INAPPLICABLE ACTIONS LEARNING FOR KNOWLEDGE TRANSFER IN REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement Learning (RL) algorithms are known to scale poorly to environments with many available actions, requiring numerous samples to learn an optimal policy. The traditional approach of considering the same fixed action space in every possible state implies that the agent must understand, while also learning to maximize its reward, to ignore irrelevant actions such as *inapplicable actions* (i.e. actions that have no effect on the environment when performed in a given state). Knowing this information can help reduce the sample complexity of RL algorithms by masking the inapplicable actions from the policy distribution to only explore actions relevant to finding an optimal policy. This is typically done in an ad-hoc manner with hand-crafted domain logic added to the RL algorithm. In this paper, we propose a more systematic approach to introduce this knowledge into the algorithm. We (i) standardize the way knowledge can be manually specified to the agent; and (ii) present a new framework to autonomously learn these state-dependent action constraints jointly with the policy. We show experimentally that learning inapplicable actions greatly improves the sample efficiency of the algorithm by providing a reliable signal to mask out irrelevant actions. Moreover, we demonstrate that thanks to the transferability of the knowledge acquired, it can be reused in other tasks to make the learning process more efficient.

1 INTRODUCTION

The field of Deep Reinforcement Learning (DRL), using neural networks as function approximators in Reinforcement Learning (RL) algorithms, has seen many successes in recent years (Mnih et al., 2015; Silver et al., 2016; Vinyals et al., 2017). Despite the impressive results this learning technique has shown, DRL is often criticized for being data hungry and sample inefficient. These algorithms are notorious to scale poorly to larger problems as they require a large amount of time and resources to learn how to perform relatively simple tasks.

Incorporating domain knowledge has been shown to be a valid approach to increase the learning effectiveness of DRL algorithms (Vinyals et al., 2017; Kool et al., 2018; Even-Dar et al., 2006; Barreto et al., 2018; Fernández et al., 2010; Spooner et al., 2021). We focus our attention to a specific type of knowledge in the form of *inapplicable actions*, i.e. actions that do not modify the environment when performed in a particular state. While this knowledge is explicitly defined in other types of planning systems, as preconditions of actions Ghallab et al. (2004), it is rarely used within the RL community. This information that can be interpreted as the *rules of the game*, can help reduce the sample complexity of RL algorithms by pruning irrelevant state-action pairs from the search space the RL agent needs to explore. This *action masking* technique contributed to several breakthroughs in the field (Vinyals et al., 2017; Kool et al., 2018). Unfortunately, a systematic way to incorporate this type of knowledge into the RL algorithm seems to be lacking from the literature.

For cases where the applicability of an action in a given state is unknown, we propose to turn one of RL algorithms' weaknesses into an advantage and use the data collected by the RL agent through exploration, to train a model able to identify whether an action is (in)applicable in a given state. As the classifier learns to identify the features encoding the constraints of the environment, it provides an increasingly accurate signal to the RL algorithm to mask out inapplicable actions, leading to a more sample efficient algorithm.

By learning the task-agnostic constraints, the classifier not only helps improving the agent's learning process, but it also encapsulates knowledge about the environment in a manner that is both interpretable and transferable. The information acquired solving a particular task can be shared to new tasks. Instead of solving the new problem from scratch, the RL algorithm makes use of the trained classifier to reduce the amount of time spent exploring inapplicable actions.

In this work, we direct our efforts to improve the sample efficiency of Policy Gradient algorithms that have been proven to converge when action masking is used (Huang & Ontañón, 2022). Our main contributions are: **(i)** propose an extended MDP formulation, State Dependent Action Space MDP (SDAS-MDP), that incorporates an explicit representation of actions constraints; **(ii)** show how various levels of hand-coded knowledge about the environment can significantly decrease the time required to learn a policy; **(iii)** provide an algorithm to jointly train a policy and an inapplicable actions classifier and empirically show that our method is more sample efficient than standard Policy Gradient algorithms; and **(iv)** present a new transfer learning technique for RL, where the knowledge previously acquired about the constraints of the environment is shared across domains by reusing a trained inapplicable actions classifier.

2 BACKGROUND

We consider the T-episodic Reinforcement Learning problem in a Markov Decision Process with a discrete actions space, denoted as $\langle \mathbb{S}, \mathbb{A}, P(s,a,s'), R(s,a,s'), T, \mu_0, \gamma \rangle$. $\mathbb{S}$ is the state space, $\mathbb{A}$ the discrete action space, $P : \mathbb{S} \times \mathbb{A} \times \mathbb{S} \rightarrow [0,1]$ the Markovian state transition probability, $R : \mathbb{S} \times \mathbb{A} \times \mathbb{S} \rightarrow [0,1]$ the reward function, T the maximum episode length, μ_0 the initial state distribution, and γ the discount factor. A stochastic policy $\pi_\theta : \mathbb{S} \times \mathbb{A} \rightarrow [0,1]$ characterizes a function, parametrized by θ, assigning a probability to an action given a state. The objective of an RL agent is to learn the optimal policy π_θ such that its expected discounted return is maximized.

$$J = \mathbb{E}_{\substack{s_0 \sim \mu_0 \\ a_t \sim \pi_\theta(\cdot|s_t) \\ s_{t+1} \sim P(\cdot|s_t, a_t)}} \left[\sum_{t=0}^{T-1} \gamma^t R(s_t, a_t, s_{t+1}) \right] = \mathbb{E}_{\mu_0, \pi_\theta} \left[\sum_{t=0}^{T-1} \gamma^t R(s_t, a_t, s_{t+1}) \right]$$

Where s_{t+1} is sampled from the state transition probability distribution $P(\cdot|s_t, a_t)$, a_t from the policy $\pi_\theta(\cdot|s_t)$ and s_0 from the initial state distribution μ_0.

The family of Policy Gradient algorithms, initially introduced in Sutton et al. (1999), aims at finding the optimal value for parameter θ such that the resulting policy generates the maximum expected returns. The parameter θ is updated in a gradient ascent fashion $\theta_{i+1} = \theta_i + \alpha \nabla_\theta J_i$ where the policy gradient $\nabla_\theta J$ has been shown to take the form:

$$\nabla_\theta J = \mathbb{E}_{\mu_0, \pi_\theta} \left[A_t^{\pi_\theta} \nabla_\theta \log \pi_\theta(a_t|s_t) \right]; \quad \text{with } A_t^{\pi_\theta} = Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t) \tag{1}$$

$$Q^{\pi_\theta}(s,a) = \mathbb{E}_{\pi_\theta} \left[\sum_{k=0}^{T-1} \gamma^k R(s_{t+k}, a_{t+k}, s_{t+k+1}) \big| s_t = s, a_t = a \right]$$

$$V^{\pi_\theta}(s) = \mathbb{E}_{\pi_\theta} \left[\sum_{k=0}^{T-1} \gamma^k R(s_{t+k}, a_{t+k}, s_{t+k+1}) \big| s_t = s \right]$$

$V^{\pi_\theta}(s)$ is the value function and represents the expected returns received starting in state s and following the policy π_θ. Similarly, $Q^{\pi_\theta}(s,a)$ is the action value function and indicates the expected returns when action a is taken in state s and the policy π_θ is followed thereafter.

