
A Simple Approach for State-Action Abstraction using a Learned MDP Homomorphism

Augustine N. Mavor-Parker
Centre for Artificial Intelligence
University College London
United Kingdom

Matthew J. Sargent
Centre for Artificial Intelligence
University College London
United Kingdom

Andrea Banino
Deepmind
London
United Kingdom

Lewis D. Griffin*
Department of Computer Science
University College London
United Kingdom

Caswell Barry*
Department of Cell and Developmental Biology
University College London
United Kingdom

Abstract

Animals can rapidly infer, from limited experience, when sets of state-action pairs have equivalent reward and transition dynamics. On the other hand, reinforcement learning systems must painstakingly learn through trial and error that sets of state-action pairs are value equivalent—requiring a prohibitively large amount of samples from their environment. Previous works use Markov decision process (MDP) homomorphisms, which are abstracted MDPs enabling sample efficient policy learning. Consequently, impressive improvements in sample efficiency have been achieved when a suitable MDP homomorphism can be constructed *a priori*—usually by exploiting a practitioner’s knowledge of environment symmetries. Here, we propose an approach for constructing homomorphisms in discrete action spaces, which uses a partial model of environment dynamics to infer which state-action pairs lead to the same state—reducing the size of the state-action space by a factor approximately equal to the cardinality of the action space. On MDP homomorphism benchmarks, we demonstrate improved sample efficiency over previous attempts to learn homomorphisms, and achieve comparable sample efficiency to approaches that rely on prior knowledge of environment symmetries.

1 Introduction

Reinforcement learning (RL) agents outperform humans on previously impregnable benchmarks such as Go (Silver et al., 2016) and Starcraft (Vinyals et al., 2019). However, the computational expense of RL hinders its deployment in real world applications. State of the art deep RL can agents demand hundreds of millions of samples (or even hundreds of billions) to learn a policy—either within an environment model (Hafner et al., 2020) or by direct interaction (Badia et al., 2020).

In contrast, humans can abstract away details about state-action pairs that do not effect their values. Consider the Atari game Asterix, where an agent collects treasure while avoiding enemies in a 2D navigation task—a human gamer understands that collecting treasure when approaching from the left has the same value as collecting the same treasure but approaching from the right. In this work, we identify this abstraction as a Markov decision process (MDP) homomorphism Ravindran and Barto (2001); van der Pol et al. (2020b). MDP homomorphisms collapse equivalent state-actions in an observed MDP onto a single abstract state-action pair in an abstract MDP van der Pol et al. (2020b).

*Equal Contribution. Send correspondence to augustine.mavor-parker@cs.ucl.ac.uk

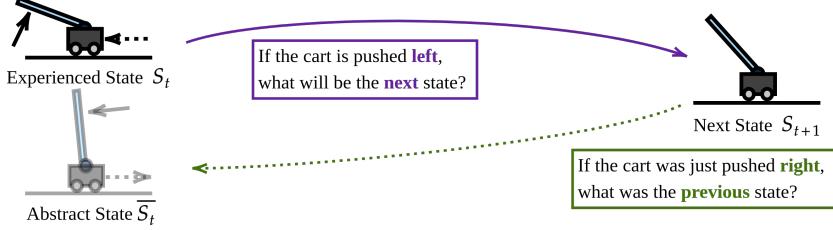


Figure 1: We find equivalent states by moving **forward** and then **backward** through a model of the environment. Unlike previous approaches (van der Pol et al., 2020b), we are *not* exploiting the horizontal symmetry of the environment. In contrast, we take advantage of the fact that state-action pairs that lead to the same state usually have equivalent values.

Given a mapping from experienced MDP to an abstract MDP, policies can be learned efficiently in the smaller abstract space and then mapped back to the experienced MDP when interacting with the environment (Ravindran and Barto, 2001). However, obtaining mappings to the abstract state-action space is challenging. Previous works hard code homomorphisms into a policy (van der Pol et al., 2020b) but developing approaches that learn homomorphic mappings from experience is an active area of research Rezaei-Shoshtari et al. (2022).

We developed *equivalent effect abstraction*, a method for constructing MDP homomorphisms from experience via a dynamics model—leveraging the fact that state-action pairs leading to the same next state frequently have equivalent values. Consider again the Asterix example—moving to a given state has the same value whether your previous action was right, left, up or down. Therefore, if we know the value of approaching from the right then, when equipped with a dynamics model, we demonstrate one can also infer the value of approaching a given state from the left, from below or from above. Consequently, equivalent effect abstraction extrapolates value judgements between equivalent state-action pairs, reducing the amount of experience required to learn a policy. We make the following contributions.

1. We develop a novel approach for constructing MDP homomorphisms (equivalent effect abstraction) that requires no prior knowledge of symmetries
2. In the tabular setting, we show equivalent effect abstraction improves the planning efficiency of model-based RL and the sample efficiency of model-free RL
3. In the deep RL setting, we show equivalent effect abstraction can be learned from experience for Cartpole and Predator-Prey environments (which are MDP homomorphisms benchmarks from van der Pol et al. (2020b))
4. When a transition model is available, we show impressive gains in sample efficiency on MinAtar’s Asterix environment

We are not the first to exploit the equivalent values of state action pairs that lead to the same next state. (Tesauro et al., 1995) use the concept of afterstates (states immediately before an opponents response to an action (Antonoglou et al., 2021)) in their backgammon program. This is similar to the equivalence we exploit and has been scaled within the MuZero framework to other board games (Antonoglou et al., 2021), but in general afterstates have not found wider applicability. (Misra et al., 2020)’s kinematic inseparability is a related but distinct abstraction that uses contrastive learning to classify whether two states will lead to the same next state for the *same* action. In contrast, our approach looks for state-action pairs (with *different actions*) that lead to the same next state, which is a broader form of abstraction. Another difference is that our equivalent effect abstraction does not require states to be discrete, which makes it readily applicable to common RL environments where it is rare to reach the same state twice. Finally Anand et al. (2015) defines a state-action equivalence that is more comprehensive than the equivalence defined by (Ravindran and Barto, 2001), which is satisfied by various equivalent state-action pairs like those found in (Ravindran and Barto, 2001; van der Pol et al., 2020b; Misra et al., 2020) as well as the state-action pairs satisfying the equivalent effect abstraction we present. However, they assume a tree of discrete states can be searched over for equivalences, which is challenging in deep RL.

