Distilling Deep Reinforcement Learning Policies in Soft Decision Trees

Youri Coppens[1,2] , **Kyriakos Efthymiadis**[1] , **Tom Lenaerts**[1,2] and **Ann Nowé**[1]
[1]Vrije Universiteit Brussel
[2]Université Libre de Bruxelles
yocoppen@ai.vub.ac.be, kyriakos.efthymiadis@vub.be, tlenaert@ulb.ac.be, anowe@ai.vub.ac.be

Abstract

An important step in Reinforcement Learning (RL) research is to create mechanisms that give higher level insights into the black-box policy models used nowadays and provide explanations for these learned behaviors or motivate the choices behind certain decision steps. In this paper, we illustrate how Soft Decision Tree (SDT) distillation can be used to make policies that are learned through RL more interpretable. Soft Decision Trees create binary trees of predetermined depth, where each branching node represents a hierarchical filter that influences the classification of input data. We distill SDTs from a deep neural network RL policy for the Mario AI benchmark and inspect the learned hierarchy of filters, showing which input features lead to specific action distributions in the episode. We realize preliminary steps towards interpreting the learned behavior of the policy and discuss future improvements.

1 Introduction

Recent advancements in Reinforcement Learning (RL) are largely attributed to the employment of deep neural network (DNN) models as a function approximation technique to estimate human-level control policies in a wide range of tasks [Mnih *et al.*, 2015; Schulman *et al.*, 2017]. Given the high number of parameters and complex architecture of these models, it is hard to directly interpret and understand the learned strategies, especially as networks grow deeper in size. An important step in Artificial Intelligence research is therefore creating mechanisms that give higher level insights into the learned behaviors of these black-box models and make the learned policies explainable.

Given a learned RL control policy, which maps state observations to probabilities over a set of finite actions, we can easily draw the analogy to a classification task in which we learn which actions correspond to a specific state. In an attempt to gain further insights into a learned deep reinforcement learning policy, we look at the applicability of a network distillation technique similar to [Frosst and Hinton, 2017], which tries to mimic DNN classification output through a Soft Decision Tree (SDT) surrogate model. More precisely, the contribution of this work is that we train an SDT to mimic the action classification of a deep neural network control policy for the Mario AI benchmark and show how the tree filters input features hierarchically, leading to the action chosen by the neural network. This approach provides a form of interpretation on how the deep control policy operates.

2 Related work

To tackle the computational complexity of machine learning models, distillation techniques are used in an attempt to transfer the knowledge from such models into smaller ones, having less computational overhead yet achieving comparable performance rates [Hinton *et al.*, 2015; Rusu *et al.*, 2016]. For the purpose of explainability however, decision tree models are one of the preferred methods, as these provide hierarchical information on the considered features for decision making. Based on the hierarchical mixture of experts from Jordan and Jacobs [1994], Frosst [2017] proposes a hybrid model of a binary decision tree with perceptron neural networks called soft decision trees. Whereas Frosst demonstrates this method with a DNN for a classification problem, our purpose is to apply soft decision trees on a Reinforcement Learning policy network. Recently, Tanno et al. [2019] generalized the concept of hybridization of neural networks and decision trees as Adaptive Neural Trees.

Recently, the RL community has started exploring approaches that lead to interpretable policies. Some approaches immediately learn interpretable RL policy models for instance through the addition of fuzzy particle swarm optimization [Hein *et al.*, 2017] and genetic programming [Hein *et al.*, 2018]. Other approaches mimic deep neural network output through heuristic program search [Verma *et al.*, 2018] or decision tree methods [Liu *et al.*, 2019]. The latter work from Liu [2019] is related to ours, as they make an attempt to distill a deep neural network representing a Q-value function into a decision tree surrogate model. Our work however focuses on mimicking an explicit policy.

3 Background

3.1 Reinforcement Learning

Reinforcement Learning addresses the problem of learning to make decisions within an environment [Sutton and Barto, 2018]. This is typically modelled through a Markov Decision

Process as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$, where $\mathcal{S}$ represents the *state space*, $\mathcal{A}$ the *action space*, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ the *state transition function*, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ the *reward function* and $\gamma \in [0, 1]$ the *discount factor*.

