TOWARDS INTERPRETABLE DEEP REINFORCEMENT LEARNING WITH HUMAN-FRIENDLY PROTOTYPES

Anonymous authors
Paper under double-blind review

ABSTRACT

Despite recent success of deep learning models in research settings, their application in sensitive domains remains limited because of their opaque decision-making processes. Taking to this challenge, people have proposed various eXplainable AI (XAI) techniques designed to calibrate trust and understandability of black-box models, with the vast majority of work focused on supervised learning. Here, we focus on making an "interpretable-by-design" *deep* reinforcement learning agent which is forced to use human-friendly prototypes in its decisions, thus making its reasoning process clear. Our proposed method, dubbed Prototype-Wrapper Network (PW-Net), wraps around any neural agent backbone, and results indicate that it does not worsen performance relative to black-box models. Most importantly, we found in a user study that PW-Nets supported better trust calibration and task performance relative to standard interpretability approaches and black-boxes.

1 INTRODUCTION

Deep reinforcement learning (RL) models have achieved state-of-the-art results in Go (Silver et al., 2016), Chess (Silver et al., 2017), Atari (Mnih et al., 2015), self-driving cars (Kiran et al., 2021), and robotic control (Kober et al., 2013). However, the usage of these agents in truly sensitive domains is limited due to the opaque nature of such systems. Extracting a deep model's rationale in a human interpretable format remains a challenging problem, but doing so would be highly useful to troubleshoot an agent's actions and by extension its possible failure states (Rudin et al., 2022). One popular approach to do so is with *post-hoc* explanation methods, which give after-the-fact rationales for model predictions mostly through some form of saliency map (Bach et al., 2015) or exemplar (Lipton, 2016). However, whilst popular, these approaches may be incomplete or erroneous (Slack et al., 2020), and recent work has instead started to focus on *pre-hoc* interpretability (Rudin, 2019). The core idea behind this latter paradigm is to design inherently explainable models, so that you can clearly see and understand their decision-making process in such a way that you can calibrate user trust and predict the system's capabilities.

In this paper, we present (to the best of our knowledge) the first inherently interpretable, well performing, deep reinforcement learning (RL) agent that uses an intuitive exemplar-based approach for decision making. Specifically, we train an interpretable "wrapper" model called Prototype-Wrapper Network (PW-Net) that can be added to any pre-trained agent, which allows them to be "interpretable-by-design", offering the same benefits as popular *pre-hoc* methods (Li et al., 2018), and the same intuitive reasoning process. Crucially however, when using PW-Nets, the main advantages of *post-hoc* approaches remain in-tact, in that the black-box model's performance is *not* lost, and it does not need to be retrained from scratch, which we demonstrate across multiple domains notoriously difficult for XAI.

2 RELATED WORK

This paper builds upon recent work building prototype-based neural networks for interpretable supervised learning. Such networks are interpretable by design because they utilize these prototypes in their forward pass by classifying test instances based upon their proximity to these prototypes, thus allowing users to intuitively understand predictions. Perhaps the first notable example of this was by Li et al. (2018) who learned prototypes in latent space which classified test instances using their L_2

distance to each prototype. Work in this area was followed up by Chen et al. (2019) who used image "parts" rather than the whole instance. This helped spawn many followup works in NLP (Ming et al., 2019), fairness (Tucker & Shah, 2022), and computer vision tasks (Donnelly et al., 2022; Davoudi & Komeili, 2021). These prior works establish the value of prototype-based neural nets, but they focus on traditional classification tasks; we are similarly inspired by such methods but seek to build interpretable agents in RL settings.

Whilst we are not the first to build interpretable RL models (Vouros, 2022; Milani et al., 2022), most prior research uses interpretable proxy models (such as decision trees) to imitate neural agents in symbolic domains. However, these techniques do not apply to richer domains with high-dimensional inputs (such as raw pixels) that we focus on in this work. To date, most work in these deep RL settings has focused on post-hoc approximations involving attention weights (Zambaldi et al., 2018), or trees Liu et al. (2018), but these methods do not allow transparency of the agent's actions or intent (Rudin et al., 2022). Perhaps the most relevant approach is that by Annasamy & Sycara (2019) who learn exemplars to explain Atari games, but their method is highly specific and only works in discrete action spaces, suffers a performance gap relative to black-box counterparts, and there is no guarantee the exemplars are interpretable (which we show is important in our user study). In contrast to all these approaches, PW-Nets are deliberately designed to be human-interpretable, generalize to any neural-based agent with any action space, do not lose model performance relative to black-box models, and do not rely on post-hoc approximations prone to error.

Human studies in RL have again largely focused on domains with symbolic input features utilizing decision tree-based policies (Paleja et al., 2021). However, these approaches may prove limited for human usage, because to achieve good model performance, a relatively low depth in the tree is often required (which hampers interpretability) alongside few input features (Bastani et al., 2018; Cunningham et al., 2003), when it is common for domains to have many more. User studies in *deep* RL trying to predict an agent's actions have shown mixed results (Anderson et al., 2020), with the focus now moving towards arguable more useful applications such as identifying defective models (Olson et al., 2021). In the prototype literature, human studies have generally focused on how "similar" test instances look to prototypes used in the model (Das et al., 2020; Rymarczyk et al., 2021), but this does not evaluate if the explanation itself is useful in downstream applications, or appropriately calibrates trust in users (Sanneman & Shah, 2022). In contrast to these studies, we avoid the need for trees and demonstrate that people can gain a solid global understanding of the model after viewing a handful of prototypes, and subsequently simulate model behaviour to predict failure cases, an application recently suggested (Rudin et al., 2022), thus helping to calibrate appropriate trust and assist in downstream tasks.

3 METHOD

Section 3.1 first details our assumptions of a Markov environment and trained neural agent, before Section 3.2 describes the proposed Prototype Wrapper Network (PW-Net) which creates a "wrapper" model around the agent to make it interpretable end-to-end. Finally, Section 3.3 gives certain performance guarantees for the system which shows it will always closely approximate the performance of a black-box agent.