Our work is organised into the following sections. In Section 2 we formally describe MDP homomorphisms and then introduce equivalent effect abstraction in Section 3. In Section 4 we validate our approach using benchmarks from the MDP homomorphism literature. Related work, limitations and conclusions are in Sections 5, 6 and 7.

2 MDP Homomorphisms

Using the definition from (Silver, 2015), an MDP \mathcal{M} can be described by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ where \mathcal{S} is the set of all states, \mathcal{A} is the set of all actions, $\mathcal{R} = \mathbb{E}[R_{t+1}|S_t = s, A_t = a]$ is the reward function that determines the scalar reward received at each state, $\mathcal{P} = \mathbb{P}[S_{t+1} = s'|S_t = s, A_t = a]$ is the transition function of the environment describing the probability of moving from state to another for a given action and $\gamma \in [0, 1]$ is the discount factor describing how much an agent should favour immediate rewards over those in future states. An agent interacts with an environment through its policy $\pi(a|s) = \mathbb{P}[A_t = a|S_t = s]$ (Silver, 2015) which maps the current state to a given action. To solve an MDP, an RL agent must develop a policy that maximises the return G , which is equal to the sum of discounted future rewards $G = \sum_{t=0}^T \gamma^t R_{t+1}$ (where t is the current timestep and T is the number of timesteps in a learning episode).

(Ravindran and Barto, 2001) introduced the concept of a homomorphism which, using the notation and definitions from (van der Pol et al., 2020b), is a homomorphism $h = \langle \sigma, \{\alpha_s|s \in \mathcal{S}\} \rangle$ from an agent’s experienced MDP \mathcal{M} to an abstract MDP $\bar{\mathcal{M}} = \langle \bar{\mathcal{S}}, \bar{\mathcal{A}}, \bar{\mathcal{P}}, \bar{\mathcal{R}}, \gamma \rangle$, which satisfies the following. $\forall s, s' \in \mathcal{S}, a \in \mathcal{A}$:

$$\bar{\mathcal{P}}(\sigma(s')|\sigma(s), \alpha_s(a)) = \sum_{s'' \in \sigma^{-1}(s')} \mathcal{P}(s''|s, a) \quad (1)$$

$$\bar{\mathcal{R}}(\sigma(s), \alpha_s(a)) = \mathcal{R}(s, a) \quad (2)$$

Where σ is a mapping from experienced states to abstract states, while α_s is a state dependent mapping between experienced and abstract actions. Equations (1) and (2) demonstrate that a homomorphic map of states and actions must maintain the transition dynamics and reward functions of the experienced MDP within the constructed abstract MDP. As proved in (Ravindran and Barto, 2001) and leveraged more recently by (van der Pol et al., 2020b), if the true Q-values in the abstract MDP are known, $\bar{Q}^*(\sigma(s), \alpha_s(a))$, then equipped with a homomorphism h , the true Q-values in the experienced MDP can be “lifted” (Ravindran and Barto, 2001) from their abstract counterparts.

$$\bar{Q}^*(\sigma(s), \alpha_s(a)) = Q^*(s, a) \quad \forall s \in \mathcal{S}, a \in \mathcal{A} \quad (3)$$

Where Q is the value of state s and action a and $*$ signifies the value function is optimal (van der Pol et al., 2020b). The size of abstract state-action space $\bar{\mathcal{S}} \times \bar{\mathcal{A}}$ is often smaller than the size of the experienced state-action space $\mathcal{S} \times \mathcal{A}$. Consequently, if an agent is equipped with a homomorphic map, a policy can be learned efficiently in the abstract state-action space and then “lifted” to the experienced state-action space (Ravindran and Barto, 2001). In general, obtaining a homomorphic map is difficult. Next, we present a novel method for inferring a homomorphic map from experience.

3 Equivalent Effect Abstraction

Equivalent effect abstraction is based on a simple observation that is guaranteed to be true in most RL benchmarks if \mathcal{R} and \mathcal{P} are deterministic. Firstly, state-action pairs that lead to the same next state have equivalent reward functions in the majority of RL benchmarks (e.g. DeepMind control Tassa et al. (2018), classic control Brockman et al. (2016) and Atari Bellemare et al. (2013)). While common MDP formulations (like the one used in Section 2) define reward functions as depending on the current state and previous action, in many environments it is possible to drop the dependence on the previous action. Nevertheless, we note that we assume no dependence on the previous action for the reward function, so our approach is not suitable for optimal control settings where different actions may be defined to have a different cost (Liberzon, 2011, p. 145). Next, we develop a method to learn the mapping σ between state-action pairs that lead to the same state.

Consider an agent that takes an action a_t in a state s_t and ends up in state s_{t+1} , once in state s_{t+1} , how can we infer what other state-action pairs could also lead to state s_{t+1} ? We could make this

Algorithm 1 Equivalent Action Abstraction Q-Network

```

class EquivalentEffectQNetwork(nn.Module):
    def __init__(self, forward_model, backward_model, num_of_actions, canonical_action):
        self.forward_model = forward_model # pretrained forward model
        self.backward_model = backward_model # pretrained backward model
        self.number_of_actions = num_of_actions
        self.canonical_action = canonical_action
        self.q_net = nn.Sequential(nn.Linear(2, 250), nn.ReLU(), nn.Linear(250, 1))

    def forward(self, state):
        values = [] # list to store q-values for each action
        for i, an_action in enumerate(self.number_of_actions):
            next_state = self.forward_model(state, an_action)
            equivalent_state = self.backward_model(state, canonical_action)
            value_for_i_th_action = self.q_net(equivalent_state)
            values.append(value_for_i_th_action)
        return values

```

Figure 2: PyTorch (Paszke et al., 2017) style pseudo-code for equivalent effect abstraction Q-network. Our Q-network only has an output for one action—the hypothetical action. To transform input states into the hypothetical reference frame we move forwards and backwards through a dynamics model, with the input action on the forwards model giving the order of the equivalent Q-value outputs.

inference if we had both a forwards *and* backwards model of the environment. To train our forwards model \mathcal{F} and backwards model \mathcal{B} , we collect experience tuples $\langle \mathbf{s}_t, a_t, \mathbf{s}_{t+1} \rangle$ from the environment and then optimise the parameters of our models θ and ϕ with a MSE loss function.

$$\arg \min_{\theta} \|\mathcal{F}_{\theta}(\mathbf{s}_t, a_t) - \mathbf{s}_{t+1}\|_2^2 \quad (4)$$

$$\arg \min_{\phi} \|\mathcal{B}_{\phi}(\mathbf{s}_{t+1}, a_t) - \mathbf{s}_t\|_2^2 \quad (5)$$

