Understanding Deep RL agent decisions: a novel interpretable approach with trainable prototypes

Caterina Borzillo[1,*], Alessio Ragno[1] and Roberto Capobianco[1,2]

[1]Department of Computer, Control and Management Engineering, Sapienza University of Rome, Rome RM 00185, Italy
[2]Sony AI

Abstract

Deep reinforcement learning (DRL) models have shown great promise in various applications, but their practical adoption in critical domains is limited due to their opaque decision-making processes. To address this challenge, explainable AI (XAI) techniques aim to enhance transparency and interpretability of black-box models. However, most current interpretable systems focus on supervised learning problems, leaving reinforcement learning relatively unexplored.

This paper extends the work of PW-Net, an interpretable wrapper model for DRL agents inspired by image classification methodologies. We introduce Shared-PW-Net, an interpretable deep learning model that features a fully trainable prototype layer. Unlike PW-Net, Shared-PW-Net does not rely on pre-existing prototypes. Instead, it leverages the concept of ProtoPool to automatically learn general prototypes assigned to actions during training. Additionally, we propose a novel prototype initialization method that significantly improves the model's performance.

Through extensive experimentation, we demonstrate that our Shared-PW-Net achieves the same reward performance as existing methods without requiring human intervention. Our model's fully trainable prototype layer, coupled with the innovative prototype initialization approach, contributes to a clearer and more interpretable decision-making process. The code for this work is publicly available for further exploration and applications.

Keywords

interpretable deep learning, reinforcement learning, explainable artificial intelligence

1. Introduction

Despite the successful and promising results of deep reinforcement learning (DRL) models over the last few decades, their practical use in critical domains remains constrained due to their unclear and enigmatic decision-making processes. To address this problem, explainable AI (XAI) techniques aim to enhance the transparency and understandability of black-box models [1]. Nevertheless, most of these recent interpretable systems address conventional supervised learning problems, such as image classification [2, 3, 4, 5, 6, 7, 8], with a limited emphasis on the reinforcement learning domain.

Kenny et al. [9] address the problem of building self-interpretable DRL agents by drawing inspiration and insights from existing methodologies for image classification tasks. Indeed, they develop an interpretable wrapper model for DRL agents, PW-Net. PW-Net aims to wrap and

*XAI.it 2023: 4th Italian Workshop on Explainable Artificial Intelligence | co-located with AI*IA 2023*
*Corresponding author.

encapsulate itself around any pre-trained RL agents, creating a comprehensive prototype-based framework that sheds light on its decision-making process, making it transparent and clear.

In this paper, we expand the work of Kenny et al. [9] by introducing an interpretable deep learning model called Shared-PW-Net. In particular, differently from PW-Net, we develop a fully trainable prototype layer, removing the dependence on pre-existing prototypes. To build such architecture, we take inspiration from the work of Rymarczyk et al. [3], with ProtoPool, giving the model the possibility of learning several general prototypes that are automatically assigned to the actions during the training phase. Alongside this, we propose a novel method for initializing the prototypes, which allows us to boost the model's performance.

Overall, the contributions of this work are the following[1]:

- we propose a fully trainable prototype layer that automatically learns prototypes without human supervision, achieving PW-Net reward results;
- we apply the differentiable prototype assignment mechanism to each class, previously introduced in ProtoPool, in order to enhance prototype sharing and network efficiency;
- we propose a new prototype initialization based on the relevance of certain actions to certain states.

The remainder of this work is organized as follows: Section 2 presents the most related literature to our work; in Section 3 we provide a general background about reinforcement learning and prototype-based models' architectures; Section 4 introduces our proposed approach in detail; in Section 5 we carry out the experimental setup and the analysis of the obtained results; Section 6 wraps up the results of this paper by analyzing its limitations and setting the base for future work.

2. Related Work

XAI is a rapidly evolving field that aims to develop artificial intelligence systems that can provide human-understandable explanations for their decisions and actions. One approach within XAI is the development of self-interpretable models [10]. These models are designed to inherently possess transparency and interpretability, allowing humans to comprehend and validate their decision-making processes. One popular self-interpretable architecture is the Prototypical-Part Network (ProtoPNet) [2]: a deep neural network that aims to solve the image classification task in an interpretable way. The key aspect of ProtoPNet is that during training it learns a set of prototypes for each label which are updated iteratively as the network learns to capture the distinctive features characterizing each class. At the inference step, it utilizes the learned prototypes to make predictions on new, unseen data: this is achieved by comparing the patches inside the input images with the prototypes, allowing the model to determine which class each patch belongs to.

