ASAP: Attention-Based State Space Abstraction for Policy Summarization

Yanzhe Bekkemoen YANZHE.BEKKEMOEN@NTNU.NO
Helge Langseth HELGE.LANGSETH@NTNU.NO
Department of Computer Science, Norwegian University of Science and Technology, Trondheim, Norway

Editors: Berrin Yanıkoğlu and Wray Buntine

Abstract

Deep reinforcement learning (RL) has shown remarkable performance, but end-users do not understand how the system solves tasks due to the black-box nature of neural networks. Many methods from explainable machine learning have been adapted to RL. However, they do not focus on the unique challenges of explaining actions' short-term and long-term consequences. This work introduces a new perspective to understanding RL policies by clustering states into abstract states utilizing attention maps, giving a bird's-eye view of the policy's behavior. We learn the attention maps iteratively together with the clustering of states by masking the input features to estimate their importance. In contrast to previous works that have uninterpretable abstract states and/or clustering objectives using state values that are non-human intuitive, we only leverage attention maps in the clustering. The policy only indirectly affects the clustering via attention maps. This allows us to give global explanations from the view of feature attention, a quantity a human can relate to given interpretable features. The experiments demonstrate that our method provides faithful abstractions by capturing state semantics, policy behavior, and feature attention. Furthermore, we show that our attention maps can mask state features without affecting policy performance.

Keywords: Explainable Reinforcement Learning; Deep Learning; Interpretability

1. Introduction

Deep reinforcement learning (RL) have gained considerable popularity and shown high performance in several tasks (Silver et al., 2017; Brown and Sandholm, 2017). However, the black-box nature of neural networks has been criticized and makes it difficult for humans to understand the underlying decision-making (Rudin et al., 2022). Consequently, deploying RL agents in the real world is challenging, especially in domains that require human understanding and trust and where mistakes are costly.

In response to this problem, several explainable artificial intelligence methods tailored to deep RL have been developed and are known as explainable reinforcement learning (XRL). The most popular being saliency methods that explain the decision-making by highlighting important features the agent pays attention to (Greydanus et al., 2018; Iyer et al., 2018; Mott et al., 2019; Puri et al., 2020; Itaya et al., 2021; Shi et al., 2022). Despite their popularity, these methods only locally explain a single decision. In classification tasks, this works since the observations do not have sequential dependencies. However, in RL, actions have short-

term and long-term consequences; thus, local explanations must be complemented with global explanations. Even in a simple environment like Cart Pole, an episode can last 500 steps. Understanding which features an agent pays attention to requires first figuring out interesting time steps to investigate.

This paper proposes a new post hoc global XRL method, **a**ttention-based **s**tate space **a**bstraction for **p**olicy summarization (ASAP). ASAP aims to complement the weakness of the aforementioned local explanation methods. Our method focuses on answering the question "*how does the policy work?*", which helps to find states where it is interesting to ask "*why did the agent do action a in the state s?*". The method focuses on environments where states are low-dimensional with interpretable features. ASAP summarizes stochastic neural network policies with high-level attention maps by merging states into abstract states, which we name *hyperstates*. To cluster states into hyperstates, we leverage only attention maps obtained through learning by masking input features to estimate their importance. These attention maps highlight task-relevant information and inform us about features used in a hyperstate when making decisions, hence avoiding the difficulty of understanding and subjective interpretation of what a hyperstate represents. Our experiments indicate clustering on attention maps captures desirable grouping of states, such as states in a hyperstate both being semantically similar and treated similarly by the policy.

Similar to our work, researchers have proposed state space abstractions methods (Zahavy et al., 2016; Topin and Veloso, 2019; McCalmon et al., 2022; Bewley et al., 2022) to reduce the state space for explaining policies. These methods explain via a Markov chain with edges representing policy actions. Unlike ASAP, most of these methods do not explain the hyperstates themselves. This makes it difficult for an explainee, the person consuming the explanation, to understand what these hyperstates represent. Although McCalmon et al. (2022) provide interpretability via natural language, their approach of defining natural language predicates and binary classifiers to make interpretable hyperstates does not scale, something we aim to solve with attention maps. Conceptually, ASAP differs from these methods based on information used to merge states. While previous methods leverage state description, action, and/or state value, we only use attention maps to cluster states. The policy only indirectly affects the clustering via attention maps. As a result, we can explain hyperstates and actions taken in hyperstates from the perspective of feature attention rather than referring to 1) state values that we lack an intuitive understanding of or 2) actions causing circular reasoning since actions themselves are used in the clustering.

Figure 1 shows an explanation produced by ASAP for a policy trained in the Mountain Car environment. The main component is the attention maps, answering which features the policy uses to make decisions in hyperstates. Additionally, there are two supporting components. The first is representative states of each hyperstate with accompanying action taken in the hyperstate. The second component is a Markov chain, showing how the policy behaves in the hyperstate space. These two additions suggest our clustering objective captures the generative process of the environment and the policy. Moreover, we show in the experiments that with enough hyperstates, each hyperstate is often related to one action. As a result, we can make statements like, "we observe that H_0 are states where the policy only cares about the velocity and not the position when accelerating left. And that H_4 are states similar to H_0, but the velocity matters less when deciding to accelerate left".