With a learned forwards model \mathcal{F}_{θ} and backwards model \mathcal{B}_{ϕ} we are ready to reduce the size of our MDP. First, we select an action \bar{a} to be our *hypothetical* action. Next for a given state-action pair (\mathbf{s}, a) the equivalent (hypothetical) state-action pair can be computed by moving forwards through our model and then backwards with the hypothetical action \bar{a} .

$$\bar{\mathbf{s}} = \sigma(\mathbf{s}, \bar{a}) = \mathcal{B}_{\phi}(\mathcal{F}_{\theta}(\mathbf{s}, a), \bar{a}) \quad (6)$$

In theory, one could calculate multiple equivalent state-action pairs for a given state-action pair by querying the backwards model with multiple different actions. However, our aim is to simplify the state-action space that our policy learns in, so instead we select one action to be our “hypothetical” action and all other state-action pairs are mapped into its reference frame. In Figure 1, the hypothetical action is moving right—meaning our Q-network only has to learn about moving right and we can map all right values to their equivalent left values with our forwards and backwards models. The hypothetical action is a hyperparameter. In some environments such as Cartpole, the hypothetical action can be selected randomly. However, in other environments backwards state predictions are not always possible for a given state-action pair—meaning hypothetical actions need to be selected more carefully.

By finding an equivalent hypothetical state-action pair for each experienced state-action pair we reduce the number of state-action pairs that an off-policy algorithm must learn to estimate—reducing the size of the space of values from $\mathcal{S} \times \mathcal{A}$ to $\mathcal{S} \times 1 + N$, where $N \in \mathcal{S} \times \mathcal{A}$ are the states where the hypothetical action is not computable because of edge cases that will be discussed shortly (e.g. stochastic environments, no unique backwards action or transitions that do not exist in the environment).

It is important to stress that as our mapping satisfies Equation 3, it also assumes the MDP

homomorphism properties developed by Ravindran and Barto (2001), meaning the policy we learn with our hypothetical state action pairs can be “lifted” van der Pol et al. (2020b) and used as a policy in a given experienced environment. Concretely, when performing value inference on a given state-action pair we apply Equation (6) to find the equivalent state and then predict a single action value with a Q-network.

We have assumed that for every state-action pair, at least one equivalent hypothetical state-action can be computed. This is often true but it is not guaranteed. For example, near borders in a grid world there is no way to travel left to a border state that has a border on its right. We only found this to significantly effect results in the tabular setting, which we mitigate by reverting to vanilla Q-learning table when necessary. This backup Q-learning value table is learned in parallel. Furthermore, in stochastic environments it may be difficult to predict equivalent states. In Section 6, we suggest recurrent state space models could be used to sample from a distribution of predicted transitions in future work (Hafner et al., 2019). Lastly, when forward dynamics are deterministic the previous state could still be unpredictable as a unique previous state for a given previous action may not exist (Yu et al., 2021). Again the model based RL provides some ideas for addressing this (Yu et al., 2021).

4 Experiments

We test our approach on a range of tasks, from tabular RL to using a convolutional DQN Silver et al. (2016). Where possible, we try to have overlap in our experiments with the MDP homomorphism literature—using Cartpole to overlap with the well known van der Pol et al. (2020b) and van der Pol et al. (2020a) as well as Predator Prey to overlap with van der Pol et al. (2020a). Hyperparameter details can be found in the Appendix. Shaded regions indicate the standard error.

4.1 Tabular Maze

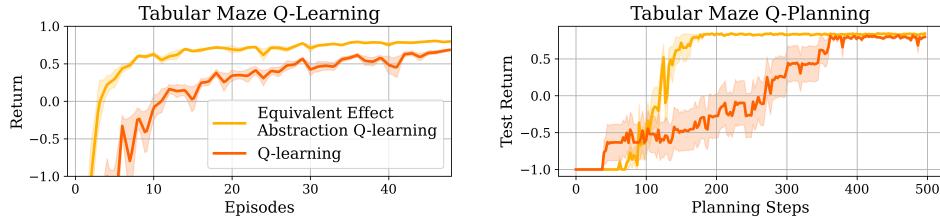


Figure 3: (a), (b) In a gridworld, removing redundancies in the Q-table improves sample efficiency. For the vanilla baselines, an agent must learn the value of each state-action pair for all actions in the action space. With equivalent effect abstraction, we reduce the number of Q-values that need to be learned. This yields improvements in both model-free Q-learning and model-based approaches Q-planning. 50 seeds are used for Q-learning while 10 seeds are used for Q-planning.

We begin with the maze environment from (Sutton and Barto, 2018, p. 165). The maze consists of 6×9 cells. The agent starts on the far west of the maze and must navigate east and find its way around walls in the middle of the environment. After passing the walls, the agent must then travel further east to reach a corridor, before moving north to reach a goal location. In this particular experiment, we assume a model of the environment is known beforehand—which we move through forwards and then backwards to create a homomorphic map.

We use an open-source Q-learning (Watkins and Dayan, 1992) implementation as a baseline² (along with the maze code from the same authors). We then adapt the Q-learning implementation to make use of a homomorphic map as shown in Algorithm 1. We show in Figure 3(a) that our approach compares favorably to vanilla Q-learning, converging much faster to the optimal policy. In the gridworld environment, we reduce the size of the state action space and hence the number of Q-values (when going left) from 216 to 67. In this case, the number of states in the reduced action space is not reduced by the cardinality of the action space exactly. This is because of the 13 states where it is not

²github.com/thehawkgriffith/dyna-maze

possible to reach them by traveling left because there is a border to the right.

To test planning efficiency, we performed a further experiment on the gridworld with model-based Q-planning (Sutton and Barto, 2018, p. 161). Our Q-planning method is equivalent to the Q-learning implementation but altered to learn by randomly sampling model transitions. At each Q-planning update we pause training and evaluate performance in a 100 step episode. Planning performance is shown in Figure 3(b). Similarly to the Q-learning results, leveraging knowledge of equivalent actions is able to significantly improve learning efficiency.