3 MASKING INAPPLICABLE ACTIONS

In many real-world problems, the set of actions that an agent can perform varies depending on the state of the environment: $\mathbb{A}(\boldsymbol{s})$. These variations can be explained by multiple factors such as expiration of resources (an agent running out of inventory will no longer be able to sell its product), the rules of the game (not being able to move its rook diagonally when playing chess), etc. Despite being supported by the standard Markov Decision Process framework, the ability for each state $\boldsymbol{s}$ to have its own set of feasible actions seems to have been mainly ignored by standard RL algorithms. Their implementations typically assume a fixed set, concatenating all the possible actions that an agent can execute throughout the episode. We focus our attention to *inapplicable actions* (Definition 3.1), which are actions that have no effect on the environment when executed in a given state [1].

Definition 3.1 *Inapplicable action*

Given a T-episodic MDP defined as $\langle \mathbb{S}, \mathbb{A}, P(\boldsymbol{s}, a, \boldsymbol{s}'), R(\boldsymbol{s}, a, \boldsymbol{s}'), T, \mu_0, \gamma \rangle$ *and a distance measure between two states:* $d(s, s')$*, an action is said to be* **inapplicable in a given state** *if the distance between the state of the environment before and after the action was taken is lower than a small value* ε*. We use* $\mathbb{I}(\boldsymbol{s})$ *to denote the set of inapplicable actions in state* $\boldsymbol{s}$*.*

$$\forall a \in \mathbb{I}(\boldsymbol{s}) \subseteq \mathbb{A} \; ; \; P(d(\boldsymbol{s}_{t+1}, \boldsymbol{s}) \leq \varepsilon | \boldsymbol{s}_t = \boldsymbol{s}, a_t = a) = 1 \tag{2}$$

This formulation of *inapplicable action* assumes a certain level of observability of the environment. While full observability is not necessary, the agent requires however to have visibility on the features of the environment that the action a has effect on. Addressing this limitation could be the object of future work.

Leveraging the definition of *inapplicable actions*, we consider a new class of augmented MDPs with an additional component $C(\boldsymbol{s}, a)$, called the applicability function, returning the probability of an action a to be applicable in state $\boldsymbol{s}$. These new MDPs, referred to as State Dependent Action Space MDP (SDAS-MDP), provide a direct signal to identify *inapplicable actions* that can be used to prune the search space of the problem and therefore improve the sample efficiency of RL algorithms.

Definition 3.2 *State Dependent Action Space MDP (SDAS-MDP)*

Given a T-episodic MDP defined as $\langle \mathbb{S}, \mathbb{A}, P(\boldsymbol{s}, a, \boldsymbol{s}'), R(\boldsymbol{s}, a, \boldsymbol{s}'), T, \mu_0, \gamma \rangle$*, and* $C : \mathbb{S} \times \mathbb{A} \rightarrow [0, 1]$ *the* **applicability function** *returning the probability that the action* a *is applicable in state* $\boldsymbol{s}$*.*

The State Dependent Action Space MDP takes the form:

$$\begin{gathered}\langle \mathbb{S}, \mathbb{A}, P(\boldsymbol{s}, a, \boldsymbol{s}'), R(\boldsymbol{s}, a, \boldsymbol{s}'), T, \mu_0, \gamma, C(\boldsymbol{s}, a) \rangle \\ \textit{with } C(\boldsymbol{s}, a) = P(a \notin \mathbb{I}(\boldsymbol{s}))\end{gathered} \tag{3}$$

The SDAS-MDP formulation makes the definition of an action masking function $\boldsymbol{m} : \mathbb{S} \rightarrow \mathbb{R}^{|\mathbb{A}|}$ straightforward in the discrete action space case by simply calling the applicability function $C(\cdot, \cdot)$ for every element of the action space. As C returns the probability of being applicable, we evaluate whether this value is below a certain cut-off threshold τ passed as parameter of the algorithm. This threshold is typically set to 0.5 but can be adjusted depending on the problem, to make the identification of inapplicable actions more specific or more sensitive. The mask generated is thus a vector of 0 for inapplicable actions, and 1 for applicable actions that will be multiplied by the probability distribution returned by the policy. The distribution is then re-normalized using the `softmax` function. For most of the existing policy gradient algorithms however, the output of the policy is the logit associated with each of the actions. In such case, the procedure consists of replacing the logit of the inapplicable actions by a very large negative number.

$$\begin{gathered}\pi_\theta^{\text{mask}}(\cdot | \boldsymbol{s}, \pi_\theta, \boldsymbol{m}(\boldsymbol{s})) = \texttt{softmax}(\boldsymbol{m}(\boldsymbol{s}) \cdot \pi_\theta(\cdot | \boldsymbol{s})) \\ \text{with} \quad m_i(\boldsymbol{s}) = \begin{cases} 0 & \text{if } C(\boldsymbol{s}, a_i) < \tau \\ 1 & \text{otherwise} \end{cases} \quad \forall i \in \{1..|\mathbb{A}|\}\end{gathered} \tag{4}$$

In the following sections, we look at how the applicability function can be formalized to be used by Policy Gradient algorithms and how knowledge can be passed into the algorithm to help make the learning process more efficient.

3.1 Inapplicable actions masking via domain knowledge

The knowledge about inapplicable actions is, in some cases, directly associated with the rules specified by the environment (e.g the rules of the game) or simply common sense (e.g trying to drop an

[1]These are different from forbidden or terminal actions typically implemented with reward shaping or by interrupting the episode abruptly.

object that we do not hold). While it seems trivial for a human to use implicit knowledge previously acquired to explore only the relevant actions, RL algorithms do not have this information available by default.

People have started using the concept of actions masking to improve the learning efficiency of RL algorithm, but the rules are typically hand-coded in the algorithm itself (Vinyals et al., 2017; Kool et al., 2018), introducing environment specific logic into the RL algorithm. Implementations such as RLLib Parametric Action Spaces (Liang et al., 2017) takes one step towards abstracting away the action masking logic from the RL policy and propose to include the mask in the observation space. However, no consensus has been found on the best way to encode this information in a generic way.

Similarly to the concept of *pre-condition* defined in the STRIPS language (Fikes & Nilsson, 1971) and used by the Automated Planning community, we argue that the *applicability function* C is domain specific and should therefore be part of the environment definition. We thus propose to extend the OpenAI Gym environment interface (Brockman et al., 2016) to include a new standardized method `is_applicable` returning whether an action is applicable in the current state of the environment. This method formalizing the applicability function C of the SDAS-MDP will be called by the RL algorithm to create the actions mask to only sample applicable actions from the policy.