Inspired by ProtoPNet, Rymarczyk et al. [3] propose ProtoPool, an interpretable prototype-based model for fine-grained image classification. Unlike other recent prototype-based approaches, ProtoPool presents two main novelties: first, it implements a fully differentiable assignment of prototypes to classes that simplifies the training process and reduces the number

[1]The code of this work is fully available at the following link: https://github.com/KRLGroup/shared_pw_net

of prototypes used for classifying images; consequently, the prototypes can be shared among classes, supporting the idea that many common visual features can occur in different classes. As second novelty, ProtoPool introduces a new focal similarity function that allows to better focus on salient visual features of an image, in contrast to other similarity metrics that focus on wider image areas making the interpretation harder to comprehend.

In the field of XRL, some works propose post-hoc methods involving attention weights [11, 12] or trees [13], however, these approaches lack transparency in revealing the agent's actions. Another compelling method involves distilling recurrent neural network (RNN) policies into finite-state machines [14, 15]. Nevertheless, this approach may not consistently yield easily analyzable outcomes and it is limited to RNNs.

In the context of this study, which combines RL and self-interpretable XAI models, highly significant work in the field of RL is the research conducted by Kenny et al. [9], which closely aligns with the objectives of our study. The paper, published early in 2023, introduces the Prototype-Wrapper Network (PW-Net) that, in contrast to the other studies, aims to build an "interpretable-by-design" DRL agent. The architecture benefits from human supervision since authors manually define both the prototypes (for network interpretability) and the last layer weights (to regulate the relationship between the prototypes and the corresponding actions). In PW-Net, in fact, the only trainable component is a set of projection networks, separately defined for each action-specific prototype. Kenny et al. [9] train PW-Net by means of distillation of a black-box agent and show that it does not lose any performance relative to the black-box agent.

Incorporating the concepts and methodologies proposed in the aforementioned works [9, 3, 2], the primary objective of Shared-PW-Net is to address the main task pursued by PW-Net, which takes advantage of the prototype mechanism to make an RL agent interpretable. At the same time, the proposed model integrates significant ideas from ProtoPool and ProtoPNet, such as the learning of the prototypes at training time and the concept of differentiable assignment of prototypes to classes with multiple slots per class. In addition to this, we introduce a novel prototype initialization.

3. Background

3.1. Reinforcement Learning

Reinforcement Learning (RL) is a prominent subfield of machine learning that focuses on teaching autonomous agents how to perceive and interpret their environment in order to take actions. Unlike supervised learning, in fact, where labeled examples are provided, or unsupervised learning, which seeks patterns in unlabeled data, RL focuses on the concept of an agent which interacts with an environment, receives feedback in the form of rewards or penalties, and learns to behave trying to maximize cumulative rewards.

RL relies on the concept of Markov Decision Process (MDP), a stochastic decision-making process that uses a mathematical framework to model the decision-making of a dynamic system. In an MDP, defined by the (S, A, T, R, γ) tuple [16], an agent takes actions A based on its observations of the current state S. Once the agent selects an action, it interacts with the environment by receiving rewards as feedback and by transitioning to a new state based on the probabilistic transition $T : S \times A \rightarrow S$. The agent's objective is to learn an optimal

policy $\pi : s \in S \rightarrow A$ that maximizes the expected discounted reward and serves as a strategic mapping from states to actions. By continuously exploring and exploiting the environment, the agent aims to refine its understanding of the state-action relationships, seeking the most advantageous course of action at any given state.

In this study, we deal with the so-called "black-box policies", pre-trained neural networks' models that operate without providing clear and transparent explanations for their decision-making process.

3.2. Prototype-based models

3.2.1. ProtoPool

ProtoPool [3] is an extension of ProtoPNet where the prototypes are assigned through a differentiable mechanism and shared among classes. The ProtoPool architecture consists of a convolutional network f, a prototype pool layer g, and a fully connected layer l. In g, Rymarczyk et al. [3] introduce a pool of M trainable prototypes $P = \{p_i \in R^D\}_{i=1}^{M}$ and K slots per class. Each slot is represented by a distribution $q_k \in R^M$ where the values in q_k indicate the probabilities of assigning successive prototypes to the k-th slot, with the sum of these probabilities $\|q_k\|$ that is equal to 1.

Given an input image x, $f(x)$ is a set of $H \times W$ vectors of dimension D, each corresponding to a specific patch of the image:

$$Z_x = \{z_i \in f(x) : z_i \in \mathbb{R}^D \, i = 1, \ldots, H \cdot W\}. \tag{1}$$

