representation. The model-free baseline performs slightly worse than our method on this task, even after seeing much more data. Additionally, even though it performs reasonably well on the training goals, it does not generalize to new goals at all. The WM-AE performs worse on this task than our approach, but generalizes much better than the model-free baseline, due to its planning, while the LD-AE baseline does not find plannable representations of this task.

Continuous State Spaces



(a) WM-AE Baseline

(b) LD-AE Baseline

(c) DMDP-H Baseline

(d) This chapter

Figure 4.4.4: Abstract MDP for four approaches in CartPole. Nodes are PCA projections of abstract states, edges are predicted $\bar{T}_\phi$, colors are predicted values.

We evaluate whether we can use our method to learn plannable representations for continuous state spaces. We use OpenAI's CartPole-v0 environment [24]. We include again a model-free baseline that is trained until completion as a reference for the performance of a good policy. We also compare DMDP-H, WD-AE and LD-AE. We expect that the latter two would perform well here; after all, the representation that they reconstruct is already quite compact. We additionally evaluate performance when the amount of data is limited to only 100 trajectories (and we limit the number of training epochs for all planning approaches to 100 epochs). We plot the found latent space for our approach and the baselines in Figure 4.4.4. The goal in this problem is to reach the all-zero reward vector, which we set as the goal state with reward 1, and all other states to reward 0. For our approach and both auto-encoder baselines, the latent space forms a bowl with the goal in its center. The DMDP-H again shows a shrunk latent space, and does not have this bowl structure.

Results are shown in Table 4.4.3. Our approach performs best out of all planning approaches. When trained fully, the model-free approach performs better. However, when we limit the number of environmental interactions to 100 trajectories, we see that the planning approach still finds a reasonable policy, while the model-free approach fails com-

Average episode length ↑	**Standard**	**Only 100 trajectories**
GC-Model-free	**197.85** ± 2.16	23.84 ± 0.88
WM-AE	150.61 ± 30.48	114.47 ± 17.32
LD-AE	157.10 ± 11.14	154.73 ± 50.49
DMDP-H ($J = 0$)	39.32 ± 9.02	72.81 ± 20.16
Ours, $J = 1$,	174.64 ± 22.43	127.37 ± 44.02
Ours, $J = 3$	166.05 ± 24.73	148.30 ± 67.27
Ours, $J = 5$	186.31 ± 12.28	**171.53** ± 34.18

Table 4.4.3: CartPole results. Comparing average episode length over 100 episodes, reporting mean and standard deviation over 5 random seeds. The left column has standard settings, in the right column only 100 trajectories are encountered, and planning models are trained for only 100 epochs.

pletely. This indicates that our approach is more data efficient.

Generalizing over Goals and Objects

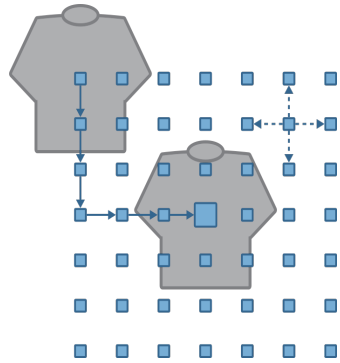
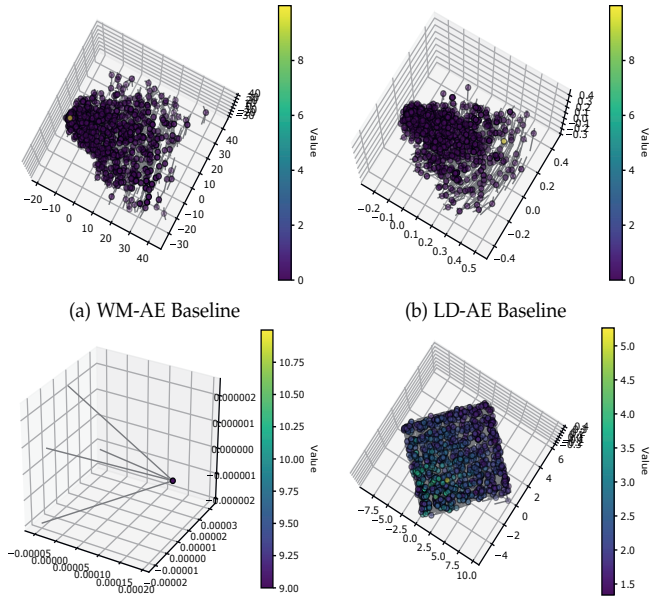


Figure 4.4.5: Transitions in the image manipulation task.

In many tasks we need to be able to generalize not only over goals, but also object instances. We evaluate if our abstract state space generalizes to unseen objects in a problem class. For this we construct an object manipulation task. On each episode, an image of a piece of clothing is sampled from a set of training images in Fashion MNIST [196], and a goal translation of the image is sampled from a set of train goals (translations with negative x-offset: $(-3, \cdot)$ up to and including $(-1, \cdot)$). Thus, the underlying state space is a 7×7 grid. The translated image is provided to the agent as a goal state. The agent receives a reward of +1 if she moves the clothing to the correct translation. See Figure 4.4.5.

At test time, we evaluate performance on test goals (translations with positive x-offset: $(1, \cdot)$ up to and including $(3, \cdot)$, seen before as states for training images but never as goals) and test images. The latent spaces for each of the four representation learning approaches are shown in Figure 4.4.6. For DMDP-H, the latent space collapses to all but a few points. For WD-AE and LD-AE, the latent space does not exhibit clear structure. For our approach, there is a clear square grid structure present in the latent space. However, the underlying trans-

lations for the images do not neatly align across images. Clustering such states together is interesting future work.



(a) WM-AE Baseline (b) LD-AE Baseline

(c) DMDP-H Baseline (d) This chapter

Figure 4.4.6: Abstract MDP for four approaches in planning in Fashion MNIST. Nodes are PCA projections of abstract states, edges are predicted $\bar{T}_\phi$, colors are predicted values.

Results are shown in Table 4.4.4. The goal-conditioned model-free baseline has an easy time finding a good policy for the training setting. It also generalizes well to unseen images. However, it has trouble generalizing to new goal locations for both train and test images. Our planning approach, on the other hand, loses some performance on the training setting, but easily generalizes to both test images and test goals. Neither WM-AE nor LD-AE find good policies in this problem. They have a difficult time learning plannable representations because their focus is on reconstructing individual images.

Avg. ep. length ↓ Dataset	**Train**		**Test**	
Goal Set	Train	Test	Train	Test
GC-Model-free	**4.82** ± 0.33	9.67 ± 5.01	**4.75** ± 0.12	8.17 ± 2.67
WM-AE	59.95 ± 4.06	63.27 ± 3.36	64.27 ± 5.33	63.41 ± 2.04
LD-AE	56.39 ± 7.07	49.35 ± 4.05	51.45 ± 6.79	51.70 ± 3.97
DMDP-H ($J = 0$)	62.86 ± 3.87	66.68 ± 4.40	65.93 ± 4.98	64.86 ± 1.57
Ours, $J = 1$,	5.07 ± 0.87	**5.27** ± 0.56	5.69 ± 0.93	5.63 ± 0.96
Ours, $J = 3$	5.60 ± 0.97	5.46 ± 0.97	6.44 ± 1.12	5.42 ± 0.89
Ours, $J = 5$	5.36 ± 0.71	5.67 ± 1.20	6.36 ± 1.21	**5.34** ± 0.93

Table 4.4.4: Comparing average episode length of 100 episodes for planning in Fashion MNIST. Reporting mean and standard deviation over 5 random seeds.

4.5 Related Work

This chapter proposes a method for learning action equivariant mappings of MDPs, and using these mappings for constructing plannable abstract MDPs. We learn action equivariant maps by minimizing MDP homomorphism metrics [165]. As a result, when the loss reaches zero the learned mapping is an MDP homomorphism [143]. MDP homomorphism metrics are a generalization of bisimulation metrics [46, 113]. Other works [71, 34, 193] consider equivariance to symmetry group actions in learning. Here, we use a more general version of equivariance under MDP actions for learning representations of MDPs. We learn representations of MDPs by 1) predicting the next latent state, 2) predicting the reward and 3) using negative sampling to prevent state collapse. Much recent work has considered self-supervised representation learning for MDPs. Certain works focus on predicting the next state using a contrastive loss [94, 6, 135], disregarding the reward function. However, certain states may be erronously grouped together without a reward function to distinguish them. [54] include both rewards and transitions to propose an objective based on stochastic bisimulation metrics [57, 46, 113]. However, at training time they focus on deterministically predicting the next latent state. Their proposed objective does not account for the possibility of latent space collapse, and for complex tasks they require a pixel reconstruction term. This phenomenon is also observed by [51], who prevent it with two entropy maximization losses.

Many approaches to representation learning in MDPs depend (partially) on learning to reconstruct the input state [36, 182, 66, 76, 67, 168, 104, 202, 85, 183, 178, 8]. A disadvantage of reconstruction losses is training a decoder, which is time consuming and usually not required for decision making tasks. Additionally, such losses emphasize visually salient features over features relevant to decision making.

Other approaches that side-step the pixel reconstruction loss include predicting which action caused the transition between two states [3], predicting the number of time steps between two states [9] or predicting objects in an MDP state using supervised learning [200].

[82] identify a set of priors about the world and uses them to formulate self-supervised objectives. In [55], the similarity between two states is the difference in goal-conditioned policies needed to reach them from another state. [152] learn representations for tree-based search that must predict among others a policy and value function, and are thus not policy-independent. Earlier work on decoupling representation learning and planning exists [36, 182, 200]. However, these works use objectives that include a pixel reconstruction term [36, 182] or require

labeling of objects in states for use in supervised learning [200].
Other work on planning algorithms in deep learning either assumes knowledge of the state graph [164, 130, 112, 86], builds a graph out of observed transitions [96] or structures the neural network architecture as a planner [132, 45, 51], which limits the search depth.

4.6 Relation to Group Equivariance

Action-equivariance is a generalization of the older notion of group equivariance [34]. In group equivariance, we require that for a function $Z : \mathcal{X} \to \mathcal{Z}$, and a group G acting on a space $\mathcal{X}$, the following holds:

$$Z(g \cdot x) = \bar{g} \cdot Z(x) \qquad \forall g \in G, x \in \mathcal{X} \tag{4.13}$$

where $\bar{g}$ indicates an equivalent group element acting on $\mathcal{Z}$. If we view the original point $x \in \mathcal{X}$ as a state, and the transformed point $x' = gx$ as a next state, we can define g as a special case of an MDP action and write:

$$Z(T(x, g)) = \bar{T}(Z(x), \bar{g}) \qquad \forall g \in G, x \in \mathcal{X} \tag{4.14}$$

which we immediately recognize as the equation in Figure 4.1.1. We can further generalize this by letting $T(\cdot, \cdot)$ and $\bar{T}(\cdot, \cdot)$ be stochastic functions, and matching their distributions. This generalization suggests that the notion of a group action and an MDP action are more similar than they at first glance look. Additionally, we can view equivariance in the general case as a body of work that finds or defines homomorphisms between spaces that respect some notion of acting on a point (a state, or e.g. an image) in order to bring it to another point (a next state, or a rotated image).

