A Thesis Title

Author Name

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
of
University College London.

Department of Something
University College London

September 5, 2023

I, Author Name, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

Abstract

My research is about stuff.

It begins with a study of some stuff, and then some other stuff and things.

There is a 300-word limit on your abstract.

Acknowledgements

Acknowledge all the things!

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The power of symmetry in understanding the physical world has been surprisingly effective. Perhaps one of the most famous examples of this is the eightfold way of Gell-Mann[Gell-Mann, 1961], which brought much deeper insight into the structure of elementary particles.

The power of symmetry is not only useful in abstract reasoning. The natural world has a bias for symmetries [Johnston et al., 2022], and the simplicity they enable. There also exists evidence that humans exploit symmetries in cognitively challenging tasks [He et al., 2022]. This apparent bias towards symmetry and evidence that humans leverage it in making decisions motivates the exploration of artificial decision-making agents with the ability to leverage the idea of symmetry, which this report focuses upon.

Symmetries are a wider concept than that of things with the same reflection down a centre line. Symmetries describe transformations that leave something unchanged, for example rotating a square by 90^o. However, the transformations are not limited to rotations and reflections, but can extend to wider ideas of time inversion, permutations. Symmetries also enable more abstract mathematical reasoning through the ideas of groups. The more abstract notions of symmetry have enabled more far-reaching results. Such as, Noether's theorem, which suggests that symmetries are a fundamental source of conservation laws, such as energy conservation, momentum, and angular momentum.

In machine learning ideas describing fundamental rules about the world and

associated rules are known as inductive biases. In order to make the agent understand these symmetries, there are two main approaches to induce these priori ideas into the agent: Data augmentation, and encoding the invariance into the agent.

Data Augmentation techniques leverage known symmetries in environment and create artificial training data by transforming the input such that the symmetry is respected, for example, flipping an input image horizontally [Laskin et al., 2020, Lin et al., 2020]. Such techniques have substantial history in supervised learning, where it is common to increase the amount of data by performing transformations on the input that preserve the output.

The second approach is to structure the agents' neural networks to only learn behaviours that respect the environment's symmetry[van der Pol et al., 2020, Wang et al., 2022, Mondal et al., 2020]. Notably, addressing the issue of generalizing to symmetries represented a key breakthrough in deep learning with the introduction of Convolutional Networks (CNNs)[LeCun et al., 1989]. These networks feature translational invariance, which is particularly important in computer vision. Further enhancements, like Group-Convolution[Cohen and Welling, 2016], extend this equivariant behaviour to accommodate a wider range of symmetries.

This report investigates the use of Group-Convolutional neural networks to introduce symmetric inductive biases into decision-making agents. These agents operate within the paradigm of Reinforcement Learning (RL), in which the agent gains and learns from experience to improve its performance. Additionally, the report explores incorporating symmetric inductive biases into a planning phase. In this phase, rather than directly learning from experience in solving a problem, the agent 'imagines' the results of its actions. This approach aims to enhance the agent's problem-solving and decision-making capabilities.

Chapter 2

Background

2.1 Markov Decision Process

[Bellman, 1957] described a framework for solving stochastic control problems. A classic example of this is playing blackjack. There is often no certain outcome to playing a given hand. However, actions exist for which the expected probability of a player winning is higher. The Markov Decision Process (MDP) formalism allows one to tackle such problems systematically, with mathematically tractable convergence bounds to optimal solutions in some cases.

The MDP describes these processes as an agent acting in an environment that is, in general, partially observable.

This is described by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, P, \gamma)$. $\mathcal{S}$ is the state space, the state of the world that the agent knows about, and $\mathcal{A}$ is the action space, the possible actions it may take. For a given state and action, the reward provided is real-valued and may be stochastic, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. P are the transition probabilities between states $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, this defines the dynamics of the process. γ determines the reward discount, how myopic the MDP is.

The agent aims to maximize its expected discounted cumulative reward, the return,

$$\mathbb{E}[G_t = \sum_{t}^{T} \gamma^t R_t]. \tag{2.1}$$

The problem may be that of an episodic setting with a clear termination, a chess player winning, for example. Alternatively, it may be in a continuous setting with

no clear termination, like managing stocks. To make decisions, an agent follows a policy. The policy, $\pi(a|s), \pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ maps states' actions to probabilities of taking action a. In the deterministic case it is just a map from state to action space.

A family of algorithms known as dynamic programming exists that, given finite states and actions, can find exact solutions to these problems, under reasonable constraints. However, these algorithms are often intractable for large state and action spaces. This is due to their algorithmic complexity being bounded by $\mathcal{O}(|\mathcal{S}||\mathcal{A}|)$.

Solutions to MDPs can be expressed in terms of the value function, $V(s) = \mathbb{E}[G_t|s_t = s]$, which is the expected return from a given state.

The optimal value function, $V^*(s)$, is the maximum value function over all possible policies. The optimal deterministic policy, $\pi^*(s)$, is the policy that maximizes the value function for all states. The optimal value function and policy satisfy the Bellman optimality equations[Bellman, 1957]:

$$V^*(s) = \max_a \mathbb{E}[R_{t+1} + \gamma V^*(s_{t+1})|s_t = s, a_t = a]. \tag{2.2}$$

The value function is very closely linked to the action-value function $Q(s, a) = \mathbb{E}[G_t|s_t = s, a_t, = a]$ which is the expected return from taking action a in state s and then following the current policy. There also exists a bellman optimality equation for the action-value function:

$$Q^*(s, a) = \mathbb{E}[R_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a')|s_t = s, a_t = a]. \tag{2.3}$$

Reinforcement learning is a common alternative to dynamic programming, which provides a principled approach to learning approximate solutions. It describes a family of techniques that learn from previous experience to improve at solving MDPs.

2.2 Reinforcement Learning

Reinforcement Learning (RL) encompasses the techniques used to solve MDPs where experience is required to solve the MDP. Especially in Deep RL, gradient based function approximations are used to parametrize either policies or value functions. Despite the approximate nature of their solutions, RL agents have achieved superhuman performance in Chess, Go, Star-Craft and have recently found diamonds in Minecraft [Silver et al., 2016, Silver et al., 2017, Hafner et al., 2023]. Many of these successes with current RL approaches highlight the challenges current algorithms face due to their hunger for data. This poor sample efficiency, is permissible in game environments as they are cheap to sample from. Even in the video game setting, these agents took millions of dollars, and many hours of compute. In real world tasks the costs of gaining samples is more expensive and presents an even larger challenge.

This sample hunger, combined with the challenge of to transfer knowledge between tasks [Kirkpatrick et al., 2017], highlights that there is a significant advantage to encoding prior knowledge into RL algorithms. The knowledge of symmetries enables agents to attempt simplified problems and draws abstract connections between different tasks. This thesis focuses on the encoding of symmetries into RL algorithms. Before introducing symmetry rigorously, this section will describe the most common approaches to solving complex RL problems.

2.3 Deep Reinforcement Learning Approaches

The underlying optimization problem in reinforcement learning is finding a policy which produces the maximum expected return. However, not all RL algorithms directly maximize return. The techniques used to train agents are broadly separated into two categories: Model-free and model-based algorithms.

Model-free algorithms choose to forgo learning the dynamics of a process and instead model the value function or policy directly.

In contrast, model-based algorithms learn the problem's dynamics and simulate the results of agent's behaviour in an environment to learn. This section will

describe the most common approaches to solving RL problems. Typically, learning algorithms face a few persistent problems in Deep RL that make solving problems difficult. A non-exhaustive list is:

- **Sample Efficiency** The algorithms require a large amount of data to learn. This is because training large neural networks are sample inefficient, taking tens hours to learn tasks such as playing Atari [Hessel et al., 2018], which humans can do in minutes. As mentioned before, breakthroughs in RL are often made in games, where data can be simulated and collected cheaply[Dulac-Arnold et al., 2019]

- **Training Robustness** The algorithms can often struggle to converge depending on the initialization of the environment and the agent, and special random seeds are required to see good performance[Henderson et al., 2018].

- **Reward Sparsity** Reward functions are often sparse, and coming across states that provide rewards, such as finding diamonds in Minecraft, is a difficult problem when initial exploration is random [Hafner et al., 2023].

- **Deadly Triad** many algorithms use their own predictions to update their current behaviour, a process known as bootstrapping. This in conjunction with off-policy learning can lead to Q-value divergence [Van Hasselt et al., 2018], and an inability to learn.

- **Catastrophic Forgetting** like in the context of supervised learning, deep networks have the problem that when trained on sequential tasks they loose performance on previous tasks in the sequence [Kirkpatrick et al., 2017]. These problems can be somewhat overcome by storing experience in the MDP and always using this to update the agent.

2.3.1 Model Free Algorithms

Model free algorithms attempt to train an agent to maximize returns from agent experience. The two key paradigms either approximating a value function are, Q-

learning. Or the agent learns the best action to take in given states these are policy gradient methods.

When combining standard dynamic programming algorithms such as Q-learning with function approximation, or direct policy learning, the challenge is finding a way to learn the form of the parametric function approximator, which is often a neural network. As such, there is a common approach between multiple algorithms, where you form a supervised learning problem, finding a function that minimizes a loss. In the RL setting, the data is gained from the MDP. The fundamental difference between Q-Learning and Policy Gradient approaches is that Q-learning uses an epsilon greedy policy to collect experience. Whereas, Actor Critics learn and gain experience with the same policy. However, the training loop is much the same;

Algorithm 1 Intuition For Model Free RL Training Loop

```
Initialize θ randomly.                                   ▷ Network Parameters
Initialize t = 0                                         ▷ Buffer Timestep
Initialize T = 0                                         ▷ Total Timestep
Initialize replay buffer D ← ∅
Sample state s
while do T < T_max
    Take action a according to policy π(a|s)
    Observe reward r and next state s'
    Store transition (s, a, r) in replay buffer D
    if t mod T_experience = 0 then                       ▷ Update Loop
        Perform gradient-based learning update
        t ← 0
        D ← ∅
```

In most cases, some operations need to be performed on the replay buffer, and the buffer will store more data than just sequential $(s, a, r)_t$ tuples. Some might argue that replay buffers are not required in certain algorithms, but this is just the case of a single transition long replay buffer. Often subsequent actions are required to calculate the loss. This can be done during the update loop. Finally, rather than interacting with one environment, training can be done in parallel in multiple environments. The process is much the same; states, rewards and actions become vectors for this. Environment vectorization has improved training stability and ro-

bustness [Mnih et al., 2016].

2.3.1.1 Deep Q-Learning

Q-Learning forgoes optimizing the true objective, the policy, to learn the optimal value function. This allows one to find an optimal policy, as there is always a deterministic greedy policy, $\pi^*(s) = \arg\max_b Q^*(S, b)$, with a stationary point at the optimal value function.

Such ideas are the foundations of Dynamic Programming, which exploits this idea in algorithms such as Value Iteration[Bellman, 1957] and Policy Iteration[Howard, 1960].

When extended to function approximation the convergence guarantee of the original dynamic programming Q-learning algorithm is absent in Deep-Q-Learning. While an exact solution may not be found due to computational constraints, excellent solutions can be found in practice using this method [Mnih et al., 2013]. Traditional Q-learning updates the value of the current state-action pair, $Q(s, a)$, using the Bellman optimality equation for the action-value function,

$$Q_{t+1}(s, a) = Q_t(s, a) + \alpha \left[r(s, a) + \gamma \max_{a'} Q_t(S_{t+1}, a') \right]. \tag{2.4}$$

Where α is the step size, all other symbols take their normal values. The important point is that this converges to the optimal value function given infinite state action space visitation. In the setting of Deep Q-Learning, we lose the guarantees of convergence in exchange for the ability to attack problems that have large on continuous state-action spaces. An important point to note about Q-Learning is that the agent uses its own predictions to learn. This is known as bootstrapping and can cause problems, famously the deadly triad [Van Hasselt et al., 2018, Sutton and Barto, 2018].

The Bellman optimality equation can be used to construct a loss function to train a parametric function approximation, a deep learning model, through gradient methods, If this method were employed in an on-policy fashion with a deterministic greedy optimal policy, there would be no exploration. As such, the agent would not

gain new experience, so an alternative exploration policy is used, which alternates between greedy and random actions. The results of its trajectory or trajectories, depending on whether it is an episodic setting, are stored in a replay buffer,

$$\mathcal{D} = \{S_0, A_0, R_1, S_1, A_1, \ldots\} \tag{2.5}$$

This can be sampled when training the network. The loss function to optimize is derived from the Bellman equation and is the squared temporal difference error[Sutton and Barto, 2018],

$$L_w(\mathcal{D}) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[\left(Q_w(s,a) - (r + \gamma(1 - \mathbb{I}(s' = S_T)) \max_{a'} Q_{w'}(s', a')\right)^2\right]. \tag{2.6}$$

Here, it is essential to note that Q'_w can be the original network if $w = w'$. However, there are some problems associated with this. The gradient with respect to the loss must now become a semi-gradient, to allow the bootstrapping off one's own predictions, where the Q'_w term must be considered a constant with respect to w. The indicator, $\mathbb{I}(s' = S_T)$, is one if s' is a terminal state.

Alternatively, the semi-gradient can be avoided with Double-Q-Learning. Double-Q-Learning uses target networks [Van Hasselt et al., 2016]. This makes two copies of the primary, Q_w, network and performs gradient updates on one, Q_w, with the secondary network, $Q_{w'}$, parameters being moved towards w after some further number of updates so that its parameters lag that of w. This can be done with weighted averages or just a direct update.

Some problems arise in continuous state spaces, where the $\max$ operation may involve an optimization loop. This is not ideal. Algorithms such as Deep Deterministic Policy Gradient (DDPG)[Lillicrap et al., 2015] use another network to parametrize an optimal policy instead of the max operation. DDPG is an example of an Actor-Critic algorithm that straddles the divide between Q-Learning and policy based methods.

2.3.1.2 Policy Based Methods

Policy based methods aim to learn an optimal policy directly. Policy based methods strive to optimize the expected cumulative return for an agent by performing gradient descent on the obtained rewards. The return for a given action, is not known beforehand and must be estimated. Reducing the variance of the estimator is a key challenge. There are multiple tricks used in deriving the gradient estimator to reduce variance as well as algorithmic improvements such as Soft Actor-Critic (SAC) [Haarnoja et al., 2018], Proximal Policy Optimization (PPO)[Schulman et al., 2017].

In the episodic setting, where it is possible to obtain complete trajectories and optimize for episodic returns, policy gradient functions can directly optimize the expected cumulative reward,

$$J_G(\theta) = \mathbb{E}_{s_0 \sim d_0} \left[v_{\pi_\theta}(S_0) \right]. \tag{2.7}$$

However, in infinite time horizon tasks, there is no termination and optimizing for the expected reward of the next action may be a more prudent objective,

$$J_R(\theta) = \mathbb{E}_{\pi_\theta} \left[R_{t+1} \right]. \tag{2.8}$$

Many common algorithms for episodic tasks can be extended for infinite time horizon problems. In the rest of the report, the episodic setting is only considered.

The general policy gradient is of the form,

$$\nabla J_G(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} \mathcal{R}_t \nabla_\theta \log(\pi_\theta(A_t|S_t)) \right]. \tag{2.9}$$

Where $\mathcal{R}_t$ is a return estimator a time t, if the return estimator is the discounted sum of rewards, this is the REINFORCE algorithm [Williams, 1992]. The most used modern policy gradient algorithms take this form: a return estimator, which is either sampled or bootstrapped with a possible constant baseline offset. This leads to Actor-Critic methods, where the actor is the policy network, and the critic is a

value function estimation network.

2.3.1.3 Actor Critic Methods

Actor critics, are a hybrid approach between policy gradient methods and value based methods. In Deep RL they consist of a pair of function approximation networks, one the actor to approximate the optimal policy, the critic to approximate the optimal value function. Because the gradient of a constant is zero, having a baseline that is independent of the policy does not affect the expectation of the policy gradient estimator, but if picked wisely, it may reduce the variance,

$$\nabla J_G(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T}(\mathcal{R}_t - b)\nabla_\theta \log(\pi_\theta(A_t|S_t))\right]. \tag{2.10}$$

As the variance of the updates depends on the magnitude of the $(\mathcal{R}_t - b)$ term, a function is learned to estimate the value of the state, $V_\phi(S_t)$, and the baseline is set to the value of the state, $b = V_\phi(S_t)$. The critic trained to minimize the mean squared error between the estimated value and the actual return. This produces an unbiased update of lower variance, which has better training behaviour[Sutton and Barto, 2018].

2.3.1.4 Generalized Advantage Estimation

Generalized advantage estimation[Schulman et al., 2015b], learns the λ-return, a weighted sum of the n-step discounted returns, in contrast to learning the discounted return of previous methods. The λ-return is defined as,

$$R_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}R_{t:t+n}^V. \tag{2.11}$$

Where $R_{t:t+n}^V$ is the n-step discounted return and lambda is a hyperparameter that controls the trade-off between bias and variance on the return estimator as it gradually uses its own estimates for states values, when $\lambda = 0$, it is an unbiased return estimate. The n-step discounted return is defined as,

$$R_{t:t+n}^V = \sum_{k=0}^{n-1}\gamma^k r_{t+k} + \gamma^n V_\phi(S_{t+n}). \tag{2.12}$$

The lambda return provides a biased but lower variance return estimator, which can benefit training and is the backbone of many modern policy gradient algorithms. Typically, the lambda return is calculated recursively backwards in episodic settings,

$$R_t^\lambda = r_t + \gamma(1-\lambda)V_\phi(S_{t+1}) + \lambda R_{t+1}^\lambda. \tag{2.13}$$

And then the advantage is the difference between the value function and the lambda return,

$$A_t^\lambda = R_t^\lambda - V_\phi(S_t). \tag{2.14}$$

Having the generalized advantage as the optimization target provides control over the variance of the gradient estimator. The generalized advantage estimator is defined as,

$$\nabla J_G(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} A_t^\lambda \nabla_\theta \log(\pi_\theta(A_t|S_t))\right]. \tag{2.15}$$

In this situation, the critic network learns to minimize the advantage. Advantage estimation is often used with regularization methods that stop the current policy from moving too far between training updates. These regularization methods penalize the KL divergence between the current and previous policies. This is the case for Trust Region Policy Optimization (TRPO) [Schulman et al., 2015a] and the computationally efficient and simple PPO[Schulman et al., 2017].