Algorithm 1 PseudoCode: Policy Gradient with inapplicable actions learning

Input: policy parameter: θ_0, classifier parameter: ϕ_0, classifier exploration threshold: ϵ_0, training flag: `train`
Output: trained policy parameters θ_K, trained classifier parameters ϕ_K

$\mathbb{D} \leftarrow \{\}$ ▷ *instantiate the rollout buffer*
for $k = 1, 2, \ldots, K$ **do**
  for worker $= 1, 2, \ldots, N$ **do**
    COLLECTTRAJECTORIES(π_{θ_k}, $\mathcal{C}_{\phi_k}$, ϵ_k, $\mathbb{D}$). ▷ *with $\mathcal{C}_{\phi_k}$ the actions classifier*
    Compute returns $\hat{R}_t$ and advantage estimates $\hat{A}_t$ using current value function V_k
  end for
  for epochs=$1, 2, \ldots, M$ **do**
    Sample data from the replay buffer $\mathbb{D}$
    Optimize $\mathcal{L}^{\text{Policy}}$ w.r.t θ via SGD with Adam optimizer
    if `train`=`TRUE` **then**
      Balance applicable and inapplicable samples ▷ *balance dataset*
      Optimize $\mathcal{L}^{\mathcal{C}}$ w.r.t ϕ via SGD with Adam optimizer ▷ *with $\mathcal{L}^{\mathcal{C}}$ the classifier loss*
    end if
  end for
  ϵ_{k+1} = SCHEDULEFUNCTION(ϵ_k, k)
end for
Return θ_K, ϕ_K

procedure COLLECTTRAJECTORIES(π_θ, $\mathcal{C}_\phi$, ϵ, $\mathbb{D}$, $\tau = 0.5$)
  Receive initial observation $\boldsymbol{s}_0$
  for $t = 1, 2, ..., T$ **do**
    $\boldsymbol{m}_t \leftarrow \mathbf{1}$
    if RANDOM$(0, 1) \leq \epsilon$ **then**
      for $a \in \mathbb{A}$ **do**
        $m_{t_a} \leftarrow \mathcal{C}_\phi(a, \boldsymbol{s}_t) \geq \tau$ ▷ *compute the actions mask*
      end for
    end if
    Select action $a_t \sim$ SOFTMAX$(\boldsymbol{m}_t \cdot \pi_\theta(\mathbf{s_t}))$ ▷ *apply the mask*
    Execute action a_t in the environment, observe reward r_t and new state $\boldsymbol{s}_{t+1}$
    $y_t \leftarrow \mathbf{1}_{\boldsymbol{s}_{t+1} \neq \boldsymbol{s}_t}$ ▷ *evaluate whether a_t was applicable*
    $\mathbb{D} \leftarrow \mathbb{D} \cup \{\langle \boldsymbol{s}_t, a_t, r_t, \boldsymbol{s}_{t+1}, y_t, \boldsymbol{m}_t \rangle\}$
  end for
end procedure

3.2 LEARNING INAPPLICABLE ACTIONS

For well understood tasks and environments, such as games where the rules are explicit and unambiguous, the definition of the applicability function is simple and straightforward to encode. However, in cases where the logic driving the (in)applicability of actions is unknown, or even partially understood, it becomes much more challenging to provide valid information to the RL algorithm to mask inapplicable actions. To address this limitation, we propose to learn the applicability function during training by leveraging the data collected by the agent to train the policy.

Decoupling the inapplicable actions learning from the policy training has the advantage of reducing the problem complexity. While the optimality of an action with respect to the total return is observed with a delay, it is possible to directly identify whether an action is applicable in a given state by evaluating the state of the environment before and after it was performed. The reduced problem of identifying the applicability of an action in a given state is thus easier to solve than finding the optimal action. Learning inapplicable actions also offers a more interpretable way to understand the policy learnt by the RL agent as we can now recognize that some actions are not only sub-optimal in certain states but also inapplicable, providing more information about the learnt policy.

The task of learning the applicability function can in fact be reduced to learning a classifier $\mathcal{C}_\phi$, parameterized by ϕ, that returns the probability for an action a to be applicable in a given state $\boldsymbol{s}$. It is possible to train the classifier via supervised learning jointly with the policy and thus provide an increasingly accurate mask of inapplicable actions to the RL algorithm.

We present in Algorithm 1 the steps of the algorithm training a classifier jointly with the policy and we detail below the important modifications made to the original algorithm (Sutton et al., 1999).

Inapplicable Actions Masking: The procedure to collect trajectories is modified to generate the mask using the classifier (`l.24`) and apply it to the policy output (`l.28`).

Exploration: To accommodate the fact that the classifier is learning and is therefore susceptible to produce false negatives (i.e masking out applicable actions) that could be part of an optimal policy, we include an exploration parameter ϵ driving the frequency at which the mask will be ignored (`l.23`). This parameter both helps the classifier collect positive and negative examples to improve its accuracy, but also drives how trusted the classifier can be. The value of ϵ can vary throughout the training as the classifier improves at identifying applicable actions.

Rollout Buffer: The data buffer collecting the rollouts is extended to include whether or not the action taken was applicable in the state the agent was in. This is computed by comparing the state of the environment before and after the action was performed (`l.30`). This value will be used as the output the classifier needs to predict. The mask used to select an action is also added to the data buffer, and is used to evaluate the effect of choosing the selected action.

Classifier Training: The classifier $\mathcal{C}_\phi$ is trained via supervised learning jointly with the policy π over multiple epochs, using the data from the rollout buffer. A key observation is that the policy is learning to act optimally and therefore avoiding taking inapplicable actions, while the classifier also helps filtering out invalid actions. This results in an imbalanced number of positive and negative examples of inapplicable actions in the buffer as the training progress and the agent converges towards an optimal policy. Classifiers are known to perform poorly on imbalanced datasets (Sun et al., 2009); to alleviate this issue we re-balance applicable and inapplicable actions samples `l.11` using a Weighted Random Sampling approach with the weight inversely proportional to the number of samples for each class.

3.3 Transfer learning

While the search for an optimal policy relies on the reward signal associated with the task at hand, the notion of inapplicable actions depends solely on the environment the agent evolves in. Decoupling the reward signal from the classifier offers the advantage of capturing knowledge about the environment only, without introducing the bias associated with the task the agent is trying to solve. This allows the classifier to be reused to solve different tasks in the same environment in a more efficient manner. Additionally, two different domains with an overlapping action space, are likely to share some properties that make an action inapplicable in a given state. Although, the classifier may not be able to perfectly identify inapplicable actions in a new domain, it already provides some valuable knowledge about the environment that can improve the learning efficiency. The knowledge encapsulated by the classifier can therefore be transferred even across different domains.