4.7 Conclusion

This chapter proposes the use of 'equivariance under actions' for learning representations in deterministic MDPs. Action equivariance is enforced by the use of MDP homomorphism metrics in defining a loss function. We also propose a method of constructing plannable abstract MDPs from continuous latent spaces. We prove that for deterministic MDPs, when our objective function is zero and our method for constructing abstract MDP is used, the map we learn is an MDP homomorphism. Additionally, we show empirically that our approach is data-efficient and fast to train, and generalizes well to new goal

states and instances with the same environmental dynamics. Finally, we show that action-equivariance is a generalization of group equivariance. Potential future work includes an extension to stochastic MDPs and clustering states on the basis of MDP metrics. Using a clustering approach as part of model training, we can learn the prototypical states rather than sampling them. This comes at the cost of having to back-propagate through a discretization step, which in early experiments (using Gumbel-Softmax [78]) led to instability.

5

Learning Factored Representations of Markov Decision Processes

5.1 Introduction

In the previous Chapter we presented work that learns to recover the structure in an environment by using *action-equivariance*. In this Chapter, we will learn the structure in individual states in order to improve decision making in object-oriented decision making problems.

Many decision-making problems are inherently structured: manipulating objects in a scene, steering agents in a multi-agent system, placing chip blocks on a chip and attaching atoms in molecule design are all examples of problems where the full problem state is decomposed into different factors, which are sometimes independent of each other. By making use of this structure, we can often reduce solving an exponentially complex problem to solving a series of simpler problems. For example, if we have a structured representation of states in a reinforcement learning problem, we can use factored approaches to decision making [65, 63, 154, 88] or use similarly structured graph neural network architectures in deep reinforcement learning [102, 180, 75, 132, 130, 37]. In Chapter 3 we have assumed to know the factorization of the environment. In this Chapter, we in-

vestigate the case where this assumption is loosened: we consider a set of decision making problems with unknown structure. This is an important research question, as structure is not always easily given a priori. Oftentimes, observations of the world are unstructured streams of e.g. image data, and structure in the state of the world must be inferred by the agent itself. As such, there is a large body of work on learning structured representations of scenes, resulting in object-based representations [28, 11, 184, 174, 92, 161, 162, 197, 200]. Most works in this area require some form of human supervision, but several works consider the fully self-supervised or unsupervised setting [59, 128, 174, 79, 197, 25, 58, 44]. Such self-supervised methods are usually based on reconstructing the visual inputs from the learned representations. There are a few issues with such approaches: reconstruction-based methods need to be able to properly reconstruct visually important yet potentially irrelevant features such as backgrounds. This means a lot of training time and model capacity is wasted on learning to accurately represent those features. Additionally, one needs to spend training time and compute to learn a decoder model which is not needed for the downstream task. Finally, such approaches tend to ignore visual features that are small but potentially important to decision-making, such as a small ball in certain Atari games. We therefore propose using a contrastive learning method based on graph embedding approaches [19, 181], where states that transition into each other are placed close together in latent space, and random state pairs are pushed further from each other.

We introduce a method for learning factored representations of object-oriented MDPs, where each state consists of a set of latent state variables, one per object in a scene. We model the latent transition model as a graph neural network [149, 114, 95, 56, 12], with the nodes the latent state variables. Due to the graph network transition model, we automatically obtain permutation equivariance on the object transitions. We call this method Contrastively-trained Structured World Models (C-SWMs). We also introduce a factored contrastive loss based on learning translational graph embeddings [19, 181], and connect contrastive learning for state representations to relational graph embeddings [129]. Finally, we introduce a novel ranking-based evaluation strategy which we use to demonstrate that C-SWMs learn object-level state representations, combinatorially generalize to unseen states and can identify objects from scenes without supervision in object manipulation, 3D physics, and Atari settings.

5.2 Background

A *Markov Decision Process* (MDP) is a tuple $M = (S, A, R, T, \gamma)$ with S the set of states, A the set of actions, $R : S \times A \to \mathbb{R}$ the reward function, $T : S \times A \times S \to [0, 1]$ the transition function and $\gamma \in [0, 1]$ a discount factor. An agent acts in an MDP with a policy $\pi : S \times A \to [0, 1]$. In this Chapter, we will not consider the reward function, which we will leave out going forward.

A *factored MDP* is an MDP whose state is a multivariate random variable $X = (X_1, \cdots, X_n)$ and state instances are $x = (x_1, \cdots, x_n)$ with for every i, $x_i \in \text{Dom}(X_i)$. In a factored MDP, the transition function $P(x'|x, a)$ can be written as a set of Dynamic Bayesian Networks (one per action) [22, 88, 154, 63]. We can write the transition function using x_i's parent set, $\mathbf{x}_{a,i}$.

$$P(x'|x, a) = \prod_i P(x'_i|\mathbf{x}_{a,i}) \tag{5.1}$$

We base our method on the graph embedding method TransE [19]. Consider a knowledge base $\mathcal{K}$ of entity-relation-entity triples $\mathcal{K} = \{(e_t, r_t, o_t)\}_{t=1}^{T}$, with $e_t \in \mathcal{E}$ the subject, $r_t \in \mathcal{R}$ the relation (not to be confused with the reward in an MDP) and $o_t \in \mathcal{E}$ the object of the knowledge base fact. We can draw a parallel between such fact triples and state transitions (s_t, a_t, s_{t+1}) in an MDP without rewards. In a sense, the MDP's action can be viewed as the relation between a state s_t and next state s_{t+1}.

TransE embeds knowledge facts with maps $F : \mathcal{E} \to \mathbb{R}^D$ and $G : \mathcal{R} \to \mathbb{R}^D$ and computes the energy of a triple as $H = d(F(e_t) + G(r_t), F(o_t))$ with $d(\cdot, \cdot)$ the squared Euclidean distance and F (and G) are embedding functions that map discrete entities (and relations) to $\mathbb{R}^D$, where D is the dimensionality of the embedding space. During training, an energy-based hinge loss [110] is used.

5.3 Structured World Models

We wish to learn structured abstract representations of MDP states and permutation equivariant transition functions which are consistent with the factored nature of the problem.

Learning Abstract Representations

Assume we have a dataset of experience $\mathcal{B} = \{(s_t, a_t, s_{t+1})\}_{t=1}^{T}$ sampled from for example an exploration policy π, with T the number of tuples in the data set. We wish to train an encoder $E : S \to \mathcal{Z}$ that maps states $s_t \in S$ to abstract representations $z_t \in \mathcal{Z} = \mathcal{Z}_1 \times \ldots \times \mathcal{Z}_K$, with K the number of object slots. We set $Z_k = \mathbb{R}^D$ for each k, with D a hyperparameter. The abstract representations z_t should contain only information needed for modeling the transitions in the environment, and no superfluous information (such as for example background color in a video game). Note that while in this Chapter we do not consider the reward, we can include reward in learning representations, as we did in Chapter 4.

Since we wish to find a structured representation of the state, we require that our encoder maps from unstructured (usually image-based) inputs to a factored, object-oriented representation. Our encoder thus consists of two modules: an object extractor E_{ext} that maps from observations to K feature maps, and an object encoder E_{enc} that maps from feature maps to object representations. The feature maps $m_t^k = [E_{\text{ext}}(s_t)]_k$ are flattened and used to predict abstract representations $z_t^k = E_{\text{enc}}(m_t^k)$ with $z_t^k \in \mathcal{Z}_k$. The feature maps can be viewed as object masks corresponding to an object slot. The object encoder E_{enc} shares weights between objects.

In this chapter we will assume a factored action space $A = A_1 \times \ldots \times A_K$, which provides an independent action for each object in the scene. This is a strong inductive bias for learning factored representations. The full architecture is shown in Figure 5.3.1.

Contrastive Learning

We can use contrastive coding on the transitions in an MDP without rewards: (s_t, a_t, s_{t+1}). However, the same action can have different outcomes in different states. We therefore base the "relation effect" on both the state and the action, resulting in the latent transition model $T(z_t, a_t)$. In essence, we therefore constrain the latent transition function to model the effects of actions as translations in latent space. The energy is then $H = d(z_t + T(z_t, a_t), z_{t+1})$.

For a single (s_t, a_t, s_{t+1}) with a negative sample $\tilde{z}_t = E(\tilde{s}_t)$, and $\tilde{s}_t$ randomly sampled from the dataset, the energy-based hinge loss is

$$\mathcal{L} = d(z_t + T(z_t, a_t), z_{t+1}) + \max(0, \epsilon - d(\tilde{z}_t, z_{t+1}))\,, \qquad (5.2)$$

where ϵ is the margin for which $\epsilon = 1$ was used in our experiments.

The full loss is an expectation of Eq. 5.2 over samples from the dataset.

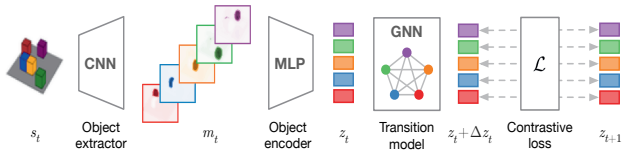


Figure 5.3.1: Visualization of general C-SWM architecture: a CNN object extractor, MLP object encoder and GNN transition model that uses the actions to compute Δz_t, together with an object-factorized contrastive loss.

5.4 Transition Model

We model the transition function as a graph neural network (GNN) [149, 114, 95, 11, 56, 12]. When using a GNN, the nodes are the different objects in the scene and the model is able to learn different pairwise interactions based on the object features, while being equivariant to the order in which the objects are assigned to slots. The input for the GNN are a set of nodes, and a set of edges. In this chapter we model the graph as a fully connected graph, so that long range dependencies do not require multiple message passing steps. The node inputs are the object representations $\{z_k^t\}_{k=1}^K$ extracted by the encoder and the actions $a_t = (a_t^1, \cdots, a_t^K)$. The actions are encoded as one-hot vectors for discrete action spaces, but can be replaced by zero vectors in case of no actions or continuous vectors in case of continuous action spaces. The transition GNN $T(\cdot, \cdot)$ predicts the effect of taking a_t from z_t:

$$\Delta z_t = T(z_t, a_t) = \text{GNN}(\{(z_t^k, a_t^k)\}_{k=1}^K)\,. \tag{5.3}$$

where $\Delta z_t = (\Delta z_t^1, \cdots, \Delta z_t^K)$. The predicted next latent state is then given by

$$z_{t+1} = (z_t^1 + \Delta z_t^1, \cdots, z_t^K + \Delta z_t^K)\,. \tag{5.4}$$

Thus, the transition function can also be viewed as predicting the individual translational effects of the action on each object in the state.