Then, for each slot, g calculates the aggregated similarity g_k between Z_x and all prototypes considering the distribution q_k:

$$g_k = \sum_{i=1}^{M} q_k^i \cdot g_{p_i}, \tag{2}$$

where g_{p_i} is a novel focal similarity function:

$$g_p(z) = \log(1 + \frac{1}{\|z - p\|^2}), \tag{3}$$

$$g_p = \max_{z \in Z_x} g_p(z) - \text{mean}_{z \in Z_x} g_p(z). \tag{4}$$

Finally, in h the K similarity scores per class are multiplied by the weight matrix w_h to obtain the output logits that then undergo a softmax transformation to produce the prediction:

$$\hat{y} = \text{softmax}(w_h \cdot g_p). \tag{5}$$

The soft prototypes' assignment has two main constraints: first, they assign one prototype per slot, and second, they assign successive slots of a class to different prototypes. The first constraint is satisfied by the use of the differentiable Gumbel-Softmax estimator [17] such that for each slot k given $q_k = (q_1, \ldots, q_M)$ and $\tau \in (0, \infty)$:

$$\text{Gumbel-softmax}(q, \tau) = (y^1, \ldots, y^M) \in R^M, \tag{6}$$

where $y^i = \frac{\exp\left(\frac{q^i+\eta_i}{\tau}\right)}{\sum_{m=1}^{M}\exp\left(\frac{q^m+\eta_m}{\tau}\right)}$ and η_m for $m \in 1,\dots,M$ are samples drawn from the Gumbel distribution.

Regarding the second constraint, the slots' orthogonality is enforced by the orthogonal loss function L_{orth}:

$$L_{\text{orth}} = \sum_{i<j}^{K} \frac{\langle q_i, q_j \rangle}{\|q_i\|_2 \cdot \|q_j\|_2}. \tag{7}$$

3.2.2. PW-Net

PW-Net [9] is an interpretable neural network that serves as a "wrapper" around any pre-trained RL agent. In particular, PW-Net uses a supervised approach for distilling an RL agent to make it interpretable: a pre-trained RL agent with black-box policy π_{bb} represents the desired behavior that the prototype-based network aims to reproduce. From $\pi_{\text{bb}}(s) = Y f_{\text{enc}}(s) + b$, the encoder f_{enc} is taken and placed within the PW-Net architecture. In light of this, the network is composed of the encoder f_{enc}, a set of projection networks $h_{i,j}$ (one for each action $i \in [1, N]$ and prototype $j \in [1, M]$) and two human-defined layers, the prototype layer and the weight matrix W'. Initially, a state s is mapped to a latent representation z through the encoder network:

$$z = f_{\text{enc}}(s). \tag{8}$$

Then, for each action i and prototype j a separate linear layer $h_{i,j}$ transforms z to a specific representation $z_{i,j}$:

$$z_{i,j} = h_{i,j}(z) \quad \forall i \in [1, N], j \in [1, M_i]. \tag{9}$$

Successively, the similarity function from Chen et al. [2] is used to score the distance between $z_{i,j}$ and the prototype $p_{i,j}$:

$$\text{sim}(z_{i,j}, p_{i,j}) = \log\left(\frac{(z_{i,j} - p_{i,j})^2 + 1}{(z_{i,j} - p_{i,j})^2 + \varepsilon}\right). \tag{10}$$

.

In the end, output actions are generated by combining the similarity scores with W':

$$a'_i = \sum_{j=1}^{M_i} W'_{i,j}\text{sim}(z_{i,j}, p_{i,j}). \tag{11}$$

4. Proposed Approach

Inspired by the work of Kenny et al. [9], we present Shared-PW-Net. This neural network can transform pre-trained neural network-based algorithms for RL agents, which we refer to as "black-box models", into self-interpretable prototype-based ones. The main idea behind our approach is to wrap the RL agent with a prototypical layer trained through the distillation of the original one. In the following sections, we present the overall architecture of Shared-PW-Net, some detailed information about the novel prototype initialization, and the differentiable assignment of prototypes to actions.