2.3.1.5 PPO

Proximal policy optimization (PPO) follows the same lines as TRPO, as it is a regularized form of gradient-based policy optimization with a critic that learns a value function as a bias reduction method. The PPO paper introduces two forms: a hard threshold method PPO-Clip, and a soft regularization method PPO-Penalty. The clip optimization target, that removes the need for explicitly calculating a KL-divergence for a single interaction is,

$$L(s,a,\theta_k,\theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}A^{\pi_{\theta_k}}(s,a), g\left(\epsilon, \frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}\right)A^{\pi_{\theta_k}}(s,a)\right). \tag{2.16}$$

Where the update rule is defined by the gradient descent algorithm, and θ_k is the previous value of θ. The advantage of a policy π is given by a truncated λ-return,

$$A^{\pi}(s,a) = V_{\phi}^{\pi}(s) - R_t(s,a). \tag{2.17}$$

Where the $V_{\phi}^{\pi}(s)$ is the critic, typically a neural network. $R_t(s,a)$ is the sampled return for the state action pair. As the algorithm is on-policy, there is an importance-sampling correction to the advantage. To stop wide variance in the policy, like TRPO, the magnitude of the update is capped with the g function,

$$g(\epsilon, A) = (1 + \text{sign}(A)\epsilon)A. \tag{2.18}$$

The ϵ hyperparameter must be set. This loss function ensures that there are no overly large policy updates.

2.3.2 Model Based Algorithms

Model-based reinforcement learning is the process of the agent learning the dynamics of a system and then performing "planning" to decide how best to act next. In concrete terms, this corresponds to an agent learning the transition dynamics of the MDP, thus learning both the subsequent state of a state action pair, $P(s'|s,a)$ and the expected reward from a state action pair $r(s,a)$. From the learned model agent "plans" by simulating the environment and acting in the simulated environment. Then the agent learns a policy with the simulated data. Common algorithms like Dyna-Q [Sutton and Barto, 2018], use Q-learning over a combination of real and simulated data to learn a policy.

Model-based algorithms have the potential to provide much better sample efficiency than model-free algorithms, as they can learn from simulated data. However, this in itself can be difficult to learn. In cases where the state-action space is infinite the model must generalize to states-actions that have not been seen before. However, the Dreamer models have shown substantial success in tackling difficult problems[Hafner et al., 2023, Hafner et al., 2020].

2.4 Dyna

Dyna style algorithms[Sutton and Barto, 2018, Sutton et al., 2012] augment model free algorithms, while using the same experience based updates, such as Q-learning 2.3.1.1 or PPO 2.3.1.5, they train, $T(S_{t+1}|S_t, A_t)$ a transition model. This can improve the sample efficiency of an agent. The basic Dyna formulation is outlined in pseudocode below. Here; L_a, L_{mb}, and L_{rw} are the agent, transition

Algorithm 2 Dyna

```
Iritialize θ, φ, ψ, randomly
for Num Epochs do
    for Num Dyna Iterations do
        for Num Acting Updates do
            Sample transition tuples from policy π_θ ∼ (s, a, s') on M
            Construct replay buffer from transitions D = {(s, a, s', r)_i}_1^N
            θ' ← Minimize L_a(θ, D)
            φ' ←Minimize L_mb(φ, D)
            ψ' ←Minimize L_rw(ψ, D)
        θ ← θ'
        φ ← φ'
        for Num Planning Updates do
            Sample transition tuples from policy π_θ ∼ (s, a, s') on T_φ
            Construct replay buffer from planned transitions D_p =
            {(s, a, s', r)_i}_1^N
            θ' ← Minimize L_a(θ, D_p)
        θ ← θ'
        ∅ ← D
        ∅ ← D_p
```

model and reward model losses, respectively. The minimization is typically through stochastic gradient descent.

Dyna provides a conceptually simple model based augmentation on conventional model free algorithms. By simulating transitions using previous experience, higher sample efficiency may be gained at the cost of a more complex training regime.

Next, the mathematical abstractions to deal with symmetries are introduced such that the methods to form inductive biases in RL agents can be understood.

2.5 Groups, Symmetries, Homomorphisms

2.5.1 Groups

Groups are an abstract mathematical idea on a set with a binary operation, $(\_ \cdot \_)$. To form a group, the members of a set must satisfy the following:

1 Closure: applying the group's operation maps all elements back onto another element.

2 Possession of an identity element: there must be an element of the set such that it and any element is mapped onto itself.

3 Possession of inverse elements: every element in the group has an inverse element.

4 Associativity: $(a \cdot b) \cdot c = a \cdot (b \cdot c)\ \forall a, b, c$

The key point is that specific symmetries form groups of all the transformations that leave the object/space invariant. An example of a group table can be found below for the cyclic group C_2 in section 4.2.

2.5.2 Representation and Actions

Members of groups like rotation operations or flip operations maintain their properties no matter what space you are in. For example flipping an image or a function the group is still the same, in that if you perform two flips you get the same image/function. As such, the group is a very general and abstract concept. When dealing with groups, in a concrete setting then the group is a set of matrices or functionals that operate on a space or a function, these are in general described as group actions, for our purposes only the left action is needed. The left and right actions refer to the side of the operand the operator is applied. The group action is defined as,

$$\ell_g : \mathcal{X} \times G \to \mathcal{X}, \forall g \in G, \forall x \in \mathcal{X}. \tag{2.19}$$

Where $\mathcal{X}$ is an arbitrary set and the left action obeys the following properties:

1 $\ell_{g_1}(\ell_{g_2}(x)) = \ell_{g_1 \cdot g_2}(x)$.

2 $\ell_e(x) = x$.

Additionally, because of the definition of a group action, the application of a group action to a function is equal to the function applied to the inverse of the group action applied to the function's domain, i.e.

$$\ell_g(f(x)) = f(\ell_{g^{-1}}(x)). \tag{2.20}$$

The inverse group action is commonly written as $\ell_{g^{-1}}(x) = g^{-1}x$, when acting on the domain of a function.

In the special case of vector spaces the group actions are invertible matrices and are called a representation;

$$\pi_g^{\mathcal{V}} : G \to GL(\mathcal{V}), \forall g \in G. \tag{2.21}$$

Where $\mathcal{V} \in \mathbb{R}^n$ is a vector space, and the group representation $\pi_g^{\mathcal{V}}$ is an invertible matrix. These representations also follow the properties of the group, i.e. they are invertible, and the identity element is mapped to the identity matrix.

2.5.3 Invariances

Invariances are properties maintained under a transformation of the input space, e.g. mirror symmetry, where the distance to a point on the mirror line is the same from the left and the right if the object is symmetric. If you have a function $f : \mathcal{X} \to \mathcal{Y}$ that is invariant to a transformation $\ell_g(x) : \mathcal{X} \times G \to \mathcal{X}$, this is expressed as:

$$f(x) = f(\ell_g(x)), \forall g \in G, \forall x \in \mathcal{X}. \tag{2.22}$$

For the transformation to be a group, there must also include the identity element s.t $\ell_h(x) = x, h \in G, \forall x \in \mathcal{X}$. From this definition one sees that symmetries themselves are invariances with respect to the action of a group. For example, suppose an object has 3-fold rotation symmetry around an axis. In that case, an abstract group with operations between its elements mirrors that of rotations around the axis by $120^o, 240^o$ and 0^o. It is straightforward to check that these operations

form a group. Rotations by 360^o left or right are the identity element. The group is closed; no matter how many rotations are performed, the object with the symmetry is still the same. The inverse of the rotation by 120^o is the rotation by 240^o and vice versa. Finally, the rotations are associative. This is the group of symmetries of the object. The object is invariant to the transformations in the group.

2.5.4 Equivariance

Equivariance is related in that if there is some transformation $\ell_g : \mathcal{Y} \times G \to \mathcal{Y}$, where it is a different representation of the group,

$$\ell_g^{\mathcal{Y}}(f(x)) = f(\ell_g^{\mathcal{X}}(x)), \forall g \in G, \forall x \in \mathcal{X}. \tag{2.23}$$

Here, both ℓ_gs are actions of the same group member. However, the spaces they act upon are different. This notion is especially important in the space of RL. From this definition, we can see that invariance is a particular case of equivariance where $\ell_g^{\mathcal{Y}}$ is the identity, $\forall g \in G$.

2.5.5 Homomorphisms

A homomorphism describes a map between two structures that preserves an operation. In the context of reinforcement learning, the preservation is that of the dynamics and reward structure of a Markov Decision Process.

2.6 MDP Homomorphism

An MDP homomorphism describes a surjective mapping between one MDP and another; Ravindran and Barto first described this notion in the early 2000s[Ravindran, 2003, Ravindran and Barto, 2001].

Definition: MDP Homomorphism Given some MDP $\mathcal{M}$, where there exists a surjective map, from $\mathcal{S} \times \mathcal{A} \to \overline{\mathcal{S}} \times \overline{\mathcal{A}}$ where all abstract quantities are denoted with an $\overline{\text{overline}}$. The MDP $\overline{\mathcal{M}}$ is an abstract MDP over the new space. The homomorphism h, then is the tuple of $(\sigma, \alpha_s | s \in \mathcal{S})$, where $\sigma : \mathcal{S} \to \overline{\mathcal{S}}$ and $\alpha_s : \mathcal{A} \to \overline{\mathcal{A}}$. This surjective map must satisfy two constraints for it to be a valid MDP homomorphism;

1. $R(s,a) = R(\sigma(s), \alpha_s(a))$

2. $P(s'|a,s) = P(\sigma(s')|\alpha_s(a), \sigma(s))$

Given this formulation from [van der Pol et al., 2020], they show that there are equivalences between the real optimal value function and the optimal abstract value function,

$$V^*(s) = \overline{V}^*(\sigma(s)).$$
$$Q^*(s,a) = \overline{Q}^*(\sigma(s), \alpha_s(a)).$$

This is *optimal value equivalence*. Further, they introduced the idea of "lifting", where a policy learned in the abstract space can be mapped to the original space. The lifted policy $\pi^{\uparrow}$ is related to the abstract policy $\overline{\pi}$ by dividing by the preimage, the set of inputs to the homomorphic mapping that map to the same output.

$$\pi^{\uparrow}(a|s) = \frac{\overline{\pi}(\alpha_s(a)|\sigma(s))}{|a \in \alpha_s^{-1}(\bar{a})|}. \tag{2.24}$$

As such any solution found in an abstract MDP can easily be transferred to the original MDP if a homomorphism exists.

2.6.1 Group Structured MDP Homomorphisms

A natural homomorphic mapping is that of an MDP that possesses symmetry, which is concretized by having the homomorphisms be the representations of the group in state and action space,

1. $R(s,a) = (\ell_g^{\mathcal{S}}(s), \ell_g^{\mathcal{A}}(a))$,

2. $P(s'|a,s) = P(\ell^{\mathcal{S}}(s')|\ell^{\mathcal{A}}(a), \ell^{\mathcal{S}}(s))$.

Where $\ell_g^{\mathcal{X}}$ is the representation of the element $g \in G$ in the space $\mathcal{X}$, in plain English, for each state or state action pair, there are a set of states with the same reward and transition probability to some new state, where these states are related to each other by the operations of the elements of g on them.

Inductive Bias	Corresponding property
Distributed representations	Inputs mapped to patterns of features
Convolution	group equivariance (usually over space)
Deep architectures	Complicated functions = composition of simpler ones
Graph Neural Networks	equivariance over entities and relations
Recurrent Nets	equivariance over time
Soft attention	equivariance over permutations
Self-supervised	pre-training $P(X)$ is informative about $P(Y\|X)$

The abstract MDP can then be solved with dynamic programming or approximate dynamic programming techniques[Van der Pol et al., 2020, Rezaei-Shoshtari et al., 2022], and the policy found in the abstract MDP can be ”lifted” to the original MDP.

2.7 Neural Networks and Inductive Biases

What is nice in many ways about the original MLP model is that it is so flexible. An infinite width MLP may produce any function [Hornik, 1991]. However, this also highlights one of their weaknesses when solving applied problems, because in theory MLPs have incredible expressive power, they are perfectly able to represent physically impossible functions. When applied to domains such as images, and physical processes where there are concrete rules governing the processes that are being modelled. Additionally, it is posited by [Wolpert et al., 1995], that there is no free lunch in machine learning, and that there must be a constrained search space for possible algorithms. Such constraints are in other words, inductive biases [Baxter, 2000]. This kind of reasoning has lead to research into encoding inductive biases into Deep Networks, from [Goyal and Bengio, 2022] a table of current approaches to introduce inductive biases;

For the purposes of this thesis, the focus will be on the induction of group equivariances. However, it is important to note that this strategy to improving machine learning models comes from a more general class of ideas.

2.7.1 G-CNNs

The Group Equivariant Convolutional Neural Network (G-CNN) is a generalization of the CNN's translational equivariance to arbitrary group structured equivariances.

The traditional Convolution layer is a discrete convolution, this is an approximation of the continuous convolution,

$$(f * k)(x) = \int_{\mathbb{R}^d} k(x - x')f(x')dy, \tag{2.25}$$

where f and k are functions on $\mathbb{R}^d$. What can be noticed is that this is the definition of the cross correlation between f and $\ell_g[k] : \mathbb{R}^d \to \mathbb{R}^d$, where $\ell_g[k]$ is the translation group $\mathbb{R}^d$ acting on the kernel k,

$$(f * k)(x) = \int_{\mathbb{R}^d} k(x - x')f(x')dx' \tag{2.26}$$

$$= \int_{\mathbb{R}^d} k(g^{-1}x')f(x')dx' \tag{2.27}$$

$$= \int_{\mathbb{R}^d} \ell_g[k](x')f(x')dx' \tag{2.28}$$

Here, the inverse of a translation by x, group action g is the translation by $-x$. This is then g^{-1}, the inverse of the group action. To demonstrate the equivariance, consider a group action on the function f;

$$(\ell_g[f] * k) = \int_{\mathbb{R}^d} \ell_g[k](x')\ell_g[f](x')dx' \tag{2.29}$$

$$= \ell_g[\int_{\mathbb{R}^d} k(x')f(x')dx'] \tag{2.30}$$

$$= \ell_g[(f * k)] \tag{2.31}$$

Thus, the layers are equivariant. When implementing this in practice, it amounts to applying all transformations to a single kernel that defines the trainable parameters for a layer. This is discussed in more detail in 2.3.1.3.

This is the backbone of the G-CNN[Cohen and Welling, 2016], where rather than a translation group, we have an arbitrary group G acting on the kernel k. When looking for more complex equivariances than C_2, multiple different groups can be used in the same layer, this increases the number of variables in the convolution's output, this complicates the form of the layers, however for our purposes this is not relevant.

Chapter 3

Literature Review

3.1 Deep Learning and MDP Homomorphisms

As discussed previously In Chapter 2 the idea of learning a group structured MDP homomorphism is a powerful tool to make the learning problem posed by an MDP simpler. This section will discuss current attempts at exploiting MDP homomorphisms, with a focus on group-structured MDP homomorphisms.

3.1.1 MDP Homomorphic Networks

[van der Pol et al., 2020] introduces the idea of performing policy-based learning that respects the symmetry of the environment, by constraining the possible policies that can be represented by a neural network.

This is achieved by using a network equivariant to Group Structured transformations on discrete action spaces. When a group-structured operation transforms the input to the network. The output policy is also transformed by this operation due to the equivariance property of the network. Thus, it exploits a group structured MDP homomorphism.

The equivariance in the deep network uses many of the same ideas as that of the G-CNNs [Cohen and Welling, 2016]. In that, the only requirement for a network to be equivariant to a discrete group action is that the individual layers of the network are equivariant to the group's actions. Despite the similarity, the [van der Pol et al., 2020] method of achieving the equivariance is quite different. The authors propose the "symmetrizer layer". In contrast to the group convolution

formulation, the symmetrizer layer achieves equivaraince by finding weight matrices that are solutions to,

$$\mathbf{W} = S(\mathbf{W}) = \frac{1}{|G|} \sum_{g \in G} \pi_g^{\mathcal{X}'^{-1}} \mathbf{W} \pi_g^{\mathcal{X}} \tag{3.1}$$

Where if $f(\vec{x}) = \mathbf{W}\vec{x}$, $f : \mathcal{X} \to \mathcal{X}'$ and $\pi_g^{\mathcal{X}}$ is the representation of g in $\mathcal{X}$, then $\pi_g^{\mathcal{X}'}$ is the representation of g in $\mathcal{X}'$. In order to find such linear systems of equations in a general manner for a group G, the authors sample many matrices randomly from the space of all possible matrices of that size $\mathcal{W}_{total}$. Then apply the symmetrizer operation, S, to all the sampled matrices.

Because the symmetrizer operation is linear, $\mathcal{W}$ is a set of solutions to the linear equation defined by the symmetrizer equation 3.1. To find this, many weight matrices are vectorised and stacked. This forms a new matrix, in which the singular value decomposition's basis vectors are orthogonal vectors of the equivariant subspace! These vectors $\{\mathbf{V}_i\}$ are all solutions to the above equation 3.1. As such, any linear combination of them is also a solution. Using the first r vectors of the SVD, where $r = \text{rank}(\mathcal{W})$. An equivariant layer can b3 formed by,

$$\mathbf{W} = \sum_{i=1}^{r} \alpha_i \mathbf{V}_i \tag{3.2}$$

These layers have interesting properties; the size of the subspace defines how many parameters the symmetrizer net has. There is no current closed-form solution to the question of how many parameters each layer will have.

This scheme of producing equivariant layers has some notable upsides, in that you only need to know how to transform the input and output of the layer, $\mathbf{W} \to \mathbf{W}\pi_g^{\mathcal{X}}$ and $\mathbf{W} \to \pi_g^{\mathcal{X}'^{-1}}\mathbf{W}$ respectively. However, the scheme does require expensive SVD calculations. In addition it requires the sampling of many matrices, which is expensive, but this only needs to be done once at the start of training for each layer. As mentioned earlier, there is no closed-form solution to how many parameters the matrices will have, and as such, the number of parameters in the network is not known until the SVD is performed.

The larger problems with this approach are that the homomorphisms must be exact and known priori. This limits the possible scenarios in which this approach can be used, as it is not always possible to know the exact symmetry. In addition, in many cases, generalizing to continuous action spaces is impossible with the current methodology.

3.1.2 Group Equivariant Deep Reinforcement Leanrning

In much the same vein as that of [van der Pol et al., 2020], [Mondal et al., 2020] proposes a method of exploiting equivariant networks for Deep RL, in comparison to [van der Pol et al., 2020], they propose using G-CNNs [Cohen and Welling, 2016] to achieve equivariance, rather than the symmetrizer layer. In contrast to learning a policy, they have a network structure such that the states are mapped to an equivariant Latent space of dimension 256. This results in a network architecture that can be thought of as an equivariant embedding function, $f(s)$, and a Q-Value function, $Q(s_{eqv}, a)$, acting on this space.

$$f : \mathcal{S} \to \mathcal{S}_{eqv} \tag{3.3}$$

$$Q : \mathcal{S}_{eqv} \to \mathbb{R}^{|A|} \tag{3.4}$$

One of the key downsides of this is that it doesn't exploit the MDP's homomorphism, which also exists in the action space. Despite this [Mondal et al., 2020] still demonstrate an improvement in sample efficiency over two baselines of DDQN [Van Hasselt et al., 2016] and DQN [Mnih et al., 2013] in Snake. However, they only see a minor improvement in sample efficiency in Pacman, which also possesses the same C_4 group symmetry.