3.1 MARKOV FRAMEWORK

Our technique assumes access to a neural agent pretrained in a Markov Decision Process (MDP). An MDP is defined by the (S, A, T, R, γ) tuple (Sutton & Barto, 2018). S is the set of states; $A \in R^M$ is the sets of $M-$dimensional actions; $T : S \times A \longrightarrow S$ is the probabilistic transition between states due to actions. We note that actions in RL, unlike outputs of simple classifiers, may be multi-dimensional (e.g., a car must control both steering and acceleration at the same time). We therefore dub each dimension in R^M a separate "action." Lastly, γ and R define the discount factor and reward function, respectively. In standard RL training methods, the goal is to find the policy, $\pi : s \in S \longrightarrow A$, that maximizes the expected discounted reward. Our approach only requires access to a pre-trained black box policy, π_{bb}; any existing method from prior art may be used to generate such a policy (Sutton et al., 2000; Williams, 1992). Assuming a neural net instantiation of this policy with a final linear layer, we may decompose π_{bb} into an encoder f_{enc}, alongside the last layer with weights $\boldsymbol{W}$ and bias $\boldsymbol{b}$ as follows: $\pi_{bb}(s) = \boldsymbol{W} f_{\text{enc}}(s) + \boldsymbol{b}$.

3.2 PROTOTYPE-WRAPPER NETWORK

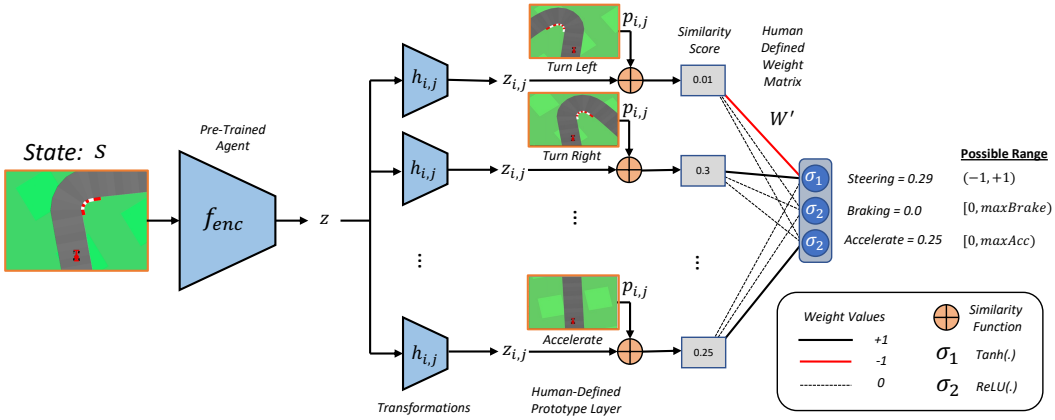

Figure 1: Prototype-Wrapper Network: The framework is instantiated here in the Car Racing domain from OpenAI's gym. A state is encoded as z using f_{enc}, before undergoing multiple separate transformations and having its similarity to pre-defined human-interpretable prototypes measured. Each output action can have one or more prototypes associated with it representing some understandable concept(s), all of which contribute to the action. For example, steering uses separate prototypes for turning left and right, whilst accelerate uses only one.

Our primary contribution is a prototype wrapper neural net model, PW-Net, that converts black-box neural models into prototype-based agents for RL by forcing them to use human-understandable concepts. The overall network diagram is depicted in Figure 1, instantiated for a car racing domain in which a simulated car must steer, brake, and accelerate.

In the wrapper model, a state, s, is mapped to a latent representation, z, via the original encoder network: $z = f_{\text{enc}}(s)$. The network defines, for each action $i \in [1, M]$, a set of prototypes that, when combined, dictate action $a[i]$. In Figure 1, for example, the steering action is associated with prototypes for a left or right turn. For each prototype, for each action, a separate "projection network," h_{ij} maps from z to a specific representation, z_{ij} for that action and prototype. Within that space for action i and prototype j, we use Chen et al. (2019)'s similarity function to score the distance between z_{ij} and prototype p_{ij}; lastly, those similarity scores are combined via a linear weight matrix, W' to produce actions. This process is defined more formally in Equation 1.

$$
\begin{aligned}
z &= f_{enc}(s) \\
z_{ij} &= h_{ij}(z) \qquad \forall i \in [1, M], j \in [1, N_i] \\
\texttt{sim}(z_{ij}, p_{ij}) &= \log\left(\frac{(z_{ij} - p_{ij})^2 + 1}{(z_{ij} - p_{ij})^2 + \epsilon}\right) \\
a'_i &= \sum_{j=1}^{N_i} W'_{ij}\,\texttt{sim}(z_{ij}, p_{ij}).
\end{aligned}
\tag{1}
$$

Prior literature in prototype-based neural networks indicates how one could train our wrapper model to learn its own prototypes (Chen et al., 2019; Ming et al., 2019; Tucker & Shah, 2022); however, one of the core novelties of our approach is that we leveraged human guidance for specifying (1) the prototypes, and (2) the final weight matrix, W' which determines how each prototype contributes to each action. In Section 6, we show how this process of defining human-friendly prototypes supported better trust calibration in user studies compared to "non-human friendly" prototypes (which may happen if algorithms learn such prototypes without human oversight, which is common practice). We created "human-friendly" prototypes by specifying a set of prototypical states (s_{pij}) that, when encoded, were used as the latent prototypes, $p_{ij} = h_{ij}(f_{\text{enc}}(s_{pij}))$, in the network. For example, in the self-driving car experiments here, one prototypical state was an example of a car turning

Algorithm 1: Training a Prototype-Wrapper Network

Input: f: A pre-trained agent encoder f_{enc} with policy π_{bb} to "wrap".
Output: A PW-Net.

Initialize wrapper, where W' is a manually specified weight-matrix.
Sample n state-action pairs from environment for Dataset $\mathcal{D} \leftarrow \{(s, \pi_{bb}(s)) \sim P(\mathcal{S})\}_{i=0}^{n}$.
Choose Human-Interpretable Prototypical States $\mathcal{S}_p \in \mathcal{S}$.
foreach batch (s, a) in $\mathcal{D}$ **do**
  $z \leftarrow f_{\text{enc}}(s)$
  foreach i in $\texttt{range}(|M|)$ **do**
    $a'[i] \leftarrow 0$
    foreach j in $\texttt{range}(|S_{pi}|)$ **do**
      $p_{ij} \leftarrow h_{ij}(f_{\text{enc}}(S_{pij})$
      $z_{ij} \leftarrow h_{ij}(z)$
      $a'[i] \leftarrow a'[i] + W'_{ij}\ \texttt{sim}\,(p_{ij}, z_{ij})$
  Minimize Loss $\mathcal{L}(a'|a; f_{enc}; \phi; W')$ with gradient descent, updating only ϕ.
Cache all $p_{ij} = h_{ij}(f_{\text{enc}}(s_{pij}))$ for testing time inference.
return Trained PW-Net

left and another state was an example of the car turning right; when encoded, these latent prototypes represent different ways of turning.

