EqR: Equivariant Representations for Data-Efficient Reinforcement Learning

Arnab Kumar Mondal [1 2 3] **Vineet Jain** [1 2] **Kaleem Siddiqi** [1 2 3] **Siamak Ravanbakhsh** [1 2]

Abstract

We study a variety of notions of equivariance as an inductive bias in Reinforcement Learning (RL). In particular, we propose new mechanisms for learning representations that are equivariant to both the agent's action, as well as symmetry transformations of the state-action pairs. Whereas prior work on exploiting symmetries in deep RL can only incorporate predefined linear transformations, our approach allows non-linear symmetry transformations of state-action pairs to be learned from the data. This is achieved through 1) equivariant Lie algebraic parameterization of state and action encodings, 2) equivariant latent transition models, and 3) the incorporation of symmetry-based losses. We demonstrate the advantages of our method, which we call Equivariant representations for RL (EqR), for Atari games in a data-efficient setting limited to 100K steps of interactions with the environment.

1. Introduction

The recent success of deep reinforcement learning (François-Lavet et al., 2018) in applications ranging from games such as Atari (Mnih et al., 2015), Go (Silver et al., 2016) and Poker (Brown & Sandholm, 2019), to robotics (Levine et al., 2016) and autonomous navigation (Bellemare et al., 2020) has demonstrated its promise as a powerful framework for sequential decision making. However, using a reward as the only signal for representation learning in contexts with high dimensional states and actions leads to tremendous data inefficiency. Notably, almost all success stories of RL rely on vast amounts of data or simulations with a substantial computational overload.

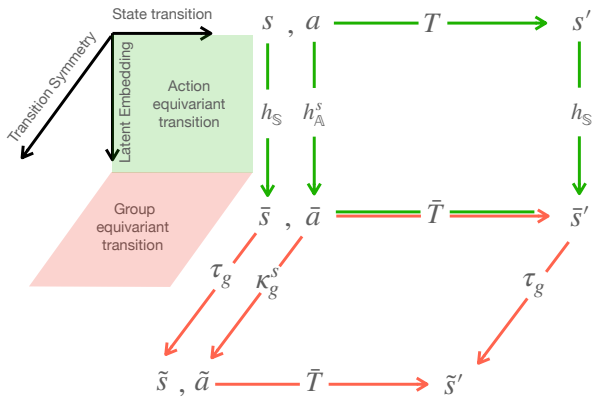


Figure 1. This figure demonstrates the relationship between two types of equivariance in latent variable modeling for an MDP with a symmetric transition function. Green arrows (vertical plane) identify a diagram for transition models in an MDP homomorphism. A model $\bar{T}$ and state embedding function $h_{\mathbb{S}}$ that are equivariant under an agent's action makes this diagram commute. Red arrows (horizontal plane) identify the commutativity diagram for a symmetric transition function of an MDP in the latent space. Here the state-action embedding $\langle \tilde{s}, \tilde{a} \rangle$ is produced through the symmetry transformation of another state-action embedding $\langle \bar{s}, \bar{a} \rangle$.

More data-efficient representation learning (Bengio et al., 2013) requires the construction of stronger inductive biases, which is a challenging problem in its own right. One general approach is to place a central emphasis on a set of *transformations* of the data, such that invariance and equivariance to them impose strong conditions on the learned representations. Intuitively, equivariance leads to better sample efficiency because by observing a specific trajectory or triplet (s, a, s'), the agent has effectively observed all of its transformations. This transformation-based viewpoint is particularly appealing in RL, where the agent is in control of some of these transformations through its actions.

In addition, transformations naturally lead to a notion of disentanglement in the representations (Higgins et al., 2018), potentially enabling better out-of-distribution generalization

[1]School of Computer Science, McGill University, Montréal, Canada [2]Mila- Quebec Artificial Intelligence Institute, Montréal, Canada [3]Centre for Intelligent Machines, McGill University, Montréal, Canada. Correspondence to: Arnab Kumar Mondal <arnab.mondal@mila.quebec>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

(Higgins et al., 2017; Thomas et al., 2017). The recent success of self-supervised learning approaches that rely on a (predefined) set of transformations (Chen et al., 2020; Zbontar et al., 2021), as also shown in the context of RL (Yarats et al., 2021; Laskin et al., 2020b), further highlights the importance of transformations in data-efficient representation learning.

Motivated by these observations, this article develops a broader perspective on the notion of equivariant representation learning within RL. In particular, we integrate equivariance under the agent's action and equivariance under the symmetries of the environment into a single latent variable model that is equivariant to an *a priori unknown* group of non-linear transformations of state-action pairs; see Figures 1 and 2. In contrast to the traditional approach of using symmetric Markov Decision Processes (MDPs), we model the larger group of state-action symmetries (separate from reward symmetries). We show how to parameterize the latent embeddings of states and actions to make the representations equivariant to continuous transformations of the environment resulting from an agent's action. We evaluate our approach, which we call Equivariant representations for RL (EqR), on the 26 games in the Atari 100K benchmark (Kaiser et al., 2019). Here we outperform other comparable methods using reliable evaluation metrics (Agarwal et al., 2021). Our approach, however, is not restricted to this domain. It is applicable in any setting where the transformations that an agent undergoes can be expressed using matrix Lie groups, including autonomous driving, navigation, and robotics.

2. Related work

The use of transformations, be it in data-augmentation or self-supervision, has become a common ingredient in recent representation learning methods for deep RL. However, theoretical work on symmetry in RL goes back to Zinkevich & Balch (2001) and Ravindran & Barto (2001), both of which use symmetric MDPs. More recent use of this formalism is in van der Pol et al. (2020b); Mondal et al. (2020), where policy networks, with built-in equivariance, are shown to improve data-efficiency. Closely related notions, that motivated the early work on symmetric MDPs, are model minimization (Ravindran & Barto, 2002), state abstraction (Ravindran & Barto, 2003; Li et al., 2006), MDP homomorphism (Ravindran & Barto, 2004) and lax bisimulations (Taylor, 2008). In particular, MDP homomorphism, which requires equivariance under an agent's action, encompasses the general idea of model-based reinforcement learning. For example, a latent MDP that matches the state dynamics and the reward distribution of the environment is learned in (van der Pol et al., 2020a; Gelada et al., 2019).

Other work in RL that is relevant to our objective includes attempts to increase data-efficiency using a learned model of the environment. While some methods such as SimPLe (Kaiser et al., 2019), learn this transition model at the pixel level, the majority of approaches use a latent space model. The latent space is either learned using reconstruction (Hafner et al., 2019a;b), or through self-supervision and contrastive methods (Laskin et al., 2020b) (CURL). However, there is evidence that the improvement in sample efficiency is largely due to image augmentation, as seen in Laskin et al. (2020a) and DrQ (Yarats et al., 2021). Using a reconstruction-based method is also inefficient because similar to pixel-level models, one needs to learn potentially irrelevant details. The fact that variations of model-free algorithms such as Data-Efficient Rainbow (DER) (van Hasselt et al., 2019) and OTRainbow (Kielak, 2019) are competitive with reconstruction-based methods without explicit representation learning components confirms this intuition. More recently SPR (Schwarzer et al., 2021) shows that data augmentation and improvements in Rainbow combined with particular forms of self-supervision, can significantly improve the sample efficiency, leading to state-of-the-art results in sample-efficient representation learning in RL.

3. Background

3.1. Groups and their Representations

A group $\mathcal{G} = \{g\}$ is a set, equipped with an associative binary operation, such that the set is closed under this operation, and each element $g \in \mathcal{G}$ has a unique inverse, such that their composition gives the identity $g^{-1}g = e$. Any subset $\mathcal{G}' \leq \mathcal{G}$ that is closed under binary operation of the groups forms a subgroup. A group $\mathcal{G}$ can *act* on a set $\mathcal{X}$ by transforming its elements $x \in \mathcal{X}$ through a bijection. We use $\alpha : \mathcal{G} \times \mathcal{X} \mapsto \mathcal{X}$ to denote the *group action*, and for brevity replace $\alpha(g, x)$ with $g \cdot x$ moving forward. The action captures some of the structure of $\mathcal{G}$ due to two constraints – the identity element acts trivially $e \cdot x = x$; and the composition of actions is equal to the action of the composition, i.e., $(gg') \cdot x = g \cdot (g' \cdot x), \forall g, g' \in \mathcal{G}$. $\mathcal{X}$ is then called a $\mathcal{G}$-set. Any $\mathcal{G}$-action partitions $\mathcal{X}$ into *orbits* $x^{\mathcal{G}} = \{g \cdot x \mid g \in \mathcal{G}\}$, and we denote the set of orbits under $\mathcal{G}$-action as $\mathcal{X}/\mathcal{G}$. A $\mathcal{G}$-action is transitive iff its action results in a single orbit.

Parameterizing Lie Groups In this work, we assume $\mathcal{G}$ is (any sub-group of) a *classical Lie group* over $\mathbb{R}$. These are the groups that can be represented using invertible matrices. We use $\rho(\mathcal{G})$ to denote a linear representation of $\mathcal{G}$, and $\rho_g : \mathbb{R}^D \to \mathbb{R}^D$ for the action (a.k.a. the representation) of $g \in \mathcal{G}$. Two other greek letters τ and κ are also used for this purpose. Many such Lie groups are identifiable by their infinitesimal generators, their *Lie algebra* $\mathfrak{g} = Lie(\rho(\mathcal{G}))$.[1] This connection enables a simple parameterization of $\rho(\mathcal{G})$

[1] This relation is bijective for "simply connected" Lie groups.

using a set of linear bases $\{\mathbf{E}^{(i)}\}_i$ for their Lie algebra – that is $\rho_g = \exp(\sum_i \beta_{g,i} \mathbf{E}^{(i)})$, where $\exp(\mathbf{Y}) = \sum_{j=0}^{\infty} \frac{\mathbf{Y}^j}{j!}$ is the matrix exponential. We refer to this parameterization later in Section 4. Such linear representations in the form of invertible matrices can be used for both continuous transformations (*e.g.*, 3D rotations) and finite groups (*e.g.*, $\times 90°$ rotations).