3.1.3 SO(2) Equivariant Reinforcement Learning

In much the same way as the above methods, [Wang et al., 2022] also exploit the equivariance property in the context of robotic control in the PyBullet suite[Coumans and Bai, 2021]. They use steerable G-CNNs[Weiler and Cesa, 2019], which provide SE(2) equivariance in robotic control environments. In contrast to previous papers where finite groups were exploited, the continuous Lie group SO(2)

constrains the degrees of freedom of the problem much more than the finite groups. They can achieve impressive improvements in sample efficiency and robustness over conventional DQN and SAC methods.

In drawer-opening, block pulling, and object Picking experiments, the equivariant networks outperform all other DQN methods. When applied to the SAC formulation, In Object picking and Drawer Opening are the only methods to solve the task.

Further, they perform tests on more difficult robotic control tasks where demonstrations are provided; this enables the agents to tackle more complex tasks such as block stacking and house building, in which they are the only successful agents again. Due to the ability to generalise to states not seen in the demonstrations but that are in the orbit of the states seen in the demonstrations.

3.2 Learning Models

In the context of exploiting symmetries in RL, there is also the opportunity to build world models learned from experience. Two possible approaches are learning a group equivariant world model. In this case, the world model has the same state action space as the original MDP. However, the equivariance inductive bias may improve the sample efficiency of learning the world model and introduces better generalization for the agent. In some ways, this is the model-based extension of the equivariant model-free methods discussed above.

An alternative approach is to learn a group invariant model. In this case, the MDP homomorphism is used to transform the state action pairs in the same orbit to a single state action pair. This produces a world model that is ”simpler” than the original MDP 3.1. From the policy learned in the world model, the policy can be lifted back to the original MDP. An example of a similar strategy is that of approximate MDP Homomorphisms.

3.2.1 Approximate MDP Homomorphisms

While [Van der Pol et al., 2020] forgo learning explicit symmetries, they learn an approximate MDP homomorphism. This approximation is exact in the case when

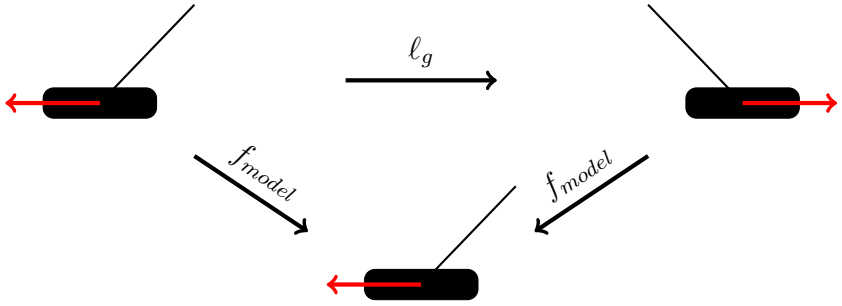

Figure 3.1: Diagram of the MDP Homomorphism created by a world model, that is invariant to ℓ_g, the group actions .

the model fits the data perfectly.

In contrast to learning symmetries, they are looking to find a state and action embedding that is equivariant to the model, such that,

$$Z(T(s,a)) = \overline{T}(Z(s), Z_s(a)) \tag{3.5}$$

Where Z is the state embedding function, and $Z_s(a)$ is the action embedding function. Thus, they construct and abstract MDP $\overline{\mathcal{M}}$, with state space $\mathcal{Z}$ and action space $\overline{\mathcal{A}}$, which they can act in. Further, rather than using deep RL, they discretize the state action space of the abstract MDP, $Z \to \mathcal{X}$, and use value iteration, repeated application of the bellman optimality operator, to learn the values of all state-action pairs. The value function can be interpolated from here to any abstract state-action pair. Pseudocode for this process is given below. The world model for the MDP consists of four jointly trained networks, Z, Z_a, $\overline{T}$,$R_z(z)$, which are trained to minimize a bisimulation metric. A bisimulation metric measures how different the dynamics of two MDPs are. The bisimulation metric used is a squared distance between the abstract MDP and the original MDP;

$$L(\theta,\phi,\xi,\psi) = \sum_{(s',s,a)\sim\tau} d(Z_\theta(s'), \overline{T}_\phi(Z_\theta(s), Z_{a_\xi}(a))) + d(R(s'), R_{z_\psi}(Z_\theta(s))) + C(\tilde{S}) \tag{3.6}$$

Where $d(z,z') = 1/2(z-z')^2$ is the squared distance between two quantities. Additionally, they add a contrastive term, $C(\tilde{S})$, to stop trivial embeddings. $\tilde{S}$ is a set of randomly sampled states from the trajectory. The contrastive term, with

model parameter dependence suppressed, is given by,

$$C(\tilde{S}) = \sum_{\tilde{s} \in \tilde{S}} \tilde{d}(Z_\theta(\tilde{s}), \overline{T}_\phi(Z(s), Z_{a_\xi}(a))) \tag{3.7}$$

Here, $\tilde{d}(z, z') = \max(0, \epsilon - d(z, z'))$, is a distance metric that encourages the embedding to not collapse to a point. This procedure produces impressive results, es-

Algorithm 3 Approximate MDP Homomorphism Pseudocode

Learn $Z : \mathcal{S} \to \mathcal{Z}$, $Z_a : \mathcal{A} \to \overline{\mathcal{A}}$
Discretise $\mathcal{Z}$ to $\mathcal{X}$.
$Q(x, a) \leftarrow$ Plan in $\mathcal{X}$.
$Q(z, a) \leftarrow$ Interpolate $Q(x, a)$
$Q(s, a) = Q(Z(s), Z_a(a))$

pecially when limited to very few episodes of interaction with the MDP. Compared with a REINFORCE baseline, on 100 interactions of CartPole, the episodic return is eight times higher. Additionally, across various tasks [Van der Pol et al., 2020], find a more structured latent space with comparable world model methods that use Encoder networks. There are, however, some limitations with the method's inability to generalize to stochastic transition dynamics, and it is not clear how the discretization would scale to larger MDPs.

3.2.2 A Simple Approach To Learning State-Action Abstraction using a Learned MDP Homomorphism

In Recent Reinforcement Learning, another approach that is related to learning an MDP homomorphism from a world model [Mavor-Parker et al., 2022] not only learns the forward transition from a given state, $T_\theta(s_t, a_t)$ but also learns the reverse transition dynamics, $B_\phi(s_{t+1}, a_t)$. Where T predicts the next state s_{t+1} and B, predicts s_t, the previous state. These two models are used to find state action pairs, $((s_t, a_t), (s'_t, \overline{a}))$, that have the same next state, which in many MDPs are states with the same value—producing an MDP homomorphism between the two sets of states with equivalent effects.

If the action a_t transitions the agent from s_t to s_{t+1}, $T(s_t, a_t) = s_{t+1}$, then

there may exists an action $\overline{a}$ such that,

$$B(T(s_t, a_t), \overline{a}) = s_t \tag{3.8}$$

This is the canonical action. There may not always be a canonical action, however.

In the case of MDPs, where the reward is only a function of the state and the transition dynamics are deterministic, then an MDP homomorphism can be found. By choosing a single canonical action, $\overline{a}$ and a start state, s_t. Equivalent effect state action pairs can be found to have the same value. Consider the set of next states $\{s_{t+1}\}$ defined by the set of actions $\{a_t\}$, such that,

$$\{T(s_t, a_i)\}_{i \in |A|} = s_{t+1}. \tag{3.9}$$

The the set state with the value $Q(s_t, a_j)$ for the cannonical action is given by $B(s_{t+1}, \overline{a}) = s'_t$. Thus $Q(s_t, a_j) = Q(s'_t, \overline{a})$, which defines a MDP homomorphism between the state action pairs $\{s_t, a_j\}$ and $\{s'_t, \overline{a}\}$. The Q values of only one set of these states need to be learned for the agent to be able to act optimally in the MDP. This reduces the sample complexity of learning in the MDP. [Mavor-Parker et al., 2022] Show this method's effectiveness in various environments.

This technique, however, becomes non-trivial to apply in stochastic environments due to the challenge of learning stochastic transition dynamics. Further, the method is not applicable if the actions define the reward.

3.3 Meta-Learning

Meta-Learning is the process of learning to learn across multiple tasks. This is a vast field where the goals are varied. Specifically, in this project, we are looking at parameter meta-learning, where a single model, f_θ, is applied to multiple different tasks, $\tau_i \sim \mathcal{T}$, drawn from a distribution. Each task has a per task loss, $L_{\tau_i}(\theta)$, a function of the model's parameters and may change its functional form.

The Model-Agnostic Meta-Learning (MAML) algorithm 4 is the canonical ex-

ample of these methods. It works by performing a gradient-based update on each task, storing the parameters,θ_i', of the model after each task's update, and then performs a gradient update on the meta loss, which is usually the sum of the per task losses $L_{\tau_i}(\theta_i')$, evaluated with their new parameters θ_i'. This is then repeated till convergence. Pseudocode is given below.

Algorithm 4 MAML Algorithm

θ is randomly initialised
while not done **do**
  Sample batch of tasks, $\tau_i \sim p(\tau)$
  for each task τ_i **do**
    $\theta_i' \leftarrow \theta - \alpha \nabla_\theta L_{\tau_i}(F\theta)$
  $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\tau_i \sim p(\tau)} L_{\tau_i}(\theta_i')$

3.3.1 Meta-Learning Symmetries By Reparametrisation

[Zhou et al., 2020], proposes a method to learn approximately equivariant networks, using the Model Agnostic Meta-Leaning (MAML) framework [Finn et al., 2017]. MAML provides gradient updates to one set of parameters. However, [Zhou et al., 2020] proposes a method of learning group convolutional layers 2.7.1. This is achieved by breaking the model into two sets of parameters, the convolutional filters, which are the conventionally trainable weights, and then the parameter sharing scheme. This is the non-trainable part of a general G-CNN.

By reparameterising, the weights $\mathbf{W} \in \mathbb{R}^{m \times n}$ of a feedforward network, into a filter v and a parameter sharing scheme $\mathbf{U}$;

$$\text{vec}(\mathbf{W}) = \mathbf{U}v \tag{3.10}$$

One can see this in the case of a group-specific $\mathbf{U}_g$, which represents a group convolutional layer. An example of the equivalence of this to the G-CNN scheme is for

C_2, where the network is equivariant to an inversion of the input,

$$\mathbf{U}_{\mathbf{C_2}} \cdot v = [\oplus_{g \in C_2} \pi_g] \cdot v, \tag{3.11}$$

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} v_1 \\ v_2 \\ -v_1 \\ -v_2 \end{pmatrix}. \tag{3.12}$$

To prove this is equivariant, we can show that the following holds,

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \cdot \left[\begin{pmatrix} v_1 & v_2 \\ -v_1 & -v_2 \end{pmatrix} \cdot x \right] = \begin{pmatrix} -v_1 x_1 - v_2 x_2 \\ v_1 x_1 + v_2 x_2 \end{pmatrix} = \begin{pmatrix} v_1 & v_2 \\ -v_1 & -v_2 \end{pmatrix} \cdot -x \tag{3.13}$$

From this insight, they propose a method of meta-learning an equivariant parameter sharing scheme $\mathbf{U}$, given a set of tasks $\mathcal{T}$, all possessing the same equivariance. Using an adapted MAML framework, v is trained on a per task loss L_{τ_i} and the training data and $\mathbf{U}$ is trained on the meta loss on the validation data; they demonstrate the ability to recover the equivariant parameter sharing scheme, for convolutional layers.

3.4 Other Related Works

Another recent development is that of [Rezaei-Shoshtari et al., 2022], which proves that the MDP homomorphism results of Optimal Value equivalence are also found in the continuous action case. Further, they use this result in conjunction with DDPG[Lillicrap et al., 2015] to learn a continuous action MDP Homomorphism, building a world model with a lax bisimulation metric that is similar to the one used in [Van der Pol et al., 2020]. While an interesting theoretical result, the method is just a generalisation of an MDP homomorphism and does not exploit the symmetry of the MDP.

In wider machine learning, symmetry as an inductive bias is not new, and more generally, requiring equivariance to input transformation is a key concept in mod-

ern Deep Learning. Geometric Deep Learning has recently emerged as a unified formalism for deep learning on structured data, such as graphs, sets, Groups, and others.

Geometric Deep Learning is a field that provides a theoretical commonality for many successful network architectures and has unified disparate architectures [Bronstein et al., 2021], such as Transformers and C-NNs, under a common framework. Geometric Deep learning provides a way to generalise the different inductive biases that arise from the world. For example, the translation invariance in object detection or order permutation in Graphs.

A recent success in this field is the $\mathrm{SE}(3)$-Transformer, [Fuchs et al., 2020], provides an equivaraint architecture, to $\mathrm{SE}(3)$ transformations. This ensures that global rotations and translations on graphs or point clouds respond similarly. This is an important equivariance to encode when dealing with proteins and other chemical molecules. For example, this architecture was used in the Alpha Fold protein structure prediction networks [Jumper et al., 2021].

Nother Networks, which take inspiration from Nother's theorem, demonstrate the ability to meta-learn symbolic conservation laws from real-world data [Alet et al., 2021]. An example of this is recovering the Hamiltonian of spring from real-world data. This is particularly impressive as the spring itself does not conserve energy due to friction, and so the network must learn the approximate conservation law. This is achieved by meta-learning a conservation loss across multiple tasks.

Chapter 4

Experiments

4.1 Environments

4.2 CartPole

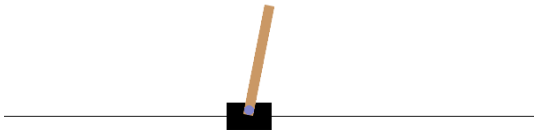

Figure 4.1: The CartPole environment.

The CartPole environment is the ”Hello World” of reinforcement learning; it describes a game where at each timestep, the aim is to balance a mass above a slider, 1 point is given for each timestep that the mass makes an angle of fewer than 12 degrees from vertical, in this instantiation. The action space$a \in 0, 1$, for the problem, is to either push the pole left or right. The state space is the position of the cart, the velocity of the cart, the angle of the pole and the angular velocity of the

pole. A state is a vector s:

$$s = \begin{pmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{pmatrix} \tag{4.1}$$

Thus, $s \in \mathbb{R}^4$.

Typically, the episodes are truncated at 500 timesteps, and the reward is 1, so the return is an integer $G_0 \in [0, 500]$. An episode terminates if the pole falls over or the cart moves too far from the centre. If a human were learning to solve this problem, one would notice that the expected result of pushing the cart left when it is leaning right at some positions is the same as pushing the cart right when it is leaning left at the displacement in the opposite direction. This is an example of symmetry in the problem, and it is this symmetry that is exploited by [van der Pol et al., 2020, Mondal et al., 2020].

To learn the transition dynamics of CartPole and agent must learn how the system evolves through time. In CartPole the transitions between different states in CartPole are governed by the PDEs:

$$\ddot{\theta} = \frac{g \sin\theta + \cos\theta \left(\frac{-F - m_p l \dot{\theta}^2 \sin(\theta)}{m_c + m_p} \right)}{l \left(\frac{4}{3} - \frac{m_p \cos^2\theta}{m_c + m_p} \right)}, \tag{4.2}$$

$$\ddot{x} = \frac{F + m_p l (\dot{\theta}^2 \sin\theta - \ddot{\theta} \cos\theta)}{m_c + m_p}. \tag{4.3}$$

Here g is the acceleration due to gravity and is positive, θ is the angle between the pole and vertical, with the pole length l. F is the action force, where the positive direction is right. The masses of the cart and the pole are m_c, m_p, respectively. Finally $\dot{}$, indicates a derivative with respect to time.

These PDEs have no closed solution and their form is taken from [Florian,] who provides slight corrections to the original dynamics in [Barto et al., 1983].

Both learning a policy and a transition model may be augmented by exploiting symmetry. The symmetry present in CartPole is the cyclic C_2 group. Other than the

[h!]

C_2	e	r
e	e	r
r	r	e

Table 4.1: The C_2 group table, where entry i, j is the result of group operation on the i^{th} and j^{th} element.

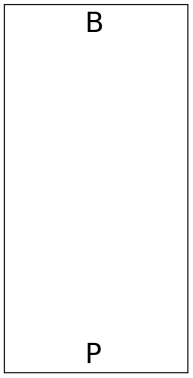

Figure 4.2: The Catch environment. The ball position is indicated by B, and the paddle position is indicated by P.

trivial group, the group with a single element, the C_2 group, is the simplest. This is a group with only two elements, the identity and inversion.

4.3 Catch

The Catch environment [Osband et al., 2020] is another simple MDP task similar to CartPole used mainly for debugging implementations. The aim of the game is to catch a falling ball. The default implementation has ten rows and 5 columns. The ball moves down by one row every time-step, and the position of the paddle is translated left or right by the action of the agent. When the ball reaches the tenth row, the reward of the system is $+1$, if the ball and the paddle are in the same location, otherwise the reward is -1. For all other time-steps the reward is 0. This produces an episodic environment, with a maximum return of 1.

The state and action spaces are both discrete, and the transitions are deterministic. These states are represented to the agent as a 5×10 array, with the ball and paddle's positions indicated by ones. The actions the agent can take are $\mathcal{A} = [0, 1, 2]$,

that translate the ball by $1 - a$. Thus, the MDP has a finite amount of transitions and state action pairs, 675.

Like CartPole, the Catch environment possesses C_2 group symmetry, however, due to the different state and action spaces the representations of the groups are different. However, because the groups are the same the structure is that in Table.**??**.

4.4 Motivation

A series of Experiments were carried out to investigate the efficacy of equivariant transition model structure in model-based RL. Where the specific model-based algorithm investigated was Dyna.

Model-based RL, is inherently more complex than actor-critic or value based methods, as not only does a policy need to be learned, but also a model of the environment dynamics must also be learned. In the case of a NN model, this will require training a model, as well as an agent.

Even with the simple models constructed here, the model based agents have two times more parameters. Within wider literature, models such as Dreamer-v3 [Hafner et al., 2023], use 8 million parameters, and multiple GPU days to learn policies on the complete B-suite environment.

Despite this increased complexity, the Dyna algorithm is constructed out of a transition model and a model-free agent. This provides a sensible progression for development and experimentation. Firstly, the model-free agent was constructed, and the symmetric inductive biase was tested only for the agent. Then, transition models were trained offline and tested. Finally these disjoint pieces were brought together to form the full Dyna implementation.