We also manually defined the weight matrix, W', to control the relationship between prototypes and actions. Intuitively, W'_{ij} corresponds to the extent in which similarity to prototype p_{ij} should increase action a_i. In our self-driving car example, a_1 was the turning rate: by setting $W'_{11} = -1; W'_{12} = +1$, we specified that the car's turning rate should equal the difference between the similarity to a prototypical left and right turn. In principle, one could combine many similarity scores for the same action, but from a performance perspective, we never found the need to use more than two (although this is likely situation- and task-dependent). Given manually-specified prototypical states and W', the only trainable parts of PW-Net were the projection networks, h_{ij}, with parameters ϕ. In experiments, we trained h_{ij} using gradient-based optimization to minimize a supervision loss between the PW-Net and the black-box policy, π_{bb}, in specific states. The training data for this process was generated by running the agent in its native domain for n iterations and collating the state action pairs into dataset $\mathcal{D} \leftarrow \{(s, \pi_{bb}(s)) \sim P(\mathcal{S})\}_{i=0}^{n}$.

3.3 Performance Guarantees

Here, we show that PW-Nets learn a comparable policy to the unwrapped black-box model, indicating that agent performance is unlikely to degrade when using our more interpretable architecture. Given that both the black-box model and PW-Net share a common encoder, f_{enc}, we therefore seek to prove that the output of a PW-Net from z, which we dub $f'_{pw}(z)$, equals the output of the black-box model from z, $f_{bb}(z)$, for all z. For simplicity, we consider only the first action, a_1 with:

$$
\begin{aligned}
a'_1 &= \sum_{j=1}^{N_i} W'_{1j}\texttt{sim}(z_{1j}, p_{1j}) \\
&= W'_{11} \log\left(\frac{(h_{11}(z) - p_{11})^2 + 1}{(h_{11}(z) - p_{11})^2 + \epsilon}\right) + W'_{12} \log\left(\frac{(h_{12}(z) - p_{12})^2 + 1}{(h_{12}(z) - p_{12})^2 + \epsilon}\right) \\
&= W'_{11} \log\left(\frac{d_{11}^2 + 1}{d_{11}^2 + \epsilon}\right) + W'_{12} \log\left(\frac{d_{12}^2 + 1}{d_{12}^2 + \epsilon}\right).
\end{aligned} \tag{2}
$$

Equation 2 simply restates how action a_1 is calculated based on prototypes. For simplicity, we only consider two prototypes per action (thus, two entries for W'_{1j}). The range of the similarity function is $(0, \log(1/\epsilon)]$; thus, the range of possible values of a'_1 is set by W'_{1j}. Assuming $W'_{11} > 0; W'_{12} < 0$, $a'_1 \in [W'_{12}\log(1/\epsilon), W'_{11}\log(1/\epsilon)]$. We note that the similarity function decreases monotonically

with distance, so it is invertible for legal values. Therefore, even if p_{ij} were fixed, a sufficiently-parametrized h_{ij} could learn to output any similarity score simply by placing z_{ij} at the correct distance from p_{ij}. Hence, as long as the black box policy outputs always fall within a finite range, a PW-Net policy can (almost always) match the output. In our experiments, we found one linear layer was not enough to accomplish this, two was sufficient (which we used, see Table 3), and more than two showed no benefit.

Two important assumptions in the above analysis are worth highlighting. First, we assumed that for each action there were at least two entries in W' and that they had opposite signs. The similarity function always generates positive outputs; thus, to match a black box policy with both positive and negative values, it is important to use both positive and negative weights in W'. In practice, we found this necessary in the self-driving car experiment. Second, we assumed independent h_{ij} for all i, j. Prior work in prototype-based neural nets often locate all prototypes within a single latent space. However, given a fixed W' and without separate h_{ij}, every state-action pair generated by π_{bb} creates a set of (potentially incompatible) constraints for the relative locations of prototype. In practice, we found that learning separate h_{ij} was necessary for good PW-Net performance, which we show in the next two sections.

The earlier discussion found that PW-Nets may mimic a black-box policy for almost all z by assuming that h_{ij} may be learned to output any desired similarity score between z_{ij} and p_{ij}. However, there are a small number of notable exceptions: precisely the z corresponding to prototype states. By definition, the distance between between such encodings and the prototype will be zero, which specifies the PW-Net output for such states. While this poses a theoretical limit, in practice we did not notice performance degradation when using PW-Net instead of black-box policies in experiments.

4 Continuous Action Spaces

Here we test the assertion that PW-Nets can mimic any black-box agent's function by testing them in a deep RL domain with multiple continuous output actions and a state space of only raw pixels. As an aside, another domain with symbolic inputs is tested to further display generality. PW-Nets are compared against several baselines and the best methods are determined using two metrics, the average reward received compared to the black-box, and how well the interpretable model mimics the black-box predictions at test time.

4.1 Experiment Setup

The deep RL domain tested here is the Car Racing problem from OpenAI's gym environment, the agent receives rewards for driving around the track as fast as possible and avoiding swerving off the road. We use a pre-trained model (partly to avoid any bias) from Jain (2022) which solved the environment using Proximal Policy Optimization (PPO). The agent has three continuous output actions (see Fig. 1). Similarity, for a symbolic problem, the Bipedal Walker environment from Open AI's gym Box2d was chosen, where the agent receives rewards for simply walking straight and not falling over. For this latter domain a pre-trained model from Barhate (2022b) was used which solved the problem with the Twin Delayed DDPG (TD3) algorithm. The agent has four continuous output actions (see Appendix A for prototypes used).

We compared first to k-Means in the space z with the same number of clusters as the number of prototypes used in PW-Net, before mapping each centroid onto the nearest training example to get the prototypes. Then, in contrast to the other methods, we allowed k-Means to learn the weight parameters in the last layer to help it. As another baseline, we followed the same protocol as PW-Net, but only a single transformation is used which houses all prototypes together in the same latent space which we call PW-Net*. In addition, this baseline learns the prototypes as trainable parameters, and once training is complete, they are mapped onto the nearest training example. This can be best thought of as an ablation test of PW-Net, however it bears resemblance to Chen et al. (2019) and their ProtoPNet model, so we also implemented the clustering and separation losses suggested by the authors (adapted for continuous domains) to help the baseline further (see Appendix B). Lastly, we also compare to VIPER by Bastani et al. (2018), but we adapt their algorithm here by using uniform sampling (which is implemented in their code base as an alternative) instead of the weighted sampling they discuss (which needs q-values), which allows us to use it in experiments.