Algorithm 1 requires little effort to make use of the knowledge previously acquired. By simply passing the parameters of a fitted classifier as input, the algorithm can directly leverage the knowledge acquired about inapplicable actions and mask them away from the policy distribution. To accommodate differences among tasks, Algorithm 1 keeps exploring the action space by ignoring the mask generated with the classifier with a random probability. Furthermore, as the classifier continues to be trained along with the policy, it adapts to the new environment and corrects itself to identify inapplicable actions in the new environment.

4 EXPERIMENTS

We present in this section a range of experiments aiming to answer the following research questions. **Q1:** Does the introduction of human knowledge to mask inapplicable actions help an RL algorithm, such as PPO, to be more sample efficient? **Q2:** Instead of hand-coding this knowledge, can we learn to identify inapplicable actions while training a policy and use the information acquired to improve the sample efficiency of the RL algorithm? **Q3:** Can the domain knowledge fed into the algorithm be completed with the knowledge acquired by the inapplicable actions classifier to further improve the performance of the algorithm? **Q4:** Is it possible to share the knowledge acquired across tasks to help improve the training efficiency of new policies?

The set of experiments presented are run on 3 different domains shown in Figure 1 where an agent (in red) starting at a random position moves in a gridworld-like environment to reach a fixed goal cell (in green). The domains are designed so that the agent will only be rewarded once it reaches the goal, and the reward received will be inversely proportional to the number of steps taken, making the agent learn how to reach the goal in the fewest possible steps. The agent receives the full image of the environment to make a decision on the action to perform. In all the environments the agent can go `up`, `right`, `down`, `left`. In the Key & Door environment, it can also decide to `pickup` a key and `open` a door. The agent must pickup the key before opening the door. But, once the door is open, it remains so and does not require to be opened again.

We present the average reward across 5 runs normalized by the maximum reward an agent can receive, the number of inapplicable actions the agent took and the classifier loss. For this set of experiments, the environment being deterministic, we use the identity function to compute the distance between two states ($d(\boldsymbol{s}_{t+1}, \boldsymbol{s}) = 1 - \mathbf{1}_{\boldsymbol{s}_{t+1}=\boldsymbol{s}}$). While we use simple environment to focus on the ability to transfer knowledge from one domain to another, we leave for future work the study of more complicated environments involving stochasticity, for which the distance measure d could be modified accordingly.

(a) Maze environment | (b) X-Island environment: task 1 | (c) X-Island environment: task 2 | (d) Key & Door environment

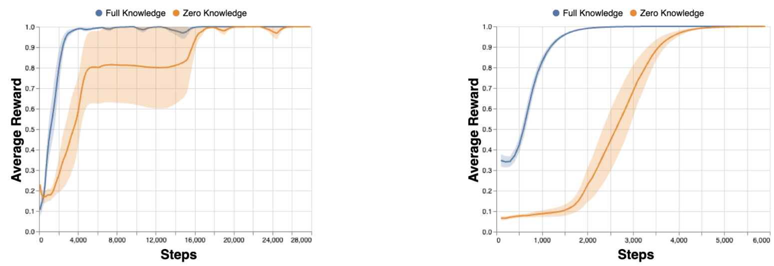
Figure 1: Maze, X-Island and Key & Door environments

4.1 POLICY LEARNING WITH DOMAIN KNOWLEDGE

Setup: We encoded domain knowledge in the X-Island and Key & Door tasks depicted in Figures 1b 1d, respectively. Then, we ran experiments to compare the learning curves of an agent with full versus zero knowledge of the inapplicable actions.

Objectives: These experiments aim at providing some evidence about the practical impact of the approach of introducing inapplicable actions knowledge, as well as assessing its potential gains.

Results: The results of these experiments are shown in Figures 2a and 2b for the X-Island (1) and Key & Door tasks, respectively. As expected, the agent converges faster in both tasks when it has perfect knowledge about the inapplicable actions at each state (blue curves), since it does not need to waste effort exploring useless state-action pairs. Although observed in both domains, the performance gain is bigger in the Key & Door domain. In this case, the inapplicable action



(a) Full vs Zero Knowledge in the X-Island task.

(b) Full vs Zero Knowledge in the Key & Door task.

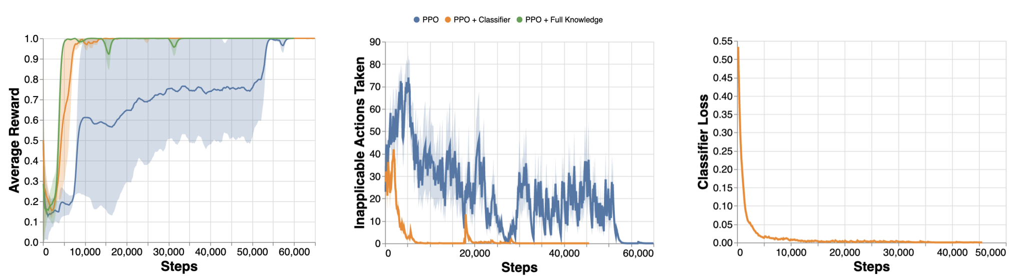
Figure 2: Average reward per episode in different tasks with different levels of inapplicable actions knowledge.

knowledge is able to prune 74% of all the possible state-action pairs, while it is *just* pruning 36% of the pairs in the X-Island domain.

4.2 Policy training with inapplicable actions learning

Setup: To evaluate the impact of learning inapplicable actions during policy training, we run Algorithm 1 in the 3 environments presented in Figure 1 with 5 different seeds and evaluate the sample efficiency gain against the original PPO algorithm. We use an exploration parameter $\epsilon = 0.5$, corresponding to a use of the mask generated by the classifier half of the time allowing the classifier to be exposed to both positive and negative examples of applicability. The full architecture of the NN used for the classifier is presented in Appendix B.

Objectives: These experiments consider the situation where no knowledge about inapplicable actions is available. We wish to evaluate if learning an estimate of the applicability function C while training the policy is feasible, and whether it can be used directly by the RL algorithm to mask out inapplicable actions from the policy distribution and avoid wasting resources.



(a) Average reward per episode

(b) Average number of inapplicable actions taken per episode

(c) Classifier loss

Figure 3: Learning the inappicable actions mask in the Maze environment.

Results: The classifier (orange curve) is able to rapidly learn to identify inapplicable actions as suggested by the classifier loss curve (Figure 3c). As we can see in Figure 3b, the number of inapplicable actions taken by the agent reduces faster than the PPO baseline throughout the training. Hence, the agent avoids exploring inapplicable actions thanks to the classifier and the quality of the mask generated. As a result, this algorithm is more sample efficient than the PPO baseline (Figure 3a). Similar results are observed for the X-Island and the Key & Door environments (Appendix D). The more constrained the environment is, the bigger the performance improvement will be when we introduce the inapplicable actions learning component. In the Maze environment, the agent has in most cases only 50% of its action space applicable which explains the significant performance improvement.