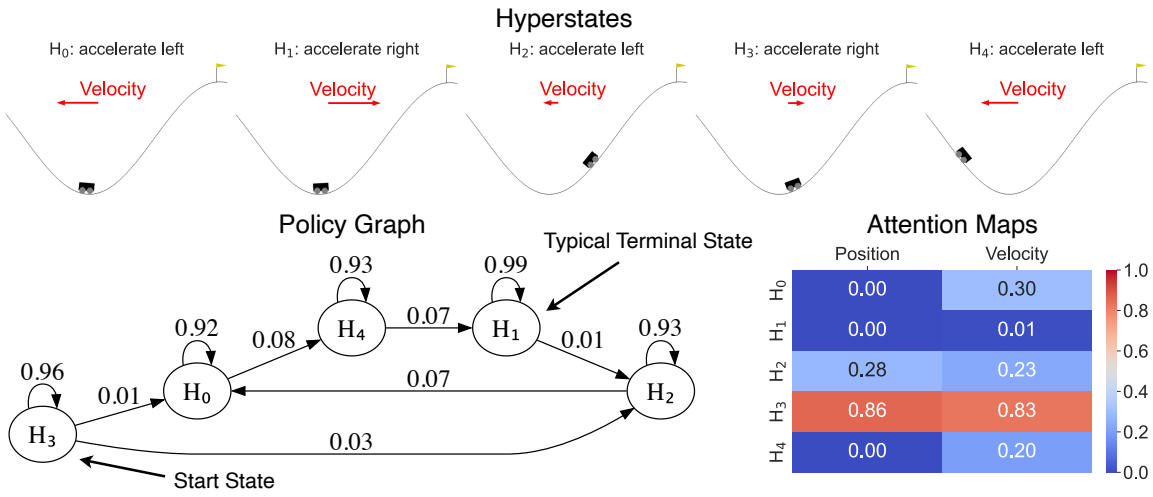


Figure 1: An RL policy explanation produced by ASAP for the Mountain Car environment. The primary component is the attention maps for hyperstates, explaining the importance of features when the policy makes decisions. Additionally, there are two supporting components. First, representative states with corresponding actions of the learned hyperstates. Second, the policy graph, a Markov chain showing how the policy traverses the hyperstate space where edges denote transitions.

The attention maps are computed relative to a baseline state that represents the "absence" of features. Our attention maps explain how the distribution over actions is affected by each feature when it is "taken away". In this example, we use the start state with features position $= \quad 0.5$ and velocity $= 0$ as our baseline state because that is when the car is at the bottom of the valley without any speed. The motivation to use a baseline is related to how humans attribute causes. Humans attribute a cause for an outcome by using the absence of a cause as the reference point for determining its impact (Sundararajan et al., 2017; Shrikumar et al., 2017).

In summary, our contributions are:

1. We propose a new method, ASAP, to provide a new perspective for understanding RL policies by presenting the state space in terms of hyperstates with attention maps. Our main contribution is the high-level attention maps that provide information about features used in the decision-making and are the only information used to create the hyperstates. The policy is leveraged to learn the attention maps but only indirectly affects the clustering. Hence, providing interpretable hyperstates and avoiding clustering on information an explainee cannot easily relate to.

2. We qualitatively and quantitatively evaluate ASAP. The experiments show that using attention maps alone results in groups of states with desirable properties. States with similar state semantics, policy behavior, and features with similar attention are mapped to the same hyperstate, providing faithful abstractions. Moreover, we show that policy performance is unaffected when attention maps mask states.

2. Related Works

Recently, XRL has gained considerable popularity (Heuillet et al., 2021). Like explainability in supervised learning, XRL methods can roughly be divided into inherent interpretability and post hoc explainability. Inherently interpretable methods aim to create agents that are interpretable out of the box. This is achieved by representing policies with function approximators where a human can easily interpret the structure. On the one hand, interpretability is intrinsic to the agent, thus fully interpretable. On the other hand, in many situations, imposing limitations on the functional form is impossible. Post hoc methods do not impose limitations on the agent but instead try to understand it as it is. The most popular are saliency methods, which highlight important features. However, these methods only explain a single time step. This alone is insufficient in RL since sequential dependencies with short-term and long-term consequences exist. More specific to RL, we have, for instance, HIGHLIGHTS (Amir and Amir, 2018) that shows explainees "critical" states where taking the wrong action has undesirable consequences on future outcomes. Other methods explain policies by directly explaining the outcome (Juozapaitis et al., 2019; Yau et al., 2020).

Methods similar to ours try to make global explanations post hoc by creating an abstraction of the state space (Zahavy et al., 2016; Topin and Veloso, 2019; McCalmon et al., 2022; Bewley et al., 2022; Luss et al., 2023). One problem with these approaches is understanding hyperstates they create. Zahavy et al. (2016) propose to apply t-distributed stochastic neighbor embedding (t-SNE) (van der Maaten and Hinton, 2008) on a DQN's activation and manually analyze the hyperstates. However, manually analyzing the structure is time-consuming. Topin and Veloso (2019) introduce a method to create a policy graph with actions, but their hyperstates lack semantics to help users understand what a hyperstate represents. Further, their method requires binary features. To resolve this issue of unintelligible hyperstates, McCalmon et al. (2022) propose to create descriptions in natural language for each hyperstate. Their clustering uses the state, action, and state value. However, creating descriptions in natural languages requires human-created predicates, which can be laborious and difficult in complex environments. Moreover, forming hyperstates in this way makes it difficult to explain hyperstates with similar state descriptions but different state values because an explainee has no intuitive understanding of state values.

To improve on the weaknesses of these approaches, we utilize attention maps to ground hyperstates and make the hyperstates interpretable without requiring human intervention. In contrast to other approaches, we only use attention to cluster states into a hyperstate, which makes it possible to create statements without referring to quantities not easily interpreted by humans, such as state values. Further, we show attention alone captures desirable properties, such as state semantics and policy behavior, without including the state description and action in the clustering process. Thus, we can explain actions in hyperstates using feature attention without circular reasoning.

Working on state space abstraction, Bewley et al. (2022) propose a method to analyze and compare several policies via multiple Markov chains, which differs from our goal of understanding a single fully trained policy. Luss et al. (2023) present a new state space abstraction method but focuses on a specific state and states within a fixed distance. Thus, their method lies between global and local explanations, which differs from our aim for global explanations.

3. Background

In RL, the environment is modeled as a Markov decision process (MDP) defined as a tuple $\langle S, A, p, r, \gamma \rangle$ with the state space S, the action space A, the discount factor $\gamma \in [0, 1]$, the transition function, and the reward function. The transition function $p(s' | s, a)$ is a conditional probability distribution that takes a state-action pair $s \in S, a \in A(s)$ and outputs the probability of the next state $s' \in S$. $A(s)$ denotes the set of actions available in the state s. The reward function $r(s, a)$ outputs the reward for taking an action $a \in A(s)$ in the state s, and can optionally depend on the next state s'. In this work, we focus on low-dimensional states with interpretable features. Further, all environments we experiment with have continuous features.