Table 1: Continuous Action Spaces: Various baselines are compared to PW-Net. Overall, only PW-Net is comparable to the performance of the original black-box agent in both domains.

	Car Racing		Bipedal Walker	
Method	Reward	MSE	Reward	MSE
PW-Net	**221.96 ± 00.47**	**2.98 ± 0.33**	**312.16 ± 0.09**	**1.28 ± 0.11**
VIPER	N/A	N/A	-89.71 ± 7.51	101.15 ± 8.16
PW-Net*	-9.48 ± -02.50	16.93 ± 3.01	190.41 ± 59.51	69.77 ± 15.16
k-Means	-2.09 ± -00.94	29.04 ± 0.01	-107.72 ± 0.13	136.90 ± 0.72
Black-Box	218.41 ± 03.00	N/A	310.71 ± 1.47	N/A

Two metrics were used, average reward and mean-squared error (MSE). Reward was calculated by training each method 5 times, and taking the average reward accumulated across 30 simulations. The second metric took the MSE of the black-box model's actions compared to the interpretable model's actions each iteration, to see which method most closely approximates the oracle.

4.2 Results

Table 1 shows the results in continuous action domains. In the deep RL domain Car Racing, only PW-Net (M=221.96, SE=0.47) was capable of achieving the same average reward as the black-box (M=218.41, SE=3.). Moreover, PW-Net achieved the closest predictions to the black-box (i.e., the MSE was smallest). In Bipedal Walker, again PW-Net (M=312.16, SE=0.09) was capable of achieving the same average reward as the black-box (M=310.71, SE=1.47), although PW-Net* did achieve notable reward too (M=190.41, SE=59.51), although not comparable to the black-box. k-Means performed the worst overall, with the lowest reward and highest MSE, with VIPER fairing only slightly better.

5 Discrete Action Spaces

Here again PW-Nets are tested, but in discrete domains. The purpose of this section is to help prove the generality of the method across more domains, models, and output action spaces. As before, a deep RL domain using raw-pixels as state representations is tested, alongside a symbolic domain as an aside. The same baselines and metrics are used, although with some minor modifications.

5.1 Experiment Setup

The deep RL domain tested is Atari Pong from OpenAI's gym environment, the agent receives rewards for scoring against the opponent, and losses rewards for being scored against. The agent has six discrete actions corresponding to moving up or down, and we use a pre-trained model from Şentürk (2022) which solved the environment using the DQN algorithm. Similarity, for the symbolic problem, the Lunar Lander environment was chosen, where the agent gets reward by landing the spaceship successfully and safely in the shortest time possible. The agent has four actions corresponding to doing nothing, or firing its engine on different sides. A pre-trained model from Barhate (2022a) was used which solved the problem with the TD3 algorithm (see Appendix for prototypes).

As baselines, we again compared to k-Means, but this time by dividing the training data in the space z using each separate action, and clustering each with $k = 1$, before mapping each centroid onto the nearest training example. Then, in contrast to the last experiment, we didn't allow k-Means to learn the weight parameters in the last layer since each can easily be associated with an appropriate action. PW-Net* was again used here, but this time using the exact same clustering and separation losses suggested by Chen et al. (2019), since they developed the losses for discrete classification problems they will work here. Lastly, we again compare to VIPER by Bastani et al. (2018) using the uniform sampling variation of their algorithm.

Two metrics were used, reward and accuracy. Reward was calculated by training each method 5 times, and taking the average reward accumulated across 30 simulations. The second metric took

Table 2: Discrete Action Spaces: Various baselines are compared to PW-Net. Overall again, only PW-Net consistently mimics the black-box function well.

	Atari Pong		Lunar Lander	
Method	Reward	Accuracy	Reward	Accuracy
PW-Net	**13.04** $\pm$ **0.29**	**86.12** $\pm$ **0.46**	222.02 $\pm$ 17.98	**94.43** $\pm$ **0.50**
VIPER	N/A	N/A	-408.81 $\pm$ 60.98	59.26 $\pm$ 1.01
PW-Net*	8.85 $\pm$ 1.69	84.84 $\pm$ 0.76	**229.97** $\pm$ **9.05**	88.89 $\pm$ 1.38
k-means	-21.00 $\pm$ 0.00	11.79 $\pm$ 4.15	-419.46 $\pm$ 119.08	10.10 $\pm$ 5.87
Black-Box	11.94 $\pm$ 0.16	N/A	212.94 $\pm$ 2.63	N/A

the accuracy of the black-box model's actions compared to the interpretable model's actions each iteration, to see which method most closely approximates the oracle at test time.

5.2 Results

Table 2 shows the results of using PW-Nets in discrete action spaces. In Pong, PW-Net achieved the highest reward (M=13.04, SE=0.29) relative to PW-Net* (M=8.85, SE=1.69), and k-Means (M=-21., SE=0.), which was also closest to the black-box's reward (M=11.94, SE=0.16). Interesting divergences happen where PW-Net* achieves lower Accuracy (M=84.84, SE=0.76) compared to PW-Net (M=86.12, SE=0.46), indicating that PW-Net was better at mimicking the black-box. In Lunar Lander, again only PW-Net variations were able to reach a comparable reward to the black box. Specifically, PW-Net achieved a reward (M=222.02, SE=17.98) which was comparable to the black-box (M=212.94, SE=2.63), as did PW-Net* (M=229.97, SE=9.05). Larger divergence happens when comparing accuracy, as PW-Net (M=94.43, SE=0.5) achieves significantly higher results than PW-Net* (M=88.89, SE=1.38), indicating that the former is again better able to mimic the original black-box function. Although marginally better than k-Means, VIPER performs badly overall (M=-408.81, SE=60.98), perhaps suggesting that decision trees will struggle in their current form to generalize beyond relatively simple domains such as Cartpole where they are usually tested successfully (Bastani et al., 2018; Silva et al., 2020). Lastly, note the pre-trained Pong model was 5% stochastic, so reaching a high level of Accuracy was impossible.

So, considering the two experiments thus far, it is clear that PW-Net generalizes the best overall both in terms of reward and MSE/Accuracy, particularly in deep RL settings which is our focus here. A natural question to ask is how sensitive the performance of PW-Net is to the prototypes chosen, so we performed a test on all four domains where the prototypes were randomly chosen from the training data 30 times with 30 simulations ran each time. Results showed no notable difference to the black-box in Car Racing (Reward: M=218.57, SE=0.09, MSE), Atari Pong (Reward; M=12.15, SE=0.15), Lunar Lander (Reward; M=213.78, SE=11.19), or Bipedal Walker (Reward: M=309.44, SE=1.79). So, overall, PW-Nets are very robust to the prototypes chosen. However, as we shall now see, user task performance crucially depends upon choosing human-friendly prototypes.

