Artificial Intelligence Research and Development
M. Villaret et al. (Eds.)

doi:10.3233/FAIA210166

Applying and Verifying an Explainability Method Based on Policy Graphs in the Context of Reinforcement Learning

Antoni CLIMENT [a], Dmitry GNATYSHAK [b] and Sergio ALVAREZ-NAPAGAO [b]
[a] *Universitat Politècnica de Catalunya, Barcelona, Spain*
[b] *Barcelona Supercomputing Center, Barcelona, Spain*

Abstract. The advancement on explainability techniques is quite relevant in the field of Reinforcement Learning (RL) and its applications can be beneficial for the development of intelligent agents that are understandable by humans and are able cooperate with them. When dealing with Deep RL some approaches already exist in the literature, but a common problem is that it can be tricky to define whether the explanations generated for an agent really reflect the behaviour of the trained agent. In this work we will apply an approach for explainability based on the creation of a Policy Graph (PG) that represents the agent's behaviour. Our main contribution is a way to measure the similarity between the explanations and the agent's behaviour, by building another agent that follows a policy based on the explainability method and comparing the behaviour of both agents.

Keywords. Explainable AI, Reinforcement Learning, Policy Graphs

1. Introduction and Motivation

Humans and complex algorithms for controlling agents do not usually share a common language. In the context of many artificial intelligence methods, including those based on Neural Networks (NN), when evaluating a specific problem we can learn the accuracy, the sensitivity, the reward or the loss with respect to an objective function, along with other metrics. These metrics can give us an idea about whether an algorithm controlling an agent, such as a robot, is learning to perform a certain task correctly or not. However, these metrics are frequently not enough to understand the agent behaviour or the rationale behind their decisions. This is known as the *explainability problem*.

Much of the current state of the art in Reinforcement Learning (RL) is usually enabled by neural network-based methods for finding a (near) optimal policy. This type of programs do not follow a code made by a human that logically solves the problem in a procedural or rule-based manner, but instead they are based on giving training input to generate a non-intelligible complex function and taking the result from it to build a behavioural policy. Furthermore, it is usually difficult to know whether the neural network is learning what it is supposed to, which causes reliability issues.

This type of problem –how can we *explain* the behaviour of an agent and its rationale– is an important sub-field of Explainability of Artificial Intelligence (XAI) [11]. Relevant topics of research in this topic include finding new ways of solving a task, elaborating insights about the agents' strategies, and analyse how an agent takes decisions in specific scenarios where they perform a task better than humans.

The main objective of our work has been to create a graph-based policy in order to analise whether the outcomes of an explainability method are able to accurately describe the behavior of a trained agent. There are currently several approaches to apply explainability methods to RL. In this paper, we briefly overview some of them (Section 2), we choose one based on the creation of graphs representing the observable agent's behaviour, and we apply it to a practical use case (Section 3). In order to validate the results of the method, we present a proposal based on creating a policy based on this generated graph and we use it to find problems on the original proposal as well as to validate the explanations produced (Section 4). The paper ends with a summary of the main conclusions and contributions from the work done (Section 5).

2. State of the art

Over the years, a vast variety of different explainability and interpretability approaches specific to the setting of RL has been proposed. In this section we provide a short overview of some of them and as well as the one that was the foundation of this paper. A more comprehensive and detailed study of the explainability methods in RL can be found in the recent surveys on the topic [1,14].

When considering ways to classify the methods to obtain explanations or describe the inner logic of RL methods, several approaches can be followed. One of the most common of them is to denote whether the explanations are *intrinsic* to the RL model or algorithm itself or are generated *post-hoc*. In addition to that, it is common to further subdivide these approaches by the scope of their explanations into *global* and *local* ones. The former showcase the global strategy used by the agent, while the latter can explain the policy's actions locally on case-by-case basis.

Before we go into specialised approaches, it is important to note that various statistics and metrics can be produced during and after the agent's construction or training, across all of these classifications. For instance, in [15] the authors propose to gather three levels of performance data: the data about the environment, about the behavior of the agent, and the data from the meta-analysis of the previous two, possibly with some domain knowledge. For each level they have outlined a wide array of statistics and metrics that may yield useful insights on agent's performance and the environment's influence on it.