Message Passing The GNN consists of MLP node update functions f_{node} and MLP edge update functions f_{edge} which share parameters between nodes and edges. A message passing round is given by

$$e_t^{(i,j)} = f_{\text{edge}}([z_t^i, z_t^j]) \tag{5.5}$$

$$\Delta z_t^j = f_{\text{node}}([z_t^j, a_t^j, \textstyle\sum_{i \neq j} e_t^{(i,j)}])\,, \tag{5.6}$$

where $e_t^{(i,j)}$ is a predicted edge representation between nodes i and j. Multiple rounds of message passing are possible, but were not found to be necessary, possibly due to the use of a fully connected graph.

Message passing in a fully connected graph is $\mathcal{O}(K^2)$, but this may be reduced to linear complexity if messages are only sent to nearest neighbors in latent space. We leave this for future work.

Factored Contrastive Loss We now adapt the original contrastive loss function in Eq. 5.2 to a factored loss, which is computed independently for the different objects. Write the predicted effect of the action on the k-th object as $\Delta z_t^k = T^k(z_t, a_t)$. Then, the energy H for positive samples is

$$H = \frac{1}{K} \sum_{k=1}^{K} d(z_t^k + T^k(z_t, a_t), z_{t+1}^k)\,, \tag{5.7}$$

and the energy $\tilde{H}$ for negative samples is

$$\tilde{H} = \frac{1}{K} \sum_{k=1}^{K} d(\tilde{z}_t^k, z_{t+1}^k)\,. \tag{5.8}$$

Here, $\tilde{z}_t = E(\tilde{s}_t)$ is the representation of the negative sample, and $\tilde{z}_t^k$ the representation of the k-th object. The full contrastive loss for a single sample is then given by

$$\mathcal{L} = H + \max(0, \epsilon - \tilde{H})\,. \tag{5.9}$$

5.5 Related Work

Here we review related work on state representation learning.

State Representation Learning State representation learning is a very active field. The general goal is to find representations of states where similar states are close together in latent space. Stochastic bisimulation [38, 57], lax bisimulation [165] or MDP homomorphisms [141, 142] are formalisms on which much of the work on state similarity is built[1]. For example, there is much work on bisimulation and other similarity metrics [46, 26, 106, 49, 47, 48, 27] and on using bisimulation (or MDP homomorphism) metrics to learn or evaluate representations [201, 170, 54, 89, 4, 20]. It is also very common to use reconstruction-based losses [36, 182, 66, 67, 108, 54, 76, 168, 104, 202, 85, 183, 178, 8]. Other self-supervised learning approaches are also common [167, 51, 82, 43, 3, 9, 55, 152].

[1] For an overview, see [113].

Contrastive Learning Contrastive approaches are common in learning graph representations [19, 139, 61, 19, 150, 176], word representations [123, 121], and image representations [135, 41, 72, 29]. They are

also becoming more common in state representation learning [94, 170, 158, 2, 116, 6, 135]. Most of these works on state representation do not focus on recovering the structure in individual states.

Structured Models of Environments Graph networks have been used to take advantage of the structure in an environment [160, 11, 73, 180, 92, 146]. Such approaches usually assume that the structured nature of the environment is already known. For problems where this structure is unknown, there is a body of literature (see [60] for a review) focusing on recovering objects from scenes directly from pixels [59, 128, 174, 79, 197, 25, 58, 44], using pixel-based losses. Recently, other forms of structure have been gaining ground as well, for example by taking symmetries into account while learning representations [126, 136, 189, 155].

5.6 Experiments

We evaluated C-SWMs on different environments to see if they can recover objects, predict transitions accurately, and generalize to new combinations of objects in scenes. Code can be found at `https://github.com/tkipf/c-swm`.

Evaluation and Training

We evaluate using rankings and compare to baselines. Settings, baselines, and evaluation metrics are described below. For more details and experiments, see [94].

Evaluation Metrics Different models result in different latent spaces. To compare their trajectory predictions in latent space, we compare the different approaches based on ranking. This is done by encoding the starting observation into latent space, followed by a prediction of the next step(s). Then the target observation is encoded, as well as a set of reference observations. The latent states are ranked based on how close they are to the predicted latent target state. This allows us to compare methods which learn very different latent spaces and removes the need for comparing based on reconstruction error or to do downstream planning (for planning performance of a related approach, see Chapter 4). We compare on Hits at Rank 1 (H@1) and Mean Reciprocal Rank (MRR) (in %) after encoding the original state, taking steps in latent space, and comparing to the encoding of the target state. We report mean and standard error on 4 runs of hold-out

environment instances.

Training and Evaluation We sample a training dataset by taking uniformly random actions in the environment and storing the interactions. We similarly sample a separate evaluation data set for each environment. We use Adam [90] with a learning rate of $5 \cdot 10^{-4}$ and a batch size of 1024. Details of architectures and hyperparameters are included in the sections below and in [94].

Baselines For baselines we compare to auto-encoder based world models [66] (both AE and VAE [91]). For the 3-body physics environment we additionally compare to Physics as Inverse Graphics (PAIG) [80]. Additionally, for 3D shapes we perform an ablation study where we compare the effect of removing the latent GNN, the state factorization, or the contrastive loss. For AE and VAE baselines we use a batch size of 512 (due to higher memory demands) and for PAIG we use a batch size of 100, as recommended by the authors.

(a) 3D blocks (b) Space Invaders (c) 3 Body Physics

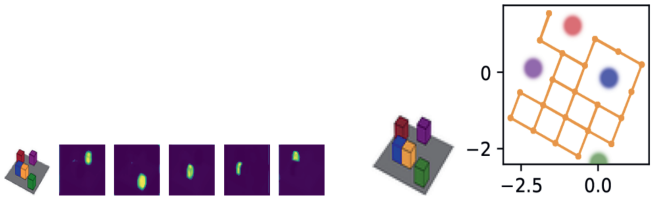
Figure 5.6.1: Example observations from a) 3D shapes block pushing world, b) Atari Space Invaders, and c) 3 Body Physics.

3D Shapes

We evaluate on a novel 3D shapes environment, where each block is moved by its own action. The shapes move in a 2D grid, but are represented in the image as blocks in 3D, making it more difficult to extract the underlying objects from visual information. The observations are $50 \times 50 \times 3$ color images. Additional details in [94]. Example observations are shown in Figure 5.6.1a. We sample 1000 training episodes with 100 environment steps. We sample 10,000 evaluation episodes with 10 steps each. Since the number of possible states is large (approximately 6.4M unique states), a complete trajectory overlap between test and train data is unlikely. We train for 100 epochs.

Qualitative Results We visualize discovered object masks on held-out test data in Figure 5.6.2a, which shows that the objects are masked out separately. In Figure 5.6.2b we show the learned abstract transition graph for one of the blocks. These results show a that the model is able to recover the underlying grid structure of a given object's transition graph. Note that no transitions are predicted where the block is

obstructed by another, even though the transition model does not have access to the image inputs.



(a) Discovered object masks in a scene from the 3D Cubes environment.

(b) Learned abstract state transition graph of the yellow cube, others fixed.

Figure 5.6.2: Discovered object masks (left) and abstract state transition graphs for a single object (right) in the 3D shapes block pushing task. Each node is an encoded state from a held out test set, and each edge (color coded by action type) is a transition predicted by the model.

Quantitative Results See Table 5.6.1 for quantitative results. C-SWMs predict almost perfectly for short term (1 step), mid-term (5 steps), and long term (10 steps) predictions, in terms of both H@1 and MRR. In comparison, both the auto-encoder world model and the VAE world model do quite well in the short term, but perform a lot worse on mid-term and long term prediction. In terms of ablations of C-SWMs, removing the latent GNN (replacing it by an object based MLP that ignores interactions) slightly hurts long term prediction. Removing the factored latent space hurts all predictions, and long term predictions most. Removing the contrastive loss hurts all predictions, resulting in a very low score for long term prediction especially.

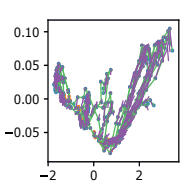
Space Invaders

We also evaluate on Space Invaders, an Atari 2600 game which has a lot of moving parts: a gun (the agent), the shields, and the aliens. The observations are two consecutive frames, given as $50 \times 50 \times 6$ tensors. Example observations are shown in Figure 5.6.1b. We sample 1000 training episodes with 10 steps each. We sample 100 evaluation episodes with 10 environment steps. To minimize train/test overlap, we warm-start the data set by first taking random actions for 50 interactions, discard them, and then start filling the data set. We additionally take care that no trajectories from the test and train set overlap completely. We train for 200 epochs.

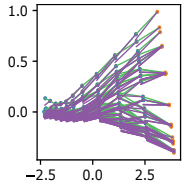
Qualitative Results See Figure 5.6.3 for object-specific filters and a state transition graph for Space Invaders. The transition graphs are much harder to interpret than those for the 3D shapes grid world. This can have multiple reasons. For one, actions are less independent compared to the grid worlds. For example, shooting the bullet

now influences the aliens in the future. Additionally, objects cannot be swapped out the way the objects can in the block world (i.e. the bullet behaves differently than the aliens). Finally, there are many objects with identical visual features (the aliens).

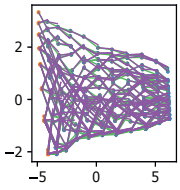
(a) Object-specific filters.



(b) Object slot 1.



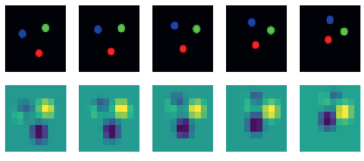
(c) Object slot 2.



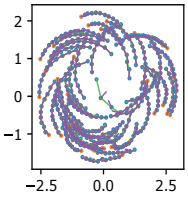
(d) Object slot 3.

Figure 5.6.3: Object filters (top) and abstract state transition graphs per object slot (bottom) in Space Invaders. Each node is an encoded state from an unseen test instance of the environment. Predictions from a trained C-SWM model with $K = 3$ object slots.

Quantitative Results See Table 5.6.1 for quantitative results. The prediction is not as good as it was in the block world, possibly for similar reasons as those listed above. Additionally, results can have a higher variance. Compared to the AE and VAE baselines, C-SWMs perform better at short, mid and long term prediction than the baselines, assuming we use the right number of slots (performance drops for $K = 1$ and is best for $K = 5$).



(a) Observations from 3-body gravitational physics simulation (top) and learned filter for one object (bottom).



(b) Abstract state transition graph.

Figure 5.6.4: Object-specific filter (left) and embedded trajectories (right) in 3-body physics. Embeddings are projected from four to two dimensions with PCA. Orange nodes are starting states, green edges are ground truth transitions, purple edges are predicted transitions.

3-Body Physics

Finally, we evaluate on 3-body physics, where the transitions of a system of 3 objects have to be predicted. Notably, this environment does not contain any actions. The observations are two consecutive frames, given as $50 \times 50 \times 6$ tensors. Example observations are shown in Figure 5.6.1c. We sample 5000 training episodes with 10 steps. We sample

1000 evaluation episodes with 10 steps. Since this environment has a continuous state space, a full trajectory overlap between test and train set is unlikely. We train for 100 epochs.