6 User Study

Although the notion of using prototypes has become increasingly popular for XAI (Chen et al., 2019; Davoudi & Komeili, 2021), most papers do not evaluate their techniques in user trials, despite such explanations ultimately being designed for humans. In contrast, here we test if PW-Nets can improve a user's ability to predict in distribution (ID) successes and out-of-distribution (OOD) failures in a simple self-driving car domain, an application seen as highly sought after (Rudin et al., 2022).

In doing so, three different conditions are compared to our "Human-Friendly Prototype" PW-Net. The first is a "Non-Human Friendly" group which uses the same prototypes as our network, but rearranges them randomly. So, the weight matrix and actions remain the same, but different prototype images are associated with each. The purpose of this comparison is to show that although the performance of the network itself does not hinge on associating human-friendly prototypes with intuitive actions, the performance of humans does. Note this is an actual PW-Net we trained that performed

just as well as the other models here (even though the prototypes were purposefully "badly chosen"). The second condition is the same as our network, but we tell users that the prototypes are random training examples of the car learning to drive, and that there are many more they are not seeing. The purpose of this group is to show that people are less able to confidently predict model performance by simply looking at a snapshot of representative simulation data. Lastly, we compare our network to a full black-box, where people are simply told *"The car has learned to drive very well by practicing for millions of simulations"*. After seeing the current situation of the car, participants are shown the explanation, and then asked to predict if the car will drive safely or not on a scale of 1-5, where 1 corresponds to "Strongly Disagree" and 5 to "Strongly Agree".

So, the three hypotheses tested are: **H1:** Users require human-interpretable prototypes to perform well in model simulation tasks. **H2:** When people receive no explanation they will struggle to predict OOD failures by overestimating the system's capabilities. **H3:** People are less able to predict OOD failures by simply observing segments of training data.

Participants. Comparing to similar studies (Kenny et al., 2021), we anticipated a medium effect size, but pilot studies showed a large effect. Hence, anticipating a medium-large effect, we chose an alpha of 0.05, a power of 0.8, and an effect size of 0.65, which predicted through a power analysis that a sample size of 30 was required in each group to avoid p-hacking. Hence, 120 people in total were recruited from prolific.co, 30 per group, all paid $12 per hour to complete the study. All participants were restricted to being native English speakers, using desktop computers, in either the U.S. or the U.K. This study passed ethics review of the institution.

Materials. 20 materials were shown to users, 10 in distribution (ID), and 10 out-of-distribution (OOD), the latter generated from the open source modification to the Car Racing environment provided by Woodcock (2016), 5 of which showed red obstacles the car should avoid, and the other 5 of which showed unusual roads not seen normally in the environment (such as t-junctions). All ID situations were gathered form the original domain, driven successfully by the car, and labelled as "driving safe". The OOD situations were real failure cases of the car, where all trained models either (1) drove off the road crashing, or (2) hit the obstacles. If the car did either of these things, it was labelled as "not driving safely". After the questions, users where shown where the car did/didn't drive safe, and asked to complete the NASA TLX perceived workload questionnaire.[1]

6.1 RESULTS.

Each participant's scores were averaged both for ID and OOD materials, which formed the distributions of interest. **H1:** Fig. 3(A) shows a large difference in predicting ID success. Specifically, those who received "Human-Friendly Prototypes" (M=4.48, SD=0.43) compared to participants in the "Non-Human Friendly Prototypes" control group (M=2.19, SD=0.78) demonstrated better ability to predict when the system would drive safe, $t(58)$=13.8, $p < .0001$, showing human-friendly prototypes had a large effect on people's task performance. **H2:** Fig. 3(B) shows a significant difference in predicting OOD failures between participants who received "Human-Friendly Prototypes" (M=2.24, SD=0.68) and participants in the "Black-Box" control group (M=3.48, SD=0.75). The prior demonstrated significantly better ability to predict when the system would *not* drive safe, $t(58)$ = -6.53, $p < .0001$, showing that prototypes give a global understanding of the system which allows people to predict OOD-based system failures better than no explanation. **H3:** Fig. 3(C) shows another significant difference in predicting OOD failures. In particular, when comparing participants who received "Human-Friendly Prototypes" (M=2.24, SD=0.68) to participants in the "Training-Data" control group (M=2.79, SD=0.57), the prior demonstrated better ability to predict when the system would *not* drive safe, $t(58)$ = -3.32, $p < .002$. This shows that prototypes give a global understanding of the system's capabilities which allow people to confidently predict OOD-based system failures better than observing segments of training data.

Aside from testing our original hypotheses, additional post-hoc analysis in Fig. 3(D) shows the groups from a pure accuracy perspective, where scores of 4/5 were taken as the user predicting the car would drive safely, and scores of 1/2 as the user predicting the car would not drive safely,

[1]Users were also asked for their confidence in predictions, and an additional system usability questionnaire was asked. However, no results were significant, so they are not reported here.