Decision tree models are a classical example of *global intrinsic* approach, with the new methods and their variations still being proposed [16,4]. Here, the RL agent simply needs to "answer" a straightforward series of questions going from the root of a tree-like question structure to get the instructions on which action to choose. Although these methods are meant to be understandable by design, building a decision tree model that can adequately perform in a complex environment usually produces an enormous tree that is too complex to be analysed as whole (thus imposing a trade-off of accuracy and explainability). A number of approaches are proposed to deal with these issues. For instance, [4] proposes to use NN models to generate decision trees, while [16] introduces differentiable decision trees that can be incrementally updated and trained via, for example, gradient descent.

Another example of a *global intrinsic* approach is using agent policies built with some high-level domain-specific programming language [19]. This way the generated policies are transparent by design, you need only to analyse their sequences of commands.

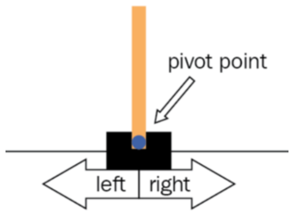
In environments with visual observations we can find a large array of saliency maps-based methods which can represent both *local intrinsic* and *local post-hoc* explainability approaches. For instance, a method proposed in [13] in addition to selecting an action, also generates intrinsic importance maps for the pixels of the image. Alternatively, the classical saliency maps described in [17] use post-hoc backpropagation to find the maps. Additionally, [6] proposes a perturbation-based approach (akin, for example, to the well-known LIME algorithm for deep NN) to generate saliency maps. Although one needs to be careful with utilizing it for some tasks with potentially critical consequences, as perturbation-based methods were recently shown to be prone to adversarial attacks [18].

To get *global post-hoc* explanations, we may analyse the behavior of the trained policy to pinpoint the most interesting or important situations or states, showcasing the behaviour of the agent. Different metrics and approaches can be used to select these execution traces. For instance in [2] the importance measure for states is defined as the difference between the discounted reward values for the best and the worst action choice in this state. Traces centered around the most important states are then shown. Another metric is proposed in [8]: here authors look for critical states which are defined as states for which choosing a random action is significantly worse than choosing a specific one in terms of reward. Further analysis about this family of methods can be found in [3].

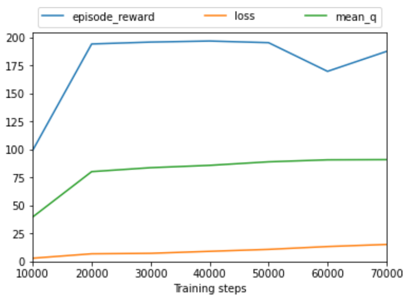
Finally, on the border between *global* and *local post-hoc* explanation approaches lie various methods that create simplified representations of the policy or the environment (or its observed version) and then use them to generate local explanations. For instance, in [9,10] authors use NN to generate a graph representation of scenes (images, for instance) that can be later used by a reasoning engine. Another way of doing it is to build a full Markov decision process and traverse it as a graph from the query state to the main reward state [12]. This allows us to ask simple questions about the chosen actions. As an alternative we can simplify the state representation (discretising it if needed) to make the process more feasible in more complex environments [7]. We base our work on the latter approach. It consists of creating a policy graph by creating a mapping from the original state to a set of predicates and then repeatedly running the agent policy, recording its interactions with the environment. This graph of states and actions can then be used for answering simple questions about agent's execution which is shown in Section 3.

3. Generating explanations for the Cartpole scenario

The approach we followed was to attempt to reproduce the method described in [7] with regards to its application on the Cartpole environment. In this paper, the authors present a list of predicates to represent and discretise the states of the environment, and a list of examples of automatically generated explanations, tested against explanations proposed by humans, while accounting for "the occasional presence of incorrect actions taken" by the trained policy. Looking into methodology, we were interested in several aspects of the original work that were not explicitly tested. First of all, we wanted to know how significant the presence of incorrect actions is from a performance point of view and how they might impact the quality of the explanations. Also, we wanted to know whether



(a) Cart Pole standing up



(b) Loss, reward and mean q-values of our trained agents

Figure 1. Cart Pole environment and training illustration

there is any method to validate the quality of the explanations complementing human validation.