Qualitative Results We show learned filters for one of the objects, and an abstract state graph for multiple trajectories in Figure 5.6.4. The learned filters are able to separately encode each of the objects. In the abstract state transition graph, we see that the model is able to encode the relevant information about the object and predict smooth trajectories, with the exception of one of the trajectories (in the center), which deviates.

Quantitative Results See Table 5.6.1. Both C-SWMs and the AE and VAE baselines are good at short and mid term prediction. C-SWM, as well as the autoencoder baseline are also good at longer term prediction. The PAIG model does not perform as well[2].

[2] Using the hyperparameter settings recommended by the authors.

Summary C-SWMs predict almost perfectly in the block pushing task, and are able to recover the grid structure of the transition function almost perfectly. The ablation study shows that all components of C-SWMs, but particularly the contrastive loss, contribute to a greater predictive performance, both for short and long term prediction. The object factorization is an important component as well, with the interaction (GNN) component providing a small final boost. For Space Invaders, C-SWMs and baseline models have a harder time representing the states well. While C-SWMs perform better than the baselines, the performance is not as good as in the 3D shapes environment. Additionally, the number of slots (a hyperparameter) has a non-trivial influence on the performance of the model, and should be chosen with care. In the Physics environment, C-SWMs and the strongest baseline (AE world model) perform well, with C-SWMs being slightly better at long term prediction.

5.7 Conclusions

We proposed C-SWMs, a method for learning structured representations of states in object-oriented MDPs that uses contrastive coding and learns a graph neural network transition model. C-SWMs make use of the learned structure, improving on multi-step prediction and providing better generalization to unseen environment configurations than models that use decoders, unstructured transitions, or unstructured representations.

	Model	1 Step H@1	1 Step MRR	5 Steps H@1	5 Steps MRR	10 Steps H@1	10 Steps MRR
3D BLOCKS	**C-SWM**	**99.9**±0.0	**100**±0.0	**99.9**±0.0	**100**±0.0	**99.9**±0.0	**99.9**±0.0
	– latent GNN	**99.9**±0.0	99.9±0.0	96.3±0.4	97.7±0.3	86.0±1.8	90.2±1.5
	– factored states	74.2±9.3	82.5±8.3	48.7±12.9	62.6±13.0	65.8±14.0	49.6±11.0
	– contrastive loss	48.9±16.8	52.5±17.8	12.2±5.8	16.3±7.1	3.1±1.9	5.3±2.8
	World Model (AE)	93.5±0.8	95.6±0.6	26.7±0.7	35.6±0.8	4.0±0.2	7.6±0.3
	World Model (VAE)	90.9±0.7	94.2±0.6	31.3±2.3	41.8±2.3	7.2±0.9	12.9±1.3
SPACE INVADERS	**C-SWM** ($K = 5$)	**48.5**±7.0	**66.1**±6.6	**16.8**±2.7	**35.7**±3.7	**11.8**±3.0	**26.0**±4.1
	C-SWM ($K = 3$)	46.2±13.0	62.3±11.5	10.8±3.7	28.5±5.8	6.0±0.4	20.9±0.9
	C-SWM ($K = 1$)	31.5±13.1	48.6±11.8	10.0±2.3	23.9±3.6	6.0±1.7	19.8±3.3
	World Model (AE)	40.2±3.6	59.6±3.5	5.2±1.1	14.1±2.0	3.8±0.8	10.4±1.3
	World Model (VAE)	1.0±0.0	5.3±0.1	0.8±0.2	5.2±0.0	1.0±0.0	5.2±0.0
3-BODY PHYSICS	**C-SWM**	**100**±0.0	**100**±0.0	97.2±0.9	98.5±0.5	**75.5**±4.7	**85.2**±3.1
	World Model (AE)	**100**±0.0	**100**±0.0	**97.7**±0.3	**98.8**±0.2	67.9±2.4	78.4±1.8
	World Model (VAE)	**100**±0.0	**100**±0.0	83.1±2.5	90.3±1.6	23.6±4.2	37.5±4.8
	Physics WM (PAIG)	89.2±3.5	90.7±3.4	57.7±12.0	63.1±11.1	25.1±13.0	33.1±13.4

Table 5.6.1: Ranking results for multi-step prediction in latent space. Reported mean and standard error of scores over 4 runs on hold-out environment instances. Highest (mean) scores in **bold**.

Scope Our approach focuses on learning representations for deterministic environments under the Markov assumption. We consider an extension to stochastic transition functions and partially observable environments a useful avenue for future work. Additionally, C-SWMs in their current form do not model the reward function, nor do they take reward into account when learning representations. This results in representations that may not contain all necessary information about a given state. Another limitation is that we do not evaluate the representations on downstream decision making tasks: their usefulness is only evaluated in terms of predictive power. In Chapter 4 we looked at unstructured representations learned in a similar way and show that they are also *plannable*. Finally, C-SWMs are not able to tell different instances of visually identical objects in a scene apart. For instance disambiguation in a task such as Space Invaders, an approach based on assigning objects into slots [83, 84, 145, 58, 44, 25, 93] could be a useful future avenue.

6

Conclusion

This thesis studied symmetry and structure in deep reinforcement learning. In particular, we focused on the main research question: *How can we incorporate and extract symmetry and structure in reinforcement learning?*

We divided the thesis up into two main parts, symmetry and structure. Part 1 considered symmetries in the state-action space of decision making problems. We considered symmetries in single agent and multi-agent reinforcement learning problems. In Part 2, we discussed structure in environments and states. We proposed learning action-equivariant representations of MDPs, and recovering object-oriented structure from environment interactions.

We start from the first research question, *How can we leverage knowledge of symmetries in deep reinforcement learning?* To answer this question, Chapter 2 introduces MDP Homomorphic networks for deep reinforcement learning. MDP Homomorphic networks are neural networks that are equivariant under *symmetries* in the joint state-action space of an MDP. We make explicit the connection between MDP Homomorphisms and equivariant networks. Current approaches to deep reinforcement learning do not usually exploit knowledge about such structure. By building this prior knowledge into policy and value networks using an equivariance constraint, we can reduce the size of the solution space. We specifically focus on group-structured symmetries (invertible transformations). Additionally, we introduce an easy method for constructing equivariant network layers numerically, so

the system designer need not solve the constraints by hand, as is typically done. We construct MDP homomorphic MLPs and CNNs that are equivariant under either a group of reflections or rotations. We show that such networks converge faster than unstructured baselines on CartPole, a grid world and Pong.

After showing the improvement in data efficiency using symmetry constraints in single agent policies, we next investigate the effect of global symmetry constraints in distributed cooperative multi agent systems. The second research question, *How can we leverage knowledge of global symmetries in distributed cooperative multi-agent systems?*, can now be discussed. Chapter 3 introduces Multi-Agent MDP Homomorphic Networks, a class of networks that allows distributed execution using only local information, yet is able to share experience between global symmetries in the joint state-action space of cooperative multi-agent systems. In cooperative multi-agent systems, complex symmetries arise between different configurations of the agents and their local observations. For example, consider a group of agents navigating: rotating the state globally results in a permutation of the optimal joint policy. Existing work on symmetries in single agent reinforcement learning can only be generalized to the fully centralized setting, because such approaches rely on the global symmetry in the full state-action spaces, and these can result in correspondences across agents. To encode such symmetries while still allowing distributed execution we propose a factorization that decomposes global symmetries into local transformations. Our proposed factorization allows for distributing the computation that enforces global symmetries over local agents and local interactions. We introduce a multi-agent equivariant policy network based on this factorization. We show empirically on symmetric multi-agent problems that globally symmetric distributable policies improve data efficiency compared to non-equivariant baselines.

In Part 2, we considered structure in learning representations of environments and states. We considered learning plannable representations that are equivariant to actions taken in the original problem using contrastive learning. We dicuss the third research question, *How can we learn representations of the world that capture the structure of the environment?* In Chapter 4, we exploit action equivariance for representation learning in reinforcement learning. Equivariance under actions states that transitions in the input space are mirrored by equivalent transitions in latent space, while the map and transition functions should also commute. We introduce a contrastive loss function that enforces action equivariance on the learned representations. We prove that when our loss is zero, we have a homomorphism of a determinis-

tic Markov Decision Process (MDP). Learning equivariant maps leads to structured latent spaces, allowing us to build a model on which we plan through value iteration. We show experimentally that for deterministic MDPs, the optimal policy in the abstract MDP can be successfully lifted to the original MDP. Moreover, the approach easily adapts to changes in the goal states. Empirically, we show that in such MDPs, we obtain better representations in fewer epochs compared to representation learning approaches using reconstructions, while generalizing better to new goals than model-free approaches. Additionally, we show that action-equivariance is a generalization of the older notion of group equivariance, which suggests that work on equivariance can be viewed from the perspective of homomorphisms that act to move from one state (point in a space) to another state (point in a space).

Thus, learned representations that are action-equivariant are plannable with exact planning methods. We next consider the problem of extracting structure within a state, in the form of object-oriented representation learning. We can now discuss the final research question: *How can we learn representations of the world that capture the structure in individual states?* Learning structured representations of the world is an important step towards better generalization in downstream tasks. Chapter 5 discusses Contrastively-trained Structured World Models (C-SWMs), an approach to representation learning that uses a contrastive method to learn compositional representations of world states. We show that C-SWMs can recover structured representations from unstructured image inputs, and can generalize to new scenes better than reconstruction-based baselines and ablated versions.

General Conclusions

The focus of this thesis was to leverage symmetry and structure in deep reinforcement learning problems. We considered two views on this problem: incorporating prior information in the form of symmetries, and finding representations that capture the structure in the world. We have shown that using symmetries in single and multi agent systems can leverage outside knowledge to reduce the amount of environment interactions necessary to finding solutions. We have also shown that it is possible to recover plannable representations by focusing on learning action-equivariant representations, and that in object-oriented problems with factored action spaces, we are able to recover the object structure, and use this to better predict future states.

Limitations

The work presented in this thesis is only the first step in this research area. Many open challenges remain. In certain problems, we might not be able to identify the symmetries a priori. In general, computing symmetries in MDPs is isomorphism complete (complexity class of testing whether two graphs are isomorphic) when the dynamics of the system are known [127]. Developing methods for learning these symmetries from data is a promising emerging research area [189]. It is important to correctly identify symmetries, as the approaches presented here are not suitable for asymmetric problems. For example, consider a cartpole problem where the right side of the tracks has more friction than the left side of the tracks. In such a case, the dynamics of the problem are not symmetric, even if they appear that way to a system designer. Another example is playing a game with symmetric dynamics with an asymmetric opponent. In football one might expect a symmetry to exist between the top and the bottom half of the field. However, if the opposing team has a bias to playing through the top half, the resulting dynamics are asymmetric. For problems that are not symmetric, using a symmetric equivariant policy network such as the methods presented here is suboptimal; we force the agent to use symmetric policies only. If the optimal policy is asymmetric, this means we exclude the optimal policy from the set of possible policies.

Enforcing symmetries in neural networks tends to introduce additional computational overhead. In practice, we need to balance computational speed with sample efficiency. Problems with a high cost per sample will benefit from using equivariance, whereas in problems with cheap samples and a necessity for fast decision making it might be preferable to use faster neural network architecturess combined with large numbers of samples.