4.2 Cartpole

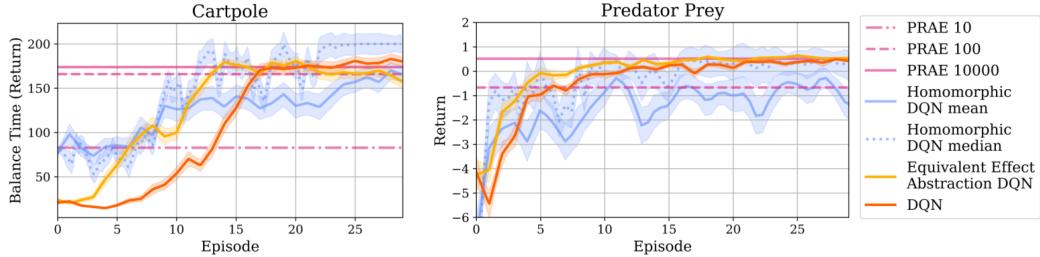


Figure 4: (a) Cartpole: we improve upon the DQN baseline significantly (the model is learned in the initial 3 episodes of experience—a different 3 episodes per repeat—that is shown on the x-axis of the plot). For vanilla DQN and equivalent effect abstraction we use 50 seeds, 5 seeds are used for each PRAE result and 30 seeds are used for MDP homomorphic networks. For clarity, PRAE 10 refers to using PRAE with 10 episodes of data to construct an environment model, which it then plans in. (b) Predator Prey: we learn a mapping to improve sample efficiency over vanilla DQN. Unlike in Cartpole the experience used to train the homomorphic map is not included in the plot for equivalent effect abstraction DQN. For equivalent effect abstraction we reuse one set of pre-trained backwards and forwards models for each RL run.

Next, we apply our approach to the Cartpole benchmark Brockman et al. (2016). In Cartpole, an agent controls a cart with a pendulum attached to it. The goal of the task is to learn to balance the pendulum upright by moving the cart horizontally left or right. The states are a four-dimensional vector (position, velocity, angle, angular velocity) and Cartpole has a discrete action space (move the cart left or right). To learn a model, we use 3 episodes of learning experience at the beginning of training to train the homomorphic map (two simple linear models, one for each action optimised with Adam (Kingma and Ba, 2015)) to generate equivalent state-action pairs for the opposite action to the one experienced. These initial training steps are included for equivalent effect abstraction in Figure 4(a). We integrate the learned homomorphism into a DQN implementation Silver (2015)³, with Q-values only being learned for an arbitrarily chosen hypothetical action.

We compare our approach to three baselines: vanilla DQN, MDP homomorphic Networks van der Pol et al. (2020b) and PRAE van der Pol et al. (2020a). For all experiments we use the baselines’ open source implementations and the libraries they build upon Stooke and Abbeel (2019). MDP homomorphic networks use specially constructed weights that are equivariant to environment symmetries (requiring prior knowledge of symmetries). PRAE trains a contrastive model to learn a mapping to a latent “plannable” MDP, that satisfies the definitions of an MDP homomorphism. PRAE then performs planning on the learned abstract MDP. For Cartpole, we learned a mapping for PRAE with datasets of 10, 100, 10000 episodes of random experience. We plot the average convergence performance of PRAE’s planning algorithm. In the Cartpole task, we did not need to avoid any hypothetical state action pairs that are not computable—meaning a reduction of the size of the state action space from $\mathcal{S} \times \mathcal{A}$ to $\mathcal{S} \times 1$.

As shown in Figure 4(a), in the low sample regime we improve upon both PRAE and vanilla DQN—with our approach converging at around episode 12 while vanilla Q-learning takes around 16 episodes to converge—note that this improvement includes the number of episodes required to learn

³<https://gist.github.com/Pocuston/13f1a7786648e1e2ff95bfad02a51521>

our mapping. We found that in many cases MDP homomorphic networks were able to achieve good performance but worst case runs brought down mean performance significantly. To demonstrate this, we plot the mean and median for MDP homomorphic networks (only the mean is plotted for other methods). The median MDP homomorphic network is also able to similarly improve upon the vanilla DQN baseline, by leveraging a practitioner’s prior knowledge of environment symmetries. MDP homomorphic networks are not necessarily a competing approach to equivalent effect abstraction and future work could conceivably combine these two approaches for even greater sample efficiency.

4.3 Stochastic Predator Prey

Following the seminal work of [van der Pol et al. \(2020b\)](#), we also benchmark our method on the Predator Prey environment, where an agent must chase a stochastically moving prey in a 2D world [van der Pol et al. \(2020a\)](#). The observations are a $7 \times 7 \times 3$ tensor encoding agent position and prey position. The objective of the agent is to catch the prey in fewest steps possible. Rewards are set to -0.1 unless the predator catches the prey, in which case the episode ends and a reward of 1 is provided.

For our method, we train action-dependent forwards and backwards models on a dataset of transitions created by taking random actions in the environment for 10^4 environment steps (equivalent to around 170 episodes of random experience). Note that the environment is stochastic, so learning a perfect environment model is impossible. We compare to the baselines introduced in Section 4.2. For PRAE, we benchmark with both 10000 and 100 episodes of experience data and perform planning to convergence. Similar to the Cartpole results, PRAE can be effective, but only when a relatively large amount of model training experience is available.

In Predator Prey, it is easy to learn a policy with around 5 episodes with a vanilla DQN. As a result, in this particular environment it is not practical to use equivalent effect abstraction as by the time a model can be learned the policy has already converged. However, what Figure 4(b) does show is that if a model can be obtained before online policy learning, then equivalent effect abstraction can deliver an improvement over vanilla DQN—even though the environment is stochastic. This is consistent with previous literature on MDP homomorphisms [van der Pol et al. \(2020b\)](#), which designed the Predator Prey environment to test whether their MDP homomorphism was robust to stochasticity.

4.4 MinAtar Asterix

MinAtar [Young and Tian \(2019\)](#) emulates a subset of games from the Atari benchmark [Bellemare et al. \(2013\)](#) but at a lower dimensionality—allowing policies similar to those found in the full Atari learning environment to be trained at lower computational cost. Consequently, we can do similar benchmarking to previous works on MDP homomorphisms (i.e. [van der Pol et al. \(2020b\)](#)’s Atari experiments) but using tasks that are easy to reproduce. We use Asterix to benchmark equivalent effect abstraction. In Asterix, an agent must discover moving treasure pots, while avoiding enemies. The effective actions are left, right, up, down and “do nothing”. Asterix has no obvious global symmetries, making it difficult for approaches that rely on symmetries [van der Pol et al. \(2020b\)](#).