In the discrete setting, an agent observes each step t the current state of the environment s_t and samples an action a_t from its current policy $\pi_t : \mathcal{S} \times \mathcal{A} \to [0, 1]$. After performing the action, the agent observes a reward $r_t = \mathcal{R}(s_t, a_t)$, and the environment transitions to the next state s_{t+1} following the transition probabilty defined by $\mathcal{T}$. With the state-action transitions the agent observes, its goal is to learn to maximize the expected return $\mathbb{E}[R_t]$, where $R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$, the sum of future discounted rewards. The discount factor γ regulates the agent's preference for immediate rewards over long-term rewards. Lower values set the focus on immediate reward, whilst higher values result in a more balanced weighing of current and future reward in R_t. Policy-based methods explicitly learn a parameterized control policy model as a conditional probability distribution over actions given a state observation $\pi_\theta(a_t|s_t)$ by performing gradient ascent on the parameters θ with a policy gradient $\nabla_\theta \log \pi_\theta(a_t|s_t) R_t$ [Williams, 1992].

3.2 Soft Decision Trees

Soft Decision Trees (SDTs) [Frosst and Hinton, 2017] are a hybrid classification model of binary decision trees with a predetermined depth and neural networks. In the model, each branching node i consists of a single perceptron, with a weight vector parameter w_i and a bias parameter b_i. On a given input x, branching nodes determine the probability to traverse to the right child node as $p_i(x) = \sigma(\beta(xw_i + b_i))$, where β is an inverse temperature parameter and σ the sigmoid logistic function.

The model learns a hierarchy of filters in its branching nodes to assign input data to a particular leaf node l with a certain path probability $P^l(x)$, the overall product of branching probabilities leading from the root to node l. The leaf nodes learn softmax distributions $Q^l_k = \frac{\exp(\phi^l_k)}{\sum_{k'} \exp(\phi^l_{k'})}$ over the possible output classes k, parameterized by ϕ_k .

The training loss of an input sample $L(x)$ is based on the entropy between leaf predictions and the target distribution T_k, weighted by its respective path probability:

$$L(x) = \log \left(- \sum_{l \in Leafs} P^l(x) \sum_k T_k \log Q^l_k \right) \quad (1)$$

In addition to the training loss, a regularizing cost C is added for the branching nodes to encourage equal usage of their respective subtrees in the decision process, based on the cross-entropy between a branching node's current distribution α_i and the discrete binary uniform distribution:

$$C = \sum_{i \in Branching} -\lambda_i (0.5 \log(\alpha_i) + 0.5 \log(1 - \alpha_i)) \quad (2)$$

where

$$\alpha_i = \frac{\sum_x P^i(x) p_i(x)}{\sum_x P^i(x)} \quad (3)$$

Each branching node receives a penalty strength λ_i, based on a start value λ which is then decayed over the tree depth by a factor of 2^{-d}. Hence, nodes higher up in the tree get a stronger penalty cost for highly deterministic branching. The overall loss is minimized using mini-batch gradient descent optimization.

4 Methods

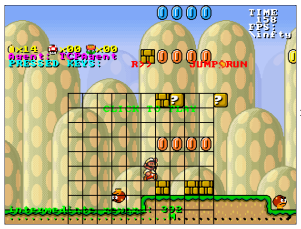
In this work, we are interested in the distillation of a deep reinforcement learning policy into a soft decision tree to gain further insights into the way the deep policy decides on an action. We evaluate the applicability of this method on the Mario AI benchmark [Karakovskiy and Togelius, 2012], based on the public Java emulator, *Infinite Mario Bros*. Agents are capable of interacting with the emulator through a TCP network connection. We train a Deep RL agent to control Mario and play a simple level of the emulator which we slightly modified in order to introduce different types of coins in the level. The available actions for the Mario agent are any possible combination of pressing the 5 emulator buttons (left, right, down, jump and run), making a total of 32 $(= 2^5)$ actions.

The goal is to reach the princess at the end of the level within the time limit of 3000 steps and collecting points in the meantime. Reaching the princess will grant the agent a reward of 100 points. Each step that brings the agent closer to the princess gives a reward of +1 and vice versa, steps leading away from the princess are punished with a reward of -1. Four different types of coins (green, blue, yellow and red) are scattered throughout the level and respectively provide a distinct reward of -20, -10, 0 and +20. Mario should thus ideally learn to collect red coins and avoid taking green and blue coins. Jumping on enemies and killing them gives a reward of +10. Mario cannot die, but does receive a penalty of -20 when hit by an enemy.