This environment has been taken from the OpenAI [1] Gym AI library. It consists of a pole attached by an un-actuated joint to a cart, which moves along a frictionless track as shown in Figure 1a. The system is controlled by applying a force going left or right to the cart. The pendulum starts upright and the gravity force will make it fall if not controlled correctly. The goal is to prevent it from falling over. The state is represented with four numbers representing cart position (*pos*), cart velocity (*vel*), pole angle (*ang*) and pole velocity (*velAtT*). It is considered fallen when the pole angle goes beyond 12^o from the center or when the cart goes out of the truck.

Connected to this environment, we have used the *keras-rl2* python library. We created an experimentation pipeline based on training Deep Q Network (*DQN*) Agents using, as in [7], a neural network with 3 convolutional layers with ReLU as an activation function, a reward function equal to the number of steps without the pole fallen and a learning rate of 1e-3. The average loss (based on MAE), episode reward and mean q-values of the training of 75 agents are summarised in Figure 1b.

Our objective was not to find the optimal policy for solving this problem, but rather to find a method to generate agents with variable levels of performance in order to analyse the outcomes of the explainability method at these different levels, and to study the impact of the presence of incorrect actions. From Figure 1b we can see that our agents receive a poor reward at 10k steps but at 20k steps they already solve the problem (which is considered to happen when the average reward is higher than 195). However, at 60k steps the reward starts being unstable while the loss function is always growing. Based on these metrics, we used a maximum of 70k steps for training due to not being seemingly useful to put a higher maximum for our study.

The explainability method proposed in [7] is based on generating a policy graph (PG) representing the states and actions observed by the trained agents acting on random environments. A policy graph G is a tuple $\langle R, N, \varepsilon, \phi \rangle$ where R is a root node representing an initial point for decision making, N is a set of nodes, representing states in our case, ε is a set of directed edges, representing actions in our case, and ϕ is the matrix of transition probabilities for the edges [5]. This formalisation allows to represent any multistage stochastic programming problem, including Markov decision processes as a special case.

[1] https://gym.openai.com

In order to build a policy graph, we need to discretise states and actions. For this, we need to define a set of predicates, which will also be useful when trying to transform the state description into natural language. The initial proposal consisted in using 10 predicates (combining terms and domains):

- $pole\_falling(X)$, with $X = left$ when $[ang < 0 \wedge velAtT < 0]$ and $X = right$ when $[ang > 0 \wedge velAtT > 0]$.
- $pole\_stabilizing(X)$, with $X = left$ when $[ang < 0 \wedge velAtT > 0]$, $X = right$ when $[ang > 0 \wedge velAtT < 0]$.
- $pole\_standing\_up()$ when $[-0.0005 < ang < 0.0005]$.
- $cart\_moving(X)$, with $X = left$ when $[vel < 0]$, $X = right$ when $[vel \geq 0]$.
- $cart\_pos(X)$, with $X = far\_left$ when $[pos \leq -2]$, $X = far\_right$ when $[pos \geq 2]$.
- $cart\_near\_middle()$ when $[-2 < pos < 2]$.

The Gym library allows us to introduce a state and an action and have in return the next state, which we can introduce into the agent that will give us the action to be taken. This cycle allows us to have full knowledge of how the run is going. With this data we have all we need to build the PG graph. We made each agent perform 2k runs of the game of 200 steps each one. We stored all the decisions that the agent took for the reached states, as well as the states visited just after taking the decisions. The created graph shows us the probability of taking a certain action for each state, as well as the probability of reaching a state after taking such action.

The policy graph can then be used to answer three questions that help explaining an agent behaviour: *1) What will you do when you are in x state?*, *2) When do you perform x action?* and *3) Why did not you perform x action in y state?*. The answer to 1) is generated by looking for the most used action in the policy graph from the input state that the user wants to check. In 2), the user inputs an action and the policy graph is used to search for the states where this action has the higher probability to be taken. And in 3), with an action and a state as inputs, the policy graph is used to look for nearby (similar) states to the input state and, for each one, there is a check on whether the contrary action is more likely to be taken, in which case the answer will consist on inferring the difference (in terms of holding predicates) between both states.

While testing the explainability algorithm, we found that not all possible states were reached during the policy graph creation phase. To account for potential edge cases, we introduced a modification in the algorithm to enable searching for nearby (similar) states in the policy graph when querying for an unknown state.