Further, the MinAtar environments follow the “sticky action” protocol [Machado et al. \(2018\)](#), meaning actions are only executed with a probability of 90% while the other 10% of the time the previous action is repeated (Figure 5(a)). To further validate our algorithm in stochastic environments we increase the stickiness of actions to 25% in Figure 5(b).

In this section, we assume a model of the environment is known, which we construct programmatically to perform equivalent effect abstraction as if the environment were deterministic—meaning every time a sticky action is chosen the model’s prediction is incorrect (see supplementary code). Thus the question addressed here is *not* about learning backwards and forwards models as we did for Cartpole (section 4.2) and Predator Prey (section 4.3). Instead we seek to understand, given a model of the environment (which may be imperfect due to stochasticity), what approach can we take to perform value-based deep RL in the smallest number of learning steps.

We use the convolutional DQN [Silver et al. \(2016\)](#) implementation from the MinAtar repository [Young and Tian \(2019\)](#) as the backbone for equivalent effect abstraction, leaving their hyperparameters unchanged. As is the case in many games (and RL environments more generally), MinAtar allows a “do nothing” action. Interestingly, it is always possible to construct our homomorphism with

the “do nothing” as our hypothetical action, meaning we never have to fall back to vanilla Q-learning (which we did previously in the tabular maze when the homomorphism was impossible to construct). In Asterix it is always possible to reach every state by doing the hypothetical action—meaning we reduce the size of the state-action space by a factor of 6 (the number of actions in the action space). We compare equivalent effect abstraction to the vanilla DQN from the MinAtar [Young and Tian \(2019\)](#) repository. As a further baseline, we construct a DQN with a equivariant feature extractor, similar to the network used to learn Pong in [van der Pol et al. \(2020b\)](#). We implement feature extractors that are equivariant to both horizontal and vertical reflections. The hyperparameter selection process for this baseline is described in the Appendix.

Asterix is a relatively complex game requiring an agent to explore for moving treasures while simultaneously avoiding enemies. During this exploration, we show empirically that traversing the state space is much more efficient when the equivalences between different state-action pairs are known. Equivalent effect abstraction converges $3\times$ faster than vanilla Q-learning, consistent with out results in previous sections. As there is no obvious symmetry—in contrast to Pong for example—homomorphic DQN does not yield an advantage over the vanilla DQN.

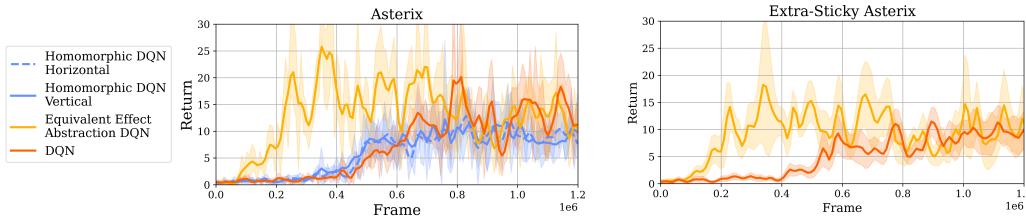


Figure 5: (a) In Asterix, Equivalent effect abstraction allows DQN to reach a return of 20 in around 3.5×10^5 frames, while vanilla DQN requires approximately (1×10^6) frames to reach this return. We run each method for five seeds. (b) Equivalent effect abstraction maintains its advantage in the presence of even more stochastic transitions (25% stochasticity as opposed to 10% previously). Both plots have been smoothed with a Gaussian kernel where $\sigma = 1$.

5 Related Work

([Ravindran and Barto, 2001](#)) developed a framework for abstraction under symmetry—deriving theoretical guarantees when lifting abstract policies. ([Givan et al., 2003](#)) proposed MDP model minimisation using bisimulation ([Larsen and Skou, 1991](#)), where states could be mapped to an abstract state if their expected transition dynamics and reward sequences were indistinguishable. These early applications of MDP homomorphisms required prior knowledge from a practitioner, with no obvious method for integrating homomorphisms into function approximation.

([Biza et al., 2021](#)) use bisimulation to inform structured hidden Markov model priors to infer a reduced discrete state space from learnt embeddings. Bisimulation can be relaxed to a similarity metric ([Ferns et al., 2011; Ferns and Precup, 2014](#))—these metrics have been used to condition embeddings with contrastive losses to generate invariant representations of states. ([Zhang et al., 2020](#)) augment their embedding encoder with an additional contrastive loss term that results in encoder distances forming an approximate bisimulation metric—allowing agents to perform control tasks in the presence of distractors. ([Rezaei-Shoshtari et al., 2022](#)) formalise continuous homomorphisms—building a homomorphism through a bisimulation metric. [Zhu et al. \(2022\)](#) use a contrastive loss and define state-action pairs with the same actions as equivalent in an abstract space. [Agarwal et al. \(2021\)](#) outline a new policy similarity metric that is used in conjunction with a contrastive loss—pulling together embeddings that result in similar trajectories. In general, contrastive learning achieves impressive results but can be sample hungry [van der Pol et al. \(2020a\)](#)).

More recently, homomorphisms in MDPs have been used in tasks with symmetries. PRAE ([van der Pol et al., 2020a](#)) trained a contrastive world model and accompanying loss function that satisfies the transition dynamics definition (Equation (1)) by design. ([van der Pol et al., 2020b](#))

derived a method for learning network weights that are equivariant to environment symmetries but their algorithm requires a practitioner to hardcode symmetric groups beforehand. Similar approaches have been adopted for environments with continuous symmetries Wang et al. (2022). Another recent approach to using MDP homomorphisms is detailed in (Biza and Platt, 2019). Here the problem of finding MDP homomorphisms is approached using Online Partition Iteration—the mappings from state to abstract states are learnt by predicting which partition a state should fall into given an action, and refining the partitions through splitting. Lastly, (Asadi et al., 2019) and (Lyu et al., 2023) explore equivalences where state-actions pairs lead to separate states but with the same probabilities.

Perhaps the most relevant related approach is the use of afterstates (Sutton and Barto, 2018, p. 136). Afterstates slightly shift an MDP out of phase with conventional state transitions, creating environment states that are in-between the initial effect of a policy’s action and the reaction of the environment to said action. Applications of the afterstates framework have generally been constrained to board games Tesauro et al. (1995) and focus on dealing with the stochasticity of an opponent rather than improved sample efficiency Antonoglou et al. (2021). Also closely related, (Misra et al., 2020) elegantly deploy a similar concept—using contrastive learning to learn an abstraction that groups together state-action pairs that will pass through or have passed through same state with the same previous action. However, this approach is not necessarily scalable to common RL environments such as Atari and classic control tasks where experiencing the exact same state twice is unlikely.