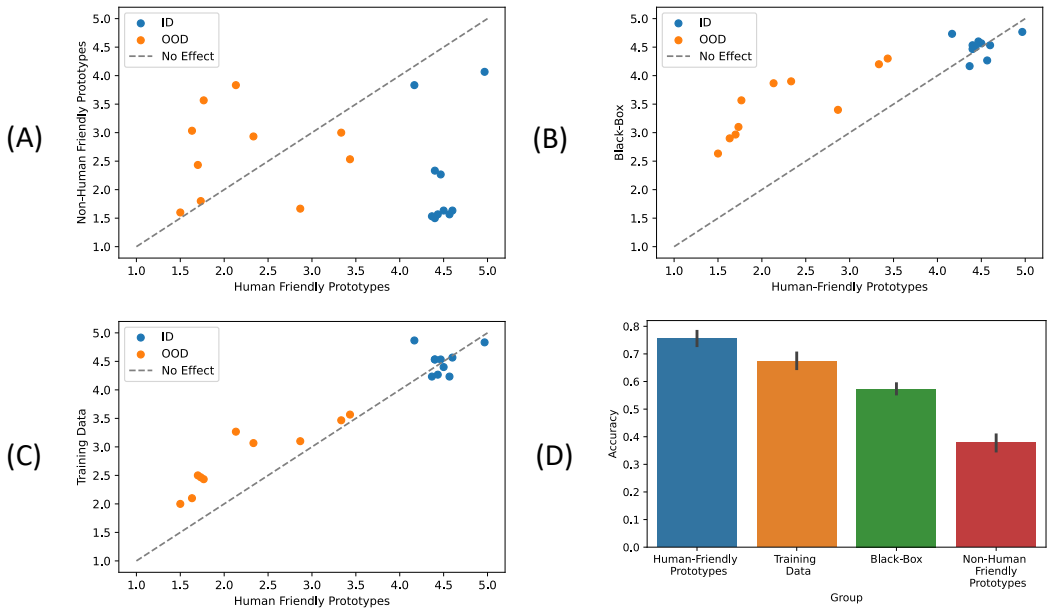

Figure 2: User Study Results: Plots A-C compare different group ratings for how confident they were the car would drive safely (5 rating) v. not drive safely (1 rating) for in distribution (ID) vs. out-of-distribution (OOD) data. Each point on the plots represents the average of a single question in the survey. The plots can be interpreted as datapoints close to the dashed line showing no difference between groups, whilst points far from it showing significant differences. Plot D shows the accuracy of all four groups in predicting if the car would drive safely or not (i.e., model simulation).

a score of 3 was always marked as incorrect, since the user is uncertain either way. The human-friendly group was better at predicting if the car would drive safe or not (M=0.75, SD=0.14), than the non-human friendly group (M=0.37, SD=0.15), training data group (M=0.67, SD=0.15), and the black-box group (M=0.57, SD=0.10), with all results achieving statistical significance $p < .05$. This ability to accurately predict a system is an important proxy metric for trust calibration (Hoffman et al., 2018; Sanneman & Shah, 2022), and commonly known as *model simulation* (Doshi-Velez & Kim, 2017). Hence, it is reasonable to posit that PW-Net may have a role to play in calibrating appropriate user trust of AI systems, something which is seen as a core application for XAI, but has proven *extremely* difficult to demonstrate in practice (Miller, 2022; Kenny et al., 2021).

Lastly, the NASA TLX questionnaire showed there was a significant difference in people's perceived workload between participants who received "Human-Friendly Prototypes" (M = 5.80, SD = 1.74) and participants in the "Non-Human Friendly" control group (M = 7.25, SD = 2.63). The control recorded a higher workload, $t(58)$ = -2.45, $p < .02$, showing that badly chosen prototypes make the task of the using the system significantly more taxing on people.

7 Conclusion and Discussion

The world at large has grown increasingly excited about possible applications for deep RL in domains such as self-driving cars and human-robot interaction. However, without complete transparency in the agent's actions and intent, it is not feasible to use the systems in truly sensitive domains (Rudin, 2019). Here, we have showcased an accessible way to *force* any deep RL agent to use human understandable concepts in an interpretable way, hopefully taking a step towards having a common language between AI and humans as has been suggested lately (Kim, 2022). Results demonstrated that the technique does *not* loose model performance and generalizes across discrete and continuous action spaces. In addition, a user study showed real applicability of the technique to assist human task performance in predicting model failure (and success) states, alongside calibrating *appropriate* user trust.

REPRODUCIBILITY STATEMENT

All original code used is uploaded with this submission via a zip file, making reputability trivial. For the user study, we also included all four groups surveys as PDFs in the same zip file, so people may recreate the study results if they wish (anonymized during peer review).

ETHICS STATEMENT

One of the potential ethical considerations of using prototypes for explanations is that the examples used must perserve user privacy. Considering this, if PW-Nets were used in downstream applications, some consideration is going to need to be given to this. However, when compared to more generic nearest neighbor algorithms, PW-Nets have a big advantage here in that there is a much smaller group of exemplars which may be possibly presented to users, so the efforts on algorithm designers to preserve privacy will be much easier.

REFERENCES

Andrew Anderson, Jonathan Dodge, Amrita Sadarangani, Zoe Juozapaitis, Evan Newman, Jed Irvine, Souti Chattopadhyay, Matthew Olson, Alan Fern, and Margaret Burnett. Mental models of mere mortals with explanations of reinforcement learning. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 10(2):1–37, 2020.

Raghuram Mandyam Annasamy and Katia Sycara. Towards better interpretability in deep q-networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pp. 4561–4569, 2019.

Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015.

Nikhil Barhate. Nikhilbarhate99/actor-critic-pytorch: Policy gradient actor-critic pytorch: Lunar lander v2, 2022a. URL https://github.com/nikhilbarhate99/Actor-Critic-PyTorch.

Nikhil Barhate. https://github.com/nikhilbarhate99/td3-pytorch-bipedalwalker-v2, 2022b. URL https://github.com/nikhilbarhate99/Actor-Critic-PyTorch.

Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. *Advances in neural information processing systems*, 31, 2018.

Chaofan Chen, Oscar Li, Daniel Tao, Alina Barnett, Cynthia Rudin, and Jonathan K Su. This looks like that: Deep learning for interpretable image recognition. *Advances in Neural Information Processing Systems*, 32:8930–8941, 2019.

Pádraig Cunningham, Dónal Doyle, and John Loughrey. An Evaluation of the Usefulness of Case-Based Explanation. In *Case-Based Reasoning Research and Development*, volume 2689, pp. 122–130. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003. ISBN 978-3-540-40433-0. doi: 10.1007/3-540-45006-8_12. URL http://link.springer.com/10.1007/3-540-45006-8_12.

Subhajit Das, Panpan Xu, Zeng Dai, Alex Endert, and Liu Ren. Interpreting deep neural networks through prototype factorization. In *2020 International Conference on Data Mining Workshops (ICDMW)*, pp. 448–457. IEEE, 2020.

Seyed Omid Davoudi and Majid Komeili. Toward faithful case-based reasoning through learning prototypes in a nearest neighbor-friendly space. In *International Conference on Learning Representations*, 2021.

Jon Donnelly, Alina Jade Barnett, and Chaofan Chen. Deformable protopnet: An interpretable image classifier using deformable prototypes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10265–10275, 2022.

Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.

Robert R Hoffman, Shane T Mueller, Gary Klein, and Jordan Litman. Metrics for explainable ai: Challenges and prospects. *arXiv preprint arXiv:1812.04608*, 2018.

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

Eoin M Kenny, Courtney Ford, Molly Quinn, and Mark T Keane. Explaining black-box classifiers using post-hoc explanations-by-example: The effect of explanations and error-rates in xai user studies. *Artificial Intelligence*, pp. 103459, 2021.

Been Kim. Beyond interpretability: Developing a language to shape our relationships with ai, May 2022. URL https://medium.com/@beenkim/beyond-interpretability-4bf03bbd9394.

B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 2021.

Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

Oscar Li, Hao Liu, Chaofan Chen, and Cynthia Rudin. Deep learning for case-based reasoning through prototypes: A neural network that explains its predictions. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Zachary C Lipton. The mythos of model interpretability. *arXiv preprint arXiv:1606.03490*, 2016.

Guiliang Liu, Oliver Schulte, Wang Zhu, and Qingcan Li. Toward interpretable deep reinforcement learning with linear model u-trees. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 414–429. Springer, 2018.

Stephanie Milani, Nicholay Topin, Manuela Veloso, and Fei Fang. A survey of explainable reinforcement learning. *arXiv preprint arXiv:2202.08434*, 2022.

Tim Miller. Are we measuring trust correctly in explainability, interpretability, and transparency research? *arXiv preprint arXiv:2209.00651*, 2022.

Yao Ming, Panpan Xu, Huamin Qu, and Liu Ren. Interpretable and steerable sequence learning via prototypes. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 903–913, 2019.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Matthew L Olson, Roli Khanna, Lawrence Neal, Fuxin Li, and Weng-Keen Wong. Counterfactual state explanations for reinforcement learning agents via generative deep learning. *Artificial Intelligence*, 295:103455, 2021.

Rohan Paleja, Muyleng Ghuy, Nadun Ranawaka Arachchige, Reed Jensen, and Matthew Gombolay. The utility of explainable ai in ad hoc human-machine teaming. *Advances in Neural Information Processing Systems*, 34:610–623, 2021.

Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, 2019.

Cynthia Rudin, Chaofan Chen, Zhi Chen, Haiyang Huang, Lesia Semenova, and Chudi Zhong. Interpretable machine learning: Fundamental principles and 10 grand challenges. *Statistics Surveys*, 16:1–85, 2022.

Dawid Rymarczyk, Łukasz Struski, Jacek Tabor, and Bartosz Zieliński. Protopshare: Prototypical parts sharing for similarity discovery in interpretable image classification. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 1420–1430, 2021.

Lindsay Sanneman and Julie A Shah. The situation awareness framework for explainable ai (safe-ai) and human factors considerations for xai systems. *International Journal of Human–Computer Interaction*, pp. 1–17, 2022.

Andrew Silva, Matthew Gombolay, Taylor Killian, Ivan Jimenez, and Sung-Hyun Son. Optimization methods for interpretable differentiable decision trees applied to reinforcement learning. In *International conference on artificial intelligence and statistics*, pp. 1855–1865. PMLR, 2020.

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.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

Dylan Slack, Sophie Hilgard, Emily Jia, Sameer Singh, and Himabindu Lakkaraju. Fooling lime and shap: Adversarial attacks on post hoc explanation methods. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, pp. 180–186, 2020.

R. S. Sutton, D. Mcallester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12*, volume 12, pp. 1057–1063. MIT Press, 2000.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL `http://incompleteideas.net/book/the-book-2nd.html`.

Mycal Tucker and Julie A Shah. Prototype based classification from hierarchy to fairness. In *International Conference on Machine Learning*, pp. 21884–21900. PMLR, 2022.

George A Vouros. Explainable deep reinforcement learning: State of the art and challenges. *ACM Computing Surveys (CSUR)*, 2022.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

Mike Woodcock. Notanymike/gym: An improvement of carracing-v0 from openai gym in order to make the environment complex enough for hierarchical reinforcement learning, Jan 2016. URL `https://github.com/NotAnyMike/gym`.

Vinicius Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David Reichert, Timothy Lillicrap, Edward Lockhart, et al. Deep reinforcement learning with relational inductive biases. In *International conference on learning representations*, 2018.

Behçet Şentürk. https://github.com/bhctsntrk/openaipong-dqn, 2022. URL `https://github.com/nikhilbarhate99/Actor-Critic-PyTorch`.

A Prototypes Chosen

In this appendix we explain how the prototypes were chosen for each domain.

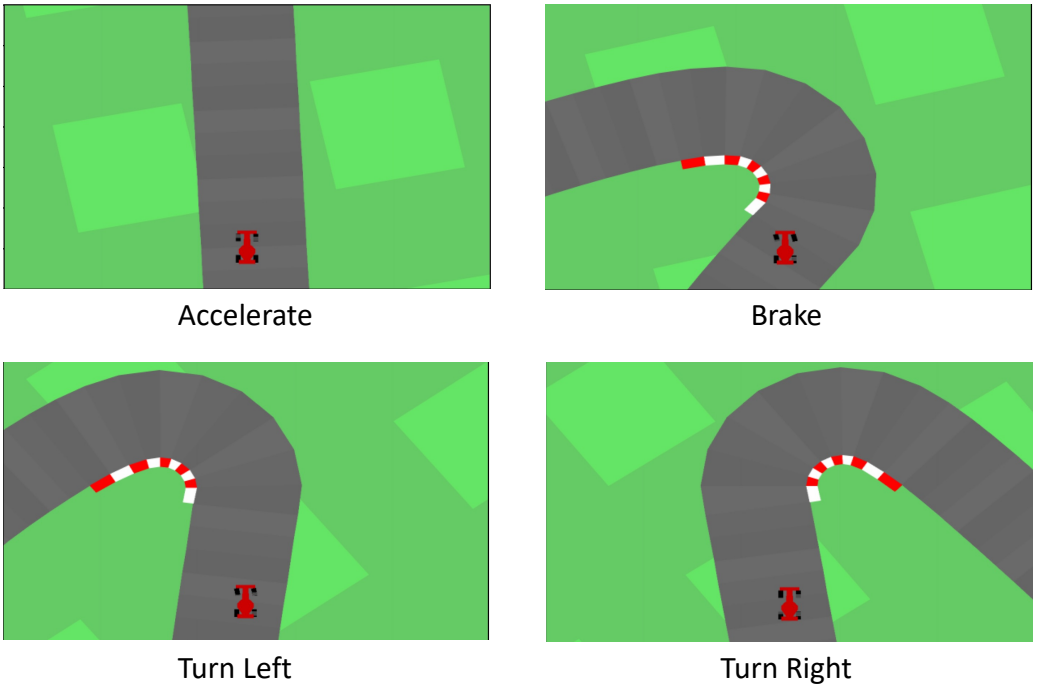

Figure 3: Human-Friendly Prototypes for Car Racing: The four prototypes chosen for Car Racing in our tests. The two prototype for "Turn Left" and "Turn Right" contribute to the output action steering. The two other prototypes singularly contribute to the actions accelerate and brake, respectively.