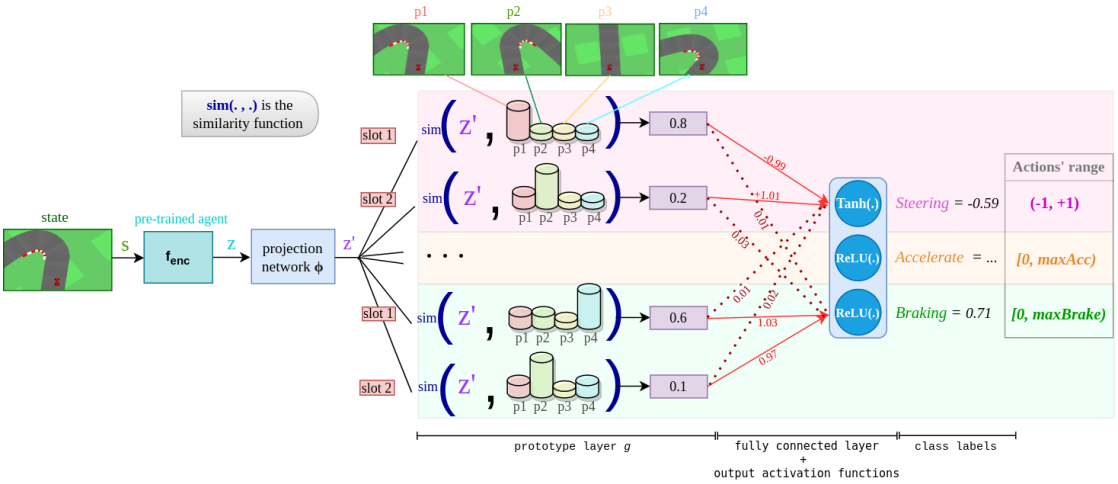

Figure 1: Shared-PW-Net architecture: The framework is instantiated here in the Car Racing domain from OpenAI's gym (Section 5.1.1). The state s first is encoded as z with f_{enc}, then, it undergoes the projection network ϕ. Successively, the network computes the similarity between the state z' and each prototype for each class and slot. Then, the similarity score is fed into the fully connected layer W to obtain the output action. A set of k prototypes, during training, is assigned to each action: this means that, for example, the network's aim is to learn that the "Steering" action can be performed in two situations that are steering on the right (output value near to +1) and steering on the left (output value near to -1) through the use of the *Tanh* activation function.

4.1. Shared-PW-Net

4.1.1. Architecture

In Figure 1, we report a scheme of Shared-PW-Net in the Car Racing environment. The network consists of a f_{enc} stage, a projection network ϕ, a prototype layer g, and a fully connected layer W. As in Kenny et al. [9], f_{enc} represents the encoded pre-trained RL agent that our interpretable network uses as desired behavior. g contains a pool of M trainable prototypes $P = \{p_i \in R^D\}_{i=1}^{M}$ and $k \in [1..K]$ slots for each action $n \in [1..N]$. The prototypes are initialized through our novel initialization technique and are assigned to each action by employing the Rymarczyk et al. [3] soft assignment technique. According to this strategy, only one prototype is assigned to each slot and, within one action, successive slots are assigned to different prototypes.

The network takes the RL agent's current state s, already fed in f_{enc}, and passes it through to ϕ, which projects it in the latent space:

$$z = f_{\text{enc}}(s), \tag{12}$$

$$z' = \phi(z) \quad \forall i \in [1, N], j \in [1, M_i]. \tag{13}$$

Successively, the network performs the similarity function from Chen et al. [2] between the state z' and the prototype p_{k_n} associated with that slot k which belongs to a certain action n:

$$\text{sim}(z', p_{k,n}) = \log\left(\frac{(z' - p_{k,n})^2 + 1}{(z' - p_{k,n})^2 + \varepsilon}\right). \tag{14}$$

After this, the similarity score is fed into W and then to the output activation function corresponding to that particular action n in order to obtain the predicted action a_i'.

$$out = W \cdot \text{sim}(z', p_{k,n}), \tag{15}$$

$$a_i' = \sigma(out), \tag{16}$$

where σ is the general activation function. In the specific case of the Car Racing domain (see Section 5.1.1), for the *steering* action we use the *Tanh* activation function because we suppose that the network, through the fully connected layer, learns that the value +1 is associated to "Turn right" action and the value -1 is associated to "Turn left" action. For *braking* and *accelerate* actions, instead, we employ the *ReLU* activation function because we want a unique value for expressing the intensity of the actions.

4.1.2. Novel prototype initialization

Apart from attempting the conventional initialization of prototypes with random numbers drawn from a standard distribution, we also investigate a novel prototype initialization technique. This novel approach involves a clustering operation on all states from which the pre-trained RL agent executes a specific action. In detail, given the list of states S and actions a of the dataset Q, for each action, we perform a clustering on the states where the black-box model performs that action. In particular, we use the number of slots K as the number of clusters in order to have at least K prototypes for each action. This procedure allows us to extract $K \cdot N$ centroids of the clusters. We select the first M centroids, and we use them as prototypes.

In the specific case of the continuous action space for the Car Racing environment, at each step i the agent performs the action $a_i \in R^3$ where each of the three components is the continuous action:

$$\begin{cases} steering & \in [-1, +1] \\ accelerate & \in [0, maxAcc) \\ braking & \in [0, maxBrake) \end{cases} \tag{17}$$

This indicates that the agent, for example, can steer to the right and brake simultaneously. For this reason, for the prototype initialization, we simply find the mean value m of each action performed by the RL agent in Q. Then, for each action, we collect the set of states where that action is executed with an absolute value that is greater than m. In the end, we use the K-means clustering to identify the centroids and we use them as prototypes.