More broadly, model-based RL has enabled sample-efficient superhuman performance in Atari (Hafner et al., 2020), but the number of planning steps required to learn a policy is still large. Backwards models have been proposed to improve sample efficiency of world model representation learning (Yu et al., 2021), which would be interesting if integrated with equivalent effect abstraction. Our work is related to approaches that learn data augmentation policies for RL (Raileanu et al., 2021), however instead of augmenting experience with a more diverse distribution of states, we are trying to learn values for a narrower distribution of hypothetical states.

6 Limitations and Future Work

Equivalent effect abstraction requires a dynamics model. For our approach to be practical, the number of transitions required to learn the environment model must be smaller than those required to learn a policy—this is true in many environments (see Cartpole results) but it is not guaranteed (see Predator Prey results). What is more, despite recent progress Hafner et al. (2019, 2020); Saxena et al. (2021); Ye et al. (2021), learning a dynamics model can be difficult to learn for two reasons.

Stochastic dynamics can make environment transitions multimodal, meaning point predictions of future or previous states are not adequate. In theory, this can be dealt with using stochastic models Hafner et al. (2019) of forwards and backwards dynamics, that could be sampled when abstracting state-action pairs. Even worse, dynamics can be completely unpredictable Kendall and Gal (2017); Mavor-Parker et al. (2021), in which case uncertainty predictions could be used to signal that equivalent effect abstraction should temporarily revert to vanilla Q-learning. Further, backwards transitions may be unpredictable because an equivalent state does not exist—see Section 1(Yu et al., 2021). Uncertainty predictions could also be used to avoid out-of-distribution mappings Kendall and Gal (2017). We note that there is a silver lining for equivalent effect abstraction, which is that only one step transition models are required, avoiding the challenge of learning multi-step models Saxena et al. (2021).

Equivalent effect abstraction is formulated within the framework of value based RL using discrete actions. While many of the recent breakthroughs in deep RL are value based (e.g. Silver et al. (2017), Badia et al. (2020)), actor-critic approaches Schulman et al. (2017); Mnih et al. (2016) are often the natural choice for control tasks. Embedding equivalent effect abstraction into actor-critic architectures is a fruitful avenue for future research. It is also conceivable to formulate equivalent effect abstraction within a continuous action space by simply discretising the action space, which has allowed discrete action space methods to obtain state of the art performance on RL tasks Banino et al. (2021). Additionally, equivalent effect abstraction could also be integrated into existing homomorphic MDP methods that rely on symmetries to further reduce the size of the abstract state-action space van der Pol et al. (2020b). In practice, this would mean using a homomorphic Q-network van der Pol et al. (2020b) but modified to only have one hypothetical action.

7 Conclusion

Equivalent effect abstraction is a simple method that reduces the size of a state-action space. It is easy to implement and, provided with dynamics models, it requires no prior knowledge of environment symmetries. We show that equivalent effect abstraction improves the sample efficiency of policy learning in tabular environments, control tasks with continuous state spaces, stochastic deep RL environments, and in MinAtar’s Asterix. An exciting next step would be to integrate equivalent effect abstraction into model-based RL (Hafner et al., 2020; Yu et al., 2021) to improve planning efficiency.

References

- Agarwal, R., Machado, M. C., Castro, P. S., and Bellemare, M. G. (2021). Contrastive behavioral similarity embeddings for generalization in reinforcement learning. *arXiv preprint arXiv:2101.05265*.
- Anand, A., Grover, A., Mausam, M., and Singla, P. (2015). Asap-uct: Abstraction of state-action pairs in uct. In *Proceedings of the 24th International Conference on Artificial Intelligence*, pages 1509–1515.
- Antonoglou, I., Schrittwieser, J., Ozair, S., Hubert, T. K., and Silver, D. (2021). Planning in stochastic environments with a learned model. In *International Conference on Learning Representations*.
- Asadi, M., Talebi, M. S., Bourel, H., and Maillard, O.-A. (2019). Model-based reinforcement learning exploiting state-action equivalence. In *Asian Conference on Machine Learning*, pages 204–219. PMLR.
- Badia, A. P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, Z. D., and Blundell, C. (2020). Agent57: Outperforming the atari human benchmark. In *International Conference on Machine Learning*, pages 507–517. PMLR.
- Banino, A., Badia, A. P., Walker, J., Scholtes, T., Mitrovic, J., and Blundell, C. (2021). Coberl: Contrastive bert for reinforcement learning. *arXiv preprint arXiv:2107.05431*.
- Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279.
- Biewald, L. (2020). Experiment tracking with weights and biases. Software available from wandb.com.
- Biza, O. and Platt, R. (2019). Online abstraction with mdp homomorphisms for deep learning. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*.
- Biza, O., Platt, R., van de Meent, J., and Wong, L. (2021). Learning discrete state abstractions with deep variational inference. *Advances in Approximate Bayesian Inference*.
- Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.
- Ferns, N., Panangaden, P., and Precup, D. (2011). Bisimulation metrics for continuous markov decision processes. *SIAM Journal on Computing*, 40(6):1662–1714.
- Ferns, N. and Precup, D. (2014). Bisimulation metrics are optimal value functions. In *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence*, UAI’14, page 210–219, Arlington, Virginia, USA. AUAI Press.
- Givan, R., Dean, T., and Greig, M. (2003). Equivalence notions and model minimization in markov decision processes. *Artificial Intelligence*, 147(1-2):163–223.
- Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. (2019). Learning latent dynamics for planning from pixels. In *International conference on machine learning*, pages 2555–2565. PMLR.
- Hafner, D., Lillicrap, T. P., Norouzi, M., and Ba, J. (2020). Mastering atari with discrete world models. In *International Conference on Learning Representations*.