Given the MDP, an agent seeks to learn a policy π to solve the task by maximizing the discounted cumulative reward received by employing the policy in the environment. The policy is a function that takes a state s_t and outputs either the corresponding action to take or the distribution over actions in that state. The policy can be deterministic or stochastic. In this work, we focus on differentiable stochastic policies. To learn the policy, a value function is often learned. The value function can either be state-based or state-action-based. The state-based value function is the expected discounted cumulative reward, known as the return, received by following the policy π from a state s and is defined by $V^t_\pi(s) = \mathbb{E}_{\tau \sim \pi}[r(\tau) | S_t = s]$, where $\tau = (s_t, a_t, s_{t+1}, a_{t+1}, \dots)$ is a trajectory of state-action pairs. Similarly, the state-action-based function is defined by $Q^t_\pi(s, a) = \mathbb{E}_{\tau \sim \pi}[r(\tau) | S_t = s, A_t = a]$ as the return of taking the action a in the state s and follow the policy thereafter.

4. Learning hyperstates

The agent seeks to learn a high-performing policy π given an MDP. Our goal for this work is to understand how this policy works post hoc, that is, comprehending its behavior after training. We do this by clustering states into hyperstates. Contrary to previous methods, we only use information about feature attention when determining how to cluster states. The policy only indirectly affects the clustering via attention maps. Our method jointly learns attention maps of states and clusters them using neural networks trained end-to-end.

Our setup consists of a network that predicts the probability of a state belonging to a hyperstate. The hyperstate probability is used to determine attention maps. As a result, we get an abstraction of the state space with hyperstates and attention maps grounding them. First, we discuss properties desirable for a hyperstate in Section 4.1. Afterward, Section 4.2 formulates the learning objective and describes the components of our method, shown in Figure 2, in detail.

4.1. Hyperstate Properties

We wish to fulfill several properties when finding an abstraction of the state space. First, we want states forming a hyperstate to be semantically similar. By being semantically similar, it becomes easier and quicker to understand what a hyperstate represents. Second, since our goal is to explain the policy, we must ensure that the policy perceives states in a hyperstate similarly. In other words, given any two states $s, s' \in H_j$, the policy should be such that $\pi_\theta(\cdot | s) \approx \pi_\theta(\cdot | s')$. Consequently, we can make explanations on how hyperstates are related

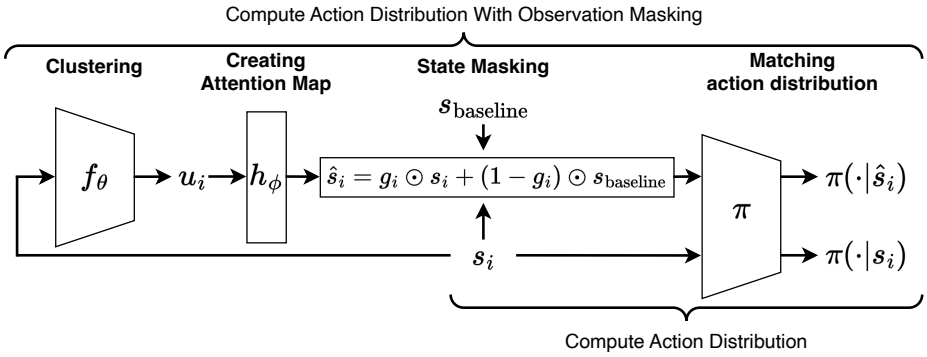


Figure 2: A stochastic policy works by taking in a state s_i and outputting a distribution over actions, shown at the bottom. Our method is trained by first computing the distribution over actions with the masked observation $\hat{s}_i$, shown at the top. The state s_i is used to compute 1) the clustering of the state via the function f_θ and 2) the masked state $\hat{s}$ utilizing the baseline state s_{baseline} weighted by the attention map g_i. Afterward, we minimize the difference between the distribution over actions conditioned on s_i and $\hat{s}_i$. Doing this allows us to create hyperstates with faithful attention maps learned iteratively end-to-end.

to actions. Lastly, we want an unambiguous representation of hyperstates through attention maps humans can understand. Hence, each state in a hyperstate should have approximately the same attention map. Given the attention maps $d, d^{\prime}$ for states $s, s^{\prime}$ 2 H_j, they should be so that d $d^{\prime}$. The experiments show that our approach captures these elements by clustering only on attention maps and without using additional information.

4.2. Approach: How to Learn Hyperstates

Given a trained policy, we propose a new state clustering approach that merges states with similar attention maps into a hyperstate. The method aims to estimate hyperstate probability for states and uses it to create attention maps via mixtures of probabilities. Specifically, we compute these two quantities in the following two steps:

1. First, given a state s_i 2 R^m, we calculate $u_i = f_\theta(s_i)$, which corresponds to *Clustering* in Figure 2. Each dimension of the m dimensional state s_i is a feature. For example, in the mountain car environment, the dimension of a state s_i is $m = 2$, and the features are *position* and *velocity*, which are continuous values. u_i 2 $[0, 1]^k$ is a categorical distribution denoting the probability for the state s_i belonging to hyperstates H = $\{H_0, \ldots, H_k\}$.

2. After obtaining the probability of the state s_i being a member of the hyperstates, u_i is inputted to a function h_ϕ to compute the attention map $g_i = h_\phi(u_i)$. g_i 2 $[0, 1]^m$ is the attention map for the state s_i and corresponds to *Creating Attention Map* in Figure 2. In addition to informing explainees about feature attention, we create a new state $\hat{s}_i$ by applying the attention map. $\hat{s}_i$ represents s_i where features are masked

according to how important they are. $\hat{s}_i$ is used in the learning objective to ensure faithful attention maps.