The state representation that our agent uses during learning is a 10 by 10 receptive field around Mario, i.e. a two-dimensional grid describing the world around the agent with block resolution. Each cell contains a numerical label related to the game object present in the environment at that position. Figure 1 illustrates the transformation from a game frame to a receptive field. As a result of this transformation, our Deep RL agent is trained on 100 input features.

We trained our Deep RL agent using the actor-critic Proximal Policy Optimization (PPO) algorithm [Schulman *et al.*, 2017] for 10 million steps. Given the state observations as input, two separate neural networks are trained to respectively represent a policy (actor) and state-value function (critic). The state-value network output is used as a baseline for updating the policy network and consists of 2 multi-layer perceptron (MLP) hidden layers of 128 units with hyperbolic tangent activation functions and a final linear layer to produce the state value. Similarly, the policy network also consists of 2 MLP hidden layers of 128 units with hyperbolic tangent activations and a final linear layer produces the 32 action probabilities.

After training, we distill the output of the deep policy network to a soft decision tree and analyze whether this tree-



$$\Rightarrow \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 9 & 9 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 4 & 4 & 4 & 4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 16 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 9 & 0 & 9 & 9 \\ 0 & 0 & 0 & 23 & 0 & 11 & 11 & 11 & 11 & 11 \\ 11 & 11 & 11 & 11 & 11 & 11 & 11 & 11 & 11 & 11 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 1: Transformation from a game frame to a 10 by 10 receptive field grid centered around Mario. Each cell of the grid has the size of a block in the game. The game objects in each of the grid's cells are translated into numerical labels as state representation for the learning agent. Whenever the receptive field ranges out of the game frame, it is padded with zeros (e.g. the bottom row). Note that the agent sees no distinction between breakable and unbreakable blocks.

based model enables us to get insights about the way our Deep RL agent controls Mario and chooses actions. We generated a training and test dataset, each containing recordings of 50 episodes played by the trained PPO policy. The datasets consist of DNN in- and output pairs $(s, \pi(s))$ of state observations and predicted action probability distributions for each step of the performed episodes. We fitted SDTs of several depths, ranging from 3 to 7, on the training set for 40 epochs using Adam optimization [Kingma and Ba, 2014]. In all settings, the penalty weight for the regularization cost λ was 0.1 and the mini-batch size was 64.

5 Results

In this section, we show how SDTs can provide an interpretation of a trained deep RL policy and elaborate on the performance of these models when one would consider to replace the DNN with the surrogate SDT for task execution.

5.1 Interpreting soft decision tree explanations

The essence of soft decision trees is that this kind of model does not rely on hierarchical features to classify data, but instead relies on hierarchical decisions. Each branching node directly processes the whole input sample, hence making each decision at a level of abstraction that the user can immediately engage with. The branching nodes filter each feature with different weights, thus changing the focus on the input sample at each level of the tree. By examining the learned filters along the traversed path from the root to leaf, one can thus understand which features the soft decision tree considers to assign a particular action distribution to a particular state. Figure 2 shows the filters of the root node and its two direct children of a soft decision tree of depth 3 that we distilled from our PPO policy. Given the spatial aspect of the state representation of the Mario environment, we can visualize each filter as a heatmap of 10 by 10 cells. The higher the magnitude of the weight of a particular feature is, the more intense the color in the heatmap will be. Shades of red relate to positive weights while blue shades represent negative weights. The root puts a lot of negative weight on a specific feature in the upper area at position (6,6). This cell is positioned at the near top right of Mario. As Mario tends to move to the right, it is important to