In partially observable problems, symmetries can depend on the whole history of a trajectory [74]. This means that any network that uses such information needs to propagate geometry information through processing the whole trajectory. In practice, this means developing methods that propagate equivariant information through recurrent networks.

Throughout this work, we have focused on exact symmetries. In many problems, the symmetries may only be approximate. For example, in continuously rotating a robot arm we may wish to treat rotations of -90° and 90.5° as symmetric. Some notion of ϵ-symmetry may be useful in problems like these. Additionally, some symmetries may be *partial*: opening the door on the left is symmetric to opening the door

on the right, except the door on the right has a different color handle. Dealing with situations like these will require us to find state-action representations that abstract away irrelevant differences, for example through learning MDP homomorphisms (see Chapter 4).

For planning in learned representations, there are limits to using dynamic programming, as we cannot always consider the full state space before acting. Additionally, as the world is dynamic, there is value in developing methods that are dynamic as well, changing representations on the fly as the agent encounters changes in the environmental dynamics. Finally, our representation learning methods have focused primarily on deterministic and Markov problems. Generalizing these methods to stochastic and partially observable problems is an important step to later applications.

7

Acknowledgments

I'd like to take this section to thank everyone who so generously gave to me of their time, their support, or their insight. First of all, I'm extremely grateful to my promotor, Max Welling, who beyond his brilliant intellect is a kind and creative person. Max, thank you for your support, guidance, and the trust and freedom you gave me to pursue the research that inspired me. I am furthermore deeply indebted to my co-promotor Frans Oliehoek, who besides his sharp mind and encyclopedic knowledge of the pre-deep learning RL literature, has championed for me since before the PhD. Frans, I cannot express how much your support has meant to me in finding my way as a researcher. Further, I would like to express my deepest appreciation to my co-promotor Herke van Hoof, whose insightful questions and conscientious attitude have made my work stronger and my life easier. Herke, thank you for being a supervisor I could always fall back on.

I'd like to extend my gratitude to the esteemed committee for my PhD: Dr. Doina Precup, Dr. Jan Peters, Dr. Evangelos Kanoulas, Dr. Maarten de Rijke, Dr. Erik Bekkers.

Thanks Yuge Shi and Karen Ullrich. Yuge, over the past few years you have become one of my closest friends. It's hard to describe what your friendship means to me. Karen, you have been an incredible support and cheerleader for me and I'm grateful for your wonderful friendship.

I furthermore want to extend my thanks to Rianne van den Berg and Zeynep Akata for their time, advice, and friendship through the years.

I'd also like to thank Bert Bredeweg, for encouraging me to continue my studies after undergrad, and Leo Dorst, for always being kind, honest, and a wonderful teacher. Many thanks also to Cees Snoek, Katja Hofmann, Jan-Willem van de Meent, Diederik Roijers, and Sam Devlin for their time, advice, and support.

I would like to express my appreciation to my close collaborators Daniel Worrall and Thomas Kipf. Thank you for being brilliant and a delight to work with. Thanks as well to my other co-authors: Johannes Brandstetter, Rob Hesselink, Ondrej Biza, Jose Gallego-Posada, Laurens Weitkamp, Jakob Foerster, Darius Muglich, Christian Schroeder de Witt, Shimon Whiteson, Tejaswi Kasarla, Gertjan Burghouts, Max van Spengler, Rita Cucchiara, Rahul Savani, Roderich Groß.

Doing a PhD in AMLAB has been an incredible experience. A heartfelt thanks to the many insightful, kind, creative, smart, and funny labmates from AMLAB and beyond who I had the good fortune of doing a PhD with. Thank you Sadaf, Maxi, Bas, Victor, Tim, Ivan, Sindy, Taco, Jakub, Marco, Qi, Maartje, Gabriele, Maurice, Artem, Emiel, Tessa, Matthias, Christos, Wendy, Sara, Sharvaree, Putri. Thank you as well to Felice and Virginie for taking many tasks out of our hands, and to Dennis for solving all of our computer problems.

Many thanks to the good people at DeepMind, who put a lot of effort into turning a remote internship into a wonderful experience. Thanks to my supervisors Ian Gemp, Richard Everett, and Yoram Bachrach. Many thanks also to Danielle Belgrave, Feryal Behbahani, Luisa Zintgraf and all the other kind folks at DeepMind. Additionally, I'd also like to thank Michael Herman, Christian Daniel, and everyone else at the Bosch Center for AI.

Finally, I'd like to thank my family and friends. Pascal, je bent mijn steun en toeverlaat. Bedankt voor je grenzeloze enthousiasme, alle middernachtelijke peptalks, en de cuba libres. Mam, bedankt voor je standvastige vertrouwen in mijn kunnen, en bedankt dat je me hebt opgevoed met het idee dat ik alles kan waar ik mijn best voor doe. Emma, ik ben je zeer dankbaar voor je vriendschap van de afgelopen 16 jaar. Ik kan mijn leven niet voorstellen zonder jou. Ik ben heel dankbaar voor mijn andere lieve vrienden, die een bron waren van energie, comfort, en afleiding tijdens de PhD: Eszter, Bas, Tessa, Chiel, Sander, Jelte, Willemijn, Jorn, Fabien. Bedankt familie Mazier, dat jullie mij lang geleden opgevangen hebben. Bedankt familie Mettes, dat jullie mij zo volledig in de familie opgenomen hebben.

Bibliography

[1] Farzad Abdolhosseini, Hung Yu Ling, Zhaoming Xie, Xue Bin Peng, and Michiel van de Panne. On learning symmetric locomotion. In *ACM SIGGRAPH Motion, Interaction, and Games*. 2019.

[2] Rishabh Agarwal, Marlos C. Machado, Pablo Samuel Castro, and Marc G. Bellemare. Contrastive behavioral similarity embeddings for generalization in reinforcement learning. In *International Conference on Learning Representations*, 2021.

[3] Pulkit Agrawal, Ashvin Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. In *Advances in Neural Information Processing Systems*, 2016.

[4] Nele Albers, Miguel Suau, and Frans A. Oliehoek. Using bisimulation metrics to analyze and evaluate latent state representations. In *Belgian-Dutch Conference on Machine Learning*, 2021.

[5] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety. *arXiv:1606.06565*, 2016.

[6] Ankesh Anand, Evan Racah, Sherjil Ozair, Yoshua Bengio, Marc-Alexandre Cote, and R. Devon Hjelm. Unsupervised state representation learning in Atari. In *Advances in Neural Information Processing Systems*, 2019.

[7] Brandon Anderson, Truong Son Hy, and Risi Kondor. Cormorant: Covariant molecular neural networks. In *Advances in Neural Information Processing Systems*, 2019.

[8] Masataro Asai. Unsupervised grounding of plannable first-order logic representation from images. In *International Conference on Automated Planning and Scheduling*, 2019.

[9] Yusuf Aytar, Tobias Pfaff, David Budden, Thomas Paine, Ziyu Wang, and Nando de Freitas. Playing hard exploration games by watching youtube. In *Advances in Neural Information Processing Systems*, 2018.

[10] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, 1983.

[11] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and Koray Kavukcuoglu. Interaction networks for learning about objects, relations and physics. In *Advances in Neural Information Processing*, 2016.

[12] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[13] Erik J. Bekkers. B-spline CNNs on Lie groups. In *International Conference on Learning Representations*, 2019.

[14] Erik J. Bekkers, Maxime W. Lafarge, Mitko Veta, Koen A.J. Eppenhof, Josien P.W. Pluim, and Remco Duits. Roto-translation covariant convolutional networks for medical image analysis. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, 2018.

[15] Richard E. Bellman. *Dynamic Programming*. 1957.

[16] Ondrej Biza and Robert Platt. Online abstraction with MDP homomorphisms for deep learning. In *International Conference on Autonomous Agents and Multi-Agent Systems*, 2019.

[17] Ondrej Biza, Elise van der Pol, and Thomas Kipf. The impact of negative sampling on contrastive structured world models. In *ICML Workshop on Self-Supervised Learning for Reasoning and Perception*, 2021.

[18] Wendelin Böhmer, Vitaly Kurin, and Shimon Whiteson. Deep coordination graphs. In *International Conference on Machine Learning*, 2020.

[19] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Advances in Neural Information Processing*, 2013.

[20] Nicolò Botteghi, Mannes Poel, Beril Sirmacek, and Christoph Brune. Low-dimensional state and action representation learning with MDP homomorphism metrics. *arXiv preprint arXiv:2107.01677*, 2021.

[21] Craig Boutilier. Planning, learning and coordination in multiagent decision processes. In *Conference on Theoretical aspects of rationality and knowledge*, 1996.

[22] Craig Boutilier, Richard Dearden, Moisés Goldszmidt, et al. Exploiting structure in policy construction. In *International Joint Conference on Artificial Intelligence*, 1995.

[23] Johannes Brandstetter, Rob Hesselink, Elise van der Pol, Erik J. Bekkers, and Max Welling. Geometric and physical quantities improve E(3) equivariant message passing. In *International Conference on Learning Representations*, 2021.

[24] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI gym. *arXiv preprint arXiv:1606.01540*, 2016.

[25] Christopher P Burgess, Loic Matthey, Nicholas Watters, Rishabh Kabra, Irina Higgins, Matt Botvinick, and Alexander Lerchner. Monet: Unsupervised scene decomposition and representation. *arXiv preprint arXiv:1901.11390*, 2019.

[26] Pablo Samuel Castro. Scalable methods for computing state similarity in deterministic Markov decision processes. In *AAAI Conference on Artificial Intelligence*, 2020.

[27] Pablo Samuel Castro, Tyler Kastner, Prakash Panangaden, and Mark Rowland. MICo: Improved representations via sampling-based state similarity for Markov decision processes. In *Advances in Neural Information Processing Systems*, 2021.

[28] Michael B. Chang, Tomer Ullman, Antonio Torralba, and Joshua B. Tenenbaum. A compositional object-based approach to learning physical dynamics. In *International Conference on Learning Representations*, 2017.

[29] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International Conference on Machine Learning*, 2020.

[30] Christopher Clark and Amos Storkey. Teaching deep convolutional neural networks to play Go. In *International Conference on Machine Learning*, 2015.

[31] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In *International Conference on Machine Learning*, 2019.

[32] Taco S. Cohen, Mario Geiger, Jonas Koehler, and Max Welling. Spherical CNNs. In *International Conference on Learning Representations*, 2018.

[33] Taco S. Cohen, Mario Geiger, and Maurice Weiler. A general theory of equivariant CNNs on homogeneous spaces. In *Advances in Neural Information Processing Systems*. 2019.

[34] Taco S. Cohen and Max Welling. Group equivariant convolutional networks. In *International Conference on Machine Learning*, 2016.

[35] Taco S. Cohen and Max Welling. Steerable CNNs. In *International Conference on Learning Representations*, 2017.