- Kendall, A. and Gal, Y. (2017). What uncertainties do we need in bayesian deep learning for computer vision? *Advances in neural information processing systems*, 30.
- Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *ICLR (Poster)*.
- Larsen, K. G. and Skou, A. (1991). Bisimulation through probabilistic testing. *Information and computation*, 94(1):1–28.
- Liberzon, D. (2011). *Calculus of variations and optimal control theory: a concise introduction*. Princeton university press.
- Lyu, Y., Côme, A., Zhang, Y., and Talebi, M. S. (2023). Scaling up q-learning via exploiting state-action equivalence. *Entropy*, 25(4):584.
- Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M., and Bowling, M. (2018). Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562.
- Mavor-Parker, A. N., Young, K. A., Barry, C., and Griffin, L. D. (2021). How to stay curious while avoiding noisy TVs using aleatoric uncertainty estimation. *Proceedings of the 39th International Conference on Machine Learning*, 162:15220–15240.
- Misra, D., Henaff, M., Krishnamurthy, A., and Langford, J. (2020). Kinematic state abstraction and provably efficient rich-observation reinforcement learning. In *International conference on machine learning*, pages 6961–6971. PMLR.
- Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR.
- pandas development team, T. (2020). pandas-dev/pandas: Pandas.
- Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch.
- Raileanu, R., Goldstein, M., Yarats, D., Kostrikov, I., and Fergus, R. (2021). Automatic data augmentation for generalization in reinforcement learning. *Advances in Neural Information Processing Systems*, 34.
- Ravindran, B. and Barto, A. G. (2001). Symmetries and model minimization in markov decision processes.
- Rezaei-Shoshtari, S., Zhao, R., Panangaden, P., Meger, D., and Precup, D. (2022). Continuous mdp homomorphisms and homomorphic policy gradient. *arXiv preprint arXiv:2209.07364*.
- Saxena, V., Ba, J., and Hafner, D. (2021). Clockwork variational autoencoders. *Advances in Neural Information Processing Systems*, 34:29246–29257.
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.
- Silver, D. (2015). Lecture notes in reinforcement learning. *Computer Science Department, University College London*.
- Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489.
- Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017). Mastering the game of go without human knowledge. *nature*, 550(7676):354–359.
- Stooke, A. and Abbeel, P. (2019). rlpyt: A research code base for deep reinforcement learning in pytorch. *arXiv preprint arXiv:1909.01500*.

- Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.
- Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., Casas, D. d. L., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., et al. (2018). Deepmind control suite. *arXiv preprint arXiv:1801.00690*.
- Tesauro, G. et al. (1995). Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68.
- van der Pol, E., Kipf, T., Oliehoek, F. A., and Welling, M. (2020a). Plannable approximations to mdp homomorphisms: Equivariance under actions. In *AAMAS*.
- van der Pol, E., Worrall, D., van Hoof, H., Oliehoek, F., and Welling, M. (2020b). Mdp homomorphic networks: Group symmetries in reinforcement learning. *Advances in Neural Information Processing Systems*, 33:4199–4210.
- Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. (2019). Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354.
- Wang, D., Walters, R., and Platt, R. (2022). So (2) equivariant reinforcement learning. In *International conference on learning representations (ICLR)*.
- Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3):279–292.
- Ye, W., Liu, S., Kurutach, T., Abbeel, P., and Gao, Y. (2021). Mastering atari games with limited data. *Advances in Neural Information Processing Systems*, 34.
- Young, K. and Tian, T. (2019). Minatar: An atari-inspired testbed for thorough and reproducible reinforcement learning experiments. *arXiv preprint arXiv:1903.03176*.
- Yu, C., Li, D., Hao, J., Wang, J., and Burgess, N. (2021). Learning state representations via retracing in reinforcement learning. *arXiv preprint arXiv:2111.12600*.
- Zhang, A., McAllister, R. T., Calandra, R., Gal, Y., and Levine, S. (2020). Learning invariant representations for reinforcement learning without reconstruction. In *International Conference on Learning Representations*.
- Zhu, Z.-M., Jiang, S., Liu, Y.-R., Yu, Y., and Zhang, K. (2022). Invariant action effect model for reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 9260–9268.

A Appendix

A.1 Impact Statement

We do not believe that EEA introduces any novel dangers or concerns in terms of broader impact. Like all reinforcement learning algorithms, our agents likely do not generalise very well and could behave unexpectedly when deployed in real world environments. We suggest extreme caution when using any RL algorithm for consequential applications.

A.2 Hyperparameter Search

Below we show the hyperparameters swept through for homomorphic MDP, DQN and Equivalent Effect Abstraction agents, broken down by environment. The PRAE architectures were generally kept the same as the hyperparameters provided in ([van der Pol et al., 2020a](#)), with the exception of the learning rate which we evaluate at 0.0001, 0.001 and 0.1

A.2.1 Sutton and Barto Tabular Gridworld

We use the hyperparameters specified in ([Sutton and Barto, 2018](#), p. 165): namely, Learning Rate = 0.1, $\gamma = 0.95$ and $\epsilon = 0.1$

A.2.2 Cartpole

Table 1: Hyperparameters swept through for the Cartpole environment. Learning rate decay refers to decaying the learning rate by a factor of ten at after a specified number of episodes have elapsed.

Hyperparameter	Values
Learning Rate	0.00001, 0.0001, 0.001, 0.01
ϵ decay schedule	No decay, exponential $\tau = \frac{-1}{200}$
γ	0.8, 0.99
Activation	ReLU, tanh
Learning Rate decay	No decay, 5, 10, 15, 20

Homomorphic MDP best hyperparameters: Learning Rate = 0.001, ϵ decay schedule = No decay, $\gamma = 0.8$, activation = tanh, Learning Rate decay = No decay

Equivalent Effect Abstraction best hyperparameters: Learning Rate = 0.001, ϵ decay schedule = No decay, $\gamma = 0.8$, activation = tanh, Learning Rate decay = 10

Vanilla DQN best hyperparameters: Learning Rate = 0.001, ϵ decay schedule = No decay, $\gamma = 0.8$, activation = tanh, Learning Rate decay = 15

A.2.3 Stochastic Predator Prey

Table 2: Hyperparameters swept through for the Predator Prey environment.

Hyperparameter	Values
Learning Rate	0.0001, 0.001, 0.01
γ	0.8, 0.99

Homomorphic MDP best hyperparameters: Learning Rate = 0.001, $\gamma = 0.99$

Equivalent Effect Abstraction best hyperparameters: Learning Rate = 0.01, $\gamma = 0.8$

Vanilla DQN best hyperparameters: Learning Rate = 0.001, $\gamma = 0.99$

A.2.4 Minatar

All hyperparameters were kept at the tuned DQN values provided in Young and Tian (2019). For the Homomorphic MDP network, we use a symmetric group for reflection about the vertical axis when training on Breakout; on Asterix we show results for both reflection about the vertical and horizontal axes.