3.2. MDP Homomorphism and Symmetric MDPs

We define an MDP as the 4-tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, R, T \rangle$ where $\mathcal{S}$ and $\mathcal{A}$ are respectively the sets of states and actions, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, and $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}^{\geq 0}$ is the state transition function.[2] For two MDPs $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, R, T \rangle$ and $\bar{\mathcal{M}} = \langle \bar{\mathcal{S}}, \bar{\mathcal{A}}, \bar{R}, \bar{T} \rangle$, an MDP homomorphism can be defined as a tuple $\mathcal{H} = \langle h_{\mathcal{S}}, h_{\mathcal{A}} \rangle$ where $h_{\mathcal{S}} : \mathcal{S} \to \bar{\mathcal{S}}$ is the state mapping and $h_{\mathcal{A}} : \mathcal{S} \times \mathcal{A} \to \bar{\mathcal{A}}$ is the state dependent action mapping. These two mappings satisfy the following invariance and equivariance conditions:

(1) Invariance of the reward:

$$\bar{R}(h_{\mathcal{S}}(s), h_{\mathcal{A}}(s,a)) = R(s,a), \quad \forall s, a \in \mathcal{S} \times \mathcal{A} \tag{1}$$

(2) Equivariance of the deterministic transition model under the agent's action:

$$\bar{T}(h_{\mathcal{S}}(s), h_{\mathcal{A}}(s,a)) = h_{\mathcal{S}}(T(s,a)), \quad \forall s, a \in \mathcal{S} \times \mathcal{A} \tag{2}$$

A probabilistic variation of the above equation for a stochastic MDP (Bloem-Reddy & Teh, 2020) is:

$$\bar{T}(h_{\mathcal{S}}(s') \mid h_{\mathcal{S}}(s), h_{\mathcal{A}}(s,a)) = \sum_{s'' \in [s']_h} T(s'' \mid s, a) \tag{3}$$

for all $s, s' \in \mathcal{S}, a \in \mathcal{A}$, where $[s']_{h_{\mathcal{S}}} = {h_{\mathcal{S}}}^{-1}(h_{\mathcal{S}}(s'))$ is the equivalence class of s' under $h_{\mathcal{S}}$.

In related literature, MDP homomorphism is often used for *minimization* of the MDP, because the optimal policy of $\bar{\mathcal{M}}$ can be lifted to obtain the optimal counterparts for $\mathcal{M}$.

Symmetric MDPs The automorphism group $\mathcal{G}_{\mathcal{M}} = \mathrm{Aut}(\mathcal{M})$ of an MDP identifies the set of symmetry transformations of state-actions that preserve the reward and the transition dynamics:

$$R(s,a) = R(g \cdot \langle s, a \rangle)), \quad \forall g \in \mathcal{G}_{\mathcal{M}}, s \in \mathcal{S}, a \in \mathcal{A} \tag{4}$$

$$T(s' \mid s, a) = T(g \cdot s' \mid g \cdot \langle s, a \rangle) \text{ and}$$

$$g \cdot T(s,a) = T(g \cdot \langle s, a \rangle), \quad \forall g \in \mathcal{G}_{\mathcal{M}}, s, s' \in \mathcal{S}, a \in \mathcal{A} \tag{5}$$

[2] We ignore the discount factor for brevity.

We refer to a reward function R that satisfies Equation (4) as a $\mathcal{G}_{\mathcal{M}}$-invariant reward function and a deterministic transition function T that satisfies Equation (5) as a $\mathcal{G}_{\mathcal{M}}$-equivariant transition function. Note that this is a distinct notion from invariance and equivariance under the agent's action in the context of MDP homomorphism. Here, the action refers to the action of a symmetry group, while in MDP homomorphism, the equivariance is to the action of the agent. We use group action or $\mathcal{G}$-action to make this distinction clear when necessary.

For a symmetric MDP that satisfies both Equation (4) and Equation (5), both the optimal action-value and optimal policy functions become invariant under $\mathcal{G}_{\mathcal{M}}$ action (Ravindran & Barto, 2001) – that is,

$$Q(s,a) = Q(g \cdot \langle s, a \rangle) \quad \text{and}$$

$$\pi(a,s) = \pi(g \cdot \langle a, s \rangle), \quad \forall g \in \mathcal{G}_{\mathcal{M}}\ s, a \in \mathcal{S} \times \mathcal{A}. \tag{6}$$

The connection of symmetric MDPs to MDP homomorphism is due to the fact that symmetries can be used to define a homomorphism $\mathcal{H} : \mathcal{M} \mapsto \bar{\mathcal{M}}$ by collapsing the state-actions that form an orbit under $\mathcal{G}_{\mathcal{M}}$. Formally, the collapsed MDP $\bar{\mathcal{M}} = \langle \bar{\mathcal{S}}, \bar{\mathcal{A}}, \bar{R}, \bar{T} \rangle$ is defined by $\bar{\mathcal{S}} = \mathcal{S}/\mathcal{G}_{\mathcal{M}}, \bar{\mathcal{A}} = \mathcal{A}/\mathcal{G}_{\mathcal{M}}$, $\bar{R}(\langle s, a \rangle^{\mathcal{G}_{\mathcal{M}}}) = R(s,a)$ and $\bar{T}(s'^{\mathcal{G}_{\mathcal{M}}} \mid \langle s, a \rangle^{\mathcal{G}_{\mathcal{M}}}) = T(s' \mid s, a)$. This results in symmetry-based model minimization of symmetric MDPs.

4. Desiderata for Symmetry-Based Representation in RL

Separating Transition and Reward Symmetries One important choice is that between using the symmetry group of the MDP ($\mathcal{G}_{\mathcal{M}}$) versus the symmetry group of state-transitions ($\mathcal{G}_T$), where $\mathcal{G}_T$ is the group of transformations of state-action pairs that lead to equivariant deterministic transitions, as given by Equation 5. The former is a subgroup of the latter $\mathcal{G}_{\mathcal{M}} \leq \mathcal{G}_T$, i.e, the symmetries of a transition model contain the symmetries of the MDP. In fact it is easy to see that $\mathcal{G}_{\mathcal{M}} = \mathcal{G}_T \cap \mathcal{G}_R$, where $\mathcal{G}_R$ is the group of transformations of state-action pairs that preserve the one step-reward and only satisfy Equation 4. We observe that working with a larger symmetry group $\mathcal{G}_T$ has two benefits. First, it creates a stronger inductive bias for the model, because many real-world settings can involve a range of symmetries in transitions that are not present in the reward. For example, an agent's transition function in a 2D space may be equivariant to the Euclidean group, while the reward may not be invariant to the same group (*e.g.*, the reward for arriving at a particular location could break this symmetry). Second, the separate modeling of transition symmetries facilitates transfer to new tasks, where the reward is changing.

Invariance in Model-Free and Equivariance in Model-Based RL If the objective is to carry out model-free RL,

Equation (6) motivates the need to learn action-value functions, or the policies that are *invariant* to symmetries of the MDP ($\mathcal{G}_{\mathcal{M}}$). For a deterministic policy, the invariance of Equation (6) becomes an equivariance constraint: $g \cdot \pi(s) = \pi(g \cdot s)$. Since this essentially leads to model minimization, van der Pol et al. (2020b); Mondal et al. (2020) use this idea to improve sample efficiency when the group's actions in the agent's action space are known permutations. However, if our objective is to learn only a symmetry-based model of the environment (*i.e.*, transition and reward functions), Equation (5) suggests that we need to learn a $\mathcal{G}_T$-*equivariant* transition function.

Symmetries in a Latent Transition Model While it is possible to learn the state transition model in the observation space that is equivariant to the agent's action, for high-dimensional inputs this could be quite challenging since the model has to learn details of the environment that are irrelevant to the RL agent. Using state and action embeddings enables learning of the transition model in the latent space. Indeed the constraint on the model and the embedding is that of the MDP homomorphism (Section 3.2). Working in the latent space has an additional benefit when it comes to symmetries: *we can assume that the $\mathcal{G}$ action on the latent state-action pairs is linear through $\rho(\mathcal{G})$ despite having non-linear transformations in the observation space.*

From the fact that symmetries of states $\mathcal{G}_{\mathcal{S}} \leq \mathcal{G}$ is a subgroup of the state-action or transition symmetry, it follows that $\rho_g \in \rho(\mathcal{G})$ can be divided into two parts: 1) $\tau_g \in \tau(\mathcal{G}_{\mathcal{S}})$ the group representation acting on the state embedding, and; 2) $\kappa^s_g \in \kappa(\mathcal{G})$, the group representation for state-dependent action embedding.[3]

At this point we can combine the requirement for an MDP homomorphism in Equation (2), with that of the $\mathcal{G}$-equivariant transition model, Equation (5) of a symmetric MDP. The result is the following two constraints in our symmetric latent variable model (see Figure 1): $\forall s, a \in \mathcal{S} \times \mathcal{A}$ and $g \in \mathcal{G}$

$$\bar{T}(h_{\mathcal{S}}(s), h_{\mathcal{A}}(s,a)) = h_{\mathcal{S}}(T(s,a)) \tag{7}$$

$$\tau_g \bar{T}(h_{\mathcal{S}}(s), h_{\mathcal{A}}(s,a)) = \bar{T}\left(\tau_g h_{\mathcal{S}}(s), \kappa^s_g h_{\mathcal{A}}(s,a)\right) \tag{8}$$

Matrix Embedding of States and Actions We now consider a design choice which can significantly simplify the constraints discussed above, though strictly speaking it is not required. We propose to use group representations for our state, and state action embeddings $h_{\mathcal{S}} : \mathcal{S} \to \tau(\mathcal{G}_{\mathcal{S}})$ and $h_{\mathcal{A}} : \mathcal{S} \times \mathcal{A} \to \kappa(\mathcal{G})$ – that is we use matrices to represent both states and actions. This choice assumes that a $\mathcal{G}$ action on state and state-action pairs is *transitive*, so that each state, and state-action pair can be mapped to a group member. To emphasize this in our notation, we use $\kappa(s)$ instead of $h_{\mathcal{S}}(s)$ and similarly use $\tau(s,a)$ instead of $h_{\mathcal{A}}(s,a)$ for state, and state-dependent action embedding respectively. This choice of embedding has several benefits: First, the learned embeddings are automatically equivariant to symmetry transformations of the state, and state-actions:

$$\tau(g \cdot s) = \tau_g \tau(s) \quad \text{and} \quad \kappa(g \cdot \langle s, a \rangle) = \kappa^s_g \kappa(s,a), \quad \forall s, a \in \mathcal{S} \times \mathcal{A}, g \in \mathcal{G}. \tag{9}$$