4.5 Baseline

The baseline that was chosen was a proximal policy optimization, Sec.2.3.1.5, agent from PureJaxRL, [Lu et al., 2022, Schulman et al., 2015c]. This baseline provides a training framework, that trains a single agent concurrently on multiple identical environments. The training regime is outlined below in pseudocode.

Algorithm 5 PureJaxRL PPO Agent Training Structure

```
Initialize agent: actor-critic π_θ, v_φ
Initialize replay buffer: D
for Num Updates do
    Gain experience for Num Timestep
    Store trajectories: D.append((S, A, S', R))
    Calculate GAE estimate from experience timesteps
    for Num Epochs do
        Split GAE estimates into minibatches
        Mini-Batch SGD with Adam on π_θ, v_φ ▷ See 2.3.1.5 for losses to optimize
Returns(D)
```

4.6 Equivariant Actor-Critics

4.6.1 CartPole

To form an equivariant network to the group structure of CartPole the actor network must be equivariant to both the identity and inversion operator. This report provides structures for equivariant G-CNNs for both CartPole and Catch, that can easily be extended to other environments with known discrete symmetries.

4.6.1.1 Constructing a CartPole Actor-Critic

In this section, the outline for the network design is described. In the Catch section 4.6.2, a more detailed description of how to extend the procedure to other groups is outlined. The group for CartPole contains two unique elements, in both state and action space. In state space the inversion and identity operator e, r are,

$$\ell_e^{\mathcal{S}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \ell_r^{\mathcal{S}} = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}. \tag{4.4}$$

Then the action space, the inversion and identity operator are,

$$\ell_e^{\mathcal{A}} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \ell_r^{\mathcal{A}} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}. \tag{4.5}$$

Thus, to parametrize an equivariant actor network, the network π_θ must satisfy,

$$\pi_\theta(\ell_e^{\mathcal{S}} s) = \ell_e^{\mathcal{A}} \pi_\theta(s), \tag{4.6}$$

$$\pi_\theta(\ell_r^{\mathcal{S}} s) = \ell_r^{\mathcal{A}} \pi_\theta(s). \tag{4.7}$$

Instead of using a G-CNN for the Actor-Critic, a simpler solution involves employing a network with only odd operations, such as the tanh activation, and excluding biases for all hidden representations. By definition, odd functions are equivariant to both inversion and identity transformations, ensuring the network's equivariance. However, this doesn't address the issue of equivariance in the action space. To bridge this gap, a group convolution layer is incorporated to map between representations.

The network can be considered as a composition of $f_\theta : \mathcal{S} \to \mathbb{R}^{|H|}$, an odd embedding MLP and $gc_\theta : \mathbb{R}^{|H|} \to \mathbb{R}^{\mathcal{A}}$ a group convolution layer that "lifts" the equivariance to the action space. As such the parametric policy is,

$$\pi_\theta(s) = gc_\theta(f_\theta(s)). \tag{4.8}$$

The equivariance properties of the sub-networks with respect to the inversion operator are,

$$f_\theta(-s) = -f_\theta(s), \tag{4.9}$$

$$gc_\theta(-x) = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} gc_\theta(x), \tag{4.10}$$

Where $gc_\theta(x) = [P(A = a_0), P(A = a_1)]$ describes the distribution over the binary actions of CartPole.

This G-CNN network for policy learning is the first novel contribution of this report. In comparison to the work of [Mondal et al., 2020], the policy learning G-CNN is fully equivariant, rather than learning action values from an equivariant embedding. Additionally, the network parametrizes a policy directly rather than Q values. Due to the network's end to end equivariance in comparison to the Q-value

network proposed by [Mondal et al., 2020], agents parametrized by this network must take the same actions in states that are in the same orbit, which is not the case for the Q-value network, which only has an equivariant embedding. Thus, they do not guarantee equivariance in the policy.

Further, this network architecture has advantages over Symmetrizer networks, [van der Pol et al., 2020], in that it does not require a large matrix inversions to solve for the parameters of the network, while still maintaining the same equivariance qualities.

4.6.1.2 Results: Training Dynamics On the CartPole Benchmark

With the network's structure established, the benchmark task focuses on mastering an expert policy within the CartPole environment. By default, CartPole imposes a maximum episode length of 500 interactions. All non-terminal states yield a reward of +1, setting the maximum episodic return at 500. As with most traditional RL problems, the primary objective in CartPole is to optimize the agent's episodic return. Finding an expert policy in CartPole using standard deep learning methods is relatively straightforward and primarily serves as an implementation benchmark. For an equivariant network structure to truly enhance the quality of the learned policy, it should strive to approach the 500 episodic return benchmark with fewer environment interactions.

When comparing the learning dynamics of policy agents, it's crucial to ensure not just that the agent achieves expertise in the task but also that the policy learning procedure remains stable across multiple random seeds. A random seed refers to the initial state in which both the agent and environment begin. Keeping training stability in mind, examining the performance under the least favourable random seeds is informative about the training robustness. If an algorithm is particularly sensitive to its initialization, its performance may be significantly impacted, and may not converge over multiple random seeds [Henderson et al., 2018].

For all experiments, we utilize a standard MLP for the critic without imposing any equivariance constraints. While this setup might not yield optimal performance, it's essential to highlight that training is conducted across 128 random seeds to

ensure the stability of the agent's learning.

Due to constraints on the equivariant network structures, it's not always feasible to maintain an identical number of parameters across networks. In instances where the exact parameter count differs, we ensure that the depth of all networks remains constant. We then adjust the width to achieve a parameter count that's within 10% of the MLP baseline.

Refer to Fig.4.3 below, where the mean episodic return of three agents with distinct network architectures is illustrated. Despite the differences in their structures, all networks share the same training hyperparameters, as provided in the PureJaxRL[Lu et al., 2022] baselines. The three networks depicted are the MLP baseline from PureJaxRL, an implementation of the Symmetrizer network from [van der Pol et al., 2020], and the G-CNN policy network introduced in this report.

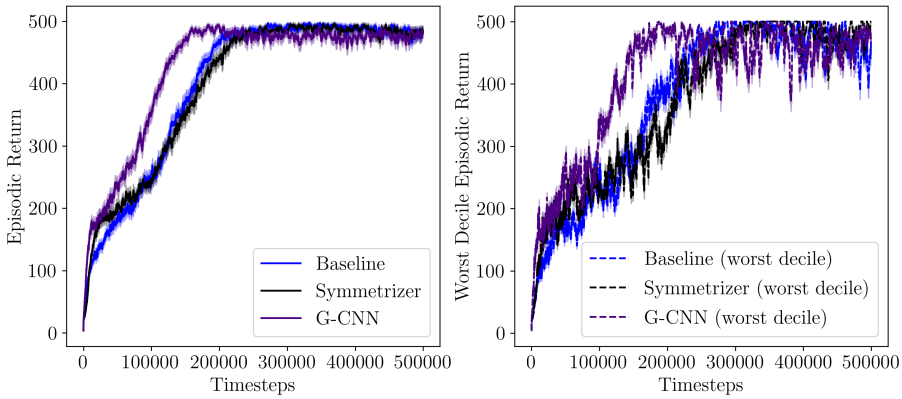

Figure 4.3: Left: Mean episodic returns for the CartPole agents across 128 random seeds plotted against number of experience time-steps in the MDP. Right: The mean cumulative episodic returns of the worst performing 128 random seeds against number of experience time-steps in the MDP. Both of the plots are moving averages, with windows of 10 time-steps. Additionally, all plots have two standard errors plotted.

Both the Symmetrizer and the G-CNN are equivariant to the actions of the C_2 group. This equivariance constraint requires a learned policy that respects the inversion symmetry present in Cart-pole. The equivariance should improve the sample efficiency of the agent as any learning from one state additionally informs the agent about the agent about the policy for the other state in the orbit. This hypothesis,

Time-steps	Baseline	Symmetrizer	G-CNN
$10,000$	102 ± 5	116 ± 6	$\mathbf{158 \pm 8}$
$100,000$	260 ± 10	240 ± 10	$\mathbf{330 \pm 10}$
$500,000$	497 ± 1	$\mathbf{500 \pm 1}$	499 ± 1

Table 4.2: Cumulative episodic returns tabulated for the three network architectures. All episodic returns are recorded with confidence intervals of two standard errors across 128 random seeds.

is supported somewhat by the observed training dynamics. Over the first period of training, both the Symmetrizer and the G-CNN, outperform the baseline. The Symmetrizer, does not maintain this performance advantage. Our implementation uses the same network size, as the original paper and the same network hyperparameters. Despite this, the Symmetrizer agent fails to learn an expert policy in fewer steps than that of the baseline.

It should be noted that here the mean plus minus two standard deviations is plotted in comparison to the median and upper and lower quartiles of cumulative returns, which is plotted in [van der Pol et al., 2020]. The performance of the Symmetrizer, is underwhelming despite the implemented network being checked for equivariance. Further tuning of the hyperparameters may yield performance that improves upon the baseline's returns. However, this was not a primary concern in this report.

The G-CNN does compare favourably with the baseline implementation of an MLP, having slightly fewer parameters. It can be seen that it both converges on average to an expert policy in fewer time-steps but also has a more favourable convergence behaviour in challenging initialization conditions. This can be seen in the right of Figure 4.3 where, the bottom tenth percentile of cumulative returns, still converges notably faster than that of the baseline.

Additionally, the mean episodic returns across all random seeds are tabulated at $10,000$, $100,000$, and $500,000$ time-steps in Table 4.2. Upon closer inspection, the G-CNN agent incorporating the equivariant inductive bias significantly outperforms the baseline.

While both models have similar parameter counts, the structure of the G-CNNs

makes their forward passes more computationally intensive. This is due to the G-CNN architecture requiring twice as many operations in a forward pass, attributed to the two group actions. However, in a scenario like Cart-Pole, where the networks are relatively small and inexpensive to evaluate—and where computation can be efficiently parallelized—both models can train 128 random seeds in under a minute. This speed is achieved using the hyperparameters listed in the Appendix and executed on a RTX 3090.

Encouraged by the promising results from the initial experiment, we decided to explore a new environment to determine whether the equivariance constraint could further enhance performance.

4.6.2 Catch

To demonstrate that G-CNN actor-critic can be extended to other environments with different group actions, we implemented the equivariant actor-critic for the Bsuite Catch environment [Osband et al., 2020]. For the Catch environment, constructing an equivariant network is challenging.

4.6.2.1 Constructing a Catch Actor-Critic

Unlike the CartPole case where layers can be made equivariant to the representation, in Catch, the entire network must be built using Group Convolutions. As in previous scenarios, we impose an equivariant constraint on the actor $\pi_\theta(s)$. In the context of Catch, the input state space is represented as $\mathcal{S} \in [0,1]^{50}$. Instead of detailing the cumbersome $\ell_r^{\mathcal{S}} \in \mathcal{R}^{50\times 50}$ matrix representation of the reflection group action $\ell_r^{\mathcal{S}}$, it is left symbolically. This action modifies the x-coordinate of both the ball and the paddle using the transformation $r(x) = -(2-x)$. Thus, the equivariance constraint is expressed as:

$$\pi_\theta(\ell_r^{\mathcal{S}} s) = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \pi_\theta(s). \tag{4.11}$$

To construct a G-CNN, which is equivariant to the group actions a group action in the hidden layers must be determined. Consider a group convolution input layer

$f : \mathcal{S} \to \mathcal{H} \in \mathbb{R}^{|G|\times|H|}$, commonly referred to as a lifting layer that maps from a state space to a hidden space of H. Lifting layers apply a group action to the input and then passes the transformed input through, $g : \mathcal{S} \to \mathbb{R}^{|H|}$, a dense/convolution layer. In the full layer the input is transformed by every group action, and then passed through g. In the absence of any pooling, this gives a hidden representation, that is the hidden dimensions of the equivalent dense/convolution layer, plus a new axis that is the size of the group. To illustrate the new equivariance constraint a single layer;

$$g(s) = \vec{h} \in \mathbb{R}^{|H|}. \tag{4.12}$$

When formed into a group convolution,

$$f(s) = \begin{pmatrix} g(s) \\ g(\ell_1^{\mathcal{S}} s) \\ g(\ell_2^{\mathcal{S}} s) \\ \vdots \\ g(\ell_{|G|} s). \end{pmatrix} \tag{4.13}$$

Once a group action is applied to the input, the output values undergo permutation. While the exact permutation is contingent on the group, the permutations' structure can be readily determined due to group closure:

$$f(\ell_n^{\mathcal{S}} s) = \begin{pmatrix} g(\ell_n^{\mathcal{S}} s) \\ g(\ell_n^{\mathcal{S}} \ell_1^{\mathcal{S}} s) \\ g(\ell_n^{\mathcal{S}} \ell_{|G|} s) \\ \vdots \\ g(\ell_n^{\mathcal{S}} \ell_{|G|} s) \end{pmatrix} = \begin{pmatrix} g(\ell_n^{\mathcal{S}} s) \\ g(\ell_i^{\mathcal{S}} s) \\ g(\ell_j^{\mathcal{S}} s) \\ \vdots \\ g(\ell_k s). \end{pmatrix} = \mathbf{P}_n f(s) \tag{4.14}$$

Where $\mathbf{P}_n$, is a permutation matrix defined by the group. There is a unique permutation matrix for each group element. This permutation relation enables one to construct further equivariant layers. Consider a subsequent, $h : \mathcal{H} \to \mathcal{H}'$, a hidden layer that must continue the equivariance to group G. This is achieved by treating

each $\mathbf{P}$ as the group action, and ensuring that the output is equivariant to its application. The new layer can be through of as taking a vector of responses, where it must be equivariant to the vectors' permutation,

$$h\left(\mathbf{P_i}\begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_{|G|} \end{pmatrix}\right) = \mathbf{P_i} h \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_{|G|} \end{pmatrix}. \tag{4.15}$$

Tying this to a concrete example in a 1D convolution, if there is an input that is one hot, and the kernel has a single weight, w_1. The output of the layer will be;

$$\text{Conv1D}\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ w_1 \\ 0 \end{pmatrix} \tag{4.16}$$

If the input is translated the output is also translated;

$$\text{Conv1D}\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} w_1 \\ 0 \\ 0 \end{pmatrix} \tag{4.17}$$

Which is the definition of the equivariant constraint, slightly more abstractly[1];

$$\text{Conv1D}(x) = \begin{pmatrix} g(\ell_1 x) \\ g(x) \\ g(\ell_{-1} x) \end{pmatrix} \tag{4.18}$$

[1]If you are slightly shocked by the order of the group actions acting on x here, dont be. It comes from the relationship between the right and left action on the kernel see [Cohen and Welling, 2016]

Then when the input is translated up by one ℓ_{-1};

$$\text{Conv1D}(\ell_{-1}x) = \begin{pmatrix} g(\ell_1\ell_{-1}x) \\ g(\ell_{-1}x) \\ g(\ell_{-1}\ell_{-1}x) \end{pmatrix} \tag{4.19}$$

And the closure of the group defines its permutation. The group table for this translation group is given below. By using this Eq.4.19 becomes;

G	-1	0	1
-1	1	-1	0
0	-1	0	1
1	0	1	-1

Table 4.3: Example translation group

$$\text{Conv1D}(\ell_{-1}x) = \begin{pmatrix} g(x) \\ g(\ell_{-1}x) \\ g(\ell_1 x) \end{pmatrix} \tag{4.20}$$

Which is a permutation of the input governed by the group structure of Eq.4.14.

Given this equivariant structure for each layer, when constructing a network out of an input layer and subsequent hidden layers that all adhere to the equivariance constraint, what remains is to identify an appropriate output. In scenarios such as an actor network operating within a discrete action space, the permutation representation proves ideal. For instance, in the "Catch" environment, the probability of moving left in state s should mirror the probability of moving right in its reflected state $s' = \ell_r^S$. Since the network produces logits—un-normalized probabilities—their permutation upon reflection possesses the desired properties. However, when a permutation doesn't fit the necessary group action, establishing equivariance becomes more intricate. We'll delve into this challenge in subsequent discussions.

With the framework of an equivariant network in place—similar to the Cart-Pole scenario—we established both an equivariant actor-critic and an MLP actor-critic. These were designed with two hidden layers and an approximate equivalent

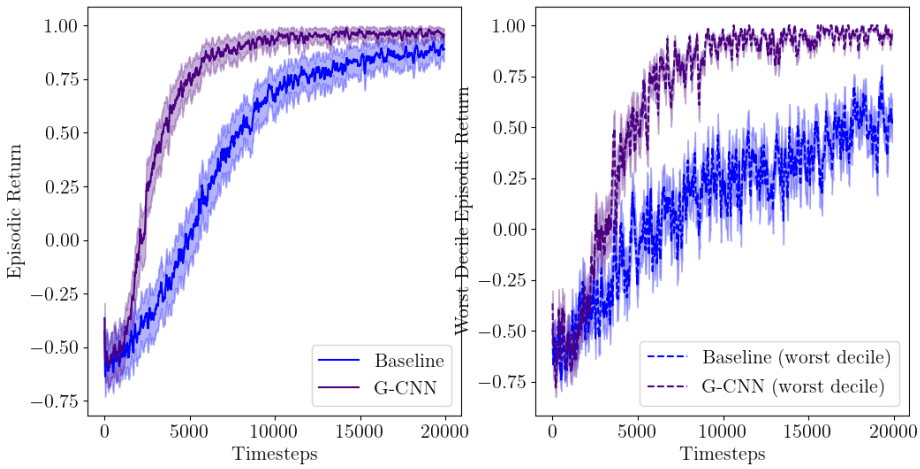

Figure 4.4: Left: Mean episodic returns for the Catch agents across 128 random seeds plotted against number of interaction time-steps in the MDP. Right: The mean cumulative episodic returns of the worst performing 128 random seeds against number of experience time-steps in the MDP. Both of the plots are moving averages, with windows of 10 time-steps. Additionally, two standard errors are plotted.

in parameter count. For this experiment, the Symmetrizer was omitted due to the difficulties encountered when trying to optimize its performance, as depicted in the primary study. Given the small state-action space of the Catch task relative to Cart-Pole, agents were allotted 20,000 MDP interactions for learning. The MLP baseline once again employed the PureJaxRL baseline actor-critic, albeit with minor modifications for adaptation to the new environment. Both network architectures were assigned identical hyperparameters, Appendix.5.2.

4.6.2.2 Results: Training Dynamics On the Catch Benchmark

Upon examining the qualitative differences in the episodic return curves, we observe a familiar yet more pronounced trend. For Catch, the inductive bias of equivariance seems especially beneficial. Not only does the agent with the G-CNN converge to an expert policy more rapidly than its counterpart, but the bottom decile of random seeds also exhibits markedly improved performance. Furthermore, by referring to the tabulated results in Table 4.4, the superiority of the Equivariant Network architecture is evident. It manages to achieve a higher proficiency in ”Catch” in merely half the time compared to the alternative.