4.1.3. Assignment of prototypes to actions

As already proposed and studied by Rymarczyk et al. [3], we adopt a soft assignment method for prototypes based on prototype distributions to enhance the interpretability of the model and to utilize the prototypes from the pool efficiently. Unlike traditional hard assignments, this soft assignment uses a differentiable *argmax* function, specifically the Gumbel-Softmax estimator [17], to assign exactly one prototype per slot for each action.

Equation 6 shows the computations performed to ensure differentiability for each slot. At training time, we initialize the Gumbel-Softmax distribution with $\tau = 1$ and gradually reduce it

to 0.001 over 30 epochs. Moreover, we extend the loss function to enforce orthogonality between prototypes assigned to the same action. This orthogonality constraint (previously shown in Equation 7) prevents multiple slots from being assigned to the same prototype, maximizing the potential of the prototype pool layer.

4.1.4. Prototype projection

Prototype projection is an essential stage in the training process, as it aids in visualizing the learned prototypes and helps in better understanding the features they represent. In this step, abstract prototypes, which the model learns during training, are replaced with the actual representations of the nearest training data. By using the representations of actual training data, the model's prototypes become more interpretable, allowing one to gain insights into the learned features and their relevance to the task. We perform prototype projection every two epochs, only after the first 10 epochs.

5. Experiments

In this study, we investigate how Shared-PW-Net performs compared to its counterpart, PW-Net, and the black-box agents in different DRL settings. We conduct tests in two different types of environments: first, we assess Shared-PW-Net performance in a DRL domain with multiple continuous output actions, and then, we conduct tests in a different domain with discrete output actions. These two experiments use a state space solely represented by raw pixel data; however, PW-Net could also be applied to other state-action representations, such as tabular environments. Finally, we study the influence of the proposed initialization procedure and the role of the number of prototypes and slots of the model.

5.1. Experimental Setup

In this section we present the setup we use to carry out the experiments. In particular, we evaluate each model on 30 episodes for 15 seeds. We compare our proposed method with PW-Net [9] and the black-box we use for the distillation procedure. As PW-Net, differently from

Table 1
Agents performance on the Car Racing and Atari Pong environments where the highest reward results of the RL agent are highlighted in bold.

	Car Racing		Atari Pong	
Model	**Reward**	**MSE**	**Reward**	**Accuracy**
Shared-PW-Net	**219.35 ± 0.41**	0.44 ± 0.03	**13.03 ± 0.74**	88.08 ± 0.01
PW-Net	218.57 ± 0.08	0.06 ± 0.00	10.72 ± 0.26	88.93 ± 0.00
PW-Net*	112.05 ± 0.30	343.80 ± 11.4	8.85 ± 1.69	84.84 ± 0.76
PW-Net**	111.97 ± 0.35	2.88 ± 0.15	9.97 ± 0.84	81.73 ± 0.00
Black-Box	218.41 ± 3.00	-	11.94 ± 0.16	-

Shared-PW-Net, uses fixed prototypes, for a fair comparison, we also compare the following two variations of PW-Net: PW-Net* employs a single projection network for state projection instead of P projection networks, it includes both prototypes and the final weight matrix trainable (hence, not human-defined) and it performs the prototype projection only at the end of the training phase; PW-Net**, on the other hand, shares the same structural characteristics of PW-Net*, except that it undergoes the same prototype projection scheme of Shared-PW-Net.

5.1.1. Continuous action space

We test our model in Car Racing continuous action domain from OpenAI's gym environment [18]. For a fair comparison, we use the same environment settings from Kenny et al. [9]. In the Car Racing environment, the agent is rewarded for driving around the path as fast as possible while avoiding any deviations from the road and having as possible actions the following three continuous output actions: *steering*, *accelerate*, and *braking*. The pre-trained model used as black-box in this domain is the one by Jain [19], which is trained with the Proximal Policy Optimization (PPO) algorithm. In the first place, to determine the best hyperparameters for our network that produce the most desirable results, we perform a grid search over the number of prototypes P or the number of slots S.

5.1.2. Discrete action space

For the discrete action domain, we use Atari Pong (from OpenAI's gym environment). In this environment, the agent has to control a paddle and try to hit a ball back and forth, with the goal of preventing the ball from going past its paddle. The agent is rewarded for successfully scoring against the opponent, while it incurs penalties for being scored against. The agent's discrete actions are 6, each allowing the agent to move up or down. We use the pre-trained model from Şentürk [20], trained with the DQN algorithm. As Şentürk [20] employ a technique called "frame stacking" that involves taking multiple consecutive frames from the game as input to the neural network, we use four state representations for describing each prototype.

5.2. Agents performances

Table 1 illustrates the results achieved for the Car Racing and Atari Pong environments. We report the mean and standard deviation values over the different seeds for statistical significance. In the following paragraphs, we comment on the results obtained on the two environments.