This means that the symmetries of the state-action pairs are preserved and now take a linear form in the latent space. While the embeddings are automatically equivariant, they may be equivariant to irrelevant non-linear transformations of the input. The world modeling constraint (Equation (7)) ensures the relevance of the non-linear transformations that are captured by the group equivariant embeddings above. Moreover, this embedding enables matrix multiplication for the transition model

$$\bar{T}(\tau(s), \kappa(s,a)) = \kappa(s,a)\tau(s), \tag{10}$$

which simply transforms the state-embedding $\tau(s)$ through the linear group action of state-dependent action encoding $\kappa(s,a)$. Using this transition model, the action equivariance constraint of Equation (7), and $\mathcal{G}$-equivariance constraint of Equation (8), simplify to:

$$\tau(s') = \kappa(s,a)\tau(s) \tag{11}$$

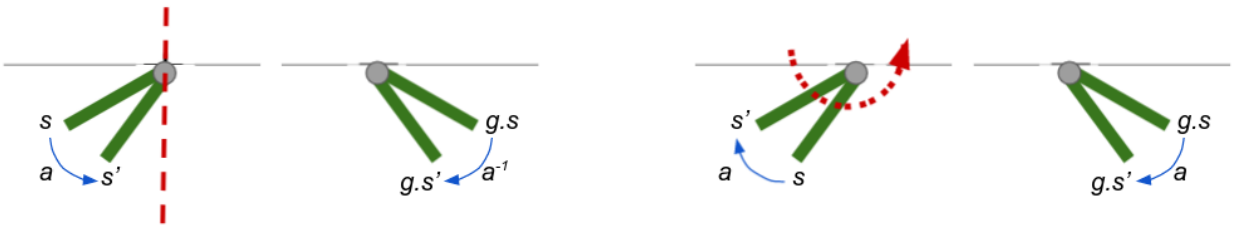
$$\tau_g \kappa(s,a)\tau(s) = \kappa^s_g \kappa(s,a)\tau_g \tau(s) \tag{12}$$

for any state transition triplet $\{s, a, s'\}$. In practice our model seeks to satisfy these two constraints via the direct minimization of appropriate loss functions, as will be discussed in Section 5.

Decomposition of the Latent Space The decomposition $\mathcal{G} = \mathcal{G}_1 \times \ldots \times \mathcal{G}_K$ into a direct product of subgroups can disentangle the factors of variation in the dataset (Higgins et al., 2018).[4] This gives us a way to represent the latent embedding space as a direct product of K factors, where each factor varies independently by actions of a subgroup of $\mathcal{G}$. Intuitively such a symmetry-based disentanglement provides an effective inductive bias particularly when there is modularity so that temporally coherent changes in the environment are due to the change of a (sparse) subset of factors. In our case this constraint takes the form of block-diagonal matrices for state and action embeddings. More precisely, we have a direct sum for state representation and

[3] This is because $\rho(\mathcal{G})$ can be seen as a representation that is *induced* by the representation $\tau(\mathcal{G}_{\mathcal{S}})$ of its subgroup: $\rho = \mathrm{Ind}^{\mathcal{G}}_{\mathcal{G}_{\mathcal{S}}} \tau$.

[4] As noted by Caselles-Dupré et al. (2019), simply having a product structure in the latent space does not guarantee disentanglement, and further constraints are required. In this work, we do not impose any additional constraints for disentanglement.



(a) (b)

Figure 2. An illustration of typical symmetries in a pendulum, and the corresponding transformations of the state and action for a group equivariant transition model: (a) shows how reflection of the agent's state results in a permutation of the action, denoted by a^{-1}. (b) shows how rotation of the agent's state results in invariance of the action in the absence of gravity. The state transitions can be modeled as group actions (2D rotations in this example), which can be captured by our symmetry transformation-based transition model. Note that rotational symmetry can hold even when gravity is present. In this case, symmetry transformations include rotations (and reflections) that preserve the Hamiltonian. Such non-linear energy-preserving transformations of state-actions in the pixel space can become linear in the embedding space.

the state-dependent action representation:

$$\tau(s) = \bigoplus_k \tau_k(s) \quad \text{and} \quad \kappa(s,a) = \bigoplus_k \kappa_k(s,a)$$

where $k \in \{1,\dots,K\}$ and $g = (g_1,\dots,g_K)$. Accordingly, the representation of the symmetry group $\mathcal{G}$ acting on the state embedding and the state-dependent action embedding is decomposed as $\tau_g = \oplus_k \tau_{g_k}$ and $\kappa^s_g = \oplus_k \kappa^s_{g_k}$. Combining this block structure with the Lie parameterization of Section 3.1 we obtain

$$\tau_\theta(s) = \bigoplus_k \exp\left(\sum_i \beta_{i,k,\theta}(s)\mathbf{E}^{(i)}\right) \quad \text{and}$$
$$\kappa_\phi(\tau_\theta(s),a) = \bigoplus_k \exp\left(\sum_i \alpha_{i,k,\phi}(\tau_\theta(s),a)\mathbf{E}^{(i)}\right) \quad (13)$$

where we use any standard neural network to implement the α_ϕ and β_θ functions that represent coefficients for the bases of the Lie algebra[5]. As we can backpropagate through this function, the network parameters θ, ϕ can be learned end to end. We refer the readers to Appendix A.2 for more detail. The choice of the subgroup depends on the symmetries of the RL environment, and this choice only affects the set of bases $\{\mathbf{E}^{(i)}\}_i$ in Equation (13). For example, in Atari games, the screen often has multiple objects undergoing 2-D translations and rotations, and one can use blocks of the 2-D Special Euclidean ($SE(2)$) group, that comprise translation and rotations of Euclidean space. For more realistic 3D environments, such as those of interest in robotics, self-driving cars and third-person games, one can use $SE(3)$, which is the group of 3-D translations and rotations. Also, in theory, we only need to specify a group that "contains" the group of interest as a subgroup. For example, if our state-actions only have 90° rotational symmetry, we may use a more general group for the representation (*e.g.*, $SE(2)$). The embedding function can define a homomorphism into the relevant subgroup.

[5]We denote both the neural networks which map to group representations and the network parameters by lowercase greek letters.

5. Loss Functions

We consider a standard RL setup where the agent interacts with its environments in episodes and we have access to $(\{s_t, a_t, r_t, s_{t+1}\})_{t=1,..,T}$ where s_t is the state, a_t is the action taken by the agent, r_t is the reward received and s_{t+1} is the observed next state at timestep t. Below we describe three loss functions that encode the equivariance/invariance constraints of Equations (1), (11) and (12).

Action Equivariant Transition Loss - Equation (11) Given triplets $\langle s_t, a_t, s_{t+1}\rangle$ from our dataset we simply apply a loss function ℓ such as a square loss[6] that penalizes the difference between two arguments:

$$L_{AET}(\theta,\phi) = \ell\left(\tau_\theta(s_{t+1}), \kappa_\phi(s_t,a_t)\tau_\theta(s_t)\right). \quad (14)$$

The choice of the embedding space and the latent transition function ensure that state embeddings are transformed by linear group action of the action embeddings. Minimization of L_{AET} encourages these symmetry transformations to capture state transitions resulting from the agent's action.

Group Equivariant Transition Loss - Equation (12) For this we need a $s_{t'}$ in addition to $\langle s_t, a_t\rangle$, where t' can be any state (at a different time step in the same or in a different episode.) We find the group transformation that maps s to

[6]In practice we use the normalized square loss $\ell(\mathbf{Y},\mathbf{Y}') = \left\| \frac{\mathbf{Y}}{\|\mathbf{Y}\|_2} - \frac{\mathbf{Y}'}{\|\mathbf{Y}'\|_2} \right\|_2^2$

s' in the latent space using $\tau_g = \tau_\theta(s_{t'})\tau_\theta(s_t)^{-1}$. Using this we can rewrite Equation (12) as

$$\underbrace{\tau_\theta(s_{t'})\tau_\theta(s_t)^{-1}}_{\tau_g}\kappa_\phi(s_t,a_t)\tau_\theta(s_t) = \kappa_g^s\kappa_\phi(s_t,a_t)\underbrace{\tau_\theta(s_{t'})}_{\tau_g\tau_\theta(s_t)}. \tag{15}$$

Since the state-dependent action encoding $\kappa_\phi(s_t,a_t)$ for the pair $\langle s_t,a_t\rangle$ is also produced by a neural network, the only missing part in the equation above is κ_g^s, the state-dependent action transformation. We use a neural network $\rho_\omega : \tau_g \mapsto \kappa_g^s$ to infer it from state transformation τ_g.

Example 1. To get an intuition for what this network is doing consider the example of a pendulum without gravity, with rotation and reflection symmetry $O(2)$ as shown in Figure 2, where inputs to the networks (s) are image sequences and the (ideal) embeddings $\tau_\theta(s), \kappa_\phi(s,a)$ are the angle plus angular velocity and torque respectively. If we rotate the pendulum using a rotation matrix τ_g, we expect the state-dependent action embedding to remain the same since the effect of torque remains similar after rotation. However, if we transform the pendulum by reflection around the vertical axis, we expect that the effect of torque will be negated. ρ_ω parameterizes this dependence.