Time-steps	Baseline	G-CNN
$2,000$	-0.51 ± 0.07	$\mathbf{-0.06 \pm 0.09}$
$10,000$	0.70 ± 0.06	$\mathbf{0.95 \pm 0.03}$
$20,000$	0.875 ± 0.04	$\mathbf{0.96 \pm 0.02}$

Table 4.4: Cumulative episodic returns tabulated for the two network architectures. All episodic returns are recorded with confidence intervals of two standard errors across 128 random seeds.

4.6.3 Conclusion

Equivariant actors were tested in both environments, and showed a substantially more efficient learning. These results are inline with other experiments, where agents are trained with inductive biases about the task's symmetry outperform conventional methods. Further, they also demonstrated the same improvement in cases with challenging initialization conditions.

The equivariance constraint itself however, is quite limiting. For any given task the group must be known beforehand. Not only this but it must also be an exact symmetry. In many settings these constraints are rare, especially with discrete symmetry groups.

Additionally, there is a forward pass cost to the equivariant network structure, especially with larger groups, despite the same number of parameters being used. With the way G-CNNs are constructed, the number of operations in a forward pass of the network is $\mathcal{O}(|G|)$. As such, the inference time increases when including the inductive bias. However, because these operations can be executed in parallel, the evaluation time increase is limited. For a rough estimation of the increased overhead the G-CNN. The implementations of the G-CNN for catch and the MLP were benchmarked. The MLP's mean forward pass over 10000 states was $27.5 \pm 0.1\mu s$, and $29.9 \pm 0.1\mu s$ for the G-CNN. This is an 8% increase in evaluation time, with similar standard error across 1000 tests. An increased inference overhead is also not a problem in multiple environments such as robotic control where samples are more expensive than inference.

Despite the mild performance overhead, if a task is known to poses a discrete group symmetry, constructing a network with an inductive bias that accounts for

this is demonstrably useful in increasing the training efficiency and reliability of an Agent, and should be exploited. However, the case of an exact symmetry is not universal.

The limited applicability of equivariant models motivates the next section of the report. In the case where the environment has a group structured symmetry or an approximate group structured symmetry, building an equivariant transition model may further improve the agent's learning dynamics.

4.7 Model-Based RL

In the previous section, Actor-Critic methods were implemented on both the Cart-Pole and Catch environments. The inclusion of an inductive bias exploiting the symmetry of an MDP led to improvements in the sample efficiency of learning an expert policy, while maintaining robustness to initialization conditions. In the case where the transition dynamics of the MDP are group-structured, or approximately group-structured, this report proposes the use of an equivariant world model. With a world model that has inductive biases designed to aid in learning the MDP's transition dynamics, the world model may be learned more efficiently and accurately from fewer samples from the environment, improving the effectiveness of planning for the agent.

The specific algorithm implemented to create a model-based agent is a Dyna, Alg.2, with a PPO agent ,Sec.2.3.1.5. Using the Dyna-PPO agent means that when the number of planning steps is set to zero, the agent becomes a model-free PPO agent, enabling direct comparison with the baseline MLP PPO implementation, found in Sec.4.5.

Both Catch and CartPole are environments where the reward function is only a function of the state the agent is in, and not a state-action pair. Further, because their rewards are simple functions of their state, this report forgoes learning the reward dynamics of the environments and uses their existing reward structure on simulated states, such that:

$$R_{t+1} = R(T(s, a)). \tag{4.21}$$

Where, R, is the reward function defined by the environment.

4.7.1 Constructing Transition Models

Before using a full model-free algorithm, where the model is learned online, during the training. A simplified algorithm, Supervised-Dyna, is tested that uses offline trained world models. The process for this is outlined below.

Using offline-trained world models has benefits as an intermediate step. Firstly, the world model is stationary throughout the agent training process and can be eas-

Algorithm 6 Supervised-Dyna

Initialize T_ϕ
Sample transition tuples from a policy $\pi \sim (s, a, s')$ on MDP $\mathcal{M}$. ▷ *Here π is a random/ expert policy*
Form Dataset $\mathcal{D}$ from sampled transition tuples.
for Num Epochs **do**
    $\phi' \leftarrow$Minimise $L(\phi, \mathcal{D})$.
for Num Dyna Iterations **do**
    for Num Acting Updates **do**
        Sample transition tuples from policy $\pi_\theta \sim (s, a, s')$ on $\mathcal{M}$
        Train agent π_θ from direct samples.
    for Num Planning Updates **do**
        Sample transition tuples from policy $\pi_\theta \sim (s, a, s')$ on T_ϕ.
        Train agent π_θ from planned samples.

ily investigated. Secondly, the model can be trained with ample data. This is in contrast to full Dyna where the length of the planning and acting phases must be tuned as a hyperparameter, and the convergence of the models must be tuned. This hyperparameter of Num Acting Updates/Num Planning Updates, is referred to as the planning ratio.

4.7.2 Proximal Pooling Motivation

To produce equivariant world models, there are additional challenges beyond just implementing a G-CNN, which permutes its outputs depending on the input transformation. In contrast to an equivariant actor, where simply permuting the logits of a distribution suffices for equivariant behaviour, for the derivation refer to Sec.3.1.1. When building an equivariant world model with a G-CNN means the output of the network predicts not only the next state but also a number of nonsense next states.

To illustrate what the final layer output in the G-CNN is doing, consider the same setup as in the actor-critic networks, where there is a hidden representation whose responses get permuted depending on the application of a group-structured transformation on the input:

$$f(s,a) = \begin{pmatrix} g(s,a) \\ g(\ell_1^{\mathcal{S}} s, \ell_1^{\mathcal{A}} a) \\ g(\ell_2^{\mathcal{S}} s, \ell_2^{\mathcal{A}} a) \\ \vdots \\ g(\ell_{|G|}^{\mathcal{S}} s, \ell_{|G|}^{\mathcal{A}} a), \end{pmatrix} \tag{4.22}$$

and when a transformation is applied to the input these responses, g are permuted,

$$f(\ell_i^{\mathcal{S}} s, \ell_i^{\mathcal{A}} a) = \mathbf{P}_i f(s,a). \tag{4.23}$$

Because the permutations are defined by the group, these are the same permutations as in Eq.4.14. However, here each response $g : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$, produces a transition image. In order to obtain a transition function there must be a method to pool over the responses.

4.7.3 Group Action Transformation

When the network f performs a forward pass and outputs the vector of responses, as described by Eq.4.23, the equivariance in the response maps group actions on a state space to a permutation of responses that all have the same shape as the state. This is not the equivariance constraint that is required for a transition model, which is:

$$T(\ell_i^{\mathcal{S}} s, \ell_i^{\mathcal{A}} a) = \ell_i^{\mathcal{S}} T(s,a). \tag{4.24}$$

The first step in achieving this is to apply a group action to each row of the response of f. This is the action transformation operation, denoted as $\mathcal{T}$, where,

$$\mathcal{T} f(s,a) = \begin{pmatrix} g(s,a) \\ \ell_{-1}^{\mathcal{S}} g(\ell_1^{\mathcal{S}} s, \ell_1^{\mathcal{A}} a) \\ \ell_{-2}^{\mathcal{S}} g(\ell_2^{\mathcal{S}} s, \ell_2^{\mathcal{A}} a) \\ \vdots \\ \ell_{-|G|}^{\mathcal{S}} g(\ell_{|G|}^{\mathcal{S}} s, \ell_{|G|}^{\mathcal{A}} a) \end{pmatrix}. \tag{4.25}$$

Due to group closure, this gets us quite close to equivariance! For example consider some group element, which is the inverse of i acting on the input,

$$\mathcal{T}f(\ell^{\mathcal{S}}_{-i}s, \ell^{\mathcal{A}}_{-i}a) = \begin{pmatrix} g(\ell^{\mathcal{S}}_{-i}s, \ell^{\mathcal{A}}_{-i}a) \\ \ell^{\mathcal{S}}_{-1}g(\ell^{\mathcal{S}}_{j}s, \ell^{\mathcal{A}}_{j}a) \\ \ell^{\mathcal{S}}_{-2}g(\ell^{\mathcal{S}}_{k}s, \ell^{\mathcal{A}}_{k}a) \\ \vdots \\ \ell^{\mathcal{S}}_{-i}g(s, a) \\ \vdots \\ \ell^{\mathcal{S}}_{-|G|}g(\ell^{\mathcal{S}}_{|G|}s, \ell^{\mathcal{A}}_{|G|}a) \end{pmatrix}. \tag{4.26}$$

Then, to achieve equivariance, the correct row must be selected. This would be straightforward if the group action on the input were known, but in general, it is not. However, in many cases, in a transition model, the output is closer to the input than to any of the other states. With this proximity intuition, a pooling method is introduced to gain approximate equivariance with the transition models. This vector of next states is referred to as transition images. When using CNNs or other G-CNNs for classification or other image processing tasks, a reduction pooling method such as mean or max is used over the different groups. Both of these pooling types result in an invariant network. Specifically, if one pools over the group, taking either a mean or a max over the group dimension of f, the output remains unchanged when the group dimension is permuted, because the transformation applies only a permutation along this axis

$$\max_{\text{axis G}} \mathbf{P}_i f(s, a) = \max_{\text{axis G}} \mathbf{P}_j f(s, a), \forall i, j. \tag{4.27}$$

Here the group axis, axis G, is the leading dimension in f if $f : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^{|G| \times \mathcal{S}}$. This invariance is not helpful in our case and so an alternative technique of pooling over possible next states is proposed.

4.7.4 Proximal Pooling Layers

As a caveat, this method does not guarantee equivariance to the input transformation. However, empirically, it is effective. The proximal pooling layer, $\mathcal{P}$, takes a distance metric, $d(s, s')$, and selects the transition image, which has the minimum distance metric as such the full approximately equivariant network is;

$$T(s,a) = \mathcal{P}\mathcal{T}f(s,a) = \arg\min_{\text{axis G}} d\left(s, \begin{pmatrix} g(s,a) \\ \ell^{\mathcal{S}}_{-1} g(\ell^{\mathcal{S}}_{j} s, \ell^{\mathcal{A}}_{j} a) \\ \ell^{\mathcal{S}}_{-2} g(\ell^{\mathcal{S}}_{k} s, \ell^{\mathcal{A}}_{k} a) \\ \vdots \\ \ell^{\mathcal{S}}_{-|G|} g(\ell^{\mathcal{S}}_{|G|} s, \ell^{\mathcal{A}}_{|G|} a) \end{pmatrix}\right). \tag{4.28}$$

The $\arg\min_{\text{axis G}}$, returns the state image, which is closest to the original state, if the network predictions are better than random with respect to the distance metric.

Empirically, across 10,000 randomly sampled CartPole states, the network, $T(s,a) = \mathcal{P}\mathcal{T}f(s,a)$, is fully equivariant, when untrained. When trained, it is equivariant across the same sample, and also achieves better validation MSE than the non-equivariant model, when trained on transitions from the same data.

This is not the case for the Catch network. Catch has 675 state-action pairs, and the transition model when untrained achieves $98.3 \pm 0.6\%$ equivariance across all these state action pairs. The standard error is quoted over 1,000 random seed parameter initializations. The performance of the transition models is investigated in more depth in Sec.**??**.

When trained, the Catch model is able to achieve a validation accuracy of 1. As such, the lack of perfect equivariance may not be an issue in some scenarios. Small, discrete spaces with a high degree of symmetry are the worst in terms of a lack of equivariant predictions.

When there are few possible states, the probability that a random state is closer to the prediction increases. In the limit where there are two states, a distance metric between the two is either $0, d$ and as such a random prediction is the same distance or closer half of the time. Additionally, for MDPs where no sensible distance metric

can be constructed, this method will fail.

Proposition 1 *Given some distance metric between two states $d(s, s')$ and a G-CNN transition model of the form Eq.4.26. If at some time t a state transition from s_t to s_{t+1} takes place. And the MDP has a group structured discrete symmetry, if the state space obeys,*

$$d(s_t, s_{t+1}) < \mathbb{E}_{s' \sim \mu}[d(s_t, s')], \forall i, s_t. \tag{4.29}$$

Where μ is the steady state distribution. Then, a proximal pooling layer can be used to make the G-CNNs' forward pass approximately equivariant.

As motivation for why this occurs, when the G-CNN performs a forward pass, one of the multiple outputs of $\mathcal{T} f(\ell_i^{\mathcal{S}} s, \ell_i^{\mathcal{A}} a)$, is empirically non correlated to the other transition images, for all parts of the state affected by the transformations. However, on of the outputs is closely related to the input, due to the proximity of the previous state to the new state. As such, we can select the closest state on average, as the closest state to the input is expected to be the correct transition image. Empirically, one can see the lack of correlation in Fig.4.5, which plots the r^2 co-efficient between the transition images across 1,000 randomly initialized untrained networks. The network parameterizes the next states as two distributions, by outputting an array of 50 logits. As expected, the network responses along the centre axis should be invariant to the input transformation. Thus, a positive correlation coefficient is encouraging. For all other logits in the distribution over the next state there is no evidence of a correlation between the two transition images, at a 5% significance level.

With the two outputs being independent of each other, the correct next state can be selected using the proximal layer reliably, and the networks can be approximately equivariant.

4.7.5 Transition Models: CartPole

A CartPole transition model is a functional approximation to a deterministic, non-linear system. To learn this transition model, transitions are sampled from a policy

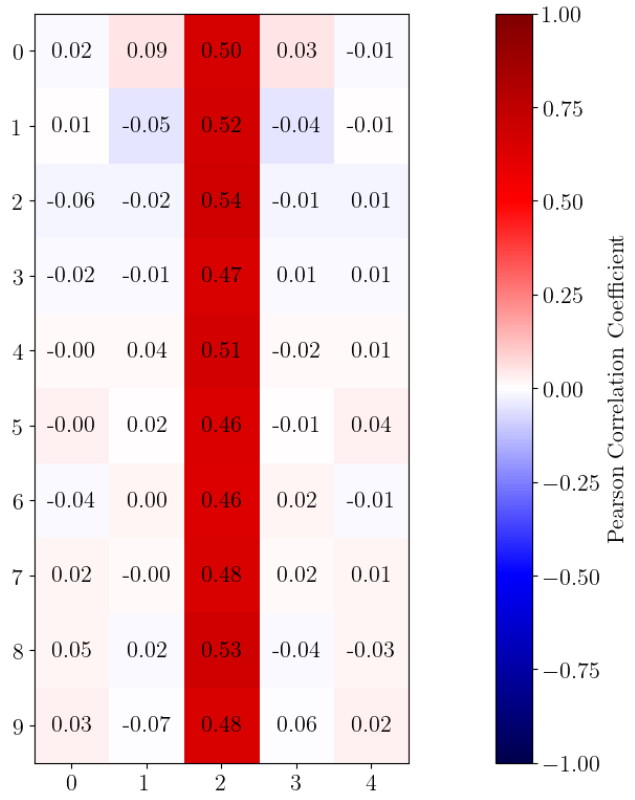

Figure 4.5: Pearson r^2 value for each logit in the output of the transition model between $g(s, a)$ and $\ell_r^{\mathcal{S}} g(\ell_r^{\mathcal{S}} s, \ell_r^{\mathcal{A}} a)$, for one of the 675 possible state action pairs, across 1000 random seeds for network initialization.

on the MDP and stored. This is then used as a dataset to perform supervised learning on. The Transition model, $T_\phi : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$, predicts next states from a state action-pair.

The loss function for the transition model is the average L2 distance between the predicted next state and the true next state across a batch of samples.

$$L(\phi) = \frac{1}{N} \sum_{(s,a,s')_i \sim \tau}^{N} (1 - \delta(\text{done}))||T(s, a) - s'||_2 \tag{4.30}$$

Here $\tau = \{(s, a, s')_1^N\}$ is a batch of transitions, of size N, sampled from the MDP, and (s, a, s'), is the state, action, next state tuple. Often, a reward model is required to also simulate an MDP. For simplicity, the CartPole reward model is used, rather than learned. The one subtlety is that the loss is only back-propagated if s, is not a terminal state. The indicator function $\delta(\text{done}) = 1$if s is terminal. Thus, the model

only learns transitions governed by the environment dynamics and not how to reset the episode.

4.7.5.1 Constructing Equivariant World Models

Like in the previous section, Sec.4.6, the equivariant transition models are deep G-CNNs. In contrast to equivariant actors in the previous section, these networks have to use the proximal pooling layer on their output. The equivariance described by these networks is to the identity and negative operation, such that;

$$T(-s, 2 * a - 1) = -T(s, a), \forall \tag{4.31}$$

These transition models are not truly equivariant but empirically, the proximal pooling layer suffices to meet the equivariance condition. As a distance metric to achieve the equivariance, the proximal pool uses the $L1$ distance.

Similarly to the actor networks, the transition models have two hidden group convolution layers. The input layer for actions also transforms the action from $[0, 1]$ to $[-1, 1]$. Thus, the same group convolution layer can be used for both the state and the action input.

4.7.5.2 Results: Comparing Convergence of Transition Models

In order to perform model based RL, a transition model must be learnt to simulate transitions, such that the agent can learn a policy that performs better in the original environment. Thus, the first set of experiments was supervised training of transition models.

Three different pairs of models were trained. The first pair was trained on data collected from an expert and random policy(joint expert-random), where half of the data is sampled from an expert policy, and half from a random policy. The second pair took the first dataset and filtered out the transitions that took the right action (left-hand). The final pair was trained only on data sampled from a random policy (random).

In Fig.4.6 it is clear that the G-CNN, despite having the same parameter count, performs better on all three transition datasets. On the joint expert-random dataset,

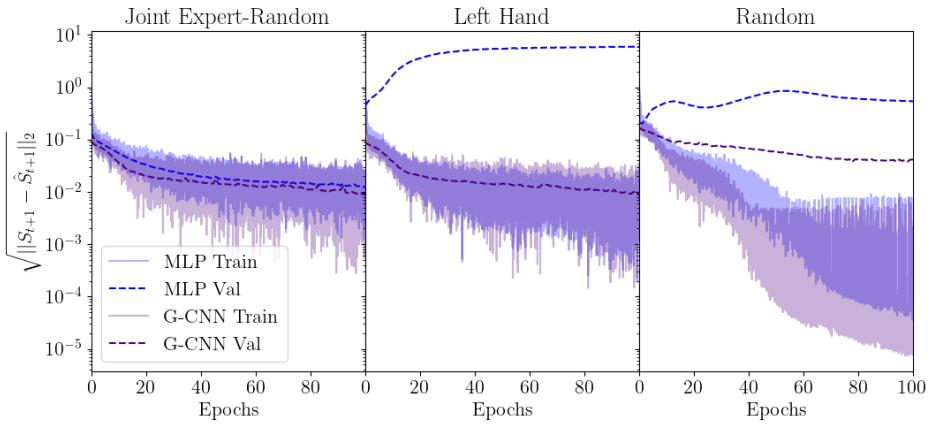

Figure 4.6: Transition Model RMS error, plotted against epochs for three different training datasets. Left, the joint expert-random dataset. Center, the same dataset as left, however, with only the left action taken. Right, solely transitions sampled from a random policy. These datasets contain 40,000 transitions. All plots have the same set of validation data with a 50/50 split of expert and random policy sampled data.

the final delta in validation root-mean-square (RMS) is ≈ 0.003.