[36] Dane Corneil, Wulfram Gerstner, and Johanni Brea. Efficient model-based deep reinforcement learning with variational state tabulation. In *International Conference on Machine Learning*, 2018.

[37] Andreea Deac, Petar Veličković, Ognjen Milinković, Pierre-Luc Bacon, Jian Tang, and Mladen Nikolić. XLVIN: executed latent value iteration nets. *arXiv preprint arXiv:2010.13146*, 2020.

[38] Thomas Dean and Robert Givan. Model minimization in Markov decision processes. In *AAAI Conference on Artifical Intelligence/IAAI Conference on Innovative Applications of Artificial Intelligence*, 1997.

[39] Neel Dey, Antong Chen, and Soheil Ghafurian. Group equivariant generative adversarial networks. In *International Conference on Learning Representations*, 2021.

[40] Nichita Diaconu and Daniel E. Worrall. Learning to convolve: A generalized weight-tying approach. In *International Conference on Machine Learning*, 2019.

[41] Alexey Dosovitskiy, Jost Tobias Springenberg, Martin Riedmiller, and Thomas Brox. Discriminative unsupervised feature learning with convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2014.

[42] David Steven Dummit and Richard M. Foote. *Abstract Algebra*. Wiley, 2004.

[43] Sebastien Ehrhardt, Aron Monszpart, Niloy Mitra, and Andrea Vedaldi. Unsupervised intuitive physics from visual observations. In *Asian Conference on Computer Vision*, 2018.

[44] Martin Engelcke, Adam R Kosiorek, Oiwi Parker Jones, and Ingmar Posner. GENESIS: Generative scene inference and sampling with object-centric latent representations. *International Conference on Learning Representations*, 2020.

[45] Gregory Farquhar, Tim Rocktäschel, Maximilian Igl, and SA Whiteson. TreeQN and ATreeC: Differentiable tree-structured models for deep reinforcement learning. In *International Conference on Learning Representations*, 2018.

[46] Norm Ferns, Prakash Panangaden, and Doina Precup. Metrics for finite Markov decision processes. In *Conference on Uncertainty in Artificial Intelligence*, 2004.

[47] Norm Ferns, Prakash Panangaden, and Doina Precup. Bisimulation metrics for continuous Markov decision processes. *SIAM Journal on Computing*, 2011.

[48] Norman Ferns, Pablo Samuel Castro, Doina Precup, and Prakash Panangaden. Methods for computing state similarity in Markov decision processes. In *Conference on Uncertainty in Artificial Intelligence*, 2012.

[49] Norman Ferns and Doina Precup. Bisimulation metrics are optimal value functions. In *Conference on Uncertainty in Artificial Intelligence*, 2014.

[50] Marc Finzi, Samuel Stanton, Pavel Izmailov, and Andrew Gordon Wilson. Generalizing convolutional neural networks for equivariance to Lie groups on arbitrary continuous data. In *International Conference on Machine Learning*, 2020.

[51] Vincent François-Lavet, Yoshua Bengio, Doina Precup, and Joelle Pineau. Combined reinforcement learning via abstract representations. In *AAAI Conference on Artificial Intelligence*, 2019.

[52] Fabian B Fuchs, Daniel E Worrall, Volker Fischer, and Max Welling. SE (3)-transformers: 3D roto-translation equivariant attention networks. *Advances in Neural Information Processing Systems*, 2020.

[53] Victor Garcia Satorras and Joan Bruna. Few-shot learning with graph neural networks. In *International Conference on Learning Representations*, 2018.

[54] Carles Gelada, Saurabh Kumar, Jacob Buckman, Ofir Nachum, and Marc G. Bellemare. DeepMDP: Learning continuous latent space models for representation learning. In *International Conference on Machine Learning*, 2019.

[55] Dibya Ghosh, Abhishek Gupta, and Sergey Levine. Learning actionable representations with goal conditioned policies. In *International Conference on Learning Representations*, 2019.

[56] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, 2017.

[57] Robert Givan, Thomas Dean, and Matthew Greig. Equivalence notions and model minimization in Markov decision processes. In *Artificial Intelligence*, 2003.

[58] Klaus Greff, Raphaël Lopez Kaufman, Rishabh Kabra, Nick Watters, Christopher Burgess, Daniel Zoran, Loic Matthey, Matthew Botvinick, and Alexander Lerchner. Multi-object representation learning with iterative variational inference. In *International Conference on Machine Learning*, 2019.

[59] Klaus Greff, Sjoerd van Steenkiste, and Jürgen Schmidhuber. Neural expectation maximization. In *Advances in Neural Information Processing*, 2017.

[60] Klaus Greff, Sjoerd van Steenkiste, and Jürgen Schmidhuber. On the binding problem in artificial neural networks. *arXiv preprint arXiv:2012.05208*, 2020.

[61] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *International Conference on Knowledge Discovery and Data Mining*, 2016.

[62] Carlos Guestrin, Daphne Koller, and Ronald Parr. Multiagent planning with factored MDPs. In *Advances in Neural Information Processing Systems*, 2002.

[63] Carlos Guestrin, Daphne Koller, Ronald Parr, and Shobha Venkataraman. Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research*, 2003.

[64] Carlos Guestrin, Michail Lagoudakis, and Ronald Parr. Coordinated reinforcement learning. In *International Conference on Machine Learning*, 2002.

[65] Carlos Guestrin, Shobha Venkataraman, and Daphne Koller. Context-specific multiagent coordination and planning with factored MDPs. In *AAAI Conference on Artifical Intelligence/IAAI Conference on Innovative Applications of Artificial Intelligence*, 2002.

[66] David Ha and Jürgen Schmidhuber. World models. *arxiv preprint arXiv:1803.10122*, 2018.

[67] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, 2019.

[68] Charles R. Harris, K. Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 2020.

[69] Juris Hartmanis and R. E. Stearns. *Algebraic Structure Theory Of Sequential Machines*. Prentice-Hall, Inc., 1966.

[70] Peter Henderson, Riashat Islam, Joelle Pineau, David Meger, Doina Precup, and Philip Bachman. Deep reinforcement learning that matters. In *AAAI Conference on Artificial Intelligence*, 2018.

[71] Irina Higgins, David Amos, David Pfau, Sebastien Racaniere, Loic Matthey, Danilo Rezende, and Alexander Lerchner. Towards a definition of disentangled representations. *arxiv preprint arXiv:1812.02230*, 2018.

[72] R. Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. In *International Conference on Learning Representations*, 2019.

[73] Yedid Hoshen. VAIN: Attentional multi-agent predictive modeling. In *Advances in Neural Information Processing*, 2017.

[74] Hengyuan Hu, Adam Lerer, Alex Peysakhovich, and Jakob Foerster. "Other-play" for zero-shot coordination. In *International Conference on Machine Learning*, 2020.

[75] Wenlong Huang, Igor Mordatch, and Deepak Pathak. One policy to control them all: Shared modular policies for agent-agnostic control. In *International Conference on Machine Learning*, 2020.

[76] Maximilian Igl, Luisa Zintgraf, Tuan Anh Le, Frank Wood, and Shimon Whiteson. Deep variational reinforcement learning for POMDPs. In *International Conference on Machine Learning*, 2018.

[77] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z. Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. In *International Conference on Learning Representations*, 2017.

[78] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with Gumbel-Softmax. In *International Conference on Learning Representations*, 2017.

[79] Michael Janner, Sergey Levine, William T Freeman, Joshua B Tenenbaum, Chelsea Finn, and Jiajun Wu. Reasoning about physical interactions with object-oriented prediction and planning. In *International Conference on Learning Representations*, 2019.

[80] Miguel Jaques, Michael Burke, and Timothy Hospedales. Physics-as-inverse-graphics: Joint unsupervised learning of objects and physics from video. *arXiv preprint arXiv:1905.11169*, 2019.

[81] Jiechuan Jiang, Chen Dun, Tiejun Huang, and Zongqing Lu. Graph convolutional reinforcement learning. In *International Conference on Learning Representations*, 2020.

[82] Rico Jonschkowski and Oliver Brock. Learning state representations with robotic priors. In *Autonomous Robots*, 2015.

[83] Daniel Kahneman and Anne Treisman. *Changing views of Attention and Automaticity.* Academic Press, Inc., 1984.

[84] Daniel Kahneman, Anne Treisman, and Brian J Gibbs. The reviewing of object files: Object-specific integration of information. *Cognitive psychology*, 1992.

[85] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H. Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. Model-based reinforcement learning for Atari. In *International Conference on Learning Representations*, 2020.

[86] Peter Karkus, David Hsu, and Wee Sun Lee. QMDP-net: Deep learning for planning under partial observability. In *Advances in Neural Information Processing Systems*, 2017.

[87] Tejaswi Kasarla, Gertjan J Burghouts, Max van Spengler, Elise van der Pol, Rita Cucchiara, and Pascal Mettes. Maximum class separation as inductive bias in one matrix. *arXiv preprint arXiv:2206.08704*, 2022.

[88] Michael Kearns and Daphne Koller. Efficient reinforcement learning in factored MDPs. In *International Joint Conference on Artificial Intelligence*, 1999.

[89] Mete Kemertas and Tristan Aumentado-Armstrong. Towards robust bisimulation metric learning. *Advances in Neural Information Processing Systems*, 2021.

[90] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[91] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations*, 2014.

[92] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems. In *International Conference on Machine Learning*, 2018.

[93] Thomas Kipf, Yujia Li, Hanjun Dai, Vinicius Zambaldi, Alvaro Sanchez-Gonzalez, Edward Grefenstette, Pushmeet Kohli, and Peter Battaglia. Compile: Compositional imitation learning and execution. In *International Conference on Machine Learning*, 2019.

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

[95] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

[96] Martin Klissarov and Doina Precup. Diffusion-based approximate value functions. In *ICML Workshop on Efficient Credit Assignment in Deep Learning and Deep Reinforcement Learning*, 2018.

[97] Jelle R Kok and Nikos Vlassis. Collaborative multiagent reinforcement learning by payoff propagation. *Journal of Machine Learning Research*, 2006.

[98] Risi Kondor and Shubhendu Trivedi. On the generalization of equivariance and convolution in neural networks to the action of compact groups. In *International Conference on Machine Learning*, 2018.

[99] Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. In *International Conference on Learning Representations*, 2021.

[100] Landon Kraemer and Bikramjit Banerjee. Multi-agent reinforcement learning as a rehearsal for decentralized planning. *Neurocomputing*, 2016.

[101] Schütt Kristof, Kindermans Pieter-Jan, Sauceda Huziel, Chmiela Stefan, Tkatchenko Alexandre, and Klaus-Robert Müller. Schnet: a continuous-filter convolutional neural network for modeling quantum interactions. In *Advances in Neural Information Processing Systems*, 2017.

[102] Vitaly Kurin, Maximilian Igl, Tim Rocktäschel, Wendelin Boehmer, and Shimon Whiteson. My body is a cage: the role of morphology in graph-based incompatible control. In *International Conference on Learning Representations*, 2021.

[103] Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. In *International Conference on Learning Representations*, 2018.