A loss function ℓ could then measure the difference between the left and right hand side of the equation above

$$\begin{aligned}L_{GET}(\theta,\phi,\omega) =& \ell\big(\tau_\theta(s_{t'})\tau_\theta(s_t)^{-1}\kappa_\phi(s_t,a_t)\tau_\theta(s_t),\\ &\rho_\omega(\tau_\theta(s_{t'})\tau_\theta(s_t)^{-1})\kappa_\phi(s_t,a_t)\tau_\theta(s_{t'})\big).\end{aligned} \tag{16}$$

Action Invariant Reward Loss - Equation (1) While L_{AET} and L_{GET} enforce the equivariance of the latent transition model to an agent's action and the symmetry group, they do not encode information about the reward in the state representations. In order for the latent model to be homomorphic to the underlying MDP of the environment we match the reward at every state embedding using a reward predictor network $r_\psi : \tau_\theta(s) \mapsto \mathbb{R}$. We measure the difference between the predicted reward and the actual reward at time step $t+1$:

$$L_R(\psi,\theta,\phi) = \left(r_\psi(\kappa_\phi(\tau_\theta(s_t),a_t)\tau_\theta(s_t)) - r_{t+1}\right)^2. \tag{17}$$

Example 2 (Sliding Ball). Here, we visualize the matrix embedding produced using our loss functions on a simple toy example. We design a sliding ball environment where a ball moves on the screen vertically in a closed loop with two actions: up and down. We parameterize the latent representation to be a single block of $SO(2)$, that is a rotation matrix $\tau_\theta(s) = \begin{bmatrix}\cos(\beta) & \sin(\beta)\\ -\sin(\beta) & \cos(\beta)\end{bmatrix}$, where β is the output of the image encoder network (see Appendix A.2 for more detail on parameterization). Since in this example, the action encoding is not state-dependent, we obtain it by feeding the action to the action encoder network $\kappa(a) = \begin{bmatrix}\cos(\alpha) & \sin(\alpha)\\ -\sin(\alpha) & \cos(\alpha)\end{bmatrix}$ where β is the output of the network. As the action is also invariant to the group transformation $SO(2)$, Equation (16) becomes unnecessary, and we can only use Equation (14). To this end, we randomly sample actions (up and down) and generate trajectories to train the encoder using Equation (14). Figure 3 visualizes the learned embedding. For visualization, we transform a 2D unit vector ($[1,0]$ using the matrix embedding of the state. As the ball moves in a 1D loop in the environment, the encoder learns to map the transformation to the correct rotation matrix in $SO(2)$.

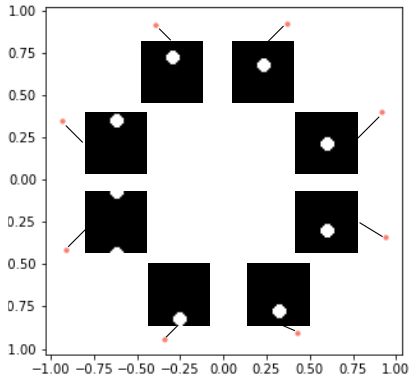


Figure 3. (a) Latent visualization of a sliding ball environment. The ball moves up or down in one dimension as dictated by the action. We show that our latent parameterization combined with L_{AET} learns the $SO(2)$ manifold of the ball's transition. To obtain the visualization, we start with a 2D unit vector ($[1,0]$ here) and transform it using the representation matrix obtained from the trained encoder by feeding the image observations. Images of eight uniformly separated positions of the ball are mapped to the red points, which denote the transformed unit vector.

6. Application to Model-free RL

While the framework discussed so far is ideal for model-based RL, here we confine our experiments to a model-free setting. Following the success of transition models for representation learning in model-free RL (Gelada et al., 2019; Schwarzer et al., 2021) we add the losses discussed above to the Temporal Difference (TD) error in Deep Q-learning. In practice, we need to make three modifications to our model/loss. These modifications are from the self-supervised representation learning literature (Grill et al., 2020) and were introduced in the RL setup in (Schwarzer et al., 2021). For ablation studies on these additional components we refer the reader to (Schwarzer et al., 2021).

Target Network A trivial solution to both equivariance enforcing losses of Section 5 is to encode all states and actions using an identity matrix. This problem in different contexts is known as the problem of collapse in representation learning. While using the reward signal helps in avoiding the collapse it is often not sufficient in sparse reward settings. Following Schwarzer et al. (2021), we use a target network to encode state s_{t+1} and $s_{t'}$ in Equation (16) in which the network parameters do not receive a gradient and are copied from the online network. We explicitly drop the subscripts to differentiate the target from the online network in this section (*e.g.*, $\tau_\theta \to \tau$).

Projection Head for Transition Losses The strict enforcement of symmetry constraints by our model can be overly restrictive when the environment has non-symmetric components, or when the our transition model is too simplistic. For this reason, following previous work, we enforce the losses on a learnable *projection* of the state embedding. That is, before application of the loss ℓ in Equations (14) and (16) we pass the embedding through a projection head.

M-step prediction Following the success of (Schwarzer et al., 2021) in long-term state embedding predictions, we predict state embeddings and rewards for M-steps.

6.1. Putting it All Together

Considering M consecutive state-actions $\{s_{t:t+M}, a_{t:t+M}\}$ and $\hat{x}_t = x_t = \tau_\theta(s_t)$, we predict the state embeddings and the rewards of the next M steps:

$$\hat{x}_{t+m} = \kappa_\phi(\hat{x}_{t+m-1}, a_{t+m-1})\hat{x}_{t+m-1} \quad \text{and}$$
$$\hat{r}_{t+k} = r_\psi(\hat{x}_{t+k}) \quad \forall m \in \{1, \dots, M\}$$

Here we are using $\hat{x}$ for M-step model prediction of the embedding to distinguish this from the latent embedding x, and the embedding produced by the target network $\bar{x} = \tau(s_{t+m})$. This also applies to the M-step predicted reward $\hat{r}$ and observed reward r. We then project these embeddings using a projection head p_ζ to produce $\hat{z}_{t+m} = p_\zeta(\hat{x}_{t+m})$ and $\bar{z}_{t+m} = p(\bar{x}_{t+m})$. Using this notation, our final expressions for L_{AET} and L_R are:

$$L_{AET} = \sum_{m=1}^{M} \left\| \frac{\hat{z}_{t+m}}{\|\hat{z}_{t+m}\|_2} - \frac{\bar{z}_{t+m}}{\|\bar{z}_{t+m}\|_2} \right\|_2^2 \quad \text{and}$$
$$L_R = \sum_{m=1}^{M} (\hat{r}_{t+m} - r_{t+m})^2 \tag{18}$$

For L_{GET} we need $\langle s_t, a_t, s_{t+1} \rangle$ and another state s'. From their embedding using the notation above we obtain $\tau_g = \bar{x}_{t'} x_t^{-1}$, the linear transformation between them, and $\kappa_g^s = \rho_\omega(\tau_g)$, the state-dependent action transformation. Now for $\bar{x}_{t'}$, we obtain the predicted next state from $\bar{x}_{t'}$ as $\bar{x}_{t'+1} = \kappa_g^s \kappa(x_t, a_t)\bar{x}_{t'} = \rho_\omega(x_{t'} x_t^{-1})\kappa(x_t, a_t)\bar{x}_{t'}$ and from $\hat{x}_{t+1}$ as $\hat{x}_{t'+1} = \tau_g \hat{x}_{t+1}$. Before penalizing the difference between these embeddings, we project them to $\hat{y}_{t'+1} = b_\eta(\hat{x}_{t'+1})$ and $\bar{y}_{t'+1} = b(\bar{x}_{t'+1})$ using projection head b_η, to get the final expression for L_{GET}:

$$L_{GET} = \left\| \frac{\hat{y}_{t'+1}}{\|\hat{y}_{t'+1}\|_2} - \frac{\bar{y}_{t'+1}}{\|\bar{y}_{t'+1}\|_2} \right\|_2^2 \tag{19}$$

Q-learning We pass the representation x_t to a Q-learning head q_ξ to learn policies based on the output of the Q-value estimator. The Q-value estimator is learnt by minimizing:

$$L_{DQN}(\xi, \theta) = \Big(q_\xi(\tau_\theta(s_t), a_t) - (r_t + \gamma \max_a q_\xi(\tau(s_{t+1}), a)) \Big)^2 \tag{20}$$

We use the data efficient adaptation of Rainbow (van Hasselt et al., 2019; Hessel et al., 2018) which combines many improvements over the original DQN(Mnih et al., 2013) such as Distributional RL(Dabney et al., 2018), Dueling DQN (Wang et al., 2016), and Double DQN (Van Hasselt et al., 2016). The total loss optimized by our model is:

$$L = L_{DQN} + \lambda_1 L_R + \lambda_2 L_{GET} + \lambda_3 L_{AET} \tag{21}$$

where λ_1, λ_2 and λ_3 are hyper-parameters. We use $\lambda_1 = \lambda_2 = \lambda_3 = 1$ in all experiments. Motivated by the performance improvements due to augmentation reported in recent literature (Yarats et al., 2021; Schwarzer et al., 2021), we also augment our states by shifting and changing the pixel intensity before encoding them. Figure 8 in Appendix B.3 shows a detailed schematic of our model. We provide the algorithm for our model in Algorithm 1 and details of network architectures in Appendix B.4.

7. Experiments

We test our method on a suite of 2D Atari games, which is a popular benchmark used in RL. The full Atari suite consists of 57 games with typically 50 million environment steps. We use the sample-efficient Atari suite introduced by Kaiser et al. (2019), which consists of 26 games with only 100,000 environment steps of training data available. In our experiments, we use three types of simple connected Lie subgroup blocks including General Linear $GL(2)$, Special Euclidean $SE(2)$, and Translation $T(2)$. Additional details are in Appendix A. Unless stated otherwise, our EqR model uses $SE(2)$ subgroup blocks, with $K = 12$ blocks and $M = 5$ steps during training. L_{AET} is always used to train EqR. L_{GET}, which makes transition model equivariant with respect to the symmetry transformation of state-actions, and L_R are optional. We build our implementation on top of SPR's (Schwarzer et al., 2021), which is based on `rlpyt`

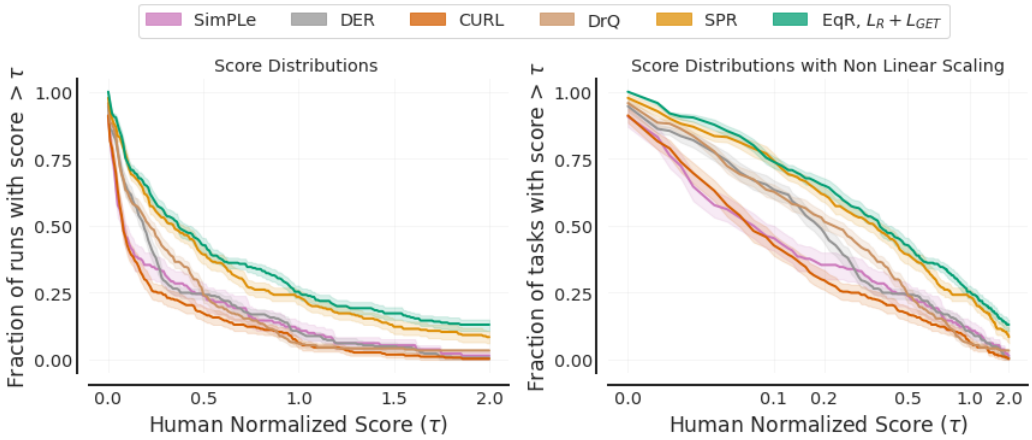


Figure 4. Performance profiles for different methods based on score distributions (a), and average score distributions (b). Shaded regions show pointwise 95% confidence bands. The higher the curve, the better the method is.