4.2.1 CLASSIFIER ANALYSIS

Setup: We analyze the trained classifier for each of the environments (see also Appendix E) and we present in Figure 4 the predictions the classifier makes on the applicability of each of the actions the agent can take in a given state. A red (respectively green) cell indicates that the classifier has predicted that the action is inapplicable (respectively applicable) when the agent stands on that cell.

Objectives: Understand what the classifier has learnt and make sure that the knowledge it acquired in the gridworld environment concurs with the common sense of treating actions that go into a wall as inapplicable.

(a) Action `up` (b) Action `right` (c) Action `down` (d) Action `left`

Figure 4: Classifier output learnt in the X-Island (1) environment

Results: We see that the classifier is able to perfectly learn inapplicable actions at each given state. For instance, all the cells where the agent has a wall on its left are red in Figure 4d. This confirms the hypothesis that the classifier provides valuable information about the environment to the RL algorithm to mask out inapplicable actions.

4.2.2 INAPPLICABLE ACTIONS LEARNING WITH PARTIAL DOMAIN KNOWLEDGE

Setup: We specify a subset of the applicability function C in the Key & Door and X-Island (1) environments. In Key & Door we only provide information about the `up`, `down`, `right` and `left` actions. In X-Island (1) we only provide information about `up` and `down`.

Objectives: Show how the classifier can learn the rest of the applicability function C when provided with partial knowledge.

Results: The results of this experiment in the Key & Door environment are shown in Figure 5 (see also Appendix D.1) As we can see, jointly using partial knowledge and the classifier speeds up the learning both over learning the classifier from scratch and just having partial knowledge.

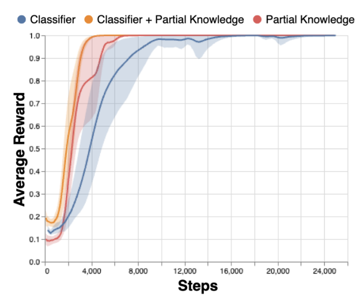


Figure 5: Different ways to acquire inapplicable actions knowledge for the Key & Door task.

4.3 KNOWLEDGE TRANSFER

Setup: We solve again the tasks presented in Figure 1, but this time we provide the algorithm with the trained classifiers obtained from the runs presented in Section 4.2. We reduce the exploration parameter ϵ to 0.25 as we have more confidence in the trained classifier. We compare our technique with an implementation of Policy Reuse (Fernández et al., 2010) using also an exploration parameter of 0.25 and a warm start technique initializing the policy with the weights of a pre-trained policy.

Objectives: Check if the knowledge acquired in one domain and encapsulated in the inapplicable actions classifier can be transferred across different tasks and domains to help improve the learning efficiency of new policies.

Results: The results when the classifier was trained in the Maze Environment are presented in Figure 6 (and in Appendix G. The transfer of the classifier yields better performance than learning the classifier from scratch in almost all configurations. The gain obtained seems to be related to the difference between the source and the destination environments. The classifier trained in the Maze environment even contributes to a small increase of performances in the Key & Door environment having a different layout and action space. As opposed to Policy Reuse or naively reusing a pre-trained policy, the transfer of the classifier is able to adapt to different environments thanks to the task-agnostic knowledge acquired.

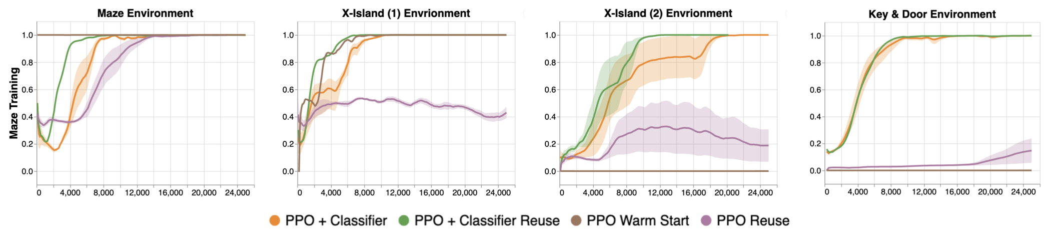


Figure 6: Average reward for different techniques of knowledge transfer reusing the knowledge acquired in the Maze environment.

5 Related work

The use of actions masking has been used by the RL community to improve the sample efficiency of RL algorithms and has contributed to some of the most impressive breakthroughs in the field (Vinyals et al., 2017; Kool et al., 2018). Despite being used, the effect of action masking had not been thoroughly analyzed until recently with Huang & Ontañón (2022), where the theoretical soundness of the approach is shown when used with Policy Gradient algorithms.

Only a few papers tackle the problem of learning the masking function. Even-Dar et al. (2006) appears to be the first to look at actions eliminations in the context of RL by using the confidence interval around the Q-function to avoid actions that are sub-optimal with high probability. Later, Zahavy et al. (2018) present a new algorithm combining DQN and an Action Elimination Network (AEN) learning to mask out sub-optimal actions. In both works the focus is put on sub-optimal actions and is therefore biased by the task to solve. Our approach on the other hand focuses on learning a particular feature of the environment only. This specificity allows us to transfer the knowledge gained in one environment to another.

To reuse previously acquired knowledge, Barreto et al. (2018) associate the Successor Representations (SR) framework, decoupling the state features and the reward distribution to estimate the value function, with the General Policy Improvement framework (GPI). This technique assumes, however, that the tasks only differ by their reward distribution. Close to our approach, the Policy Reuse algorithm (Fernández et al., 2010) uses expert policies previously trained to help orient the learning agent towards an optimal policy.

Recent works from the automated planning research community try to learn the symbolic model capturing the applicability of actions at each state from latent spaces (Asai & Fukunaga, 2018; Bonet & Geffner, 2020) with the aim of limiting the domain knowledge required to solve problems.

6 Conclusion

In this work, we presented an approach to formally specify environment specific constraints (i.e. inapplicable actions) that can be used by an RL algorithm to only explore the actions relevant to finding an optimal policy. We also proposed a technique to learn these constraints when they are not known a priori by training jointly with the policy a classifier predicting that an action is inapplicable in a given state. The information captured by this new component can directly be used to mask out inapplicable actions, leading to more sample efficient RL algorithms. The knowledge acquired about the environment can be transferred to other tasks in similar environment providing an even greater performance gain. The transferability is however subject to certain conditions including the difference of the action spaces or the environment settings. A more detailed analysis of these constraints would be an interesting topic for future research. While our approach was shown to significantly improve the performance of Policy Gradient algorithms we leave as future work a study of its usefulness to other RL algorithms.