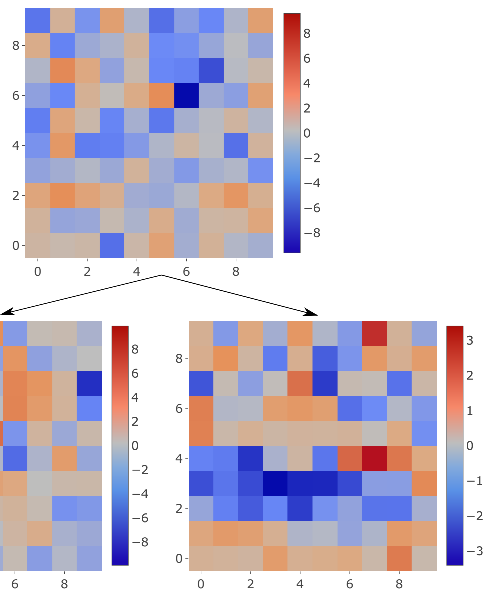


Figure 2: Heatmap visualization of the learned filters from the first split in a distilled soft decision tree of depth 3 for our Deep RL agent. Each node processes the observed state (10 by 10 receptive field of Mario) as a whole, but assigns different weights to each feature. The more intense the color, the higher the magnitude of the weight assigned to the feature (positive: red, negative: blue) and the more focus that feature receives. The root (top) focuses highly on a particular feature in the upper area, the left child mostly focuses on features in the third row and the right child assigned mainly negative weights (dark blue) underneath the center.

focus on what lies ahead of him. The left child rather puts focus on some features in the third row, which are located above Mario. The right child on its turn has an area of strongly negative weights underneath the center of the field. This particular area is in general the floor where Mario is standing on. Thus, depending on the choice made by the activation function in the root, the chosen child node at the next level of the tree will focus on the values of different subsets of features for the next branching decision.

Understanding why a particular state observation leads to a certain action distribution can be realized by examining how the input state itself is filtered along the path between the root and the chosen leaf node in the soft decision tree model. Similar to the filters visualized in Figure 2, filtered states can be visualized as a heatmap too. This way, one can see which elements in Mario's receptive field gain the most magnitude by the filter and are thus the most decisive factors to branch to a specific subtree. Figure 3 provides two examples where the predicted path of a soft decision tree of depth 3 for particular states is visualized from the root to the predicted leaf node containing the action distribution. The leaf node's distribution is shown through a bar chart. We also provide the action distribution produced by the PPO policy network as comparison. In the first example, at the top row of the figure, Mario is jumping through the air and sees a row of red coins above him and a group of blue coins below him. The branching nodes decided to always descend to the left child, ending up

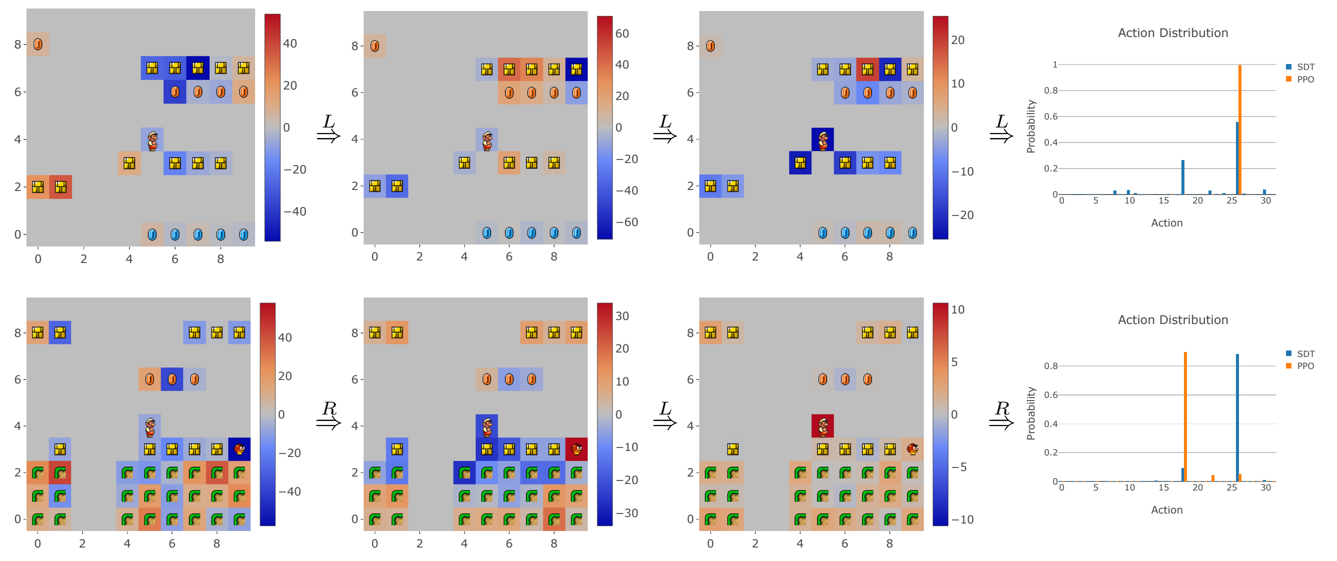


Figure 3: Two examples of an SDT classification of a particular state to a certain action distribution. Each branching node shows a heatmap representing the filtered input state before it enters the sigmoid activation function. In the first example (top row), the filters give high values to the red coins and blocks present in the state and give almost no value to the blue coins in the bottom of the field. In the second example (bottom row), the filters put value on the red coins and enemy monster present in the state.