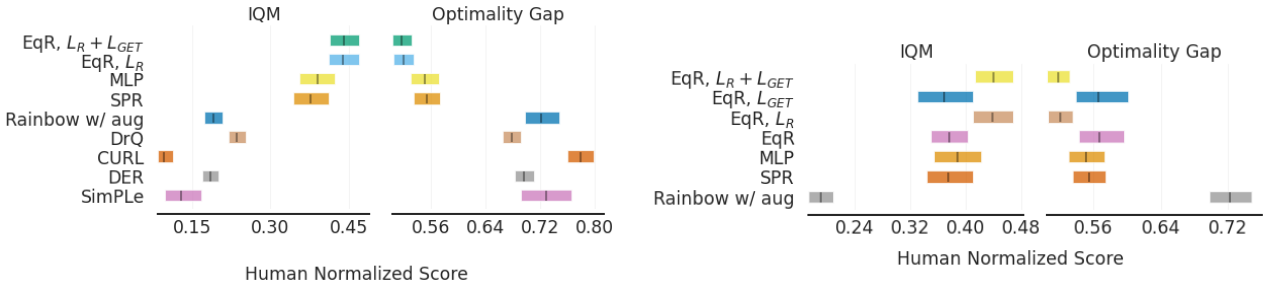


Figure 5. Plots of Interquartile Mean (IQM) and Optimality Gap (Agarwal et al., 2021) computed from human-normalized scores, showing the point estimates along with 95% confidence intervals (over 10 runs for all methods, 5 runs for SimPLe). A higher IQM and a lower optimality gap reflects better performance. (a) shows different methods for all 26 games. (b) shows our proposed method with different loss compomnents for all 26 games.

(Stooke & Abbeel, 2019) and PyTorch (Paszke et al., 2019). We use the same underlying RL algorithm and hyperparameters used by SPR for a fair comparision. Our implementation is available at https://github.com/arnab39/Symmetry-RL.

Evaluation Metrics We compute the average episodic return (the 'game score') at the end of training and normalize it with respect to human scores, as is standard practice. The human-normalized score (HNS) is given by $\frac{\text{agent score - random score}}{\text{human score - random score}}$. Since there is considerable variance across different runs, the mean and the median are not very reliable metrics. Instead, Agarwal et al. (2021) propose using bootstrapped confidence intervals (CI) with stratified sampling which is more suitable for small sample sizes (10 runs per game in our case). We report the Interquartile Mean (IQM), which is the mean across the middle 50% of the runs, as well as the Optimality Gap, which is the amount by which the algorithm fails to meet a minimum HNS of 1.0. We also provide performance profiles showing the fraction of runs above a certain normalized score, which gives a more complete picture of the performance.

Results We use 10 seeds for every game, for every variation of our model. Figure 4 shows performance profiles for our model, EqR with $L_R + L_{GET}$, along with other comparable methods. If one curve is strictly above another, the better method is said to "stochastically dominate" the other (Agarwal et al., 2021). The curves for both variations of the proposed method are almost always above the next best method, SPR (Schwarzer et al., 2021). Figure 5(a) provides results for different methods on all 26 games. The two best variations of the proposed method outperform previous methods, and the difference is statistically significant considering the CI. Table 2 in Appendix B.1 shows the full results on all games: *our best model achieves super-human*

performance on eight games and achieves a higher score than any other previous method on 13 out of the 26 games.

In order to better understand the effect of various modeling choices, loss functions and implementation details on the performance, we now consider different variations of EqR, with the same augmentation as the baseline for ablation studies.

Choice of Group To understand the role of the choice of a group in the embedding space, we use our EqR model with L_R. This variation of EqR is similar to DeepMDP (Gelada et al., 2019), except for the group structured latent embedding space and group action-based state transition. In order to investigate the effect of the above two group-related constructs, we remove them and use an action encoder to predict the next states directly, referring to this as MLP which makes it like as DeepMDP but with normalized mean square error loss for the model learning part.

Loss functions Figure 5(b) compares the performance of EqR with different loss components. Using EqR with the default L_{AET} results in a considerable improvement over Rainbow with augmentation (note that this is still using symmetry-based representation and transition with $SE(2)$ subgroup blocks.) Adding L_{GET} improves the performance slightly, while adding only L_R improves the performance even further. We hypothesize that the reward loss plays a role in both preventing representation collapse and preserving more information about the reward distribution in the latent state embeddings. Adding both L_{GET} and L_R improves the performance only slightly. The reason why the contribution of L_{GET} is not significant is likely that this prior of a equivariant transition model with respect to symmetry transformations of state-actions is too restrictive for some games, while being beneficial for others. Notably, in 17 out of a total of 26 games, including this loss term leads to a statistically significant boost in performance. These 17 games are: 'Alien', 'BankHeist', 'BattleZone', 'Boxing', 'ChopperCommand', 'CrazyClimber', 'DemonAttack', 'Freeway', 'Hero', 'Jamesbond', 'MsPacman', 'Pong', 'PrivateEye', 'Qbert', 'RoadRunner', 'Seaquest', 'UpNDown'. The full list of game-wise scores for the ablation studies are presented in Tables 3 and 4 in Appendix B.1. Further, to show that our model is learning the symmetry in the latent space we provide the combined loss plots L_{AET} and L_{GET} in Appendix B.2.

8. Conclusion

In this paper, we introduced a latent variable model for representation learning in RL, considering both equivariance to an agent's action and symmetry transformations in the environment. The proposed model has the capacity to become equivariant to non-linear symmetry transformations of state-actions.

We have considered three major symmetry-related constructs within a single coherent framework. First, we use the group equivariant state and action embedding, which we achieve through Lie parameterization. We believe our Lie parameterization will have applications beyond RL, for learning symmetric representations. The world modeling constraints further ensure that the transformations captured by the equivariant embedding are relevant. Second, the equivariance to agent's action, which when combined with group equivariant embeddings, ensures that the state transitions are captured by symmetry transformations in the latent space. Our empirical results suggest the importance of these two components on improving performance in Atari games with limited data. However, symmetry in RL can also appear in the form of a symmetric MDP. By this we mean that not only are the state and action embeddings equivariant, and that the transition model uses group transformation, but also that the latent transition model itself is equivariant under symmetry transformations of state-action pairs. This is a stricter constraint, and we found it marginally helpful in some settings of Atari games.

In future work we aim to explore the application of our approach in model-based RL, its ability to generalize across tasks, and its evaluation in more symmetric environments, such as those in Mujoco. We also plan to further investigate theoretically grounded methods for combining both symmetric and asymmetric aspects of the environment in our model.

Acknowledgments

We thank the reviewers for their valuable feedback. We are grateful to Tara Akhound-Sadegh for discussions on Lie parameterization and for sharing the sliding ball environment's implementation. This research was supported by funds from NSERC and by a CIFAR AI chair. Computational resources were provided by Mila and Compute Canada.

References

Agarwal, R., Schwarzer, M., Castro, P. S., Courville, A., and Bellemare, M. G. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, 34, 2021.

Bellemare, M. G., Candido, S., Castro, P. S., Gong, J., Machado, M. C., Moitra, S., Ponda, S. S., and Wang, Z. Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature*, 588(7836):77–82, 2020.

Bengio, Y., Courville, A., and Vincent, P. Representation learning: A review and new perspectives. *IEEE transac-*

tions on pattern analysis and machine intelligence, 35(8): 1798–1828, 2013.

Bloem-Reddy, B. and Teh, Y. W. Probabilistic symmetries and invariant neural networks. *J. Mach. Learn. Res.*, 21: 90–1, 2020.

Brown, N. and Sandholm, T. Superhuman ai for multiplayer poker. *Science*, 365(6456):885–890, 2019.

Caselles-Dupré, H., Garcia Ortiz, M., and Filliat, D. Symmetry-based disentangled representation learning requires interaction with environments. *Advances in Neural Information Processing Systems*, 32:4606–4615, 2019.

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pp. 1597–1607. PMLR, 2020.

Dabney, W., Rowland, M., Bellemare, M. G., and Munos, R. Distributional reinforcement learning with quantile regression. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

François-Lavet, V., Henderson, P., Islam, R., Bellemare, M. G., and Pineau, J. An introduction to deep reinforcement learning. *arXiv preprint arXiv:1811.12560*, 2018.

Gelada, C., Kumar, S., Buckman, J., Nachum, O., and Bellemare, M. G. Deepmdp: Learning continuous latent space models for representation learning. In *International Conference on Machine Learning*, pp. 2170–2179. PMLR, 2019.

Grill, J.-B., Strub, F., Altché, F., Tallec, C., Richemond, P., Buchatskaya, E., Doersch, C., Pires, B., Guo, Z., Azar, M., et al. Bootstrap your own latent: A new approach to self-supervised learning. In *Neural Information Processing Systems*, 2020.

Hafner, D., Lillicrap, T., Ba, J., and Norouzi, M. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019a.

Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pp. 2555–2565. PMLR, 2019b.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

Higgins, I., Pal, A., Rusu, A., Matthey, L., Burgess, C., Pritzel, A., Botvinick, M., Blundell, C., and Lerchner, A. Darla: Improving zero-shot transfer in reinforcement learning. In *International Conference on Machine Learning*, pp. 1480–1490. PMLR, 2017.

Higgins, I., Amos, D., Pfau, D., Racanière, S., Matthey, L., Rezende, D. J., and Lerchner, A. Towards a definition of disentangled representations. *ArXiv*, abs/1812.02230, 2018.

Kaiser, Ł., Babaeizadeh, M., Miłos, P., Osiński, B., Campbell, R. H., Czechowski, K., Erhan, D., Finn, C., Kozakowski, P., Levine, S., et al. Model based reinforcement learning for atari. In *International Conference on Learning Representations*, 2019.