Thus, our goal is to iteratively learn the two parameterized functions $f_\theta : \mathsf{R}^m \; ! \; [0,1]^k$ and $h_\phi : [0,1]^k \; ! \; [0,1]^m$ to obtain a state abstraction that fulfills the desired hyperstate properties. To achieve that, we have three constraints in our learning objective. Firstly, we want the clustering to be tight so that all states in a hyperstate have approximately the same attention, which is fulfilled by the objective introduced in Section 4.2.2. Secondly, we want g_i to be sparse for obtaining simpler explanations, which we elaborate on in Section 4.2.3. Finally, we need to maintain policy fidelity by making sure that $\pi(\,j s_i) \quad \pi(\,j \hat{s}_i)$ for all $s_i, \hat{s}_i$, where $\hat{s}_i$ is the masked state. We describe state masking in Section 4.2.4 and how to maintain policy fidelity in Section 4.2.5.

4.2.1. Learning Objective

Consider the dataset $\mathsf{D} = \mathsf{f}(s_i, \pi(\,j s_i))\mathsf{g}_{i=i}^n$ of state-action-distribution pairs that we generate by running simulations with the policy π in the environment. Our goal is to minimize the learning objective defined by

$$L = \sum_{i=1}^{n} \mathsf{j}\mathsf{j}\pi(\,j s_i) \quad \pi(\,j \hat{s}_i)\mathsf{j}\mathsf{j}_2^2 + \lambda_1 D_{\mathrm{KL}}(p_i \mathsf{k} u_i) + \lambda_2 \mathsf{j}\mathsf{j} g_i \mathsf{j}\mathsf{j}_1. \tag{1}$$

In this objective, we need to balance three things: 1) keep the same policy behavior while states are masked by minimizing $\mathsf{k}\mathsf{j}\pi(\,j s_i) \quad \pi(\,j \hat{s}_i)\mathsf{j}\mathsf{j}_2^2$, discussed in Section 4.2.5, 2) form tight clusters so that attention maps of all member states in a hyperstate are approximately the same, which we get by minimizing $D_{\mathrm{KL}}(p_i \mathsf{k} u_i)$, described in Section 4.2.2, 3) make attention maps spare and therefore also explanations sparse by minimizing $\mathsf{j}\mathsf{j} g_i \mathsf{j}\mathsf{j}_1$, elaborated in Section 4.2.3. By iteratively minimizing this objective, we learn the function f_θ to cluster states into hyperstates and the function g_ϕ to produce attention maps.

4.2.2. Clustering

The clustering function f_θ is represented by a feedforward neural network outputting a categorical distribution. We aim to make each state have a high probability of belonging to one of the hyperstates. In other words, we want u_i to be close to the Dirac delta distribution. In turn, outputting the same attention map for all states in a hyperstate. We iteratively refine hyperstate purity by following Xie et al. (2016)'s approach. To do so, we minimize the Kullback–Leibler divergence defined by

$$D_{\mathrm{KL}}(p_i \mathsf{k} u_i) = \sum_j p_{ij} \log \frac{p_{ij}}{u_{ij}}, \qquad \text{where } p_{ij} = \frac{u_{ij}^2 / h_j}{\sum_{j'} u_{ij'}^2 / h_{j'}} \text{ and } h_j = \sum_i u_{ij}. \tag{2}$$

p_{ij} is interpreted as the probability of assigning a state s_i to a hyperstate H_j. This objective iteratively refines the hyperstates using a normalization term to avoid all states being assigned to one large hyperstate.

4.2.3. Creating attention map

To compute the attention maps, we take u_i and feed it through a linear layer followed by the sigmoid activation defined by $g_i = h_\phi(u_i) = \sigma(\phi_w u_i + \phi_b)$. These values are bounded 0–1, which provides an understanding of how much each feature in the m-dimensional state contributes to the policy output relative to the baseline state. The baseline state represents the "absence" of features and is discussed in the next subsection. We aim to obtain a sparse g_i to make it easier to understand which features contribute to the decision-making. To do so, we minimize the L^1 norm of the attention map jjg_ijj_1. We use the L^1 norm since it performs regularization and feature selection.

4.2.4. State Masking

We mask the state s_i with the attention map g_i to create the masked state $\hat{s}_i$. We do this by first defining a baseline state s_{baseline} that represents "missingness" or "absence" of features (Sturmfels et al., 2020). The goal of s_{baseline} is to represent a state where the evidence used for prediction is removed and therefore referred to as missingness. As previously mentioned, the motivation to use a baseline is connected to how humans attribute causes. That is, using the absence of causes as the reference point, we reason about and blame causes for the resulting outcome (Sundararajan et al., 2017; Shrikumar et al., 2017).

In computer vision, it is common to represent the baseline with the constant value 0. However, using 0 is troublesome and can have unintended effects, especially when we have symbolic state representations where 0 in some features is impossible. For instance, it is impossible to observe 0 meter for the height of a person. Thus, setting a feature to 0 naively can create unrealistic and out-of-distribution baseline states. We define the start state s_0 as our baseline to combat this issue. In the start state, an agent often starts with nothing, such as no speed and no acceleration. Although not always true, it represents an easily defined valid state.

After defining a baseline representing missingness, we compute the masked state $\hat{s}_i$ by

$$\begin{aligned} \hat{s}_i &= g_i \quad s_i + (1 \quad g_i) \quad s_{\text{baseline}} \\ \hat{s}_i &= s_{\text{baseline}} + (s_i \quad s_{\text{baseline}}) \quad f_\theta(g_\phi(s_i)) \end{aligned} \tag{3}$$

where is the elementwise product. Equation (3) creates a new state $\hat{s}_i$ where features are masked and the amount of masking depends on how important they are. If the policy acts the same in both states s_i and s_{baseline} then the values in g_i will be 0 or close to 0, thus $\hat{s}_i \quad s_{\text{baseline}}$. This is seen in Figure 1 where attention on both features in H_1 are 0. Attention on the features are 0 because the distribution over actions for states in H_1 and the baseline state are approximately the same. This does not imply state semantics are neglected, as we see in Section 5.3.

4.2.5. Matching distribution over actions

To obtain faithful attention maps, we make sure $\pi(\,j\hat{s}_i) \quad \pi(\,js_i)$. This is achieved by minimizing the mean squared error between the distribution over actions, defined by $jj\pi(\,js_i)$ $\pi(\,j\hat{s}_i)jj_2^2$. For this method, we require the policy to be differentiable and that we can compute the distribution over actions. We do not update the policy and keep it fixed during training.