[104] Thanard Kurutach, Aviv Tamar, Ge Yang, Stuart Russell, and Pieter Abbeel. Learning plannable representations with causal infogan. In *Advances in Neural Information Processing Systems*, 2018.

[105] Lior Kuyer, Shimon Whiteson, Bram Bakker, and Nikos Vlassis. Multiagent reinforcement learning for urban traffic control using coordination graphs. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 2008.

[106] Charline Le Lan, Marc G Bellemare, and Pablo Samuel Castro. Metrics and continuity in reinforcement learning. In *AAAI Conference on Artificial Intelligence*, 2021.

[107] Michael Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. In *Advances in Neural Information Processing Systems*, 2020.

[108] Adrien Laversanne-Finot, Alexandre Péré, and Pierre-Yves Oudeyer. Curiosity driven exploration of learned disentangled goal spaces. In *Conference on Robot Learning*, 2018.

[109] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *IEEE*, 1998.

[110] Yann LeCun, Sumit Chopra, Raia Hadsell, M Ranzato, and F Huang. A tutorial on energy-based learning. *Predicting structured data*, 2006.

[111] Kimin Lee, Kibok Lee, Jinwoo Shin, and Honglak Lee. Network randomization: A simple technique for generalization in deep reinforcement learning. In *International Conference on Learning Representations*, 2020.

[112] Lisa Lee, Emilio Parisotto, Devendra Singh Chaplot, Eric P. Xing, and Ruslan Salakhutdinov. Gated path planning networks. In *International Conference on Machine Learning*, 2018.

[113] Lihong Li, Thomas J. Walsh, and Michael L. Littman. Towards a unified theory of state abstraction for MDPs. In *International Symposium on Artificial Intelligence and Mathematics*, 2006.

[114] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[115] Yijiong Lin, Jiancong Huang, Matthieu Zimmer, Yisheng Guan, Juan Rojas, and Paul Weng. Invariant transform experience replay: Data augmentation for deep reinforcement learning. *IEEE Robotics and Automation Letters*, 2020.

[116] Guoqing Liu, Chuheng Zhang, Li Zhao, Tao Qin, Jinhua Zhu, Jian Li, Nenghai Yu, and Tie-Yan Liu. Return-based contrastive representation learning for reinforcement learning. In *International Conference on Learning Representations*, 2021.

[117] Iou-Jen Liu, Raymond A. Yeh, and Alexander G. Schwing. PIC: Permutation invariant critic for multi-agent deep reinforcement learning. In *Conference on Robot Learning*, 2019.

[118] Anuj Mahajan and Theja Tulabandhula. Symmetry learning for function approximation in reinforcement learning. *arXiv:1706.02999*, 2017.

[119] Aditi Mavalankar. Goal-conditioned batch reinforcement learning for rotation invariant locomotion. In *ICLR Workshop Beyond Tabula Rasa in RL*, 2020.

[120] Pascal Mettes, Elise van der Pol, and Cees Snoek. Hyperspherical prototype networks. *Advances in Neural Information Processing Systems*, 2019.

[121] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[122] Shruti Mishra, Abbas Abdolmaleki, Arthur Guez, Piotr Trochim, and Doina Precup. Augmenting learning using symmetry in a biologically-inspired domain. *arXiv preprint arXiv:1910.00528*, 2019.

[123] Andriy Mnih and Yee Whye Teh. A fast and simple algorithm for training neural probabilistic language models. In *International Conference on Machine Learning*, 2012.

[124] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Tim Harley, Timothy P. Lillicrap, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, 2016.

[125] 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. In *Nature*, 2015.

[126] Arnab Kumar Mondal, Vineet Jain, Kaleem Siddiqi, and Siamak Ravanbakhsh. EqR: Equivariant representations for data-efficient reinforcement learning. In *International Conference on Machine Learning*, 2021.

[127] Shravan Matthur Narayanamurthy and Balaraman Ravindran. On the hardness of finding symmetries in Markov decision processes. In *International Conference on Machine learning*, 2008.

[128] Charlie Nash, Ali Eslami, Chris Burgess, Irina Higgins, Daniel Zoran, Theophane Weber, and Peter Battaglia. The multi-entity variational autoencoder. In *Advances in Neural Information Processing Workshops*, 2017.

[129] Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *IEEE*, 2016.

[130] Sufeng Niu, Siheng Chen, Colin Targonski, Melissa Smith, Jelena Kova evi, and Hanyu Guo. Generalized value iteration networks: Life beyond lattices. In *AAAI Conference on Artificial Intelligence*, 2018.

[131] Future of Life Institute. Autonomous weapons: An open letter from AI & robotics researchers, 2015.

[132] Junhyuk Oh, Satinder Singh, and Honglak Lee. Value prediction network. In *Advances in Neural Information Processing Systems*, 2017.

[133] Frans A. Oliehoek, Matthijs T.J. Spaan, and Nikos Vlassis. Optimal and approximate Q-value functions for decentralized POMDPs. *Journal of Artificial Intelligence Research*, 2008.

[134] Rahul Oliehoek, Frans A.and Savani, Jose Gallego, Elise van der Pol, and Roderich Groß. Beyond local Nash equilibria for adversarial networks. In *Benelux Conference on Artificial Intelligence*, 2018.

[135] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[136] Jung Yeon Park, Ondrej Biza, Linfeng Zhao, Jan-Willem van de Meent, and Robin Walters. Learning symmetric embeddings for equivariant world models. In *International Conference on Machine Learning*, 2021.

[137] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *Advances in Neural Information Processing Autodiff Workshop*, 2017.

[138] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, 2019.

[139] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *International Conference on Knowledge Discovery and Data Mining*, 2014.

[140] Martin L Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1994.

[141] Balaraman Ravindran and Andrew G. Barto. Symmetries and model minimization in Markov decision processes. Technical report, University of Massachusetts, 2001.

[142] Balaraman Ravindran and Andrew G. Barto. SMDP homomorphisms: An algebraic approach to abstraction in semi-Markov decision processes. In *International Joint Conference on Artificial Intelligence*, 2003.

[143] Balaraman Ravindran and Andrew G. Barto. Approximate homomorphisms: A framework for non-exact minimization in Markov decision processes. In *International Conference on Knowledge Based Computer Systems*, 2004.

[144] Philipp Robbel, Frans A. Oliehoek, and Mykel J. Kochenderfer. Exploiting anonymity in approximate linear programming: Scaling to large multiagent MDPs. In *AAAI Conference on Artificial Intelligence*, 2016.

[145] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In *Advances in Neural Information Processing*, 2017.

[146] Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control. In *International Conference on Machine Learning*, 2018.

[147] Victor Garcia Satorras, Emiel Hoogeboom, and Max Welling. E(n) equivariant graph neural networks. In *International Conference on Machine Learning*, 2021.

[148] Victor Garcia Satorras and Max Welling. Neural enhanced belief propagation on factor graphs. In *International Conference on Artificial Intelligence and Statistics*, 2021.

[149] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 2009.

[150] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. 2017.

[151] Nicol N. Schraudolph, Peter Dayan, and Terrence J. Sejnowski. Temporal difference learning of position evaluation in the game of Go. In *Advances in Neural Information Processing Systems*, 1994.

[152] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy P. Lillicrap, and David Silver. Mastering Atari, Go, chess and Shogi by planning with a learned model. 2019.

[153] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. In *arXiv:1707.06347*, 2017.

[154] Dale Schuurmans and Relu Patrascu. Direct value-approximation for factored MDPs. *Advances in Neural Information Processing Systems*, 2001.

[155] Dhruv Sharma, Alihusein Kuwajerwala, and Florian Shkurti. Augmenting imitation experience via equivariant representations. In *International Conference on Robotics and Automation*, 2022.

[156] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. In *Nature*, 2016.

[157] Gregor N.C. Simm, Robert Pinsler, Gábor Csányi, and José Miguel Hernández-Lobato. Symmetry-aware actor-critic for 3D molecular design. In *International Conference on Learning Representations*, 2021.

[158] Aravind Srinivas, Michael Laskin, and Pieter Abbeel. CURL: Contrastive unsupervised representations for reinforcement learning. In *International Conference on Machine Learning*, 2020.

[159] Adam Stooke and Pieter Abbeel. rlpyt: A research code base for deep reinforcement learning in Pytorch. *arxiv preprint arXiv:1909.01500*, 2019.

[160] Sainbayar Sukhbaatar, Arthur Szlam, and Rob Fergus. Learning multiagent communication with backpropagation. In *Advances in Neural Information Processing Systems*, 2016.

[161] Chen Sun, Abhinav Shrivastava, Carl Vondrick, Kevin Murphy, Rahul Sukthankar, and Cordelia Schmid. Actor-centric relation network. In *European Conference on Computer Vision*, 2018.

[162] Chen Sun, Abhinav Shrivastava, Carl Vondrick, Rahul Sukthankar, Kevin Murphy, and Cordelia Schmid. Relational action forecasting. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[163] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. In *International Conference on Autonomous Agents and Multi-Agent Systems*, 2018.

[164] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. In *Advances in Neural Information Processing Systems*, 2016.

[165] Jonathan Taylor, Doina Precup, and Prakash Panagaden. Bounding performance loss in approximate MDP homomorphisms. In *Advances in Neural Information Processing Systems*, 2008.

[166] Nathaniel Thomas, Tess Smidt, Steven Kearnes, Lusann Yang, Li Li, Kai Kohlhoff, and Patrick Riley. Tensor field networks: Rotation-and translation-equivariant neural networks for 3D point clouds. *arXiv preprint arXiv:1802.08219*, 2018.

[167] Valentin Thomas, Emmanuel Bengio, William Fedus, Jules Pondard, Philippe Beaudoin, Hugo Larochelle, Joelle Pineau, Doina Precup, and Yoshua Bengio. Disentangling the independently controllable factors of variation by interacting with the world. *arXiv preprint arXiv:1802.09484*, 2018.

[168] Valentin Thomas, Jules Pondard, Emmanuel Bengio, Marc Sarfati, Philippe Beaudoin, Marie-Jean Meurs, Joelle Pineau, Doina Precup, and Yoshua Bengio. Independently controllable factors. *arxiv preprint arXiv:1708.01289*, 2017.

[169] Elise van der Pol, Ian Gemp, Yoram Bachrach, and Richard Everett. Stochastic parallelizable eigengap dilation for large graph clustering. *arXiv preprint arXiv:2207.14589*, 2022.

[170] Elise van der Pol, Thomas Kipf, Frans A. Oliehoek, and Max Welling. Plannable approximations to MDP homomorphisms: Equivariance under actions. In *International Conference on Autonomous Agents and Multi-Agent Systems*, 2020.

[171] Elise van der Pol and Frans A. Oliehoek. Coordinated deep reinforcement learners for traffic light control. In *NeurIPS Workshop on Learning, Inference and Control of Multi-Agent Systems*, 2016.

[172] Elise van der Pol, Herke van Hoof, Frans A. Oliehoek, and Max Welling. Multi-agent MDP homomorphic networks. In *International Conference on Learning Representations*, 2022.