5.2.1. Car Racing

Concerning the Car Racing environment, we evaluate the performance of each model using the average reward of the RL agent and the mean squared error (MSE) between the interpretable model's actions and the black-box' ones. In Table 1 we observe that Shared-PW-Net succeeds in reaching the same reward levels of PW-Net (and the black-box agent) even if it is less accurate. In light of this, Shared-PW-Net represents a valid alternative and a workable solution for setting up a self-interpretable model that does not rely on human intervention. The two straightforward modifications of PW-Net with trainable prototypes, PW-Net* and PW-Net**, on the other hand,

do not achieve remarkable results. This leads to demonstrating that the application of soft assignment of prototypes to classes and the utilization of the new prototype initialization (adopted in Shared-PW-Net) are fundamental for achieving significant results.

5.2.2. Atari Pong

In this domain, we use the reward of the RL agent and the accuracy of the model with respect to the black-box to evaluate the agent's performance in the environment. Also in the Atari Pong environment, the Shared-PW-Net outperforms PW-Net* and PW-Net** and successfully attains the same rewards values of PW-Net and the black-box model (Table 1).

5.3. Prototypes influence

To study the effect of the prototype number and initialization, we test various hyperparameter combinations and report the result in reward and distillation performances (Table 2).

Table 2
Hyperparameter study on the Car Racing and Atari Pong environments. We highlight the best results, in bold, and the ones within the standard deviation of the best, underlined. We identify with an asterisk the models initialized with the novel proposed approach.

	Car Racing		Atari Pong	
Model	**Reward**	**MSE**	**Reward**	**Accuracy**
Shared-PW-Net P4-S2	211.48 ± 5.85	4.14 ± 2.63	-	-
Shared-PW-Net P4-S2*	219.20 ± 0.40	0.78 ± 0.16	-	-
Shared-PW-Net P6-S2	219.05 ± 0.31	0.49 ± 0.07	12.49 ± 0.62	87.08 ± 0.01
Shared-PW-Net P6-S2*	**219.35 ± 0.41**	**0.44 ± 0.03**	12.13 ± 1.24	86.75 ± 0.01
Shared-PW-Net P6-S3	218.86 ± 0.33	0.64 ± 0.25	-	-
Shared-PW-Net P6-S3*	219.05 ± 0.28	0.50 ± 0.08	-	-
Shared-PW-Net P8-S3	214.30 ± 4.88	2.77 ± 2.19	11.84 ± 0.96	87.29 ± 0.01
Shared-PW-Net P8-S3*	219.09 ± 0.40	0.45 ± 0.06	**13.03 ± 0.74**	88.08 ± 0.01
Shared-PW-Net P8-S2	-	-	12.97 ± 0.52	88.42 ± 0.01
Shared-PW-Net P8-S2*	-	-	12.56 ± 1.01	87.09 ± 0.01
Shared-PW-Net P10-S3	-	-	12.27 ± 0.56	88.81 ± 0.00
Shared-PW-Net P10-S3*	-	-	12.58 ± 0.49	88.72 ± 0.00

5.3.1. Car Racing

For Car Racing, we test the following combinations: (i) 4 prototypes and two slots; (ii) 6 prototypes and two slots; (iii) 6 prototypes and three slots; (iv) 8 prototypes and three slots. We find the model (ii) to be the best performing. The choice in the number of prototypes and slots is guided by the number of classes (actions) within the Car Racing environment. Moreover, for each model, we test both the random initialization and the one proposed in Section 4.1.2. We

see that all the models trained with the novel prototype initialization perform better than the randomly initialized ones. Moreover, while we find consistent differences in performances for the randomly initialized models, all the models using the novel initialization reach comparable performances.

5.3.2. Atari Pong

The model exhibits consistently high performance in Atari Pong across different hyperparameter variations, notably concerning the number of prototypes and slots. While the novel initialization technique shows success in some instances, it is the Shared-PW-Net configuration, featuring eight prototypes and two slots per action, that attains the highest reward value. In this particular setup, the novel initialization proves to be the key factor in achieving the best model performance.