5. Experiments

The aim of the experiments is to show that our method satisfies desirable properties outlined in Section 4.1 by only using attention maps to cluster states. In addition, the experiments intend to demonstrate that our attention maps are faithful to the policy they explain.

First, we introduce our experimental setup in Section 5.1. Second, we qualitatively investigate the insight provided by our method and whether it matches our intuition in Section 5.2. Third, Section 5.3 introduces the use of t-SNE and silhouette score (Rousseeuw, 1987) to check how well our method merges semantically similar states and separates semantically dissimilar states. Fourth, in Section 5.4, we test hyperstate fidelity by assigning one action to each hyperstate and acting with the hyperstates instead of the policy in the environment. Last, Section 5.5 looks into how masked states affect the policy's performance.

5.1. Experimental Setup

Environments. We tested our methods in five different environments with various state space complexity and different action spaces. Mountain Car v0, Cart Pole v1, Acrobot v1, and Flappy Bird v0 (Kubovčík, 2023) have a discrete action space. While Swimmer v4 (Todorov et al., 2012) has a continuous action space. **Policies.** We utilized the policies implemented by CleanRL v1.0.0 (Huang et al., 2022) in PyTorch v1.13.1. We used the Proximal Policy Optimization (PPO) algorithm (Schulman et al., 2017) to train policies for all the experiments. The neural network architectures used are those prespecified by the CleanRL implementation. The hyperparameters used to train the policies generally follow those prespecified by CleanRL. We used the Adam optimizer (Kingma and Ba, 2015) to optimize the parameters in this work. **Clustering.** We used K-means clustering throughout the experiment to compare to our method. For Mountain Car and Cart Pole, we used 5 hyperstates, 10 hyperstates for Acrobot and Swimmer, and 11 hyperstates for Flappy Bird. The number of hyperstates was chosen to satisfy the hyperstate properties in section 4.1 while at the same time keeping the number of hyperstates as low as possible. This required multiple runs with manually inspecting the resulting hyperstate assignment and selecting one by balancing these considerations.

5.2. Hyperstate Interpretation

This section illustrates how to leverage explanations and shows the value of providing hyperstate-level attention maps. We investigate the Flappy Bird example shown in Figure 3, where the timing of actions plays a central role. There are two actions in this environment: do nothing or flap. The symbolic feature version of this environment used in the experiments consists of 12 features. The first three features 0–2 corresponds to the first pipe's horizontal position, the upper vertical position, and the lower vertical position. Features 3–5 represent the second pipe and features 6–8 represent the third pipe, respectively. The final three features are the agent's vertical position, vertical velocity, and rotation.

As the agent usually does not flap, it is interesting to investigate when it does. From Figure 3, we see the agent flaps in two of the hyperstates H_5 and H_6. Although not seen in the figure, the agent sometimes flaps in H_8 to keep it above ground before reaching the first pipe. In H_5, we see from the attention map that the agent decides to flap based on the

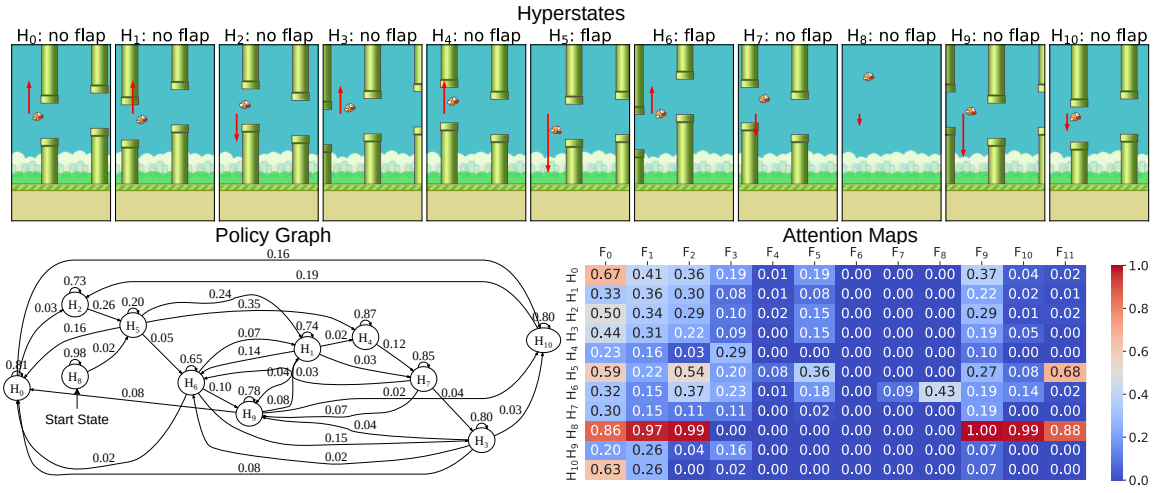


Figure 3: An explanation produced by ASAP for the Flappy Bird environment. The explanation is the hyperstate level attention maps with supporting elements. First, representative states of the learned hyperstates where the red arrow length denotes vertical velocity. Second, the policy graph, a Markov chain showing how the policy traverses the hyperstate space. Only edges where transitions have 2% or larger probability are included for ease of visualization. The attention maps show the attention on features when making decisions relative to the baseline state. The start state, which is the baseline, closely resembles the representative state of H_8.

first pipe's horizontal F_0 and vertical positions F_1 and F_2, especially on the lower vertical position F_2. This makes sense since the first pipe decides whether the agent survives. The agent also pays attention to the second pipe's horizontal and lower vertical positions F_3 and F_5 because it likely affects how much the agent should flap. Too much flapping might cause the agent to crash into the upper second pipe. Interestingly, the agent uses its rotation F_{11} which is related to when it last flapped and is not concerned about its velocity F_{10}. This may be because the vertical velocity changes instantaneously when the agent flaps. In H_6, when the agent flaps to pass the second pipe from a lower position, surprisingly, it seems like it uses the first pipe to determine its position relative to the second pipe. Similar to H_5, it cares about the vertical position of the pipe that comes after. The policy graph shows that the agent stays longer in H_6 than H_5, which makes sense since it has to move from the first piper with a lower vertical gap to the next pipe with a higher vertical gap.