A.1 Car Racing

In this domain there are three action outputs, steering, braking, and accelerating. Steering has a possible range from -1 to +1, with negative values turning left, and positive values turning right, 0 means to drive straight on. Accelerating and Braking both have values from 0 to ∞.

When choosing the prototypes for PW-Nets, we broke steering down into two concepts of "turning left" and "turning right", with a prototype representing each concept. For accelerating and braking, we only chose one prototype each. For accelerating a state of a straight road was chosen, since this is typically seen as a safe situation to accelerate, and for braking the motion of taking a turn was chosen, since it is necessary for a car to always brake when turning a corner in order to drive safely, these where seen as being somewhat interpretable for people (see Fig. 3).

Note that many more concepts could be used to dictate when the car brakes or accelerates. For example, if there are obstacles on the road, then one could correspond a prototypical example of these objects to the action "braking". That way, when the car sees such an obstacle, it will brake and explain why is is braking (i.e., due to its similarity to the prototype with the obstacle). However, here for simplicity (and to demonstrate multiple/single options), we only associate one concept with the actions braking and accelerating, whilst associating two with steering.

A.2 Atari Pong

In this domain there are six action outputs, action 0 and 1 are placeholders and do nothing, action 2 & 4 makes the racket go up, and action 3 & 5 makes the racket go down. The prototypes used in our tests are shown in Fig. 4, with near identical ones used for the duplicate actions.

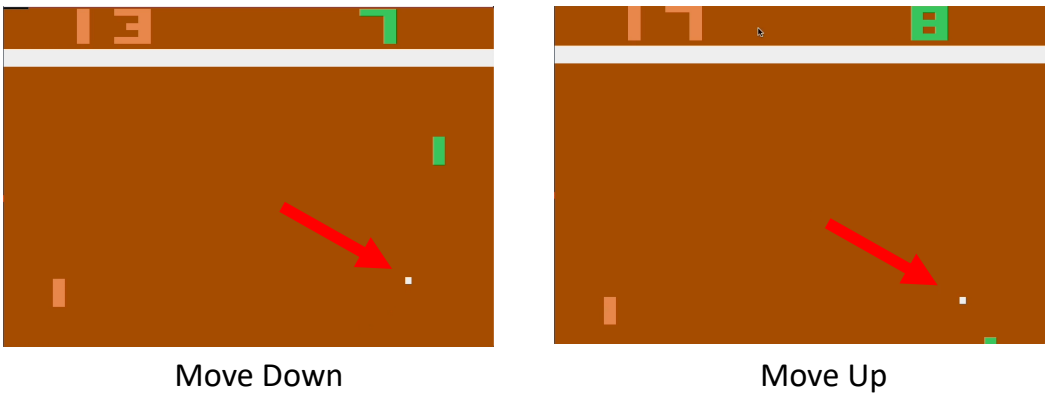

Figure 4: Human-Friendly Prototypes for Pong: The player is the green object which must hit the white ball back to the left side of the table. The red arrow shows the direction the white ball is travelling.

A.3 Symbolic Domains

Although the focus of this paper is on deep RL, these domains were included to help prove the generality of PW-Nets as proposed in Section 3.3. For Lunar Lander and Bipedal Walker we defined "ideal" prototypical examples of each concept in the action-space where the action of interest was 1 or -1 and all others 0, which we then mapped to the nearest training example. Lunar Lander had 1 for the action of interest, and 0 for the others (four prototypes in total). Bipedal Walker had two prototypes for each output action since they ranged from -1 to +1 (so 8 prototypes in total), each representing the concept of one of its limbs moving forward or backwards (e.g., "Right Hip Moving Forwards").

B Losses in PW-Net*

When learning all the prototypes in the same latent space z, we followed prior literature in Chen et al. (2019) and used their separation and clustering losses as described below:

$$Clst = \frac{1}{n}\sum_{i=1}^{n} \min_{j:p_j \in P_{y_i}} \min_{z \in patches(f(x_i))} \|z - p_j\|_2^2 \tag{3}$$

$$Sep = -\frac{1}{n}\sum_{i=1}^{n} \min_{j:p_j \notin P_{y_i}} \min_{z \in patches(f(x_i))} \|z - p_j\|_2^2 \tag{4}$$

where n is the number of instances in a batch, j is a prototype for one of the classes in P, $f(.)$ is the encoder, and z is the latent representation of some patch in test instance x_i.

However, in our domains we are considering whole instances, and not parts of them, so the losses become:

$$Clst = \frac{1}{n}\sum_{i=1}^{n} \min_{j:p_j \in P_{y_i}} \min_{z=(f(x_i))} \|z - p_j\|_2^2 \tag{5}$$

$$Sep = -\frac{1}{n}\sum_{i=1}^{n} \min_{j:p_j \notin P_{y_i}} \min_{z=(f(x_i))} \|z - p_j\|_2^2 \tag{6}$$

In the continuous actions spaces, the losses had to be modified to work, so they became:

$$Clst = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{P}\min_{z=(f(x_i))}\|z_i - p_j\|_2^2 \tag{7}$$

$$Sep = \frac{-\sum_{i=1}^{P}\sum_{j=1}^{P}\|p_i - p_j\|_2^2}{\frac{P^2-P}{2}} \tag{8}$$

where P simply represents the number of prototypes used. Intuitively speaking, the first loss forces all prototypes to be close to the data distribution, so that when they are mapped to real instances they do not change much. The second loss forces all prototypes to be far apart from each other. Each loss had to be scaled in optimization with parameters λ_1 and λ_2, respectively. In discrete domains, we set the values to 0.8, and 0.08 [following Chen et al. (2019)], but in continuous domains we found 0.08, and 0.008 worked better. The denominator in Equation (8) simply accounts for i and j occasionally being equal to each other, the loss could also be rewritten as:

$$Sep = -\sum_{i=1}^{P}\sum_{j=1}^{P}\|p_i - p_j\|_2^2 \quad \text{s.t.} \quad i \neq j \tag{9}$$

C User Study designs

Rather than include the user designs in this manuscript, we uploaded them in the accompanying zip file in the interests of keeping this document to a reasonable size (both in terms of file size and document length).

D Transformation Architecture

Each PW-Net model here used multiple transformations $h_{i,j}$ to convert z into a latent space in which it could have its similarity measured against human-defined prototypes. The architecture for each of these is specified in Table 3. In practice, the prototype sizes were usually 50 in length.

Table 3: The transformation architecture used in all experiemnts.

Transformations $h_{i,j}$	
Layer	Layer Parameters
Linear	(Latent z size, prototype size)
BatchNorm1d	prototype size
ReLU	
Linear	(prototype size, prototype size)