5.4. Model Interpretability

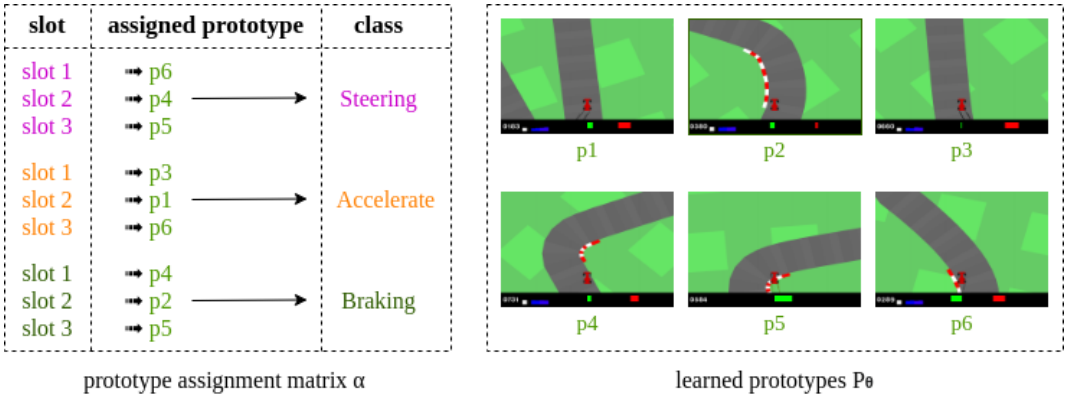

Figure 2: Illustration of the prototype assignment matrix learned by Shared-PW-Net (P6-S3 with the novel initialization procedure) together with the learned prototypes. Shared-PW-Net learns the prototype assignment matrix that assigns a prototype to each slot of each class. In the figure, the prototypes $p6$, $p4$, $p5$ are associated respectively to $slot1$, $slot2$, $slot3$ of *Steering* class; prototypes $p3$, $p1$, $p6$ are associated to $slot1$, $slot2$, $slot3$ of *Accelerate* class and the same logic is applied to the *Braking* class.

In this section, we show how to obtain insights on the reasoning process of Shared-PW-Net. In particular, in Figure 2, we report an illustration of the prototype assignment matrix learned by the model with 6 prototypes and 3 slots per action. We see that for the *Steering* action, the Shared-PW-Net successfully learns to differentiate between right ($p4$, $p5$) and left ($p6$) directions. Moreover, the model associates prototypes $p3$, $p1$, $p6$ to the *Accelerate* action, identifying with a straightforward road (or a road without pronounced curves) those situations in which acceleration is possible. The same logical reasoning can be observed for the third class, the *Braking* action, to which the model associates prototypes $p4$, $p2$, $p5$, revealing that the time to decelerate is when the car is approaching a curve. Such observations contribute to a more comprehensive understanding of the model's decision-making process. Examining the

prototypes and their assignment gives us valuable insights into how the model perceives different actions and scenarios within the car racing environment. This analysis of the Shared-PW-Net's interpretability sheds light on its strengths and limitations, providing crucial information to enhance the model's performance and adaptability in varying real-world scenarios.

6. Conclusions

This paper introduces a novel prototype-based neural network that builds upon the foundational work of PW-Net. Our proposed model is an interpretable wrapper for any pre-trained reinforcement learning agent, enabling a deeper understanding of its decision-making process. By leveraging prototypes assigned to actions, our network achieves comparable reward performances with respect to the state-of-the-art in two environments, without the need for human intervention. Indeed, the use of trainable prototypes eliminates the need for human-defined prototypes, enhancing the model's autonomy and adaptability. Although this study demonstrates the efficacy of behavior cloning on a pre-trained RL agent, showcasing promising results in various scenarios, it is essential to acknowledge the inherent limitation of relying on a pre-existing agent for this approach. While leveraging a pre-trained agent offers certain advantages, such as reduced training time and access to valuable learned knowledge, it can also introduce constraints that limit the full exploration of the model's capabilities. With respect to the introduction of the novel prototype initialization method, it has shown its effectiveness as the results indicate an improvement in the reward values when it is utilized, compared to the cases where a simple random prototypes initialization is employed.

As a natural progression of this research, a future direction could be to avoid using behavior cloning or distillation as the training method for our prototype-based neural network in order to allow the model to learn from data directly, potentially reducing the introduction of biased behaviors and increasingly improving performance from an interpretable perspective. Furthermore, the exploration of additional environments (which is ongoing) could provide an additional validation to this work.

Overall, our contribution expands the frontiers of interpretable reinforcement learning by presenting a powerful and versatile framework that enhances the interpretability of pre-trained RL agents. By empowering researchers and practitioners to gain deeper insights into agent behavior, our prototype-based neural network opens new avenues for improving the transparency and trustworthiness of RL applications. Furthermore, this framework could hold significant promise in real-world applications such as autonomous driving because it could help passengers trust the vehicle's decision-making processes by providing clear explanations for the car's driving choices and by enhancing safety and accountability in autonomous driving systems. We believe our work provides a strong foundation for interpretable DRL in real-world applications, where ethics, as seen in contexts like autonomous driving, plays a crucial role in responsible decision-making and trustworthiness.

References

[1] W. Saeed, C. Omlin, Explainable ai (xai): A systematic meta-survey of current challenges and future opportunities, Knowledge-Based Systems 263 (2023) 110273. URL: https://www.sciencedirect.com/science/article/pii/S0950705123000230. doi:https://doi.org/10.1016/j.knosys.2023.110273.