Besides the flapping situations, we observe from the policy graph that the agents stay longer in H_8 than the rest of the hyperstates. This means it takes a while before the first pipe appears after starting the game. In this hyperstate, the agent uses information about its vertical position, vertical velocity, and rotation more than in other hyperstates. This matches our intuition because the agent has no other information to act upon besides the information about itself. When the pipes appear, the agent receives a lot of redundant information and can use it to gauge its vertical position. Therefore, its vertical position

no longer matters. The policy graph shows that the agent goes to H_5 after H_8 when it encounters the first pipe. Afterward, depending on if the gap on the second pipe is higher or lower, it can take two different paths: 1) H_1 ! H_6 ! H_3 or 2) H_4 ! H_7 ! H_9. Next, when the first pipe disappears, the agent goes to H_0 and restarts the cycle. Because the environment is stochastic, there are many paths to take besides the one we outlined, as reflected in the complexity of the policy graph.

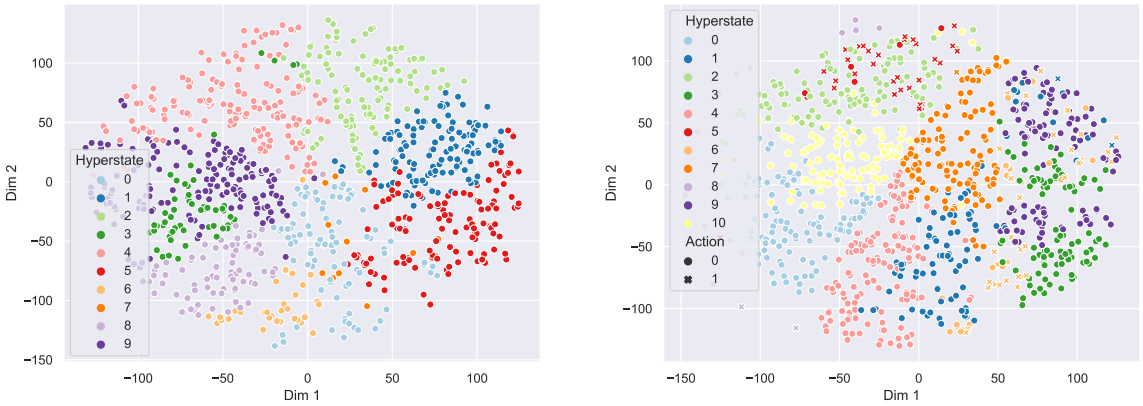
5.3. State Semantic

One important property to have when making hyperstates is to group semantically similar states. To test if ASAP can do that, we plot the state space of Swimmer and Flappy Bird using t-SNE. t-SNE visualizes similarity and dissimilarity between points in a high-dimensional space. If ASAP assigns states close in the t-SNE plot to the same hyperstates, it suggests that ASAP retains state semantics when constructing hyperstates. In Figure 4(a), we observe that our method groups states reasonably well semantically, especially since this is not explicitly encouraged via the loss function. Also, the plots are created using default hyperparameters; thus, they are not overfitted. For an environment with discrete action space, we investigate the states space of Flappy Bird in Figure 4(b). We first see that H_8 is quite spread out, but from Figure 3, we know that those are states before passing the first pipe after starting. The second thing is that H_5 and H_6 overlap with other hyperstates and are too spread out. These two hyperstates are situations where the agent flaps. Thus, it is hard to avoid the overlap while trying to capture how the agent behaves. Besides these hyperstates with low tightness and low separations from other hyperstates, there is some unwanted spread and overlap between H_3 and H_9. Nevertheless, we do not optimize for this in the objective function while still being able to capture these patterns.

t-SNE provides a qualitative measure, thus allowing different interpretations. The silhouette score is often used as a metric to evaluate cluster consistency. We compute the silhouette score using the state and Euclidean distance to test if state semantics are preserved. The silhouette score measures how similar a state is compared to other states in its own hyperstate and to states in other hyperstates. The aim is to have hyperstates that are tight and well separated from other hyperstates. We compare our method to hyperstates created by clustering on various other information. This includes action, state value, the activation of the policy's penultimate layer, and explanations created by three feature attribution methods implemented in Captum (Kokhlikyan et al., 2020): 1) deep learning important features (DeepLIFT) (Shrikumar et al., 2017), 2) feature ablation, which perturbs the input by replacing input features with baseline values, and 3) Shapley value sampling, which is based on perturbation to compute Shapley values (Strumbelj and Kononenko, 2010). Besides directly clustering on the state to create hyperstates, our method performs best or second best based on the silhouette score, as seen in Table 1. Although the silhouette score has the disadvantage of preferring spherical clusters, it quantitatively indicates that our approach does not neglect state semantics when creating hyperstates.

5.4. Hyperstate Fidelity

To give explanations such as, "if given the state s from the hyperstate H_i, then the policy will execute the action a", we need the policy to take the same action for all "critical"



(*a*) Swimmer state space (*b*) Flappy Bird state space

Figure 4: State spaces of the Swimmer and Flappy Bird environments produced using t-SNE with Scikit-learn v1.2.2 and its default hyperparameters.

	Environment				
Info	Mountain Car	Cart Pole	Acrobot	Flappy Bird	Swimmer
Action	0.030	−0.001	0.020	0.067	0.064
State Value	0.191	−0.023	−0.032	−0.092	−0.070
Activation	0.045	0.052	**0.120**	**0.142**	0.080
Feature Attribution	−0.035	−0.055	−0.076	−0.048	0.072
Shapley Values	−0.084	**0.134**	−0.143	−0.148	**0.133**
DeepLIFT	−0.073	−0.086	−0.066	−0.093	−0.025
ASAP (Ours)	**0.226**	0.127	0.057	0.103	0.101

Table 1: Silhouette score of states clustered based on information extracted from the policy.

states in a hyperstate. Thus, to test the ability of our methods to create hyperstates that group states with the same actions together, we test how well the hyperstates can be used as a substitute for the policy they explain. For each hyperstate, we take all states from the training set assigned to that hyperstate, average the action probability for those states, and assign the action with the highest probability to the hyperstate. Afterward, we simulate the environments and act with the hyperstates as a substitute for the policy.