Kielak, K. P. Do recent advancements in model-based deep reinforcement learning really improve data efficiency? 2019.

Laskin, M., Lee, K., Stooke, A., Pinto, L., Abbeel, P., and Srinivas, A. Reinforcement learning with augmented data. *arXiv preprint arXiv:2004.14990*, 2020a.

Laskin, M., Srinivas, A., and Abbeel, P. Curl: Contrastive unsupervised representations for reinforcement learning. In *International Conference on Machine Learning*, pp. 5639–5650. PMLR, 2020b.

Levine, S., Finn, C., Darrell, T., and Abbeel, P. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

Li, L., Walsh, T. J., and Littman, M. L. Towards a unified theory of state abstraction for mdps. *ISAIM*, 4:5, 2006.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *nature*, 518(7540): 529–533, 2015.

Mondal, A. K., Nair, P., and Siddiqi, K. Group equivariant deep reinforcement learning. *arXiv preprint arXiv:2007.03437*, 2020.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037, 2019.

Quessard, R., Barrett, T. D., and Clements, W. R. Learning group structure and disentangled representations of dynamical environments. *arXiv preprint arXiv:2002.06991*, 2020.

Ravindran, B. and Barto, A. G. Symmetries and model minimization in markov decision processes. Technical report, USA, 2001.

Ravindran, B. and Barto, A. G. Model minimization in hierarchical reinforcement learning. In *International Symposium on Abstraction, Reformulation, and Approximation*, pp. 196–211. Springer, 2002.

Ravindran, B. and Barto, A. G. Smdp homomorphisms: an algebraic approach to abstraction in semi-markov decision processes. In *Proceedings of the 18th international joint conference on Artificial intelligence*, pp. 1011–1016, 2003.

Ravindran, B. and Barto, A. G. An algebraic approach to abstraction in reinforcement learning. 2004.

Schwarzer, M., Anand, A., Goel, R., Hjelm, R. D., Courville, A., and Bachman, P. Data-efficient reinforcement learning with self-predictive representations. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=uCQfPZwRaUu.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Stooke, A. and Abbeel, P. rlpyt: A research code base for deep reinforcement learning in pytorch. *arXiv preprint arXiv:1909.01500*, 2019.

Taylor, J. Lax probabilistic bisimulation. 2008.

Thomas, V., Pondard, J., Bengio, E., Sarfati, M., Beaudoin, P., Meurs, M.-J., Pineau, J., Precup, D., and Bengio, Y. Independently controllable features. *arXiv preprint arXiv:1708.01289*, 2017.

van der Pol, E., Kipf, T., Oliehoek, F. A., and Welling, M. Plannable approximations to mdp homomorphisms: Equivariance under actions. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 1431–1439, 2020a.

van der Pol, E., Worrall, D., van Hoof, H., Oliehoek, F., and Welling, M. Mdp homomorphic networks: Group symmetries in reinforcement learning. *Advances in Neural Information Processing Systems*, 33, 2020b.

Van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

van Hasselt, H. P., Hessel, M., and Aslanides, J. When to use parametric models in reinforcement learning? *Advances in Neural Information Processing Systems*, 32:14322–14333, 2019.

Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., and Freitas, N. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pp. 1995–2003. PMLR, 2016.

Yarats, D., Kostrikov, I., and Fergus, R. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=GY6-6sTvGaf.

Zbontar, J., Jing, L., Misra, I., LeCun, Y., and Deny, S. Barlow twins: Self-supervised learning via redundancy reduction. *arXiv preprint arXiv:2103.03230*, 2021.

Zinkevich, M. and Balch, T. Symmetry in markov decision processes and its implications for single agent and multi agent learning. In *In Proceedings of the 18th International Conference on Machine Learning*. Citeseer, 2001.

A. Subgroup blocks and their parameterization

A.1. Choice of group

Atari games differ in their style of play, their objectives, the symmetry transformations of both the agent and other objects on the screen and associated symmetry transformation of the agent's action. But most of these games include symmetry transformations. For example, the screen often has multiple objects undergoing two dimensional translations and rotations. In this case one can use blocks of the 2D Special Euclidean Group $SE(2)$. Each such block can capture the transformation of a particular object in the screen, including the agent. One can also use more restrictive subgroup blocks like $T(2)$, which capture only 2D translations. For more realistic 3D environments, such as those of interest in robotics, self-driving cars and third person games, one can use $SE(3)$, which is the group of 3D translations and rotations. This should capture both the transformations of the objects in the environment and changes in viewpoint due to the agent's actions.

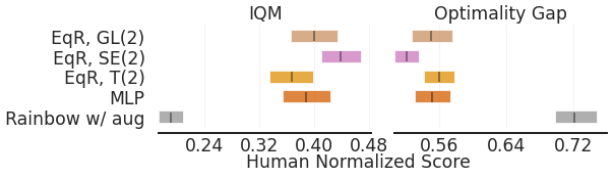


Figure 6. Plots of Interquartile Mean (IQM) and Optimality Gap computed from human-normalized scores, showing the point estimates along with 95% confidence intervals (over 10 runs for all methods). A higher IQM and a lower optimality gap reflects better performance. The plot shows the proposed model with different group choices for all 26 games.

We further test models with other subgroups including $T(2)$ and $GL(2)$. Figure A.1 shows that adding a symmetry-based inductive bias in the model by making the embeddings group representations and modeling the transitions as group actions is indeed helpful. The success of the model which uses $SE(2)$ blocks might be attributed to the fact that translations and rotations are the most common types of symmetry transformations present in Atari games. However, the more restrictive $T(2)$ slightly hurts the performance, while the more general $GL(2)$ performs similarly to the MLP model.

A.2. Implementing parameterization

We consider three types of subgroups: GL(n) - the set of all invertible linear transformations, SE(n) - the set of all rotations and translations and T(n) - the set of all translations. We provide a general method to parameterize each of these, based on the type of group.

GL(n) As the matrix representation of GL(n) is the set of invertible matrices which has a measure of 1 it is easy to parameterize it. We just generate n^2 parameters using a network corresponding to each element of the matrix. This gives an element from GL(n).

T(n) As T(n) just denotes translation in a n-dimensional space with group action being addition, implementing it is straightforward. We generate n parameters using a neural network and instead of using matrix multiplication use addition for the group action. Note that we can also use a matrix representation for T(n) but it is unnecessary and inefficient.

SE(n) Unlike GL(n) and T(n), parameterizing SE(n) is a bit tricky because it involves parameterizing SO(n). We use a homogeneous co-ordinate based representation of SE(n) = $\left\{\begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix}, R \in SO(n) \text{ and } t \in T(n)\right\}$. So we need n parameters for the t and another $D = \frac{n(n-1)}{2}$ parameters for $SO(n)$ from the neural network. As explained in Section 4, we can use a Lie parameterization to get the elements of $SO(n)$ by $R = \exp(\sum_{d=1}^{D} \beta_d \mathbf{E}^{(d)})$ where $\mathbf{E}^{(d)}$ denote D bases of the space of skew symmetric matrices and the β_is are the parameters of the neural network. For example, in the case of $SO(2)$ we can use the basis $\mathbf{E}^{(1)} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$. Similarly, we can extend this to $SO(n)$ by using a basis given by D $n \times n$ matrices $\mathbf{E}^{(ij)}$ $\forall\{1 \le i < j \le n\}$ whose only non-zero elements are $\mathbf{E}^{(ij)}_{i,j} = -1$ and $\mathbf{E}^{(ij)}_{j,i} = 1$.

Although Lie parameterization gives us a general recipe to output a representation of simple connected Lie groups like

$SO(n)$, in our implementation we use Euler parameterization because it runs faster in Pytorch. We provide the code for both. Following (Quessard et al., 2020), we parameterize each rotation matrix in $SO(n)$ using the product of rotations on D orthogonal planes in $\mathbb{R}^n$: $R = \prod_{i=1}^{n} \prod_{1 \leq i < j \leq n} R^{ij}$. Here $R^{ij} \in \mathbb{R}^{n \times n}$ is the rotation matrix in the $i-j$ plane, and its non-zero elements besides the diagonal are the four values on the i, j rows and columns, which comprise the 2D rotation matrix that is $R^{ij}_{i,j} = \begin{bmatrix} \cos(\theta_{i,j}) & \sin(\theta_{i,j}) \\ -\sin(\theta_{i,j}) & \cos(\theta_{i,j}) \end{bmatrix}$. We have D parameters $\theta_{i,j}$ which we can obtain from a neural network.

The parameters in all the parameterization techniques mentioned here can be back-propagated. We summarize the number of parameters required from a neural network output, representation type and the associated group actions of different subgroups in Table 1.

Table 1. Group Properties

Subgroup block type	#Parameters	Representation type	Group action
General Linear - $GL(n)$	n^2	Matrix$_{n \times n}$	Matrix multiplication
Special Euclidean - $SE(n)$	$\frac{n(n-1)}{2} + n$	Matrix$_{(n+1) \times (n+1)}$	Matrix multiplication
Special Orthogonal - $SO(n)$	$\frac{n(n-1)}{2}$	Matrix$_{n \times n}$	Matrix multiplication
Translation - $T(n)$	n	Vector$_n$	Addition

B. Atari Details

B.1. Full Results

We provide individual results on the 26 Atari games after 100K training steps. Our results are averaged over 10 seeds, and the network architectures and full list of hyperparameters used to produce them are provided in Appendix B.4 and Appendix B.6.

- Table 2 compares our two best performing EqR models using $SE(2)$ subgroup blocks with other methods.
- Table 3 compares different choices of subgroup blocks with the reward loss, L_R, included for all EqR models (also see Figure 6 (a)).
- Table 4 compares EqR using $SE(2)$ subgroup blocks with different loss terms included in the training objective (see Section 5 and Figure 6 (b)). The action equivariance transition loss, L_{AET} is always included for EqR models.

Table 2. Mean game scores on the 26 Atari games after 100K environment steps. The EqR models use $SE(2)$ subgroup blocks along with an action equivariant transition loss, L_{AET}, and are averaged over 10 seeds.