4. Validating the explainability method: creation of a graph-based agent policy

Once the explainability algorithm was implemented and improved, we had access to the generation of natural language explanations as shown in [7]. In that case, the validation is carried out by comparing the generated sentences against sentences written by human experts. One of our objectives was to look for complementary methods that could be automated in order to reduce the dependency to such domain-specific experts.

With the answers from the three questions, it was possible to know if they had more or less sense from a human perspective, but it was not possible to ensure that the answers given had any relationship with the behavior (or even the strategy, if there is any) of the agent, and therefore some kind of validation was missing. However, having a PG built from the observation of the agent's behaviour can give us a powerful tool to work with.

For this validation, we propose the creation of an agent policy inferred from this PG structure, trying to mimic the original trained agent's policy, in order to compare the behaviour of both. One concern related to this proposal is that the PG graph is based on a simplification of the states and the actions (using predicates), and therefore such a policy could also be an over-simplification of a policy that is backed by a deep neural network. However, our aim was not to create equal agents but rather to ensure that the explanations generated are able to reflect the trained agent.

Following this proposal, we implemented, using the Gym API, a policy based on answering the first of the three questions (*What will you do when you are in x state?*) for each current state and using the output to determine the action executed on the environment. Once implemented, we started testing it on the environment in order to check whether the policy was functional.

In this process, we encountered one problem: in a high percentage of runs, the pole ended up falling or going out of track. This brought up two issues: the results were unsatisfactory as the performance of the policy was much lower than the trained agent's average reward; and the behaviour of the policy graph-based policy did not quite reflect the original behaviour. Our hypothesis was that this divergence was caused by a poor state representation with the 10 predicates. By our own observations, the trained agent was able to learn a concept not possible to capture by these predicates, namely the pole being displaced left or right but in a stable position (due to the inertia of the moving cart). We solved this by adding two more predicates to the state representation:

- $stuck(X)$, with $X = left$ when $[\neg pole\_standing\_up() \wedge \neg\exists x : pole\_falling(x) \wedge \neg\exists y : pole\_stabilizing(y) \wedge ang > 0]$, $X = right$ when $[\neg pole\_standing\_up() \wedge \neg\exists x : pole\_falling(x) \wedge \neg\exists y : pole\_stabilizing(y) \wedge ang < 0]$.

By making this change, the new policy inferred from the PG generated using the 12 predicates turned out to be feasible and effective, as we will see later in this section. This issue indicates that human validation might not be enough when dealing with behaviours of agents in scenarios representing complex systems. In order to analyse the behaviour of the policy graph-based agent, we trained 15 agents for a number of steps between 10k and 70k, for a total of 105 agents. For each of these agents, we generated its corresponding policy graph, with the intention of inferring a policy based on answering the first question.

However, doing a direct comparison between both agents can yield misleading conclusions due to the *curse of dimensionality*: because the Cartpole scenario defines a complex system, reducing the state from a high amount of real valued variables to discrete predicates with small domains may entail frequent deviations in the actions taken by each policy in certain states. On the one hand we have a policy that is the product of training with a Q-learning algorithm in a space of states and actions defined by combinations of continuous variables that might be difficult to interpret; on the other hand we have a policy that is based on a very reduced set of predicates that have been designed with interpretability in mind. It cannot be assumed that both policies will have the same expressive power.

However, if we assume that the policy graph is able to generalise (or else the explainability method is pointless), we should be able to ignore these deviations and consider the strategy or general behaviour of the two policies compatible. Our hypothesis is that if we had a single policy that randomly chose actions based on the DQN-policy or the policy graph indistinctly, the behaviour of the agent should stay consistent with respect to the original one. The actions chosen by the policy graph should not interfere –or rather, should be compatible with– the actions chosen by the DQN-policy, as any deviations caused by the divergence between the policies should be mitigated in the long term by the actions chosen by the more fine-grained DQN-policy. To this mixed policy, we also add control policies for checking the behaviour against random agents. Therefore, for each trained agent we generated the following policies:

- DQN: all actions are chosen by the policy trained by the Deep Q-Network.
- PGR: all actions are chosen by the policy inferred from the policy graph.
- RND: each scenario step, the action is chosen at random from all valid actions.
- HEX: each scenario step, the action is chosen at random between DQN and PGR.
- HRD: each scenario step, the action is chosen at random between DQN and RND.