Table 2 shows how dividing states into hyperstates based on various information extracted from the policy reflects the policy's behavior. We see that our approach and the activation of the policy's penultimate layer capture the behavior best, besides including the action itself in the clustering process. The only exception is in Swimmer where the feature attribution methods also capture the policy's behavior. In this environment, the policy has two outputs, so we averaged the outputs rather than computing them with respect to the

Method	Environment				
	Mountain Car	Cart Pole	Acrobot	Flappy Bird	Swimmer
State	−200.00	43.89 ± 8.14	−500.00	2.10	51.48 ± 20.69
State Value	−200.00	9.26 ± 0.77	−500.00	2.10	13.91 ± 7.82
Activation	**−114.38 ± 1.35**	201.47 ± 8.29	**−88.58 ± 24.63**	2.10	309.94 ± 7.90
Feature Ablation	−200.00	19.48 ± 4.70	−247.19 ± 77.03	2.10	334.10 ± 7.44
Shapley Values	−200.00	49.66 ± 26.03	−412.97 ± 100.66	2.10	309.65 ± 7.74
DeepLIFT	−200.00	15.05 ± 4.93	−490.09 ± 28.23	2.10	39.69 ± 23.26
ASAP (Ours)	−133.09 ± 33.71	**500.00**	−96.49 ± 34.03	**457.71 ± 121.60**	**337.67 ± 3.33**

Table 2: Policy Performance with deterministic action for each hyperstate. The sample return mean and the standard deviation is computed over 100 episodes with a different random seed for each episode.

Method	Environment				
	Mountain Car	Cart Pole	Acrobot	Flappy Bird	Swimmer
No Masking	−114.42 ± 19.17	**500.00**	−73.10 ± 11.31	479.83 ± 108.63	**305.92 ± 7.31**
Feature Ablation	−113.48 ± 0.89	**500.00**	−75.36 ± 11.76	2.44 ± 2.38	176.75 ± 14.54
Shapley Values	−119.64 ± 16.87	**500.00**	−79.58 ± 19.84	9.72 ± 1.54	88.66 ± 17.33
DeepLIFT	**−111.05 ± 4.37**	**500.00**	−74.40 ± 17.88	13.17 ± 2.01	293.64 ± 5.93
ASAP (Ours)	−113.06 ± 17.52	**500.00**	**−72.34 ± 11.30**	**492.31 ± 74.78**	305.57 ± 6.84

Table 3: Sample return mean and standard deviation over 100 episodes with input features masked according to Equation (3). Each episode is initialized with a new seed.

action of interest, which might affect performance. Regarding Flappy Bird, the timing is important, making replacing the policy with one action per hyperstate difficult.

5.5. Policy Performance With Masked Observations

To evaluate the faithfulness of our attention maps, we mask states using the attention maps generated by our attention method and three feature attribution methods that utilize baselines: DeepLIFT, feature ablation, and Shapley value sampling. Explanations computed by the other methods are with respect to the action with the highest probability. To mask the input with the other feature attribution methods, we normalized their values to the range 0–1 and utilize Equation (3) to compute the masked state.

In Table 3, we see attention maps produced by ASAP mask states without affecting policy performance. On the one hand, this is to be expected since ASAP optimizes this in the loss function. On the other hand, ASAP restricts the attention maps to be approximately the same for all states in a hyperstate, thus limiting the degree of freedom. Surprisingly, the other methods not optimized for this task mask states without the policy losing too much performance and even performing better in the Mountain Car environment. In the Swimmer environment, the other feature attribution methods perform worse. The behavior might be related to that we average outputs to compute feature attributions. As for the

Flappy Bird, they do not work because the timing of the action plays a central role, and a small error on when to flap can end the episode prematurely.

6. Conclusion and Future Work

This paper introduced a new attention-based state abstraction method, ASAP. ASAP produces global explanations and works post hoc by end-to-end learning hyperstates and attention maps describing them. The attention maps are the only information used in the clustering objective, contrasting previous methods using state, action and/or state value. The policy is only indirectly used in clustering via learned attention maps. Accordingly, this enabled us to make statements about hyperstates and policy behavior in hyperstates using attention maps, as we did not use the action in the clustering process. Also, we did not refer to non-human intuitive quantities like the state value. We qualitatively and quantitatively showed that ASAP captures state semantic and policy behavior without leveraging this information in the training objective. This resulted in hyperstates with semantically similar states where the policy behaves alike. Further, we demonstrated that the attention maps are faithful to the policy and can mask states without affecting policy performance.

For future work, we should 1) develop methods for determining the number of hyperstates, which is a difficult problem. On the one hand, the more complex environments need more hyperstates to capture the agent's behavior. On the other hand, we need to consider several aspects of the explainee's needs. If the explainee wants hyperstates to represent a single action each, how similar states in a hyperstate are, and feature attention similarity. Also, we need to avoid overwhelming the explainee by having too many hyperstates, making explanations less valuable. 2) Investigate how to create hyperstate descriptions suited for non-experts. The current representation requires understanding the features, which may require domain knowledge and is unsuited for non-experts. 3) Conduct user studies of ASAP. User studies provide new insights and are important to understanding the real-world usefulness of explanations. Nevertheless, there are some shortcomings. They are expensive to perform, thus resulting in researchers using Amazon Mechanical Turk and university students for evaluation, which are not necessarily the intended end-users for the explanations. Moreover, user studies are often not comparable across papers, indicating the importance of evaluations besides user studies. 4) Research how ASAP can be adapted to more complex environments with high-dimensional states. One question is how to represent hyperstates when the input is pixels and high-dimensional while still being human-understandable. Another is handling the number of hyperstates so that explanations are faithful and capture agent behavior without overwhelming the explainee. This may require dividing the state space into subspaces before applying ASAP.