[2] C. Chen, O. Li, C. Tao, A. J. Barnett, J. Su, C. Rudin, This Looks like That: Deep Learning for Interpretable Image Recognition, Curran Associates Inc., Red Hook, NY, USA, 2019.

[3] D. Rymarczyk, Ł. Struski, M. Górszczak, K. Lewandowska, J. Tabor, B. Zieliński, Interpretable image classification with differentiable prototypes assignment, in: S. Avidan, G. Brostow, M. Cissé, G. M. Farinella, T. Hassner (Eds.), Computer Vision – ECCV 2022, Springer Nature Switzerland, Cham, 2022, pp. 351–368.

[4] M. Tucker, J. A. Shah, Prototype based classification from hierarchy to fairness, in: K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, S. Sabato (Eds.), Proceedings of the 39th International Conference on Machine Learning, volume 162 of *Proceedings of Machine Learning Research*, PMLR, 2022, pp. 21884–21900. URL: https://proceedings.mlr.press/v162/tucker22a.html.

[5] S. O. Davoudi, M. Komeili, Toward faithful case-based reasoning through learning prototypes in a nearest neighbor-friendly space., in: International Conference on Learning Representations, 2022. URL: https://openreview.net/forum?id=R79ZGjHhv6p.

[6] J. Donnelly, A. J. Barnett, C. Chen, Deformable protopnet: An interpretable image classifier using deformable prototypes, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2022, pp. 10265–10275.

[7] B. La Rosa, R. Capobianco, D. Nardi, A self-interpretable module for deep image classification on small data, Applied Intelligence (2022). doi:10.1007/s10489-022-03886-6.

[8] O. Li, H. Liu, C. Chen, C. Rudin, Deep learning for case-based reasoning through prototypes: A neural network that explains its predictions, in: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence, AAAI'18/IAAI'18/EAAI'18, AAAI Press, 2018.

[9] E. M. Kenny, M. Tucker, J. Shah, Towards interpretable deep reinforcement learning with human-friendly prototypes, in: The Eleventh International Conference on Learning Representations, 2023.

[10] G. Schwalbe, B. Finzel, A comprehensive taxonomy for explainable artificial intelligence: a systematic survey of surveys on methods and concepts, Data Mining and Knowledge Discovery (2023). URL: https://doi.org/10.1007/s10618-022-00867-8. doi:10.1007/s10618-022-00867-8.

[11] V. F. Zambaldi, D. Raposo, A. Santoro, V. Bapst, Y. Li, I. Babuschkin, K. Tuyls, D. P. Reichert, T. P. Lillicrap, E. Lockhart, M. Shanahan, V. Langston, R. Pascanu, M. M. Botvinick, O. Vinyals, P. W. Battaglia, Deep reinforcement learning with relational inductive biases, in: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019, OpenReview.net, 2019. URL: https://openreview.net/forum?id=HkxaFoC9KQ.

[12] A. Mott, D. Zoran, M. Chrzanowski, D. Wierstra, D. J. Rezende, Towards Interpretable

Reinforcement Learning Using Attention Augmented Agents, Curran Associates Inc., Red Hook, NY, USA, 2019.

[13] G. Liu, O. Schulte, W. Zhu, Q. Li, Toward Interpretable Deep Reinforcement Learning with Linear Model U-Trees: European Conference, ECML PKDD 2018, Dublin, Ireland, September 10–14, 2018, Proceedings, Part II, 2019, pp. 414–429. doi:10.1007/978-3-030-10928-8_25.

[14] M. H. Danesh, A. Koul, A. Fern, S. Khorram, Re-understanding finite-state representations of recurrent policy networks, in: M. Meila, T. Zhang (Eds.), Proceedings of the 38th International Conference on Machine Learning, volume 139 of *Proceedings of Machine Learning Research*, PMLR, 2021, pp. 2388–2397. URL: https://proceedings.mlr.press/v139/danesh21a.html.

[15] A. Koul, S. Greydanus, A. Fern, Learning finite state representations of recurrent policy networks, 2018. arXiv:1811.12530.

[16] R. S. Sutton, A. G. Barto, Reinforcement Learning: An Introduction, second ed., The MIT Press, 2018.

[17] E. Jang, S. Gu, B. Poole, Categorical reparameterization with gumbel-softmax, in: International Conference on Learning Representations, 2017. URL: https://openreview.net/forum?id=rkE3y85ee.

[18] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba, Openai gym, arXiv preprint arXiv:1606.01540 (2016).

[19] J. Jain, Jinayjain/deep-racing: Self-driving racecar using reinforcement learning (proximal policy optimization) in pytorch, https://github.com/JinayJain/deep-racing, 2022.

[20] B. Şentürk, https://github.com/bhctsntrk/openaipong-dqn, https://github.com/bhctsntrk/OpenAIPong-DQN, 2022.