REPRODUCIBILITY STATEMENT

We describe in details the new algorithm in the text as well as with Algorithm 1. The experiments configuration is detailed in Appendix A and we provide the code used to run them as supplementary materials.

REFERENCES

Masataro Asai and Alex Fukunaga. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. In Sheila A. McIlraith and Kilian Q. Weinberger (eds.), *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pp. 6094–6101. AAAI Press, 2018. URL `https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16302`.

Andre Barreto, Diana Borsa, John Quan, Tom Schaul, David Silver, Matteo Hessel, Daniel Mankowitz, Augustin Zidek, and Remi Munos. Transfer in deep reinforcement learning using successor features and generalised policy improvement. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 501–510. PMLR, 10–15 Jul 2018. URL `https://proceedings.mlr.press/v80/barreto18a.html`.

Blai Bonet and Hector Geffner. Learning first-order symbolic representations for planning from the structure of the state space. In Giuseppe De Giacomo, Alejandro Catalá, Bistra Dilkina, Michela Milano, Senén Barro, Alberto Bugarín, and Jérôme Lang (eds.), *ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020)*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pp. 2322–2329. IOS Press, 2020. doi: 10.3233/FAIA200361. URL `https://doi.org/10.3233/FAIA200361`.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv*, Jun 2016. doi: 10.48550/arXiv.1606.01540. URL `http://arxiv.org/abs/1606.01540`. arXiv:1606.01540 [cs].

Eyal Even-Dar, Shie Mannor, and Yishay Mansour. Action elimination and stopping conditions for the multi-armed bandit and reinforcement learning problems. *Journal of Machine Learning Research*, 7(39):1079–1105, 2006. ISSN 1533-7928.

Fernando Fernández, Javier García, and Manuela Veloso. Probabilistic policy reuse for inter-task transfer learning. *Robotics and Autonomous Systems*, 58(7):866–871, 2010. ISSN 0921-8890. doi: https://doi.org/10.1016/j.robot.2010.03.007. URL `https://www.sciencedirect.com/science/article/pii/S0921889010000655`. Citation Key: FERNANDEZ2010866.

Richard E. Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3):189–208, 1971. ISSN 0004-3702. doi: https://doi.org/10.1016/0004-3702(71)90010-5. URL `https://www.sciencedirect.com/science/article/pii/0004370271900105`. Citation Key: FIKES1971189.

Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: theory and practice*. Elsevier, 2004.

Shengyi Huang and Santiago Ontañón. A closer look at invalid action masking in policy gradient algorithms. In *The International FLAIRS Conference Proceedings*, volume 35, 2022.

Wouter Kool, Herke Van Hoof, and Max Welling. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*, 2018.

Eric Liang, Richard Liaw, Philipp Moritz, Robert Nishihara, Roy Fox, Ken Goldberg, Joseph E. Gonzalez, Michael I. Jordan, and Ion Stoica. Rllib: Abstractions for distributed reinforcement learning, 2017. URL `https://arxiv.org/abs/1712.09381`.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(75407540):529–533, Feb 2015. ISSN 1476-4687. doi: 10.1038/nature14236. URL https://www.nature.com/articles/nature14236.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Thomas Spooner, Nelson Vadori, and Sumitra Ganesh. Factored policy gradients: Leveraging structure for efficient learning in MOMDPs. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=NXGnwTLlWiR.

Yanmin Sun, Andrew KC Wong, and Mohamed S Kamel. Classification of imbalanced data: A review. *International journal of pattern recognition and artificial intelligence*, 23(04):687–719, 2009.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999.

Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, John Quan, Stephen Gaffney, Stig Petersen, Karen Simonyan, Tom Schaul, Hado van Hasselt, David Silver, Timothy Lillicrap, Kevin Calderone, Paul Keet, Anthony Brunasso, David Lawrence, Anders Ekermo, Jacob Repp, and Rodney Tsing. Starcraft ii: A new challenge for reinforcement learning. *arXiv:1708.04782 [cs]*, Aug 2017. arXiv: 1708.04782.

Tom Zahavy, Matan Haroush, Nadav Merlis, Daniel J Mankowitz, and Shie Mannor. Learn what not to learn: Action elimination with deep reinforcement learning. In *Advances in neural information processing systems*, volume 31, 2018.

A EXPERIMENTS CONFIGURATION

All the experiments were run on a *r6i.8xlarge* EC2 instance. The results presented are the mean value over 5 different seeds using the standard error to construct the confidence interval. For all the experiments presented, the policy was trained until convergence or stopped after $100,000$ steps. The implementation of the PPO algorithm from (Raffin et al., 2021) was used as a baseline, and modified to implement the Algorithm 1 introducing the inapplicable actions masking components.

The code for the Open AI Gym environments used and the RL algorithms including Algorithm 1 is provided in the supplementary materials.

For the experiments learning inapplicable actions, the classifier is trained jointly with the policy, using the same batch size (64), the same number of epochs (10) per training iteration and with an Adam optimizer using a learning rate of $3 \cdot 10^{-4}$ for the *Inapplicable Actions Learning* experiments and $1 \cdot 10^{-4}$ for the *Knowledge Transfer* experiments. The classifier used is a neural network composed of three elements. The first element called the `ObservationsExtractor` is a Convolutional Neural Network (CNN) in charge of extracting features from the image representing the current state of the system. The second component, `ActionsExtractor`, simply one-hot encodes the action that needs to be evaluated. Finally, the classifier is a binary classifier implemented with a Multi-Layer Perceptron using ReLU and Dropout layers, that takes as an input the concatenation of the vectors from the two extractor components and output the logit of the action being applicable in the given state. The policy neural network uses a features extractor network generating features in a latent space that will then be fed into the policy and the value networks of the PPO algorithm (Raffin et al., 2021).

The full architecture details of the neural networks are provided in Appendix B.

B NEURAL NETWORK ARCHITECTURES

All the neural networks' weights are initialized using the *Xavier* initialization, also known as *Glorot* initialization, associated to the uniform distribution. We use the `Pytorch` framework (Paszke et al., 2019) to implement all the neural network described in this paper.