[173] Elise van der Pol, Daniel E. Worrall, Herke van Hoof, Frans A. Oliehoek, and Max Welling. MDP homomorphic networks: Group symmetries in reinforcement learning. In *Advances in Neural Information Processing Systems*, 2020.

[174] Sjoerd van Steenkiste, Michael Chang, Klaus Greff, and Jürgen Schmidhuber. Relational neural expectation maximization: Unsupervised discovery of objects and their interactions. In *International Conference on Learning Representations*, 2018.

[175] Pradeep Varakantham, Yossiri Adulyasak, and Patrick Jaillet. Decentralized stochastic planning with anonymity in interactions. In *AAAI Conference on Artificial Intelligence*, 2014.

[176] Petar Veličković, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R. Devon Hjelm. Deep graph infomax. In *International Conference on Learning Representations*, 2019.

[177] K.P. Wabersich and M.N. Zeilinger. Linear model predictive safety certification for learning-based control. In *IEEE Conference on Decision and Control*, 2018.

[178] Angelina Wang, Thanard Kurutach, Kara Liu, Pieter Abbeel, and Aviv Tamar. Learning robotic manipulation through visual planning and acting. In *Robotics: Science and Systems*, 2019.

[179] Dian Wang, Robin Walters, and Robert Platt. SO(2)-equivariant reinforcement learning. In *International Conference on Learning Representations*, 2022.

[180] Tingwu Wang, Renjie Liao, Jimmy Ba, and Sanja Fidler. Nervenet: Learning structured policy with graph neural networks. In *International Conference on Learning Representations*, 2018.

[181] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In *AAAI Conference on Artificial Intelligence*, 2014.

[182] Manuel Watter, Jost Tobias Springenberg, Joschka Boedecker, and Martin Riedmiller. Embed to control: a locally linear latent dynamics model for control from raw images. In *Advances in Neural Information Processing Systems*, 2015.

[183] 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. *arxiv preprint arXiv:1905.09275*, 2019.

[184] Nicholas Watters, Daniel Zoran, Theophane Weber, Peter Battaglia, Razvan Pascanu, and Andrea Tacchetti. Visual interaction networks: Learning a physics simulator from video. In *Advances in Neural Information Processing*, 2017.

[185] Hua Wei, Guanjie Zheng, Vikash Gayah, and Zhenhui Li. A survey on traffic signal control methods. *arXiv preprint arXiv:1904.08117*, 2019.

[186] Maurice Weiler and Gabriele Cesa. General E(2)-equivariant steerable CNNs. In *Advances in Neural Information Processing Systems*, 2019.

[187] Maurice Weiler, Mario Geiger, Max Welling, Wouter Boomsma, and Taco S Cohen. 3D steerable CNNs: Learning rotationally equivariant features in volumetric data. In *Advances in Neural Information Processing Systems*, 2018.

[188] Maurice Weiler, Fred A. Hamprecht, and Martin Storath. Learning steerable filters for rotation equivariant CNNs. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[189] Matthias Weissenbacher, Samarth Sinha, Animesh Garg, and Yoshinobu Kawahara. Koopman Q-learning: Offline reinforcement learning via symmetries of dynamics. In *International Conference on Learning Representations*, 2022.

[190] Laurens Weitkamp, Elise van der Pol, and Zeynep Akata. Visual rationalizations in deep reinforcement learning for Atari games. In *Benelux Conference on Artificial Intelligence*, 2018.

[191] Marysia Winkels and Taco S. Cohen. 3D G-CNNs for pulmonary nodule detection. In *Medical Imaging with Deep Learning Conference*, 2018.

[192] Daniel E. Worrall and Gabriel J. Brostow. CubeNet: Equivariance to 3D rotation and translation. In *European Conference on Computer Vision*, 2018.

[193] Daniel E. Worrall, Stephan J. Garbin, Daniyar Turmukhambetov, and Gabriel J. Brostow. Harmonic networks: Deep translation and rotation equivariance. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[194] Daniel E. Worrall and Max Welling. Deep scale-spaces: Equivariance over scale. In *Advances in Neural Information Processing Systems*, 2019.

[195] Wenxuan Wu, Zhongang Qi, and Li Fuxin. Pointconv: Deep convolutional networks on 3D point clouds. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[196] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms. *arxiv preprint arXiv:1708.07747*, 2017.

[197] Zhenjia Xu, Zhijian Liu, Chen Sun, Kevin Murphy, William T Freeman, Joshua B Tenenbaum, and Jiajun Wu. Unsupervised discovery of parts, structure, and dynamics. In *International Conference on Learning Representations*, 2019.

[198] Dmitry Yarotsky. Universal approximations of invariant maps by neural networks. *arxiv preprint arXiv:1804.10306*, 2018.

[199] KiJung Yoon, Renjie Liao, Yuwen Xiong, Lisa Zhang, Ethan Fetaya, Raquel Urtasun, Richard Zemel, and Xaq Pitkow. Inference in probabilistic graphical models by graph neural networks. In *ICLR Workshop Track*, 2018.

[200] Amy Zhang, Adam Lerer, Sainbayar Sukhbaatar, Rob Fergus, and Arthur Szlam. Composable planning with attributes. In *International Conference on Machine Learning*, 2018.

[201] Amy Zhang, Rowan McAllister, Roberto Calandra, Yarin Gal, and Sergey Levine. Learning invariant representations for reinforcement learning without reconstruction. In *International Conference on Learning Representations*, 2020.

[202] Marvin Zhang, Sharad Vikram, Laura Smith, Pieter Abbeel, Matthew Johnson, and Sergey Levine. SOLAR: Deep structured representations for model-based reinforcement learning. In *International Conference on Machine Learning*, 2019.

[203] Xupeng Zhu, Dian Wang, Ondrej Biza, Guanang Su, Robin Walters, and Robert Platt. Sample efficient grasp learning using equivariant models. In *Robotics: Science and Systems (RSS)*, 2022.

[204] Martin Zinkevich and Tucker Balch. Symmetry in Markov decision processes and its implications for single agent and multi agent learning. In *International Conference on Machine Learning*, 2001.

8

Summary

In this thesis, we study symmetry and structure in deep reinforcement learning. We divide the thesis into two different parts. In the first, we explore how to leverage knowledge of symmetries in reinforcement learning problems. In the second, we propose methods to learning about the structure of an agent's environment and individual states.

Our contributions are as follows: In Part 1 we use existing knowledge of symmetries to gain improvements in data efficiency in MDPs (Chapter 2) and knowledge of symmetries and structure to improve data efficiency in multi-agent MDPs (Chapter 3).

- We propose *MDP Homomorphic Networks* (Chapter 2) [173]. MDP homomorphic networks are neural networks that are equivariant under symmetries in the joint state-action space of an MDP. Due to their equivariance, we find improved data efficiency compared to non-equivariant baselines.

- We propose *Multi-Agent MDP Homomorphic Networks* (Chapter 3) [172]. Multi-Agent MDP Homomorphic Networks form a class of networks that allows distributed execution using only local information, yet is able to share experience between global symmetries in the joint state-action space of cooperative multi-agent systems. We show that global equivariance improves data efficiency compared to non-equivariant distributed networks on symmetric coordination problems.

In Part 2 we consider learning the underlying graphs of MDPs (Chapter 4) and structure in individual states (Chapter 5).

- We propose *PRAE* (Chapter 4) [170]. PRAE exploits action equivariance for representation learning in reinforcement learning. Equivariance under actions states that transitions in the input space are mirrored by equivalent transitions in latent space, while the map and transition functions should also commute. We prove that under certain assumptions, the map we learn is an MDP homomorphism and show empirically that the approach is data-efficient and fast to train, generalizing well to new goal states and instances with the same environmental dynamics.

- We propose *C-SWMs* (Chapter 5) [94]. C-SWMs find object-oriented representations of states from pixels, using contrastive coding and graph neural network transition functions. We show improvement in multi-step prediction and generalization to unseen environment configurations compared to models that use decoders, unstructured transitions, or unstructured representations.

9

Samenvatting - Dutch Summary

In deze dissertatie bestuderen we symmetrieën en structuur in *deep reinforcement learning*. De dissertatie bestaat uit twee delen. In het eerste deel onderzoeken we hoe we kennis over symmetrieën kunnen gebruiken bij het oplossen van taken in *reinforcement learning*. In het tweede deel stellen we methoden voor om over de structuur van de omgeving van een agent en de structuur van individuele omgevingstoestanden te leren.

Onze bijdragen zijn als volgt: In Deel 1 gebruiken we voorafgaande kennis over symmetrieen om verbeteringen wat betreft data efficiëntie in *MDP*s (Hoofdstuk 2) en *multi-agent MDPs* (Hoofdstuk 3) te verkrijgen.

- We stellen *MDP Homomorphic Networks* (Hoofdstuk 2) [173] voor. *MDP homomorphic networks* zijn neurale netwerken die equivariant zijn onder symmetrieen in de gezamenlijke ruimte van omgevingstoestanden en acties in een *MDP*. Door deze equivariantie ontdekken we een verbetering in data efficiëntie vergeleken met netwerken die niet equivariant zijn.

- We stellen *Multi-Agent MDP Homomorphic Networks* (Hoofdstuk 3) [172] voor. *Multi-Agent MDP Homomorphic Networks* zijn een klasse neurale netwerken die gedistribueerde uitvoering toestaan en daarvoor alleen lokale informatie nodig hebben, doch alsnog ervaringen kan

delen tussen globale symmetrieen in de gezamenlijke ruimte van omgevingstoestanden en acties in cooperatieve *multi-agent* systemen. We laten zien dat in symmetrische coordinatie problemen het gebruik van globale equivariantie de data efficientie verbetert ten opzichte van niet-equivariante gedistribueerde netwerken.

In Deel 2 beschouwen we het leren van de onderliggende grafen van *MDPs* (Hoofdstuk 4) en het leren van de structuur in individuele omgevingstoestanden (Hoofdstuk 5).

- We stellen *PRAE* (Hoofdstuk 4) [170] voor. PRAE maakt gebruik van actie-equivariantie voor het leren van representaties in *reinforcement learning*. Equivariantie onder acties betekent dat transities in de invoerruimte gespiegeld worden door equivalente transities in de latente ruimte, terwijl de functie naar de latente ruimte en de transitiefunctie commutatief moeten zijn. We bewijzen dat de functie naar de latente ruimte die we leren onder bepaalde aannames een MDP homomorphisme is. Verder laten we zien dat de aanpak data-efficient is en snel leert, en goed generaliseert naar nieuwe doelen en instanties met dezelfde omgevingsdynamiek.

- We stellen *C-SWMs* (Hoofdstuk 5) [94] voor. C-SWMs ontdekken object-georienteerde representaties van omgevingstoestanden vanuit pixels, daarbij gebruik makende van *contrastive coding* en *graph neural network* transitiefuncties. We laten vergeleken met modellen met decoders, ongestructureerde transities of ongestructureerde representaties verbetering zien in voorspellingen over meerdere stappen en in generalisatie naar nieuwe configuraties van de omgeving.