Figure 2 shows a histogram of the last steps the different policies achieved before failing (or succeeding if reaching 200). The two agents with a random selection component, HRD (5.54% success rate) and RND (0.00%) perform very poorly while DQN (78.65%), PGR (78.16%) and HEX (72.95%) have very good and similar success rates. As predicted, the PGR agent success rate deviates considerably at the early steps, as can be seen in the non-marginal frequencies in the histogram between around steps 25 and 50. This can be attributed to the aforementioned deviations due to the state simplification and discretisation, having an impact on the aptitude to stabilise on edge situations, i.e. when the pole is very far from the center. However, DQN and HEX – which is 50% based on PGR – have a very similar histogram, which points at the fact that the PGR policy has not had a strong effect, adjusting well to the original behaviour.

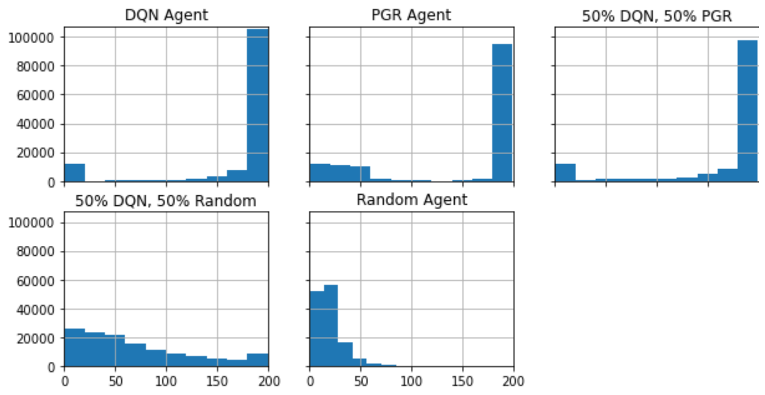


Figure 2. Histogram of last stable step before failing (or succeeding if $step = 200$)

This analysis can be reinforced by analysing the average cart movement, which is a variable that is causally related to the actual behaviour of the agent. In Figure 3 we can see the relationship between the cart movement and the last step before failing or succeeding.

When looking at DQN, we can appreciate two main patterns (ignoring most of the cases, which are successes and they cluster at the right-most border). Across the whole range of steps, most of the failures happen after having moved the cart a low distance (avg.

of around 0.01 and 0.15), forming a wide band from side to side. There seems to be a second pattern coming from those failures in where the cart has moved a higher distance, with the last step not being lower than 100. These patterns indicate the presence of at least two distinct behaviours, which seem to be also distinguishable in HEX and HRD. Again, the effect of the PGR actions in HEX seem to have little effect on the original policy. However, while the patterns seem to be also visible in the PGR case, they are heavily simplified, which may be a consequence of the simplification of the state representation.

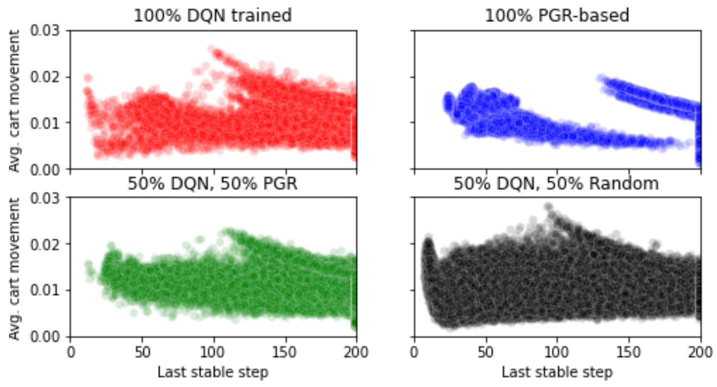


Figure 3. Relationship between last step explored and the average cart movement per step

This raises concerns about the generalisation power of the reduction of the DQN policy into the PGR one. In a more in-depth analysis, we can look at the evolution of the performance based on the amount of training steps. As we saw in Section 3, with our training configuration DQN reaches almost optimal reward between 20k and 50k steps but it becomes unstable after that. In Figure 4 we can see the effect on several metrics: performance, cart movement, pole velocity and pole rotation. The performance of PGR between 10k and 40k steps is above 80% while the performance of DQN is always below 80%, which means that the PGR graph is capable of generalising well until 50k steps.