References

Dan Amir and Ofra Amir. HIGHLIGHTS: Summarizing Agent Behavior to People. In *Proc. of AAMAS*, 2018.

Tom Bewley, Jonathan Lawry, and Arthur Richards. Summarising and Comparing Agent Dynamics with Contrastive Spatiotemporal Abstraction. *IJCAI Workshop on XAI*, 2022. doi: 10.48550/arXiv.2201.07749.

Noam Brown and Tuomas Sandholm. Libratus: The Superhuman AI for No-Limit Poker. In *Proc. of IJCAI*, 2017. doi: 10.24963/ijcai.2017/772.

Samuel Greydanus, Anurag Koul, Jonathan Dodge, and Alan Fern. Visualizing and Understanding Atari Agents. In *Proc. of ICML*, 2018.

Alexandre Heuillet, Fabien Couthouis, and Natalia Díaz Rodríguez. Explainability in deep reinforcement learning. *Knowl. Based Syst.*, 2021. doi: 10.1016/j.knosys.2020.106685.

Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. CleanRL: High-quality Single-file Implementations of Deep Reinforcement Learning Algorithms. *JMLR*, 2022.

Hidenori Itaya, Tsubasa Hirakawa, Takayoshi Yamashita, Hironobu Fujiyoshi, and Komei Sugiura. Visual Explanation using Attention Mechanism in Actor-Critic-based Deep Reinforcement Learning. In *Proc. of IJCNN*, 2021. doi: 10.1109/IJCNN52387.2021.9534363.

Rahul Iyer, Yuezhang Li, Huao Li, Michael Lewis, Ramitha Sundar, and Katia P. Sycara. Transparency and Explanation in Deep Reinforcement Learning Neural Networks. In *Proc. of AIES*, 2018. doi: 10.1145/3278721.3278776.

Zoe Juozapaitis, Anurag Koul, Alan Fern, Martin Erwig, and Finale Doshi-Velez. Explainable Reinforcement Learning via Reward Decomposition. In *IJCAI Workshop on XAI*, 2019.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *Proc. of ICLR*, 2015. doi: 10.48550/arXiv.1412.6980.

Narine Kokhlikyan, Vivek Miglani, Miguel Martin, Edward Wang, Bilal Alsallakh, Jonathan Reynolds, Alexander Melnikov, Natalia Kliushkina, Carlos Araya, Siqi Yan, and Orion Reblitz-Richardson. Captum: A unified and generic model interpretability library for PyTorch. *CoRR*, 2020. doi: 10.48550/arXiv.2009.07896.

Martin Kubovčík. Flappy Bird for Gymnasium, 2023.

Ronny Luss, Amit Dhurandhar, and Miao Liu. Local Explanations for Reinforcement Learning. *Proc. of AAAI*, 2023. doi: 10.48550/arXiv.2202.03597.

Joe McCalmon, Thai Le, Sarra Alqahtani, and Dongwon Lee. CAPS: Comprehensible Abstract Policy Summaries for Explaining Reinforcement Learning Agents. In *Proc. of AAMAS*, 2022. doi: 10.5555/3535850.3535950.

Alexander Mott, Daniel Zoran, Mike Chrzanowski, Daan Wierstra, and Danilo Jimenez Rezende. Towards Interpretable Reinforcement Learning Using Attention Augmented Agents. In *Proc. of NeurIPS*, 2019.

Nikaash Puri, Sukriti Verma, Piyush Gupta, Dhruv Kayastha, Shripad Deshmukh, Balaji Krishnamurthy, and Sameer Singh. Explain Your Move: Understanding Agent Actions Using Specific and Relevant Feature Attribution. In *Proc. of ICLR*, 2020.

Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *J. Comput. Appl. Math.*, 1987. doi: 10.1016/0377-0427(87)90125-7.

Cynthia Rudin, Chaofan Chen, Zhi Chen, Haiyang Huang, Lesia Semenova, and Chudi Zhong. Interpretable machine learning: Fundamental principles and 10 grand challenges. *Statistics Surveys*, 2022. doi: 10.1214/21-SS133.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *CoRR*, abs/1707.06347, 2017.

Wenjie Shi, Gao Huang, Shiji Song, Zhuoyuan Wang, Tingyu Lin, and Cheng Wu. Self-Supervised Discovering of Interpretable Features for Reinforcement Learning. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2022. doi: 10.1109/TPAMI.2020.3037898.

Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning Important Features Through Propagating Activation Differences. In *Proc. of ICML*, 2017.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. *CoRR*, 2017. doi: 10.48550/arXiv.1712.01815.

Erik Strumbelj and Igor Kononenko. An Efficient Explanation of Individual Classifications using Game Theory. *J. Mach. Learn. Res.*, 2010. doi: 10.5555/1756006.1756007.

Pascal Sturmfels, Scott Lundberg, and Su-In Lee. Visualizing the Impact of Feature Attribution Baselines. *Distill*, 2020. doi: 10.23915/distill.00022.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic Attribution for Deep Networks. In *Proc. of ICML*, 2017.

Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *Proc. of IROS*, 2012. doi: 10.1109/IROS.2012.6386109.

Nicholay Topin and Manuela Veloso. Generation of Policy-Level Explanations for Reinforcement Learning. In *Proc. of AAAI*, 2019. doi: 10.1609/aaai.v33i01.33012514.

Laurens van der Maaten and Geoffrey Hinton. Visualizing Data using t-SNE. *JMLR*, 2008.

Junyuan Xie, Ross B. Girshick, and Ali Farhadi. Unsupervised Deep Embedding for Clustering Analysis. In *Proc. of ICML*, 2016.

Herman Yau, Chris Russell, and Simon Hadfield. What Did You Think Would Happen? Explaining Agent Behaviour through Intended Outcomes. In *Proc. of NeurIPS*, 2020.

Tom Zahavy, Nir Ben-Zrihem, and Shie Mannor. Graying the black box: Understanding DQNs. In *Proc. of ICML*, 2016.