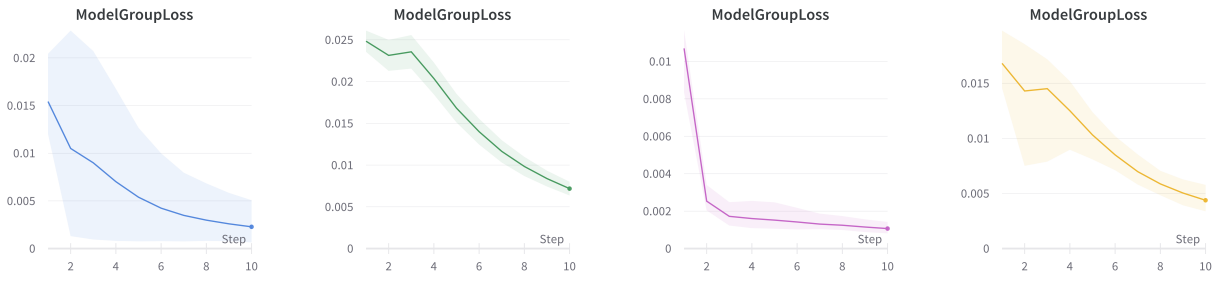
Game	Random	Human	SimPLe	DER	CURL	DrQ	SPR	EqR, L_R	EqR, $L_R + L_{GET}$
Alien	227.8	7127.7	616.9	739.9	558.2	771.2	801.5	774.0	**872.9**
Amidar	5.8	1719.5	88.0	**188.6**	142.1	102.8	176.3	140.9	138.4
Assault	222.4	742.0	527.2	431.2	600.6	452.4	571.0	**753.8**	734.3
Asterix	210.0	8503.3	**1128.3**	470.8	734.5	603.5	977.8	923.2	902.5
Bank Heist	14.2	753.1	34.2	51.0	131.6	168.9	380.9	395.1	**397.4**
BattleZone	2360.0	37187.5	5184.4	10124.6	14870.0	12954.0	**16651.0**	13044.0	13255.0
Boxing	0.1	12.1	9.1	0.2	1.2	6.0	35.8	37.5	**39.2**
Breakout	1.7	30.5	16.4	1.9	4.9	16.1	17.1	**17.2**	16.0
ChopperCommand	811.0	7387.8	**1246.9**	861.8	1058.5	780.3	974.8	1073.5	1142.2
Crazy Climber	10780.5	35829.4	**62583.6**	16185.3	12146.5	20516.5	42923.6	49399.0	52008.1
Demon Attack	152.1	1971.0	208.1	508.0	817.6	**1113.4**	545.2	531.4	532.1
Freeway	0.0	29.6	20.3	**27.9**	26.7	9.8	24.4	24.1	25.2
Frostbite	65.2	4334.7	254.7	866.8	1181.3	331.1	1821.5	**1855.6**	1699.4
Gopher	257.6	2412.5	771.0	349.5	669.3	636.3	715.2	**1010.0**	912.1
Hero	1027.0	30826.4	2656.6	6857.0	6279.3	3736.3	**7019.2**	5775.2	6118.5
Jamesbond	29.0	302.8	125.3	301.6	**471.0**	236.0	365.4	312.8	319.7
Kangaroo	52.0	3035.0	323.1	779.3	872.5	940.6	3276.4	**3569.3**	3296.0
Krull	1598.0	2665.5	4539.9	2851.5	4229.6	4018.1	3688.9	**5614.5**	5467.7
Kung Fu Master	258.5	22736.3	17257.2	14346.1	14307.8	9111.0	13192.7	**18511.0**	17510.9
Ms Pacman	307.3	6951.6	1480.0	1204.1	1465.5	960.5	1313.2	1317.1	**1663.5**
Pong	-20.7	14.6	**12.8**	-19.3	-16.5	-8.5	-5.9	-6.0	-6.1
Private Eye	24.9	69571.3	58.3	97.8	**218.4**	-13.6	124.0	76.6	88.9
Qbert	163.9	13455.0	**1288.8**	1152.9	1042.4	854.4	669.1	773.8	814.9
Road Runner	11.5	7845.0	5640.6	9600.0	5661.0	8895.1	**14220.5**	13385.0	13708.8
Seaquest	68.4	42054.7	683.3	354.1	384.5	301.2	583.1	650.3	**697.9**
Up N Down	533.4	11693.2	3350.3	2877.4	2955.2	3180.8	28138.5	44295.4	**52118.4**
Mean Human-Norm'd	0.000	1.000	0.443	0.285	0.381	0.357	0.704	0.859	**0.886**
Median Human-Norm'd	0.000	1.000	0.144	0.161	0.175	0.268	0.415	**0.418**	0.398
# Superhuman games	0	N/A	2	2	2	2	7	**8**	7

B.2. Loss Plots

We provide the combined loss plots for symmetry-based losses (L_{AET} and L_{GET}) for 4 of the 26 games to verify that the model is indeed minimizing the losses and learning symmetry in the latent. The loss plots of the rest of the games follow the same pattern.

Table 3. Mean game scores on the 26 Atari games after 100K environment steps for different choices of subgroup blocks, averaged over 10 seeds. The reward loss, L_R, is included in addition to the default loss L_{AET}.

Game	Random	Human	MLP	EqR, T_2	EqR, SE_2	EqR, GL_2
Alien	227.8	7127.7	780.1	846.4	774.0	**881.3**
Amidar	5.8	1719.5	**143.3**	139.7	140.9	132.2
Assault	222.4	742.0	701.5	684.0	**753.8**	692.3
Asterix	210.0	8503.3	973.6	**1004.4**	923.2	889.5
Bank Heist	14.2	753.1	402.1	353.5	395.1	**430.2**
BattleZone	2360.0	37187.5	12722.6	11500.0	13044.0	**13114.0**
Boxing	0.1	12.1	**38.0**	28.9	37.5	33.4
Breakout	1.7	30.5	16.2	14.8	**17.2**	15.4
ChopperCommand	811.0	7387.8	989.5	1028.9	1073.5	**1088.1**
Crazy Climber	10780.5	35829.4	43705.8	50822.1	49399.0	**55018.5**
Demon Attack	152.1	1971.0	518.6	**544.2**	531.4	510.2
Freeway	0.0	29.6	20.3	18.5	**24.1**	21.4
Frostbite	65.2	4334.7	1702.4	1653.8	**1855.6**	1797.7
Gopher	257.6	2412.5	720.2	**1012.5**	1010.0	894.4
Hero	1027.0	30826.4	**6840.0**	5779.8	5775.2	5934.7
Jamesbond	29.0	302.8	**337.4**	313.25	312.8	334.8
Kangaroo	52.0	3035.0	2994.8	2942.5	**3569.3**	3186.4
Krull	1598.0	2665.5	3801.5	5293.0	5614.5	**5772.6**
Kung Fu Master	258.5	22736.3	13780.4	14924.2	**18511.0**	16002.8
Ms Pacman	307.3	6951.6	1220.8	1166.8	**1317.1**	1147.7
Pong	-20.7	14.6	-6.1	-11.5	**-6.0**	-8.2
Private Eye	24.9	69571.3	72.4	65.1	**76.6**	55.9
Qbert	163.9	13455.0	678.4	763.2	**773.8**	635.2
Road Runner	11.5	7845.0	12765.2	**13654.2**	13385.0	12560.4
Seaquest	68.4	42054.7	**656.9**	647.3	650.3	633.3
Up N Down	533.4	11693.2	23130.6	**58164.4**	44295.4	43767.2
Mean Human-Norm'd	0.000	1.000	0.681	0.829	**0.859**	0.833
Median Human-Norm'd	0.000	1.000	0.398	0.361	**0.418**	0.380
# Superhuman games	0	N/A	6	6	**8**	7



(a) (b) (c) (d)

Figure 7. Loss L_{AET} + L_{GET} vs. training steps (in 10,000s) of (a) Boxing (b) Krull. (c) Kung Fu Master (d) Up N Down

Table 4. Mean game scores on the 26 Atari games after 100K environment steps for EqR using $SE(2)$ subgroup blocks with different loss terms included in the training objective. The action equivariance transition loss, L_{AET}, is included for all EqR models and the scores are averaged over 10 seeds.

Game	Random	Human	EqR	EqR, L_R	EqR, L_{GET}	EqR, $L_R + L_{GET}$
Alien	227.8	7127.7	856.5	774.0	862.5	**872.9**
Amidar	5.8	1719.5	134.7	**140.9**	135.0	138.4
Assault	222.4	742.0	643.1	**753.8**	701.3	734.3
Asterix	210.0	8503.3	824.8	**923.2**	864.9	902.5
Bank Heist	14.2	753.1	**407.3**	395.1	335.9	397.4
BattleZone	2360.0	37187.5	12805.6	13044.0	12990.4	**13255.0**
Boxing	0.1	12.1	32.7	37.5	34.8	**39.2**
Breakout	1.7	30.5	14.6	**17.2**	14.8	16.0
ChopperCommand	811.0	7387.8	1015.6	1073.5	934.8	**1142.2**
Crazy Climber	10780.5	35829.4	38483.9	49399.0	43085.6	**52008.1**
Demon Attack	152.1	1971.0	523.8	531.4	504.6	**532.1**
Freeway	0.0	29.6	22.1	24.1	22.5	**25.2**
Frostbite	65.2	4334.7	1635.2	**1855.6**	1563.9	1699.4
Gopher	257.6	2412.5	695.3	**1010.0**	789.3	912.1
Hero	1027.0	30826.4	5763.9	5775.2	5603.8	**6118.5**
Jamesbond	29.0	302.8	**388.4**	312.8	344.9	319.7
Kangaroo	52.0	3035.0	2667.9	**3569.3**	2848.7	3296.0
Krull	1598.0	2665.5	4209.2	**5614.5**	4411.2	5467.7
Kung Fu Master	258.5	22736.3	12287.9	**18511.0**	16394.6	17510.9
Ms Pacman	307.3	6951.6	1141.3	1317.1	1514.7	**1663.5**
Pong	-20.7	14.6	-9.9	**-6.0**	-6.5	-6.1
Private Eye	24.9	69571.3	73.2	76.6	87.5	**88.9**
Qbert	163.9	13455.0	696.7	773.8	736.8	**814.9**
Road Runner	11.5	7845.0	12659.2	13385.0	13110.4	**13708.8**
Seaquest	68.4	42054.7	593.6	650.3	641.0	**697.9**
Up N Down	533.4	11693.2	29425.4	44295.4	39076.6	**52118.4**
Mean Human-Norm'd	0.000	1.000	0.682	0.859	0.749	**0.886**
Median Human-Norm'd	0.000	1.000	0.337	**0.418**	0.377	0.398
# Superhuman games	0	N/A	6	**8**	6	7