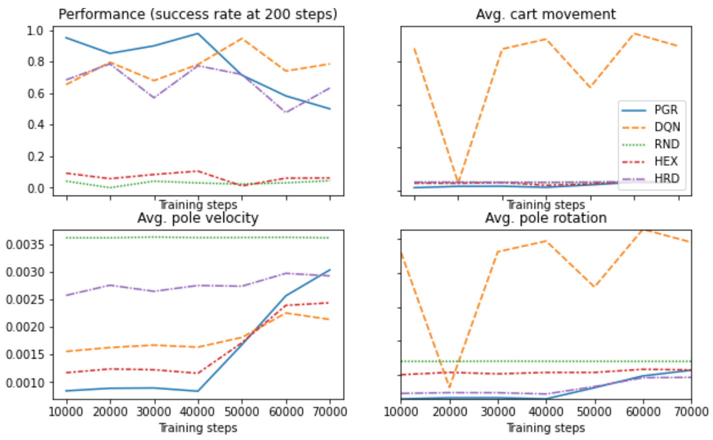


Figure 4. Evolution of metrics through training steps

Finally, we analysed the correlations between the three Cartpole-specific metrics (Figure 5), using Spearman due to these metrics being ordered (all the policies were run on the same random scenarios) but not following a normal distribution. The metric more causally connected to the behaviour (the actions) of the agent, the cart movement, does not entail very high correlations. The highest values are between DQN and HEX (0.60, $p<0.001$) and between PGR and HEX (0.53, $p<0.001$), which makes sense as this policy is a combination of both. The correlation between DQN and PGR (0.26, $p<0.001$) is statistically significant but it is quite low. If we look at the effect of the actions on the pole (both velocity and rotation) we can find higher correlations: DQN and PGR (0.55, $p<0.001$), DQN and HEX (0.72, $p<0.001$), PGR and HEX (0.67, $p<0.001$).

In summary, from the performance results combined with these correlations we can infer that if we look at all the runs globally, the behaviours of the two agents might yield similar results, both in performance and in the effect of the actions (the behaviour of the pole). However, if we look at the individual runs, we will find many frequent deviations. In other words: by applying the explainability method on the Cartpole scenario we can extract rough explanations that can approximate the behaviour or strategy of the original agent *in general*, but the method might not be able to explain with precision the behaviour in every instance of the scenario.

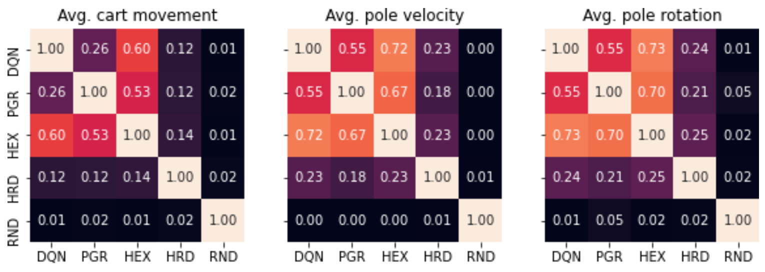


Figure 5. Cross-correlations between cart movement, pole velocity and pole rotation, averaged by step

5. Conclusions

The topic of XAI applied to Reinforcement Learning is growing in relevance and can be key to tackle issues such as the possibility to assess the quality of the behaviour of an agent or aid in the interaction between humans and AI-based agents. There are already some approaches in the literature that try to provide explainability in this context. However, they need to be tested in practical use cases in order to assess their effectiveness.

In this paper, we show our work in this direction in which we choose a method and try to reach a baseline for further research. After an analysis of the literature, we chose a method based on the generation of PGs by discretising the state representation into predicates, and applied it to a simple yet complex scenario (Cartpole). The result was a policy graph that allowed us to produce explanations, that according to the original work should be validated by human experts with domain-specific knowledge.

In order to understand how good the baseline that we were getting was and to be able to validate the results inferred from the policy graph, we propose a method for extending this validation with an automatic process by creating several policies based on both the original policy and the policy graph, and testing them along with the original policy in random new scenarios. By applying this method we were able to 1) detect predicates that were missing in order to have a complete state representation, and 2) analyze the quality of the policy graph as a tool to store a simplified representation of the original behaviour.