Table 1: Neural Network architectures

Observations Extractor	**Classifier**	**Policy Features Extractor**
BatchNorm2d(N)	BatchNorm1d(M)	Conv2d(N, 32, kernel_size=8, stride=4)
Conv2d(N, 32, kernel_size=8, stride=4)	Linear(M, 256)	ReLU()
ReLU()	ReLU()	Conv2d(32, 64, kernel_size=4, stride=2)
Conv2d(32, 64, kernel_size=4, stride=2)	Dropout(0.3)	ReLU()
ReLU()	BatchNorm1d(256)	Conv2d(64, 64, kernel_size=3, stride=1)
Conv2d(64, 64, kernel_size=3, stride=1)	Linear(256, 96)	ReLU()
ReLU()	ReLU()	Flatten()
Flatten()	Dropout(0.3)	
	BatchNorm1d(96)	
	Linear(96, 1)	

C Domain knowledge

C.1 Different levels of domain knowledge

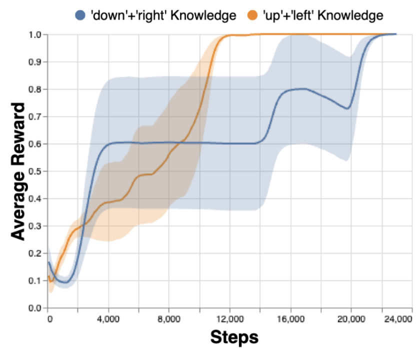
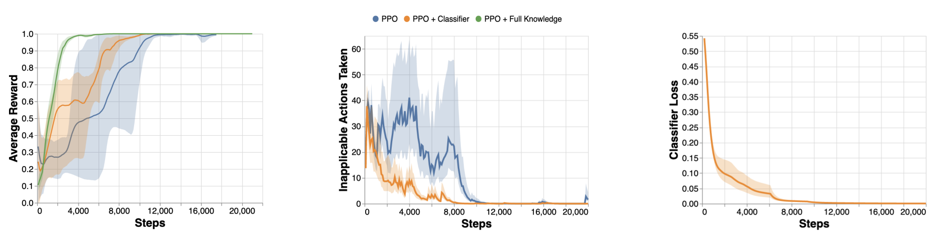


Figure 7: Importance of the level of inapplicable actions knowledge provided to the algorithm

The performance gain provided by knowing actions (in)applicability not only depends on the task but also on the specific actions we have information from. Consider the X-Island (1) task depicted in Figure 1b. In this case, the optimal policy involves the agent taking a number of down and right actions. Therefore, having knowledge about when the agent cannot move `up` or `left` (actions that the agent does not need to execute) should accelerate the learning more than having information about the move `down` and `right` actions (actions that the agent will need to execute). This hypothesis is confirmed by the results in Figure 7, where we can observe that the agent converges faster when it has access to the `up` and `left` actions knowledge than when it has access to the `down` and `right` knowledge.

D Inapplicable actions learning

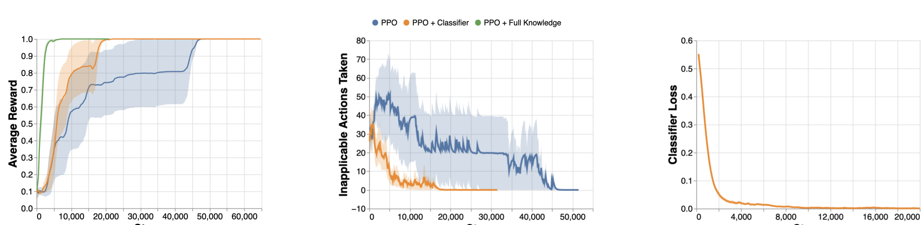


(a) Average reward per episode

(b) Average number of inapplicable actions taken per episode

(c) Classifier loss

Figure 8: X-Island (1) environment

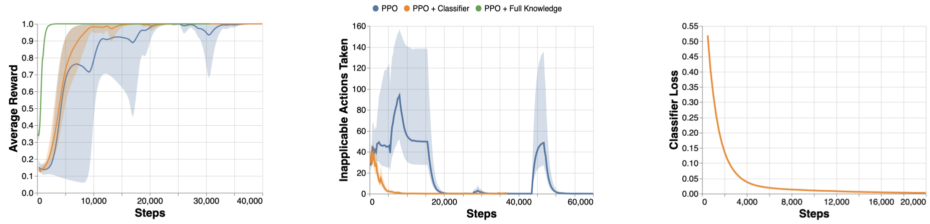


(a) Average reward per episode

(b) Average number of inapplicable actions taken per episode

(c) Classifier loss

Figure 9: X-Island (2) environment



(a) Average reward per episode

(b) Average number of inapplicable actions taken per episode

(c) Classifier loss

Figure 10: Key & Door environment

Introducing the inapplicable actions learning classifier helps reduce the sample complexity of the PPO algorithm in all the tasks Figures 8,9 and 10. The classifier is able to quickly learn to identify the inapplicable actions, reducing the number of inapplicable actions explored.

D.1 Inapplicable actions learning with partial domain knowledge

As we can see in Figure 11, in this case jointly using partial knowledge and the classifier to learn the rest of the applicability function C perform similarly, both outperforming learning the classifier from scratch. This is because learning the classifier takes a similar time than learning the optimal policy in this particular task.

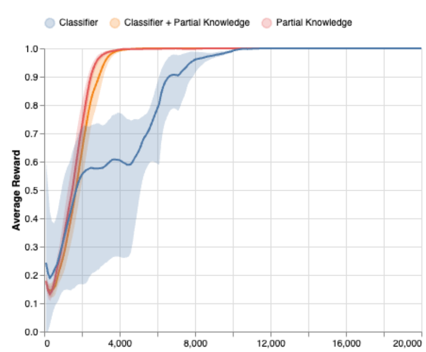
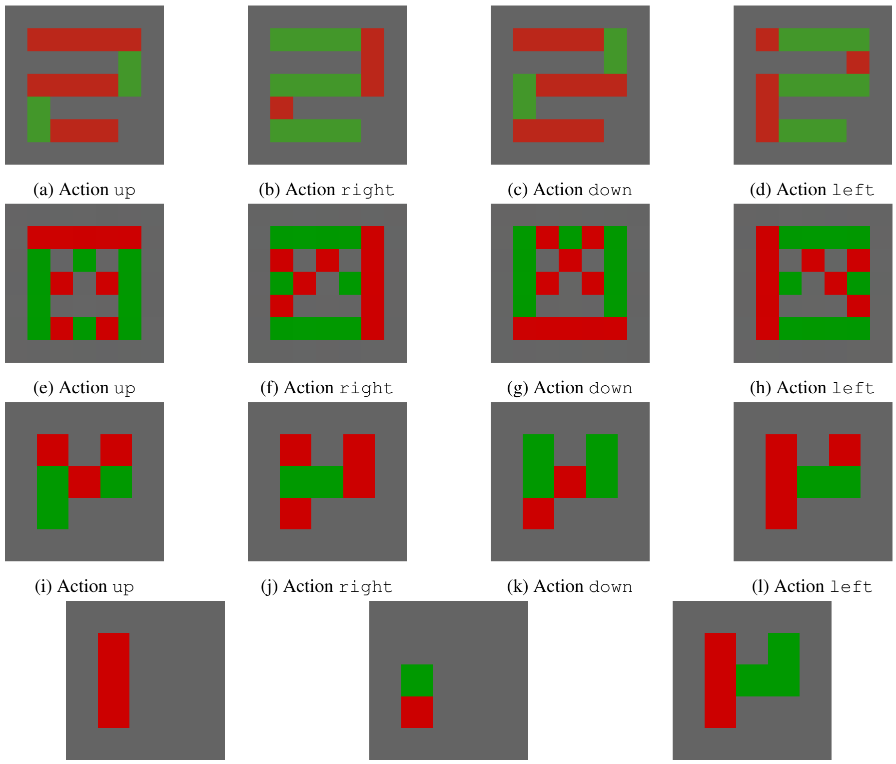


Figure 11: Average reward for different combinations of Classifier and Partial Knowledge.