A.3 Model Architectures

A.3.1 Cartpole

Listing 1: Homomorphic MDP Network (van der Pol et al., 2020b)

```
BasisLinear*(repr_in=4, channels_in=1, repr_out=2, channels_out=64)
ReLU() / tanh()
Basislinear(repr_in=2, channels_in=64, repr_out=2, channels_out=64)
ReLU() / tanh()
BasisLinear(repr_in=2, channels_in=64, repr_out=2, channels_out=1)
```

*BasisLinear refers to the symmeterised layers used in (van der Pol et al., 2020b) to create homomorphic networks. This network is identical to the Cartpole network presented in that paper, but with only one output head that outputs state-action values.

Listing 2: Value Network architecture for DQN and Equivalent Effect Abstraction

```
Linear(input_size=4, output_size=1024)
tanh()
Linear(input_size=1024, output_size=1024)
tanh()
Linear(input_size=8, output_size=1024)
tanh()
Linear(input_size=1024, output_size=2)
```

Listing 3: Transition Model Architecture for Equivalent Effect Abstraction

```
Linear(input_size=2, output_size=2)
```

Listing 4: PRAE Architectures (van der Pol et al., 2020a)

```
# state encoder
Linear(input_size=4, output_size=64)
ReLU()
Linear(input_size=64, output_size=32)
ReLU()
Linear(input_size=32, output_size=50)
#action encoder
Linear(input_size=54, output_size=100)
ReLU()
Linear(input_size=100, output_size=2)
# reward prediction network
Linear(input_size=50, output_size=64)
ReLU()
Linear(input_size=64, output_size=1)
```

A.3.2 Predator Prey

Listing 5: Homomorphic MDP Network (van der Pol et al., 2020b)

```
BasisConv2d(repr_in=1, channels_in=1, repr_out=4, channels_out=4,
filter_size=(7,7), stride=2, padding=0)
ReLU()
BasisConv2d(repr_in=4, channels_in=4, repr_out=4, channels_out=8,
filter_size=(5,5), stride=1, padding=0)
ReLU()
GlobalMaxPool()
```

```

BasisLinear(repr_in=4, channels_in=8, repr_out=4, channels_out=128)
ReLU()
BasisLinear(repr_in=4, channels_in=8, repr_out=4, channels_out=128)
ReLU()
BasisLinear(repr_in=4, channels_in=128, repr_out=5, channels_out=1)

```

This is again the same network used in [van der Pol et al. \(2020b\)](#), albeit with a different output head.

[Listing 6: Value Network Architecture for DQN and Equivalent Effect Abstraction](#)

```

Linear(input_size=441, output_size=1024)
ReLU()
Linear(input_size=1024, output_size=8)
ReLU()
Linear(input_size=8, output_size=1024)
ReLU()
Linear(input_size=1024, output_size=5)

```

[Listing 7: Transition Model Architecture for Equivalent Effect Abstraction](#)

```

Linear(input_size=882, output_size=512)
ReLU()
Linear(input_size=512, output_size=8)
ReLU()
Linear(input_size=8, output_size=512)
ReLU()
Linear(input_size=512, output_size=441)

```

[Listing 8: PRAE Architectures \(\[van der Pol et al., 2020a\]\(#\)\)](#)

```

# state encoder
Linear(input_size=441, output_size=64)
ReLU()
Linear(input_size=64, output_size=32)
ReLU()
Linear(input_size=32, output_size=50)
#action encoder
Linear(input_size=54, output_size=100)
ReLU()
Linear(input_size=100, output_size=2)
# reward prediction network
Linear(input_size=50, output_size=64)
ReLU()
Linear(input_size=64, output_size=1)

```

A.3.3 MinAtar

[Listing 9: Homomorphic MDP Network \(\[van der Pol et al., 2020b\]\(#\)\)](#)

```

BasisConv2d(repr_in=1, channels_in=4, repr_out=2, channels_out=16,
filter_size=(3,3), stride=1, padding=0)
ReLU()
Linear(input_size=1408, output_size=256)
ReLU()
Linear(input_size=256, output_size=6)

```

[Listing 10: Value Network Architecture for DQN and Equivalent Effect Abstraction](#)

```

Conv2d(channels_in=4, channels_out=16, filter_size=(3,3), stride=1, padding=0)
ReLU()
Linear(input_size=1024, output_size=128)
ReLU()
Linear(input_size=128, output_size=6)

```

A.4 Additional Minatar Results with an Incorrect Model

Here we present some initial results where we tried to naively reuse the Asterix dynamics model for Breakout (ignoring upwards and downwards movements as they are not possible in Breakout). However, this model is imperfect as it always assumes the ball is reflected from the paddle back towards the direction it came in, which is not always true in Breakout depending on the side of the paddle that the ball hits. What is more, the vanilla DQN agent cannot solve the game and all value based methods almost instantly converge to a suboptimal policy, suggesting that alternative learning algorithms should be used in the future on this environment.

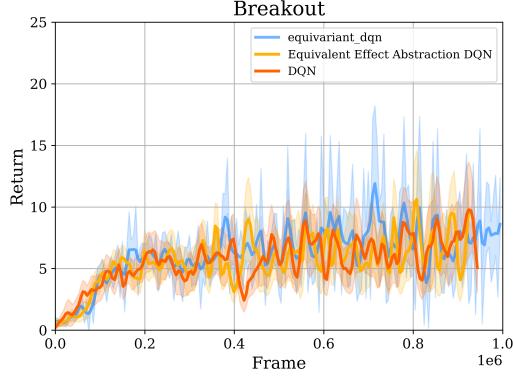


Figure 6: Here we get similar results to [van der Pol et al. \(2020b\)](#) using our imperfect model. On this particular environment all versions of DQN converge almost instantly to a suboptimal policy. To obtain more meaningful results, future work should integrate equivalent effect abstraction into actor-critic methods that are able to solve Breakout [Young and Tian \(2019\)](#).

B Syncing Log arrays

For the sticky MinAtar results we logged frames and return values with wandb while performing our experiments [Biewald \(2020\)](#). We performed this logging suboptimally as we logged steps for frames and returns separately—meaning we ended up with two arrays where one was NaN when the other was logging a float and vice versa. We used Panda’s ([pandas development team, 2020](#)) linear interpolator to estimate the NaN values in our final plots. This process does not effect our results significantly as the points were logged at a very high frequency.