Further, when the right action is filtered out, there was a reassuringly stark delta in performance between the equivariant model and the conventional MLP transition model. As expected, the equivariant model recovered the majority of the performance in this scenario, recovering slightly better RMS error, 0.0093 and 0.0091 for the joint expert-random and left-hand datasets respectively. This may not be a significant delta. The MLP, on the other hand, appears to overfit to left actions heavily. Then when the MLP is asked to generalize to transitions that it has not seen it fails.

Interestingly, when only random data is sampled, the equivariant model has lower validation loss. To further investigate these trends, the final transition model validation loss is plotted by angle for each of the datasets in Fig.4.7.

In the figure on the left, the RMS error is substantially lower for the equivariant model, for $\theta < |5^o|$. This indicates that the equivariant constraints and the structure of the model improves the accuracy of the model in comparison to the MLP, especially in the region where CartPole's dynamics are linear due to the limited moments applied by the pole. This trend is seen across all datasets. For all transition models there is a sharp increase in RMS error around 8^o. The RMS error then falls as the pole deviates further from horizontal. Currently, this is a topic for further

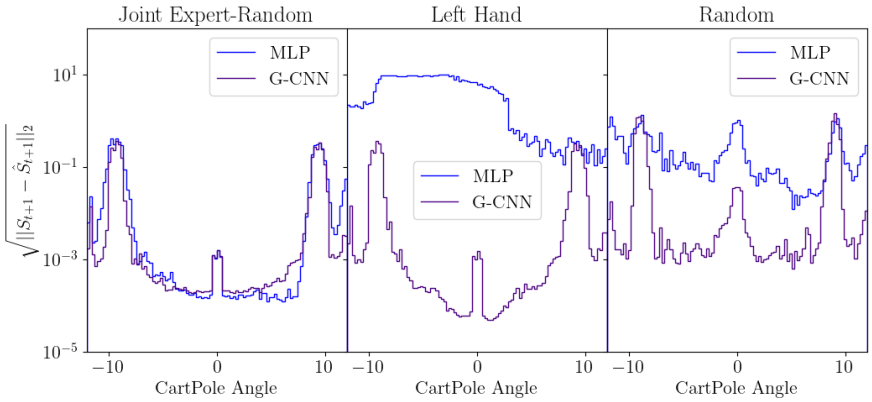

Figure 4.7: RMS Error plotted by angle on validation transitions sampled from a 50/50 split of expert and random data. Left, models trained on a dataset of 50/50 sampled expert and random policy transitions. Center, only transitions taking the left action. Right, transitions only sampled from a random policy.

investigation.

The Centre plot in Fig.4.7, is as expected. The RMS errors were similar for the equivariant G-CNN, and the MLP performs slightly better on a pole leaning to the right.

In the last plot, where the model is trained solely on transitions sampled from random actions, a striking result is obtained. The equivariant model performs notably better across all angels, with an RMS validation error of 0.039, in comparison of that to the MLP of 0.19. On validation data that is sampled jointly from an expert policy, where the majority of transitions are at small angles.

This superior generalization is demonstrated when the model is trained on random data but validated on join expert random data, which contains far more transitions at small angles.

This property may be important in training Dyna agents. Where initially the agents' policies are suboptimal and may tend to be worse at keeping the pole upright. However, the agents' learning in the simulated environments may be augmented by the improved generalization to small angels. As the policies improve from simulated experience, they tend towards having more transitions at small angels.[2]

[2]The transition distributions can be found in the Appendix.5.1.

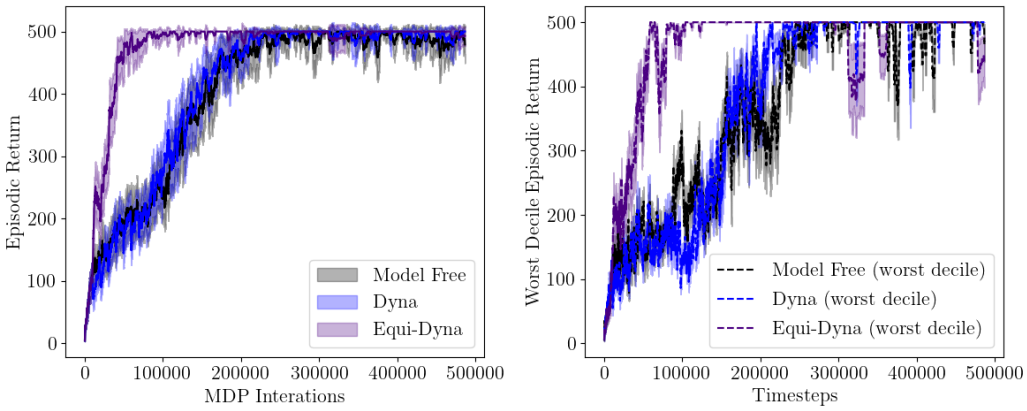

Figure 4.8: Left: Mean episodic returns for the CartPole agents across 128 random seeds plotted against number of interaction time-steps in the MDP. Right: The mean cumulative episodic returns of the worst performing 128 random seeds against number of experience time-steps in the MDP. Both of the plots are moving averages, with windows of 10 time-steps. Additionally, two standard errors are plotted. The Equi-Dyna model uses a planning ratio of 8 and the Dyna model uses a planning ratio of 1.

Time-steps	Baseline	Dyna	Equi-Dyna
$10,000$	102 ± 5	110 ± 10	$\mathbf{120 \pm 10}$
$100,000$	260 ± 10	245 ± 20	$\mathbf{498 \pm 2}$
$500,000$	497 ± 1	$\mathbf{500 \pm 1}$	498 ± 13

Table 4.5: Cumulative episodic returns tabulated for the three network architectures. All episodic returns are recorded with confidence intervals of two standard errors across 128 random seeds.

4.7.5.3 Results: Supervised-Dyna

Using the off-line trained world models above on the joint expert-random data. Both actor-critic agent were trained using the Supervised-Ayna algorithm, Alg.6. With only equivariant transition models, and conventional MLP actor-critics. The model-based agents have 50 Dyna iterations, and the full hyperparameter set can be found in Appendix.5.2. Further, a hyperparameter sweep was performed over planning ratio and the best planning ratios were chosen. The results are displayed in Fig.4.8. Where the Equi-Dyna model was found to perform best with a high planning ratio. The small difference in validation loss between the transition models appears to have a large impact on the quality of the planning available to the agents, with the Dyna model only providing a small uplift over a model-free baseline.

This trend, when investigated at the $20\%, 40\%$, and 100% completion in Table.4.5 is clear. Particularly impressively, at 40% of the sampled experience the Equi-Dyna model is able to achieve an expert policy on average. With further training, it seems that the policy diverges further. This is the most sample efficient model yet, and suggests that with a sufficiently good world model, the agent may be able to achieve strong generalization performance.

4.7.6 Transition Models: Catch

Again construction of the G-CNN is much the same as in the case for CartPole. The basic network was built from group convolutions, with two hidden layers. There is one notable difference between these models and the CartPole models. The output for the Catch transition models are two distributions over the ball and paddle locations. In contrast to the real valued outputs of the CartPole model. These distributions are $T(s,a)_b, T(s,a)_p$ which are the distribution over the ball and paddle locations, respectively.

$$L(\phi) = \frac{1}{N} \sum_{(s,a,s')\sim\tau}^{N} (1 - \delta(\text{done})) \left[BCE(s', T(s,a)_b) + BCE(s', T(s,a)_p) \right]. \tag{4.32}$$

Where BCE is the binary cross entropy between two distributions. The same indicator function is used to ensure that the model only learns from non-terminal states.

In order to make the model equivariant the proximal pooling layer requires a distance metric. This needs to quantify how far apart the predictions are. The metric used was an L1 loss between the x, y position in s and $T(s,a)$ of both the ball and the paddle. Additionally, to make the possible number of distances greater when the distances are summed, the ball's displacement is multiplied by 0.13. This constant value was found to improve the equivariance behaviour of the proximal pooling layer. The distance metric is then given by,

$$d(s,s') = ||\text{mode}(T(s,a)_b) - s'_b||_1 + 0.13||\text{mode}(T(s,a)_p) - s'_p||_1. \tag{4.33}$$

The subscripts, b, p indicate the ball and the paddle, respectively.

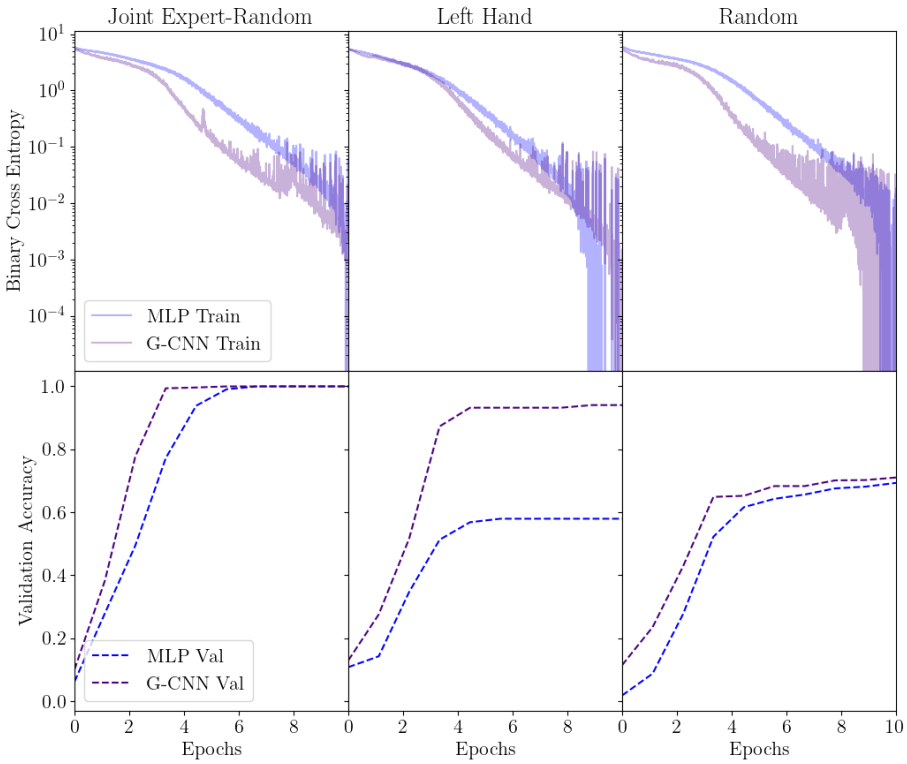

Figure 4.9: Transition Model BCE error, plotted against epochs for three different training datasets, With model validation accuracy plotted below. Left, the dataset is joint expert-random. Center, the same dataset as left, however, with only the left action taken. Right, solely transitions sampled from a random policy. These datasets contain 400,000 transitions. All plots have the same set of validation data with a 50/50 split of expert and random policy sampled data.

4.7.6.1 Results: Comparing Convergence of Transition Models

As before for CartPole, the transition models were trained on three different datasets to assess their convergence behaviour. Again the datasets were a random and expert policy sampled data, only left actions from the previous dataset, and only random policy sampled transitions.

In Fig.4.9, the equivariant model again demonstrates the benefits of inductive biases. The equivariant and MLP models achieve 0.998 validation accuracy on transitions sampled from the joint expert random policy, with slightly faster convergence. When the model acts on a dataset, which has been filtered for only the left actions, the equivariant G-CNN generalizes to both actions as expected. The

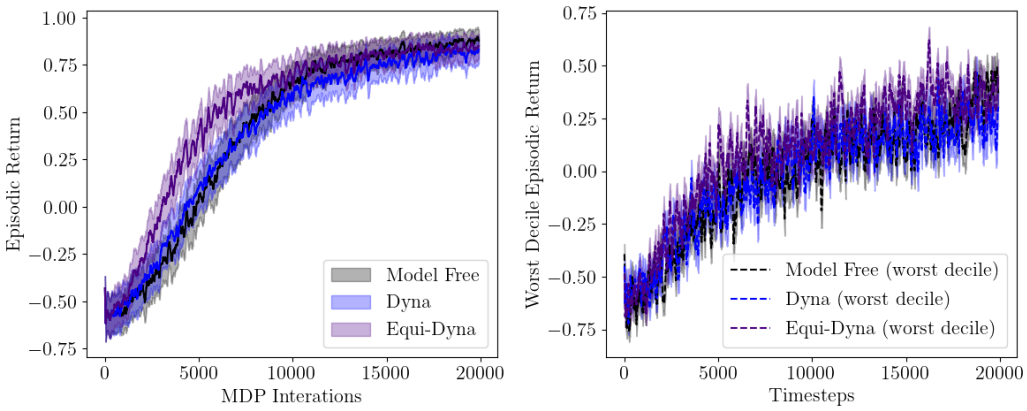

Figure 4.10: Left: Mean episodic returns for the Catch agents across 128 random seeds plotted against number of interaction time-steps in the MDP. Right: The mean cumulative episodic returns of the worst performing 128 random seeds against number of experience time-steps in the MDP. Both of the plots are moving averages, with windows of 10 time-steps. Additionally, two standard errors are plotted. A planning ratio of 8 is used for Equi-Dyna and 4 for Dyna.

Equivariant model preforms slightly better on the random data. However, the performance is lack luster. On inspection the random dataset contains 60 of the unique transitions found in the join expert-random dataset. This implies that both models are struggling to generalize.

4.7.6.2 Results: Supervised-Dyna

Again, three agents were trained over 128 random seeds. To compare the performance the equivariant G-CNN Dyna against an MLP transition model with a baseline actor-critic. The number of Dyna iterations was set to 50 for both the MLP transition model Dyna agent (Dyna) and the equivariant transition model (Equi-Dyna) agent.

Inspecting the returns in Fig.4.10 we see that the equivariant transition model agent (Equi-Dyna) slightly outperforms the Dyna model. The performance delta is most notable early on in the training. This is despite both of the transition models achieving near perfect accuracy on the validation dataset. Notably, though the worst decile of random seeds do perform better across both of the model based agents. In the case where the equivariant transition model, and the conventional MLP have the same validation error. One would expect that their ability to plan trajectories would

Time-steps	Baseline	Dyna	Equi-Dyna
$2,000$	-0.51 ± 0.07	-0.41 ± 0.09	$\mathbf{-0.29 \pm 0.09}$
$10,000$	0.70 ± 0.06	0.64 ± 0.06	$\mathbf{0.67 \pm 0.05}$
$20,000$	0.875 ± 0.04	0.88 ± 0.06	$\mathbf{0.90 \pm 0.05}$

Table 4.6: Mean cumulative episodic returns tabulated for the two network architectures. All episodic returns are recorded with confidence intervals of two standard errors across 128 random seeds.

be approximately equivalent. This does seem to be the case. However, the quality of the models seems insufficient to be a high quality substitute for actual transitions.

Again tabulating the returns' mean distribution in Table.4.6. In the table only the best planning ratios are listed. This enables comparison with the G-CNN actor-critic, which vastly outperforms all Dyna agents. Additionally, the distributions of the means are all within two standard errors for 50% and full sample completion.

4.7.7 Conclusion

For both Catch and CartPole, the equivariant G-CNN transition models with a proximal pooling layer are capable of matching and/or surpassing the performance of the MLP transition models. Additionally, there is ample evidence to suggest that, when combined with the correct distance metric, the proximal pooling layer is sufficient to produce approximately equivariant transition models. This evidence is apparent in both Catch and CartPole, where a notable improvement in accuracy or reduction in loss on unseen data is observed. The G-CNN's ability to generalize to states not encountered in the training data is particularly significant if these states fall within the orbit of states seen during training. When using the aforementioned transition models for performing Supervised-Dyna, mixed results were obtained. In the context of CartPole, the equivariant model's generalization improved significantly, allowing the agent to effectively utilize the planning phase and markedly enhance its sample efficiency. As a result, the agent required far fewer MDP time-steps to learn an expert policy

In the Catch environment, both the MLP and G-CNN models achieved identical accuracy after training. When employed to plan trajectories for a Dyna agent, these models resulted in a slight improvement in the initial episodes compared to

a model-free implementation. Furthermore, they exhibited significantly improved stability across the worst-performing random seeds.

The results of the Supervised-Dyna experiments indeed demonstrate the effectiveness of the Dyna implementation. In certain scenarios, the transition model can effectively enhance an agent's sample efficiency. However, the marginal improvement in performance gained from models that generalize slightly worse suggests that the quality of the model plays a pivotal role in enabling effective model-based reinforcement learning.

4.8 Dyna

Previously, it was demonstrated that equivariant world models provide advantages over world models without an inductive bias.

However, for such techniques to be truly useful, the models must demonstrate an improvement in sample efficiency. This was not the case with the offline trained world models. As the world models required interacting with the MDP many times to create a dataset to learn from. Because of this a set of agents were trained with online learnt transition models in a full Dyna Alg.2 setting.

In the same fashion as the Supervised-Dyna agents before, 128 agents were trained at each hyperparameter configuration initialized with independent random seeds. In comparison to the Supervised-Dyna experiments before agents are trained across five planning ratios and four values for the number of Dyna iterations. The planning ratio is much the same as before and controls the fraction of time spent planning to acting. The number of Dyna iterations controls the size of the buffer of transitions for both the transition model. To help with reasoning about the training the size of the replay buffer is,

$$|\mathcal{D}| = 50000/\text{Num Dyna Iterations}. \tag{4.34}$$

Further, when considering the planning phase, the planning buffer size is related to the acting buffer by,

$$|\mathcal{D}_p| = \mathbf{PR}|\mathcal{D}|. \tag{4.35}$$

As the number of timesteps is fixed the number of Dyna iterations provides a mechanism for controlling the frequency of planning.

4.8.1 Results: CartPole Dyna

The matrix of plots in Fig.4.11 shows underwhelming performance for the Dyna implementation. Both the Dyna and Equi-Dyna agents underperform the baseline actor-critic in all plots. Further, the model's performance is strikingly similar across all planning ratios and Dyna iterations.