in the leftmost leaf containing an action distribution. As blue coins result in negative reward, Mario should ideally decide to only collect red coins. Note that the filters do not give a lot of value to the blue coins, but rather to the red coins and the blocks near Mario where he can land on. From this we can tell why Mario will indeed not consider taking any actions to move towards the blue coins. The action with the highest probability in the determined leaf node is to jump to the right in running mode (action 26). Looking at the PPO policy output in the bar chart, the neural network was deterministic for this state and chooses the same action.

Since the distributions in the leafs need to generalize and provide an action strategy for multiple state samples, it is to be expected that the resulting action distributions in the soft decision trees can differ from the neural network output of the PPO agent for individual input samples. This is clearly visible in our second example, at the bottom row of Figure 3. This time, Mario stands under a group of red coins and an enemy monster is sighted on the right side of the receptive field. The path predicted by the tree was to first branch to the right node, then left and finally to the right action distribution. The action with the highest probability is to jump to the right in running mode (action 26), allowing Mario to collect the red coins and landing on top of the enemy. The distribution has also assigned some probability to the 18th action, which is running to the right. This would lead Mario to the enemy in order to land on it and kill it. The filters from the nodes along the path respectively focus on the present coins, then the enemy monster and finally on Mario itself. For this state, the predicted action distribution differs from the PPO agent's output, which mostly assigns probability to the 18th action.

A final examination of the soft decision trees can consist of inspecting all the action distributions it provides in its leafs. Figure 4 shows the learned action distributions of a soft decision tree of depth 3 distilled from our PPO agent, thus having 8 leaf nodes. We notice that most of the distributions have set probabilities on a small number of the available actions. Most leaf distributions assign probabilities to actions that relate to jumping to the right (action 10) with or without the addition to use the running mode (actions 18 and 26) or pressing the down button which makes Mario duck (actions 22 and 30). We can hereby determine that the soft decision tree in general has distilled a policy to move to the right as quick as possible. Depending on the case however, the model tells us that there are some specific alternative actions that the agent could opt for. Two of the leaf nodes give a more uniform distribution over the actions, but with more probability assigned to actions that do not move to the right, as sometimes Mario would have to reposition itself in order to be able to jump over obstacles.

5.2 Performance: DNN vs. SDT

Once we have distilled an interpretable surrogate model of the PPO policy, one could consider using these models as a replacement for the deep neural network in the case these models achieve a similar performance in task execution. We therefore look at the performance of several distilled soft decision trees, ranging from a depth of 3 to 7, in terms of achieved reward collection over 100 episodes in Figure 5. On average, all the generated trees performed less in terms of reward collection during the episodes than our PPO agent. We noticed that the soft decision trees with a depth of 4 and 5 controlling Mario even sometimes tended to get stuck at specific states in the episode, e.g. inside a pit or in front of flower pot obstacles and not being able to let Mario jump out of these