B.3. Model schematic

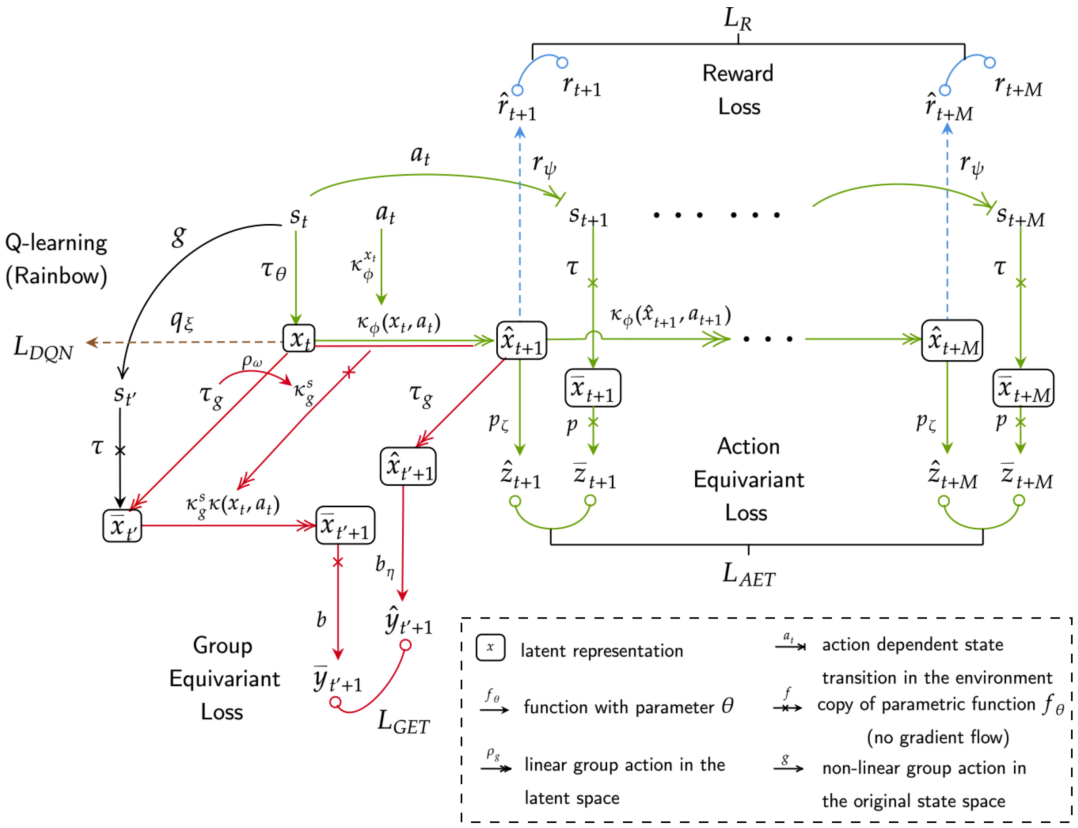


Figure 8. A schematic of the EqR model, applied to model-free RL. Green in the framework corresponds to learning equivariance under the agent's action and red corresponds to learning equivariance of the transition model with respect to symmetry transformation of the state-action. This color scheme is consistent with Figure 2. The part of the framework that corresponds to reward matching and Q-learning is shown in blue and brown respectively. The arrows in the schematic are differentiated by their heads and are described in the legend.

B.4. Network Architecture

We follow the baseline RL implementation of DrQ (Yarats et al., 2021) and SPR (Schwarzer et al., 2021) by using the 3-layer convolutional encoder from (Mnih et al., 2015) and then use a linear layer to get the parameters for the Group Parameterization. The output size of this layer varies depending on the group type, the number of blocks used and the size of the group. This defines our τ_θ. Note that the output of our encoder is a matrix for $GL(n)$ and $SE(n)$. We flatten it before we feed to other neural network like the Q-head $q_\xi(\cdot)$.

For the action encoder $\kappa(\cdot)$ we use a simple 1 layer MLP with batchnorm, ReLU and a hidden size of 256. We concatenate the one-hot encodings of the actions with the state representations coming from τ_θ and pass it through the action encoder to get matrix representation of the group after parameterization.

For the reward predictor network r_ψ we use a 2-layered MLP with batchnorm, ReLU and a hidden size of 256.

For the Q-head $q_\xi(\cdot)$ we use 2-layered MLP as well.

For the projection head $p_\zeta(\cdot)$ we share the first layer of Q-head whereas for projection head $b_\eta(\cdot)$ we use a single layer MLP.

B.5. Algorithm

Algorithm 1 Equivariant Representations for RL

Denote the parameters of online networks τ_θ, κ_ϕ, p_ζ, b_η as Θ_o
Denote the parameters of target networks τ, κ, p, b as Θ_c
Denote the parameters of networks ρ_ω, q_ξ as Φ
Denote the dept of the prediction as M and batch size as N
Initialize the replay buffer $\mathcal{B}$
while Training **do**
  Collect $\{s, a, r, s'\}$ using policy with (Θ_o, Φ) and add to the buffer $\mathcal{B}$
  Sample a minibatch of M length sequences $\{s_{0:M}, a_{0:M}, r_{0:M}\} \sim \mathcal{B}$
  for i in range$(0, N)$ **do**
    if augmentation **then**
      $s^i_{0:M} \leftarrow augment(s^i_{0:M})$
    end if
    $x^i_0 \leftarrow \tau_\theta(s^i_0)$ {state representation}
    $\hat{x}^i_0 \leftarrow x^i_0$
    $l^i \leftarrow 0$
    for k in range$(1, M+1)$ **do**
      $\hat{x}^i_k \leftarrow \kappa_\phi(\hat{x}^i_{k-1}, a^i_{k-1})\hat{x}^i_{k-1}$ {state transition by group action}
      $\bar{x}^i_k \leftarrow \tau(s^i_k)$ {target state representation}
      $\hat{z}^i_k \leftarrow p_\zeta(\hat{x}^i_k)$, $\bar{z}^i_k \leftarrow p_\zeta(\bar{x}^i_k)$ {projections}
      $l^i \leftarrow l^i + \lambda_2 \| \frac{\hat{z}_{t+k}}{\|\hat{z}_{t+k}\|_2} - \frac{\bar{z}_{t+k}}{\|\bar{z}_{t+k}\|_2} \|^2_2$ {compute L_{AET} at step k}
      $\hat{r}^i_k \leftarrow r_\psi(\hat{x}^i_k)$ {predict rewards}
      $l^i \leftarrow l^i + \lambda_1 \|\hat{r}^i_k - r^i_k\|^2_2$ {compute L_R at step k}
    end for
    $j \sim \{0, .., N-1\}$ {uniformly sample an index}
    $\bar{x}^j_0 \leftarrow \tau(s^j_0)$ {encode the state for that index from the batch}
    $\tau^i_g = \bar{x}^j_0 {x^i_0}^{-1}$ {find the group representation}
    $\hat{x}^j_1 \leftarrow \tau^i_g \hat{x}^i_1$ {next state by group action}
    $\bar{x}^j_1 \leftarrow \rho_\omega(\tau^i_g)\kappa(x^i_0, a^i_0)\bar{x}^j_0$ {next state by action-embedding}
    $\hat{y}^i_1 \leftarrow b_\eta(\hat{x}^j_1)$, $\bar{y}^i_1 \leftarrow b(\bar{x}^j_1)$ {projections}
    $l^i \leftarrow l^i + \lambda_3 \| \frac{\hat{y}^i_1}{\|\hat{y}^i_1\|_2} - \frac{\bar{y}^i_1}{\|\bar{y}^i_1\|_2} \|^2_2$ {compute L_{GET}}
    $l^i \leftarrow l^i + RLloss(\hat{x}^i_0, a^i_0, r^i_0, \bar{x}^i_1; q_\xi)$
  end for
  $l \leftarrow \frac{1}{N} \sum_{i=0}^{N} l_i$ {average over minibatch}
  $\Theta_o, \Phi \leftarrow optmize((\Theta_o, \Phi), l)$ {update online networks}
  $\Theta_c \leftarrow \Theta_o$ {copy weights to target networks}
end while

B.6. Hyperparameters

In this section, we provide the full set of hyperparameters in our model. As mentioned earlier, our baseline RL algorithm closely follows SPR's (Schwarzer et al., 2021) implementation of Rainbow and hence we use most of their hyperparameters setting in order to be able to compare to them. Note that the weights of L_R - λ_1, L_{AET} - λ_2 and L_{GET} - λ_3 are set to one whenever they are used in the model.

Table 5. Hyperparameters for ErQ (including variations) on Atari.

Parameter	**Setting**
Gray-scaling	True
Observation down-sampling	84×84
Frames stacked	4
Action repetitions	4
Reward clipping	$[-1, 1]$
Terminal on loss of life	True
Max frames per episode	108K
Update	Distributional Q
Dueling	True
Support of Q-distribution	51
Discount factor	0.99
Minibatch size	32
Optimizer	Adam
Optimizer: learning rate	0.0001
Optimizer: β_1	0.9
Optimizer: β_2	0.999
Optimizer: ϵ	0.00015
Max gradient norm	10
Priority exponent	0.5
Priority correction	$0.4 \rightarrow 1$
Exploration	Noisy nets
Noisy nets parameter	0.5
Training steps	100K
Evaluation trajectories	100
Min buffer size for sampling	2000
Replay period every	1 step
Updates per step	2
Multi-step return length	10
Prediction depth, M	5
λ_1	1
λ_2	1
λ_3	1
Data Augmentation	Random shifts (±4 pixels) Intensity(scale=0.05)

Parameter	**Setting ($\mathbf{T_2}$)**	**Setting ($\mathbf{SE_2}$)**	**Setting ($\mathbf{GL_2}$)**
Num Blocks (K)	32	12	12
Group Action	Addition	MatMul	MatMul