When only considering a planning ratio of 0.5, where, the agent learns from

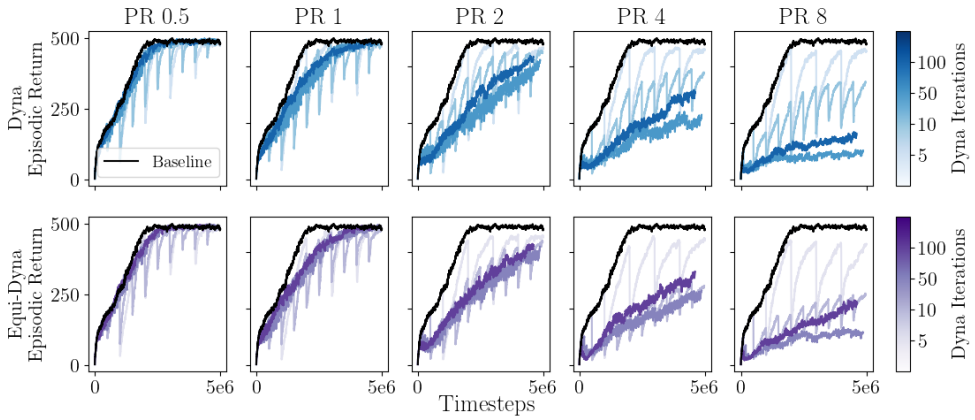

Figure 4.11: A matrix of plots for mean episodic returns for the CartPole Dyna agents across 128 random seeds plotted against number of interaction time-steps in the MDP. The matrix varies planning ratio horizontally, and the top row is the conventional MLP transition model in blue. The equivariant transition models are all on the bottom row, in purple. Each plot contains 5 lines, in black is the baseline actor-critic implementation. The colorbar indicate the number of dyna iterations used to train the agents.

twice the number of real transitions as simulated transitions. The agents with higher Dyna iteration count, thus fewer gradient updates[3] from simulated transitions, perform similarly to the baseline. They did not demonstrate the discontinuities seen in models with higher Dyna iterations.

These discontinuities are caused by the actor-critic network gaining too much experience in the planning environment. If the planning environment does not simulate the true environment well enough.

The evidence for this is threefold. Firstly such discontinuities, are clearly absent in the expert transition model seen in Fig.4.8. Where, the pretrained transition model, has access to more data than is seen by the online trained world models throughout the whole training, and achieves much lower validation loss 0.009 compared with the lowest transition model training loss across all hyperparameters of 0.039. This factor of four loss indicates a far more accurate transition model.

Secondly, considering higher planning ratios it can be seen that the discontinuities are even larger. As higher planning ratios correspond to the actor-critic performing more simulated training. This results in further fitting the actor-critic

[3]Batch size is constant, so parameter updates is a function of dataset size.

to the simulated environment. When the agent returns to the true environment the return delta is larger for higher planning ratios, indicating the negative effect on the policy in performing simulated training.

Finally, when considering the higher planning ratios and higher Dyna iterations, we see that when the majority of policy updates are in the simulated environment, and the transition model learns from a limited amount of simulated data, that the effectiveness of the models collapses in the true environment.

This results in a set of Dyna agents where the equivariance inductive bias, provides no improvement in performance. This is attributed to the poor quality of the online learnt world models. Where the equivariant G-CNN transition model's superior data efficiency and generalization is not sufficient to overcome the limited sample count available for training transition models.

4.8.2 Catch

In comparison to the CartPole environment the Catch environment is discrete and as such may provide a simpler task for learning. In the same manner as CartPole, a set of experiments were performed, varying the planning ratio and number of Dyna iterations. Despite the small discrete state space, none of the agents are able

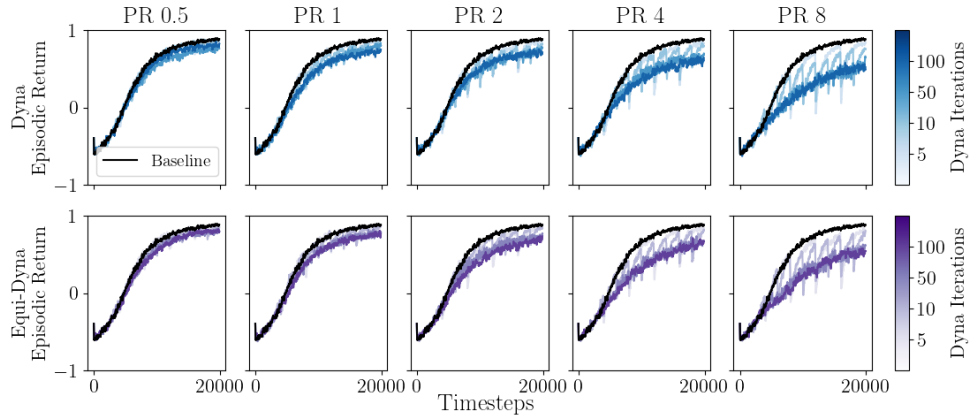

Figure 4.12: A matrix of plots for mean episodic returns for the Catch Dyna agents across 128 random seeds plotted against number of interaction time-steps in the MDP. The matrix varies planning ratio horizontally, and the top row is the conventional MLP transition model in blue. The equivariant transition models are all on the bottom row, in purple. Each plot contains 5 lines, in black is the baseline actor-critic implementation. The colorbar indicate the number of dyna iterations used to train the agents.

to improve upon the actor-critic baseline. The similarity, in the patterns shown in the return curves between Catch and CartPole imply that the overfitting to poor simulation is also present.

In the mean returns of the Catch agents over 128 seeds, again minimal difference is seen between the Equi-Dyna agent and the Dyna agent. However, their training dynamics do seem to be similar. At higher iteration count, the return curves are smoother, due to reduced number of updates in the planned environment, not allowing the policy to drift from the value learnt in the true environment dramatically. When the planning ratio is increased, the impact upon the quality of the policy is more sever. This degradation is attributed to overfitting to a simulated environment.

In comparison to CartPole, the Catch environment only has 675 possible transitions. However, when testing the expert policy and the random policies in Section 4.7 the joint expert random policy, did not sample every possible state action pair, in 10,000 sampled transitions. When training the agent's world model online from the replay buffer, not all transitions may be present, and the model's inability to generalize to unseen states is still a problem. This poor generalization performance is clear in the return curves as the agent's transition models are not able to produce sufficiently accurate transitions for the policy to learn from.

4.9 Conclusion

In both environment similar behaviour was observed. The Dyna and Equi-Dyna have very comparable performance at the same hyperparameters, which always underperformed the baseline. As such, planning with online models was ineffective.

The poor performance was attributed to transition models that do not deliver sufficient accuracy in producing trajectories that simulate the environment well enough to provide an alternative to samples in the true environment.

The inaccurate transition models suggest providing the models with more data and/or not using the transition model before some point in the training such that enough data can be sampled from the MDP that the transition model can generalize effectively. This However, would not have been straightforward to implement.

Implementation constrains of the numerical computing library JAX and available memory, meant that there was no ability to grow array size in code compiled to the XLA backend. This in conjunction with the large memory footprint of thousands of transitions, meant that the workarounds tried with pre allocated replay buffers were infeasible due to out of memory errors on the GPU and RTX 3090.

Chapter 5

Conclusions

This report presented multiple novel contributions. With the focus of leveraging symmetries for reinforcement learning agents. The first of these contributions was a fully equivariant actor-critic architecture using discrete group G-CNNs. These equivariant agents showed superior sample efficiency and robustness to the MLP baselines in multiple environments.

In contrast to using equivariant structured actor-critic agents, equivariant transition models were investigated. Here, the Proximal Pooling layer Sec.4.7.4 was proposed, which provides a method to produce approximately equivariant transition models from G-CNNs. In experiments, once trained these models demonstrated full equivariance. The power of this was shown in their ability to predict transitions from discrete actions unseen in the training set.

Finally, the equivariant transition models were applied to Dyna agents, with MLP actor-critics. First the transition models were trained offline, with large volumes of transition samples. The pre-trained equivariant transition models agent in the CartPole environment demonstrated a substantial uplift over the actor-critic baseline. In the Catch environment, the results were less impressive.

When extended to online trained transition models, neither the equivariant transition models nor the MLP transition models were able to outperform the actor-critic baseline. This inability to perform effective planning was attributed to the transition models not achieving high enough accuracy in simulating the true environment for the leant policy to improve when planning.

As both the equivariant transition model and MLP transition model failed to generalize sufficiently, this highlights an issue with the Dyna algorithm when training a model online. Despite the fact that the equivariant model demonstrated improved generalization. This improved generalization was not sufficient to stop the actor-critic agent overfitting to an inaccurate transition model impairing the performance of the actor-critic agent in the true environment.

The report presents clear evidence that equivariance is an effective inductive bias in symmetric MDPs, providing improved sample efficiency and robustness in training. Additionally, the report provides a path for future investigation with equivariant G-CNNs world models.

5.1 Future Work

There exist multiple interesting future avenues to take. Firstly, implementing alternative model-based RL algorithms with the equivariant G-CNN transition models would be interesting to see if performance improvements can be seen, not only when the model is trained offline.

Secondly, possibly most interestingly, is that there may be the possibility of learning the group structure, in the G-CNNs using a Meta-Learning method, like that proposed by [Zhou et al., 2020]. This would entail training agents simultaneously, as we do across multiple random seeds acting in different instantiations of environments, learning the permutation matrices described in Sec.4.7.4.

Bibliography

[Alet et al., 2021] Alet, F., Doblar, D., Zhou, A., Tenenbaum, J., Kawaguchi, K., and Finn, C. (2021). Noether networks: meta-learning useful conserved quantities. *Advances in Neural Information Processing Systems*, 34:16384–16397.

[Barto et al., 1983] Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5):834–846.

[Baxter, 2000] Baxter, J. (2000). A model of inductive bias learning. *Journal of artificial intelligence research*, 12:149–198.

[Bellman, 1957] Bellman, R. (1957). A markovian decision process. *Indiana Univ. Math. J.*, 6:679–684.

[Bronstein et al., 2021] Bronstein, M. M., Bruna, J., Cohen, T., and Velič ković, P. (2021). Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*.

[Cohen and Welling, 2016] Cohen, T. and Welling, M. (2016). Group equivariant convolutional networks. In *International conference on machine learning*, pages 2990–2999. PMLR.

[Coumans and Bai, 2021] Coumans, E. and Bai, Y. (2016–2021). Pybullet, a python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`.

[Dulac-Arnold et al., 2019] Dulac-Arnold, G., Mankowitz, D., and Hester, T. (2019). Challenges of real-world reinforcement learning. *arXiv preprint arXiv:1904.12901*.

[Finn et al., 2017] Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR.

[Florian,] Florian, R. V. Correct equations for the dynamics of the cart-pole system.

[Fuchs et al., 2020] Fuchs, F., Worrall, D., Fischer, V., and Welling, M. (2020). Se (3)-transformers: 3d roto-translation equivariant attention networks. *Advances in neural information processing systems*, 33:1970–1981.

[Gell-Mann, 1961] Gell-Mann, M. (1961). The eightfold way: A theory of strong interaction symmetry.

[Goyal and Bengio, 2022] Goyal, A. and Bengio, Y. (2022). Inductive biases for deep learning of higher-level cognition. *Proceedings of the Royal Society A*, 478(2266):20210068.

[Haarnoja et al., 2018] Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR.

[Hafner et al., 2020] Hafner, D., Lillicrap, T., Norouzi, M., and Ba, J. (2020). Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*.

[Hafner et al., 2023] Hafner, D., Pasukonis, J., Ba, J., and Lillicrap, T. (2023). Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*.

[He et al., 2022] He, C., Rathbun, Z., Buonauro, D., Meyerhoff, H. S., Franconeri, S. L., Stieff, M., and Hegarty, M. (2022). Symmetry and spatial ability enhance

change detection in visuospatial structures. *Memory & Cognition*, 50(6):1186–1200.

[Henderson et al., 2018] Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. (2018). Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32.

[Hessel et al., 2018] Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. (2018). Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32.

[Hornik, 1991] Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257.

[Howard, 1960] Howard, R. A. (1960). Dynamic programming and markov processes.

[Johnston et al., 2022] Johnston, I. G., Dingle, K., Greenbury, S. F., Camargo, C. Q., Doye, J. P., Ahnert, S. E., and Louis, A. A. (2022). Symmetry and simplicity spontaneously emerge from the algorithmic nature of evolution. *Proceedings of the National Academy of Sciences*, 119(11):e2113883119.

[Jumper et al., 2021] Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., et al. (2021). Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589.

[Kirkpatrick et al., 2017] Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526.

[Laskin et al., 2020] Laskin, M., Lee, K., Stooke, A., Pinto, L., Abbeel, P., and Srinivas, A. (2020). Reinforcement learning with augmented data. *Advances in neural information processing systems*, 33:19884–19895.

[LeCun et al., 1989] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551.

[Lillicrap et al., 2015] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

[Lin et al., 2020] Lin, Y., Huang, J., Zimmer, M., Guan, Y., Rojas, J., and Weng, P. (2020). Invariant transform experience replay: Data augmentation for deep reinforcement learning. *IEEE Robotics and Automation Letters*, 5(4):6615–6622.

[Lu et al., 2022] Lu, C., Kuba, J., Letcher, A., Metz, L., Schroeder de Witt, C., and Foerster, J. (2022). Discovered policy optimisation. *Advances in Neural Information Processing Systems*, 35:16455–16468.

[Mavor-Parker et al., 2022] Mavor-Parker, A. N., Banino, A., Griffin, L. D., and Barry, C. (2022). A simple approach for state-action abstraction using a learned mdp homomorphism. *arXiv preprint arXiv:2209.06356*.

[Mnih et al., 2016] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning.

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

[Mondal et al., 2020] Mondal, A. K., Nair, P., and Siddiqi, K. (2020). Group equivariant deep reinforcement learning. *arXiv preprint arXiv:2007.03437*.

[Osband et al., 2020] Osband, I., Doron, Y., Hessel, M., Aslanides, J., Sezener, E., Saraiva, A., McKinney, K., Lattimore, T., Szepesvári, C., Singh, S., Van Roy, B., Sutton, R., Silver, D., and van Hasselt, H. (2020). Behaviour suite for reinforcement learning. In *International Conference on Learning Representations*.

[Ravindran, 2003] Ravindran, B. (2003). Smdp homomorphisms: An algebraic approach to abstraction in semi markov decision processes.

[Ravindran and Barto, 2001] Ravindran, B. and Barto, A. G. (2001). Symmetries and model minimization in markov decision processes.

[Rezaei-Shoshtari et al., 2022] Rezaei-Shoshtari, S., Zhao, R., Panangaden, P., Meger, D., and Precup, D. (2022). Continuous mdp homomorphisms and homomorphic policy gradient. *Advances in Neural Information Processing Systems*, 35:20189–20204.

[Schulman et al., 2015a] Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015a). Trust region policy optimization. In Bach, F. and Blei, D., editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1889–1897, Lille, France. PMLR.

[Schulman et al., 2015b] Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2015b). High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

[Schulman et al., 2015c] Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2015c). High-dimensional continuous control using generalized advantage estimation.

[Schulman et al., 2017] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

[Silver et al., 2016] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503.

[Silver et al., 2017] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2017). Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*.

[Sutton and Barto, 2018] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*.

[Sutton et al., 2012] Sutton, R. S., Szepesvári, C., Geramifard, A., and Bowling, M. P. (2012). Dyna-style planning with linear function approximation and prioritized sweeping. *arXiv preprint arXiv:1206.3285*.

[Van der Pol et al., 2020] Van der Pol, E., Kipf, T., Oliehoek, F. A., and Welling, M. (2020). Plannable approximations to mdp homomorphisms: Equivariance under actions. *arXiv preprint arXiv:2002.11963*.

[van der Pol et al., 2020] van der Pol, E., Worrall, D. E., van Hoof, H., Oliehoek, F. A., and Welling, M. (2020). Mdp homomorphic networks: Group symmetries in reinforcement learning.

[Van Hasselt et al., 2018] Van Hasselt, H., Doron, Y., Strub, F., Hessel, M., Sonnerat, N., and Modayil, J. (2018). Deep reinforcement learning and the deadly triad. *arXiv preprint arXiv:1812.02648*.

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

[Wang et al., 2022] Wang, D., Walters, R., and Platt, R. (2022). so2-equivariant reinforcement learning. *arXiv preprint arXiv:2203.04439*.

[Weiler and Cesa, 2019] Weiler, M. and Cesa, G. (2019). General e (2)-equivariant steerable cnns. *Advances in neural information processing systems*, 32.

[Williams, 1992] Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Reinforcement learning*, pages 5–32.

[Wolpert et al., 1995] Wolpert, D. H., Macready, W. G., et al. (1995). No free lunch theorems for search. Technical report, Citeseer.

[Zhou et al., 2020] Zhou, A., Knowles, T., and Finn, C. (2020). Meta-learning symmetries by reparameterization. *arXiv preprint arXiv:2007.02933*.

5.2 Hyperparameters

5.2.1 Actor-Critic

All of the hyperparameter choices come from PureJaxRL[Alet et al., 2021], apart from those for Catch which were slightly modified.