This paper presents part of our ongoing work, which is currently advancing in two research lines. First of all, we are extending the generation of the graph-based policy, in order to include not only one but the three types of questions that [7] defines for generating explanations. On a second topic, we are applying the same method to other environments with a much more complex state representation to check whether this method can be generalised. Currently we are applying it to the VizDoom environment.

References

[1] A. Alharin, T. N. Doan, and M. Sartipi. Reinforcement learning interpretation methods: A survey. 8:171058–171077, 2020.

[2] D. Amir and O. Amir. HIGHLIGHTS: Summarizing agent behavior to people. AAMAS '18, page 1168–1176, Richland, SC, 1 2018. IFAAMAS.

[3] Ofra Amir, Finale Doshi-Velez, and David Sarne. Summarizing agent strategies. Autonomous Agents and Multi-Agent Systems, 7 2019.

[4] O. Bastani, Y. Pu, and A. Solar-Lezama. Verifiable reinforcement learning via policy extraction. In Proceedings of the 32nd International Conference on NIPS, NIPS'18, page 2499–2509, Red Hook, NY, USA, 2018. Curran Associates Inc.

[5] O. Dowson. The policy graph decomposition of multistage stochastic programming problems. Networks, 76(1):3–23, 2020. Publisher: Wiley Online Library.

[6] S. Greydanus, A. Koul, J. Dodge, and A. Fern. Visualizing and understanding Atari agents. In Proceedings of the 35th ICML, volume 80 of Proc. of ML Research, pages 1792–1801. PMLR, 10–15 Jul 2018.

[7] B. Hayes and J. A. Shah. Improving robot controller transparency through autonomous policy explanation. In Proceedings of the 2017 ACM/IEEE Intl. Conf. on HRI, page 303–312, NY, USA, 1 2017. ACM.

[8] S. H. Huang, K. Bhatia, P. Abbeel, and A. D. Dragan. Establishing appropriate trust via critical states. 2018 IEEE/RSJ IROS, pages 3929–3936, 2018.

[9] M. Klawonn and E. Heim. Generating triples with adversarial networks for scene graph construction. CoRR, abs/1802.02598, 2018.

[10] M. Klawonn, E. Heim, and J. A. Hendler. Exploiting class learnability in noisy data. CoRR, abs/1811.06524, 2018.

[11] L. Longo, R. Goebel, F. Lecue, P. Kieseberg, and A. Holzinger. Explainable artificial intelligence. In Machine Learning and Knowledge Extraction, pages 1–16, Cham, 2020. Springer.

[12] P. Madumal, T. Miller, L. Sonenberg, and F. Vetere. Explainable reinforcement learning through a causal lens. Proceedings of the AAAI Conference, 34(03):2493–2500, Apr. 2020.

[13] D. Nikulin, A. Ianina, V. Aliev, and S. Nikolenko. Free-lunch saliency via attention in atari agents. In 2019 IEEE/CVF ICCVW, pages 4240–4249, 2019.

[14] E. Puiutta and E. M. S. P. Veith. Explainable reinforcement learning: A survey. In Machine Learning and Knowledge Extraction, pages 77–95, Cham, 2020. Springer.

[15] P. Sequeira, E. Yeh, and M. T Gervasio. Interestingness elements for explainable reinforcement learning through introspection. In IUI Workshops, 1 2019.

[16] A. Silva, M. Gombolay, T. Killian, I. Jimenez, and S.-H. Son. Optimization methods for interpretable differentiable decision trees applied to reinforcement learning. In Proceedings of the 23rd Intl. Conf. on AI and Statistics, volume 108 of Proceedings of ML Research, pages 1855–1865. PMLR, Aug 2020.

[17] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In Proc. of ICLR, 2014.

[18] D. Slack, S. Hilgard, E. Jia, S. Singh, and H. Lakkaraju. Fooling LIME and SHAP: Adversarial attacks on post hoc explanation methods. pages 180—-186, 1 2020.

[19] A. Verma, V. Murali, R. Singh, P. Kohli, and S. Chaudhuri. Programmatically interpretable reinforcement learning. CoRR, abs/1804.02477, 2018.