E CLASSIFIER ANALYSIS



(a) Action `up` (b) Action `right` (c) Action `down` (d) Action `left`

(e) Action `up` (f) Action `right` (g) Action `down` (h) Action `left`

(i) Action `up` (j) Action `right` (k) Action `down` (l) Action `left`

(m) Action `right` when door is closed (n) Action `pickup` (o) Action `open` when door is open

Figure 12: Classifier output learnt in the Maze, X-Island (2) and Key & Door environments

Figure 12 presents the output of the classifier trained in the different environments. The classifier is able to predict with high accuracy that an action is inapplicable in a given state in all the environments. In the Key & Door environment, the applicability of the action `right` changes when the door is open. We see in Figures 12m and 12j that the classifier is able to capture the change in the state that makes an action applicable. The classifier struggles however with the `open` action for states that were not visited often Figure 12o.

F Exploration parameter analysis

We explore in this section the impact of the exploration parameter (ϵ) on the reward of the agent learning to reach the goal in the Maze environment. We observe that a trade-off exists between exploration and exploitation of the knowledge acquired about inapplicable actions. When the agent explores too much ($\epsilon = 0.75$) the agent needs more time to converge to the optimal policy. With a lower $epsilon$, the agent is able to reduce the time to converge by leveraging the accuracy of the classifier to mask inapplicable actions. However, when the agent relies too much on the classifier ($\epsilon = 0.1$) and does not explore enough, the agent is not able to explore valid actions because the classifier incorrectly masks them making the reward diverge.

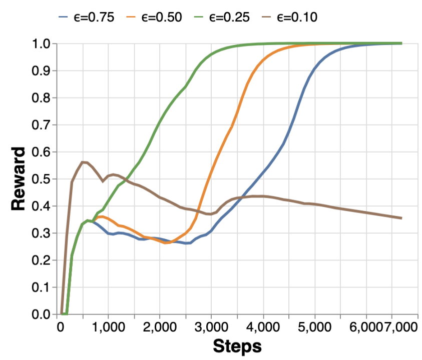


Figure 13: Reward for different exploration parameter values (ϵ) in the Maze Environment.

G KNOWLEDGE TRANSFER

PPO Warm Start: PPO Warm Start reuses a pre-trained policy as a starting point to learn a new task. The weights of the original policy are used to instantiate the current policy before the training starts.

PPO Reuse: We implement the logic of Policy Reuse (Fernández et al., 2010) in the PPO algorithm, where during the rollouts, a pre-trained expert policy is sampled with a probability ϵ and the current policy is used the rest of the time.

As observed in Figure 14, the transfer of the trained classifier from one environment to another is beneficial. The training efficiency is improved in almost all the configurations. The performance improvement is more important when the training environment and the destination environment are similar. A more constrained environment, such as the Maze environment with only 50% of the action space being applicable in most of the states, seems to provide better knowledge transfer results. Comparing the transfer of the classifier with other knowledge transfer techniques such as Policy Reuse or a naive warm start approach, our approach is more versatile. It can be transferred both across tasks and across environments. The warm start approach, clearly overfits while the Policy Reuse learns something but struggles to find an optimal policy.

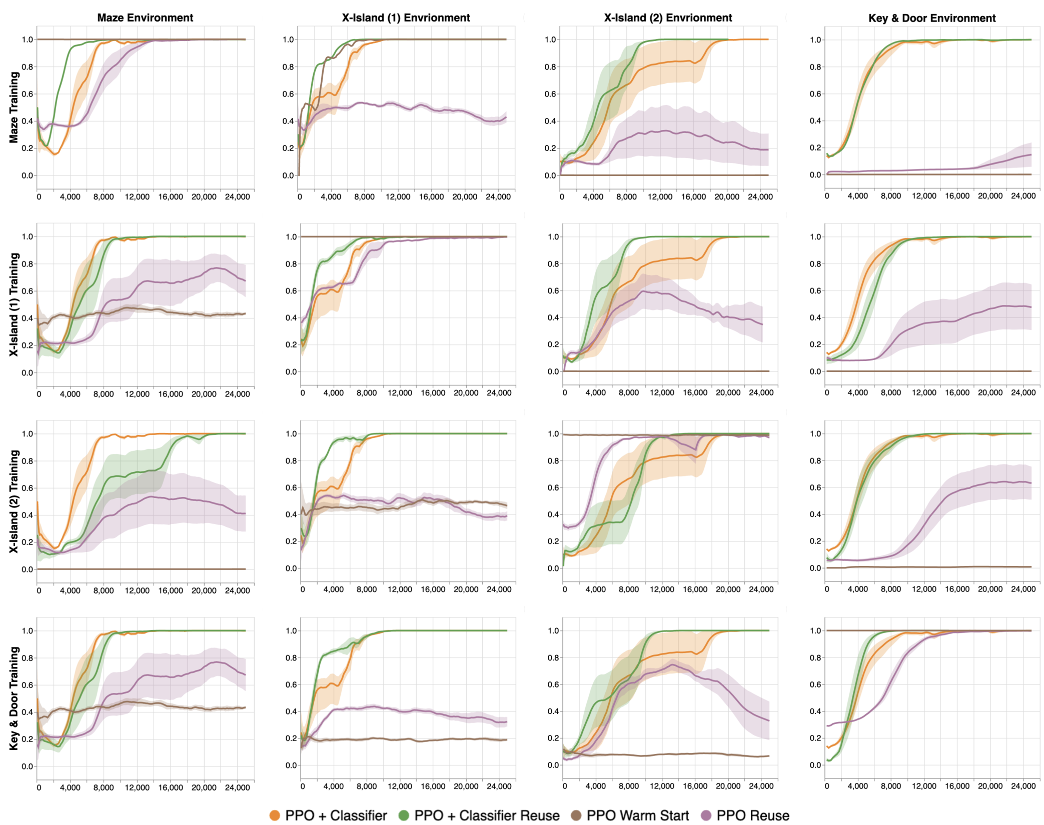


Figure 14: Average reward for different transfer learning configurations. The row represents the trained classifier used while the columns are the environment it was transferred to.