Hyperparameter	Value
Optimizer	Adam
Learning rate	$3.5{\cdot}10^{-4}$
Adam: β_1	0.9
Adam: β_2	0.999
Adam:ϵ	10^{-8}
Max Grad Norm	0.5
Mini Batch Size	128

Table 5.1: CartPole Optimizer Hyperparameters

Hyperparameter	Value
Optimizer	Adam
Learning rate	$3.5{\cdot}10^{-4}$
Adam: β_1	0.9
Adam: β_2	0.999
Adam:ϵ	10^{-8}
Max Grad Norm	0.5
Mini Batch Size	10

Table 5.2: Catch Optimizer Hyperparameters

Hyperparameter	Value
PPO:Clip-ϵ	0.2
Value coefficient	0.5
Entropy coefficient	0.01
Update length	128
Discount: γ	0.99
GAE-λ	0.95

Table 5.3: RL Hyperparameters

5.2.2 Transition Model

Network	Parameter Count
Actor-Critic	9155
Symmetrizer	9155
Equivariant Actor-Critic	8961 height

Table 5.4: Actor-Critic network parameter Counts

Hyperparameter	Value
Optimizer	Adam
Learning rate	$1{\cdot}10^{-3}$
Adam: β_1	0.9
Adam: β_2	0.999
Adam:ϵ	10^{-8}
Max Grad Norm	0.5
Mini Batch Size	64

Table 5.5: Transition Model Hyperparameters

5.3 CartPole

5.3.1 Transition Distribution By Angle

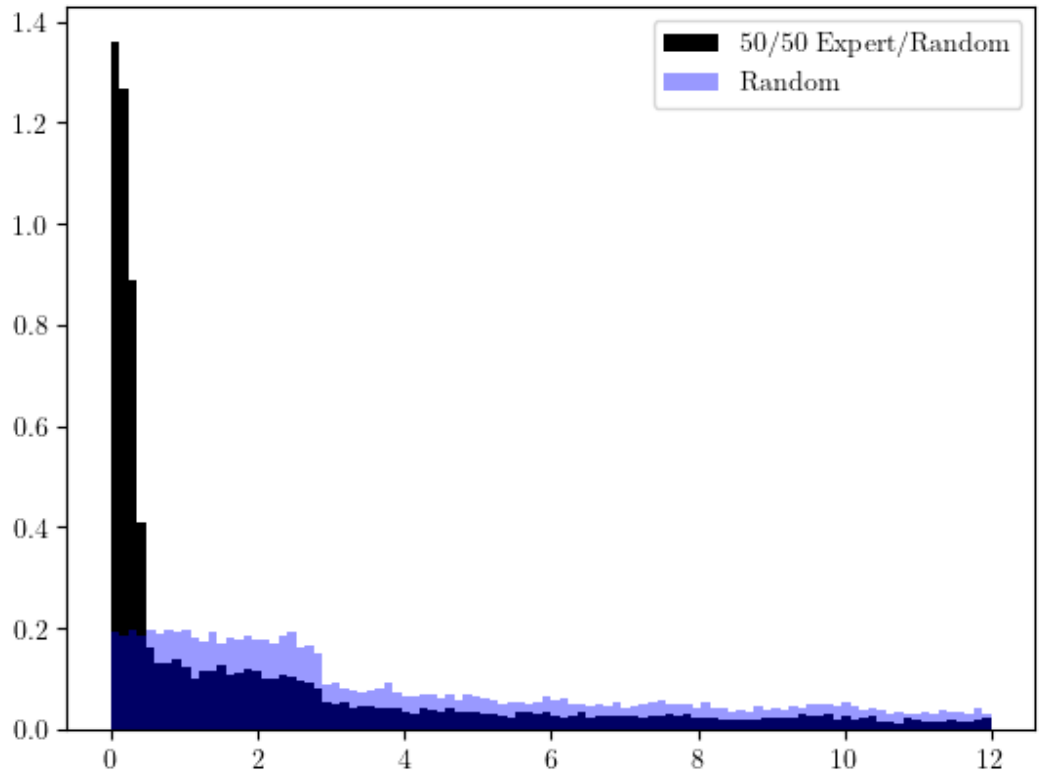

Figure 5.1: Histogram plot of transitions initial state against absolute angle. 50/50 Expert/Random refers to the joint expert-random dataset.

5.3.2 Supervised-Dyna

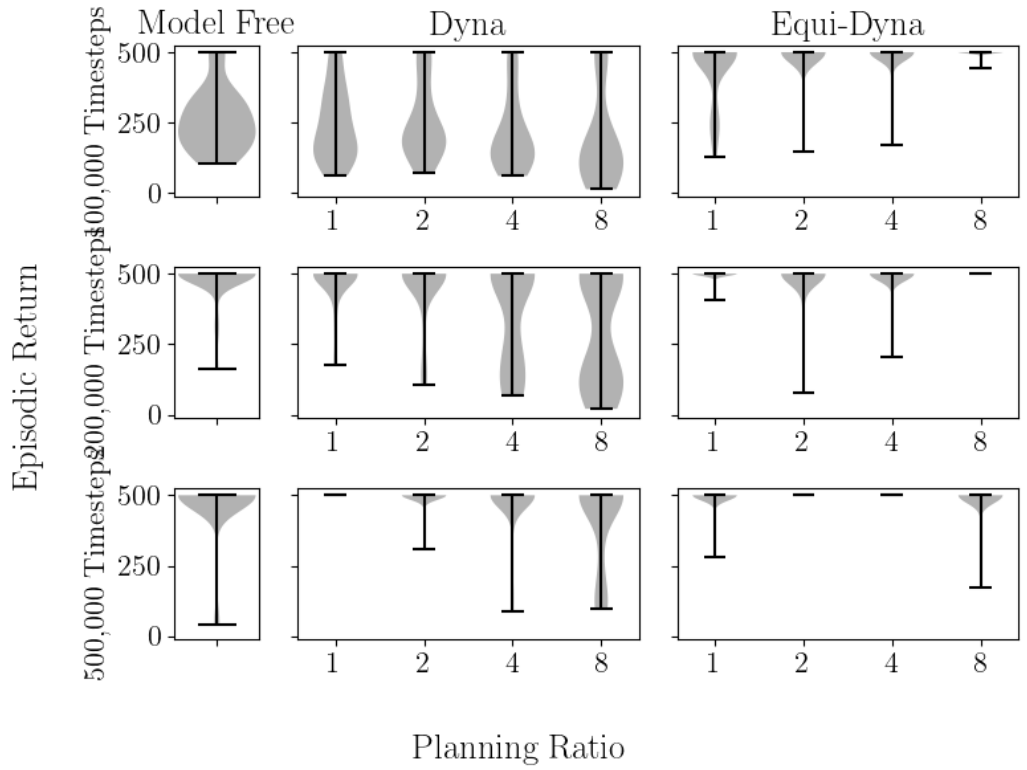

Figure 5.2: Violin Plot of episodic returns at timestep intervals across all tested planning ratios on the CartPole environment.

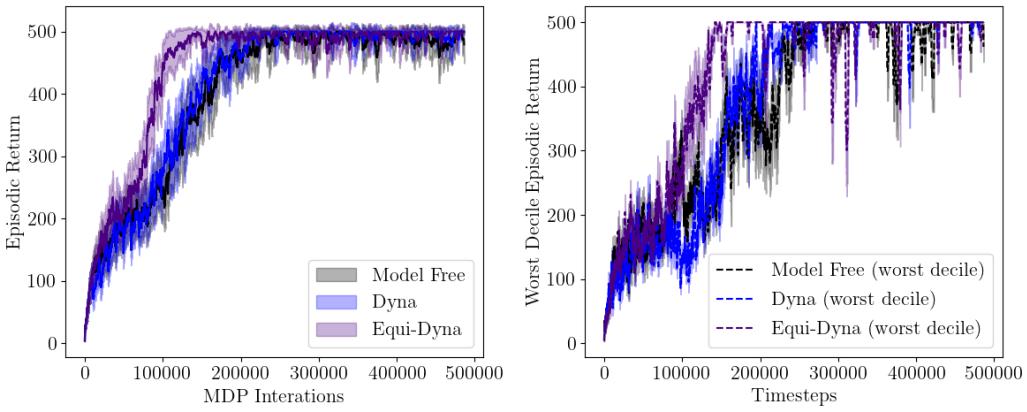

Figure 5.3: Episodic returns for Supervised-Dyna agents with a baseline on the CartPole environment. Using a planning ratio of one.

5.4 Catch

5.4.1 Supervised-Dyna

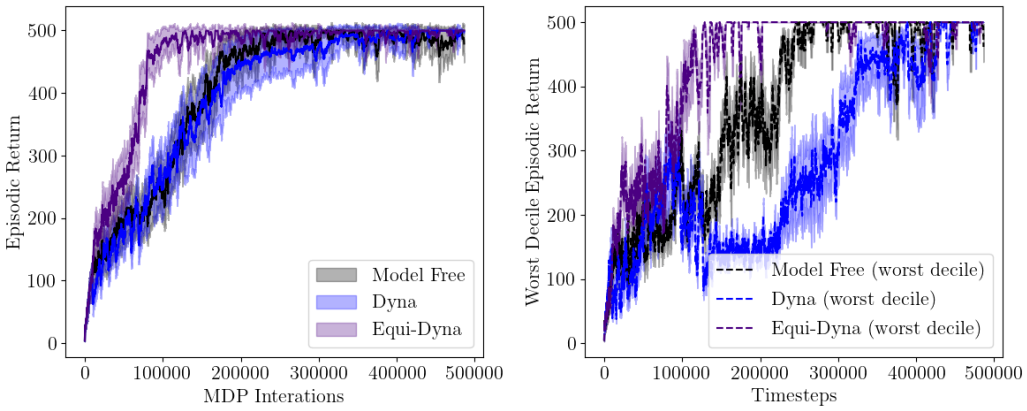

Figure 5.4: Episodic returns for Supervised-Dyna agents with a baseline on the CartPole environment. Using a planning ratio of two.

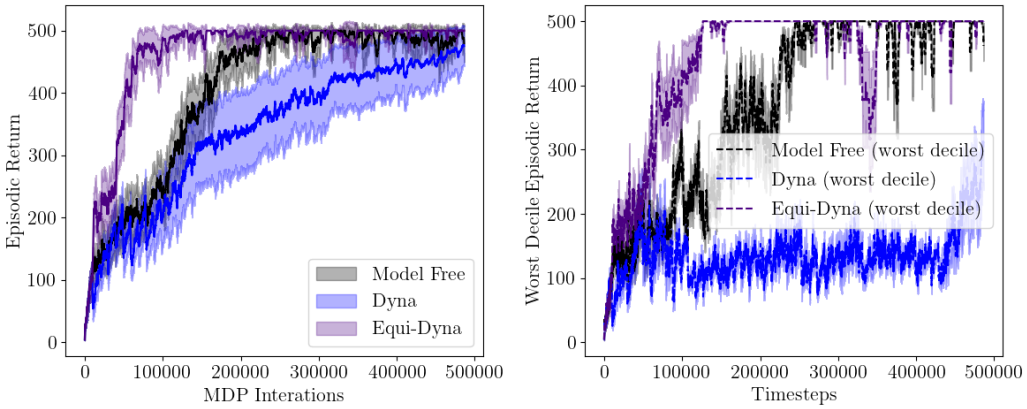

Figure 5.5: Episodic returns for Supervised-Dyna agents with a baseline on the CartPole environment. Using a planning ratio of four.

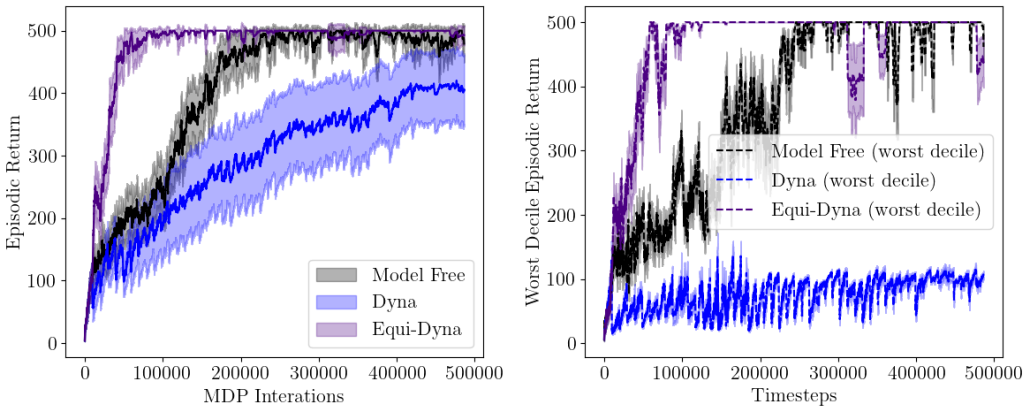

Figure 5.6: Episodic returns for Supervised-Dyna agents with a baseline on the CartPole environment. Using a planning ratio of eight.

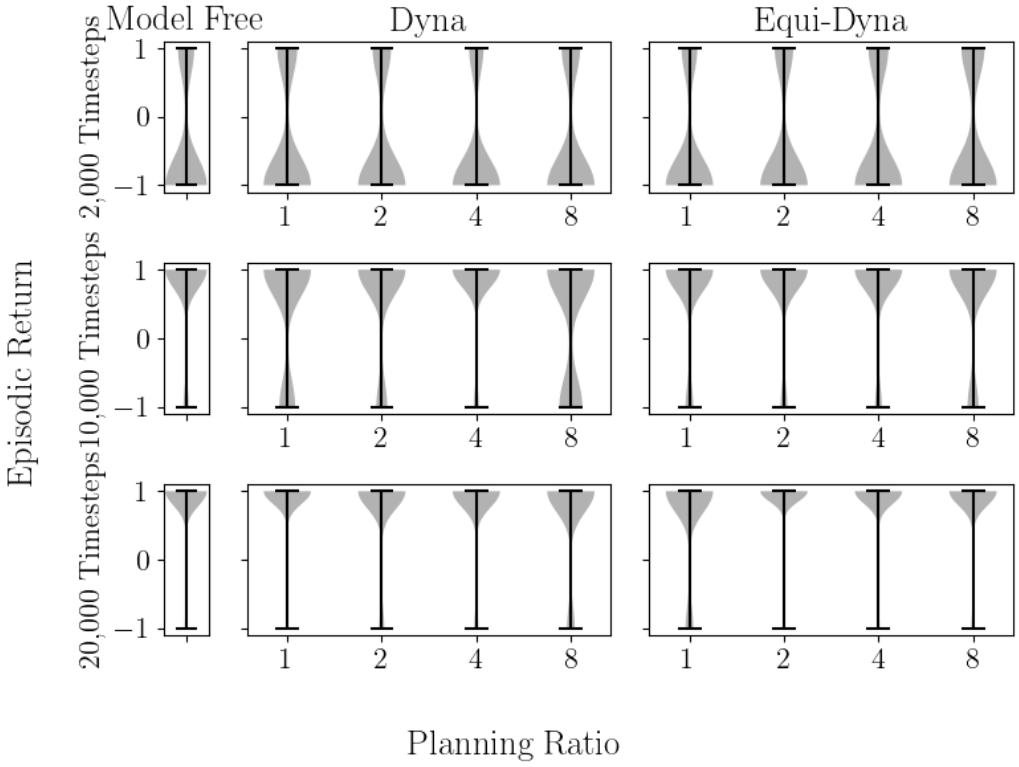

Figure 5.7: Violin Plot of episodic returns at timestep intervals across all tested planning ratios on the catch environment.

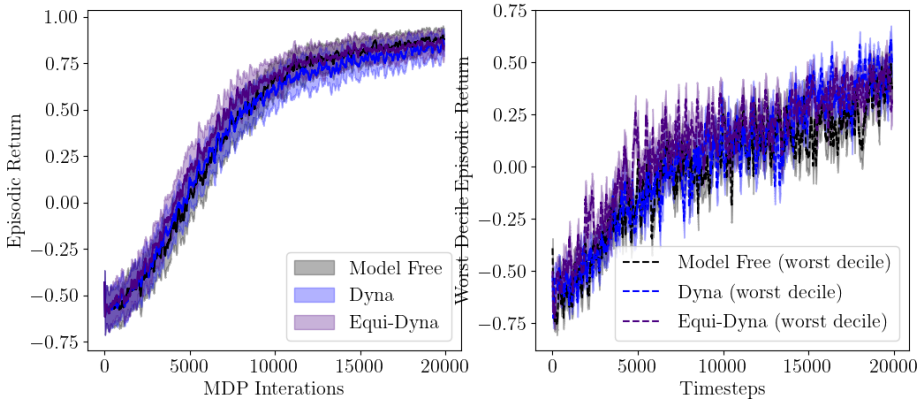

Figure 5.8: Episodic returns for Supervised-Dyna agents with a baseline on the Catch environment. Using a planning ratio of one.

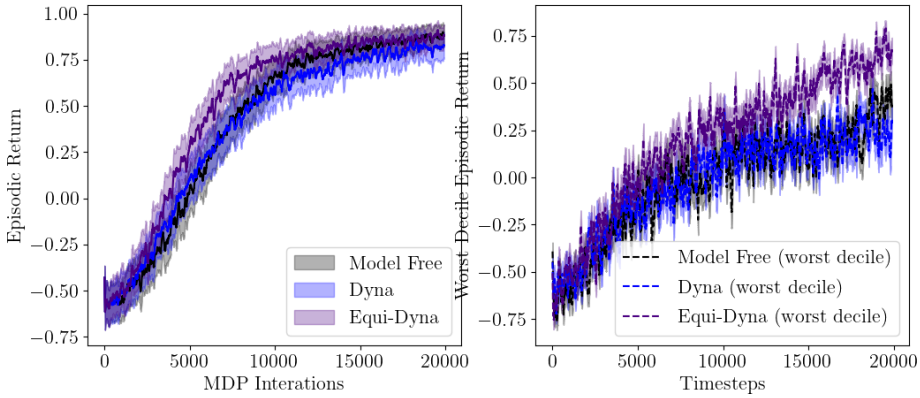

Figure 5.9: Episodic returns for Supervised-Dyna agents with a baseline on the Catch environment. Using a planning ratio of two.

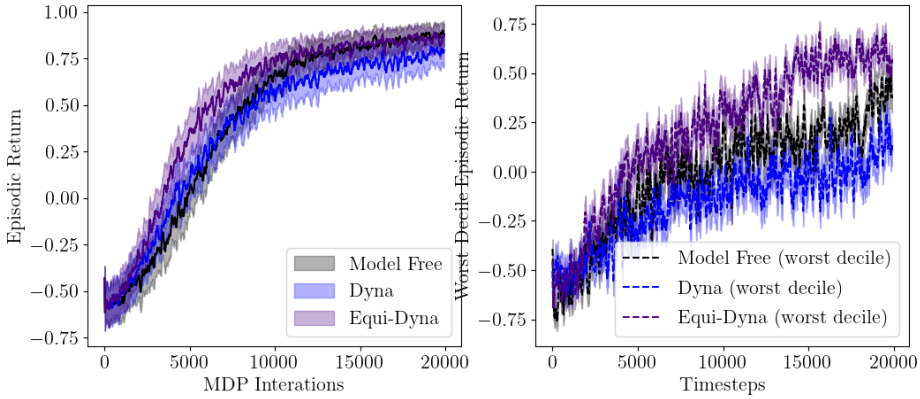

Figure 5.10: Episodic returns for Supervised-Dyna agents with a baseline on the Catch environment. Using a planning ratio of four.

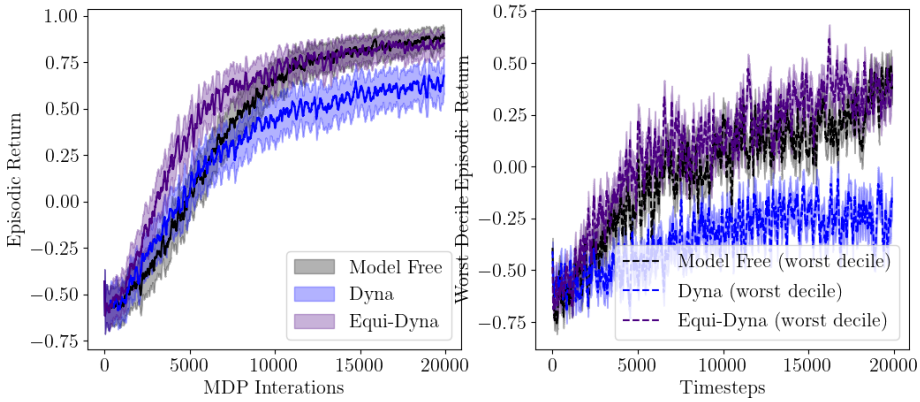

Figure 5.11: Episodic returns for Supervised-Dyna agents with a baseline on the Catch environment. Using a planning ratio of eight.