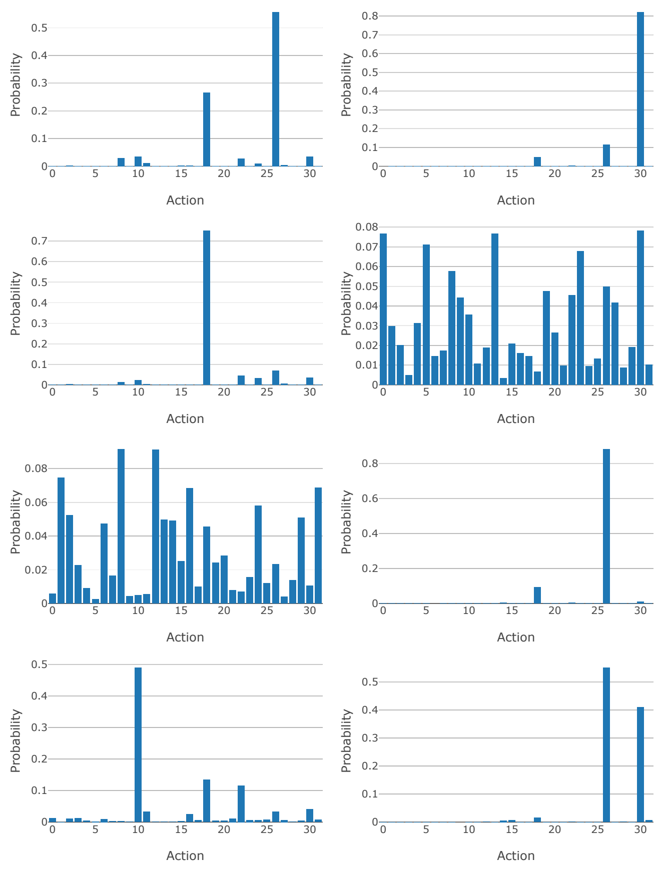


Figure 4: The action distributions learned by the leaf nodes of our distilled soft decision tree with depth 3. Most distributions assign probabilities to actions that relate to letting Mario jump to the right as quickly as possible (e.g. actions 10, 18, 22, 26 and 30).

places. Since the distributions in the leafs have to generalize and provide an action strategy suitable for multiple state samples, it is to be expected that the resulting action distributions in the smaller model do not fully cover the deep policy and thus perform less in terms of task execution. With bigger trees, we noticed an increase in average reward collection, as the increase in complexity allows the tree to generate more fine-grained distributions and hence the better the task performance becomes. However, this performance increase comes at the trade-off of a decreased interpretability, as the chain of filtering nodes leading to a specific action distribution becomes larger.

6 Discussion and Future Work

The employment of soft decision trees to interpret a deep RL policy is one the many first steps taken towards explainable reinforcement learning. Given that our evaluated environment contains spatial aspects, we were able to provide an interpretation by visualizing learned features and follow these to the actions chosen. The question remains whether these kind of soft decision tree models also achieve an increased interpretability in non-spatial settings, e.g. with biological data. Soft decision trees however, still lack a form of

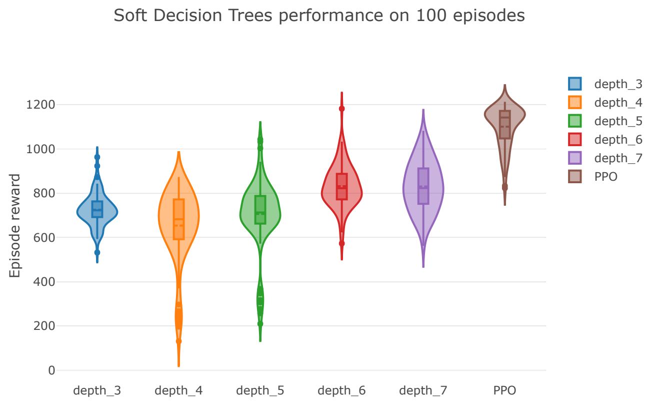


Figure 5: Violin plot showing the performance of our fitted soft decision trees on 100 episodes, in comparison to 100 episodes played by the PPO agent.

human-readable abstractions. One can argue that it does not provide explanations in e.g. a symbolic form. The addition of rule-based machine learning techniques could become an interesting asset to provide more symbolic explanations of the RL policy. By performing user tests, we could also try to answer whether these type of models can be generally accepted as explanations by humans for decisions made by RL agents.

Rather than mimicking a pre-trained DNN, there is the possibility to fit a soft decision tree directly on data. Frosst [2017] demonstrated this for classification tasks, but reported lower performance rates when soft decision trees were trained immediately on the data set. As a consequence, we have not yet explored this in a RL setting. We assume adjustments have to be made to the loss function to allow us to train a soft decision tree policy directly from agent experience. Some of the considered future steps are therefore to examine the methods used by Liu's [2019] and draw relations to the policy level instead of Q-values. This could potentially allow us to train soft decision trees directly as a policy.

Given that our fitted surrogate models have a lower performance in terms of episode rewards, the question rises whether the SDTs are actually explaining the underlying policy. Therefore, further attention has to be spent on how well the SDTs match the mimicked policy. In order to increase the performance of SDTs, we could opt for online finetuning of the surrogate model in the environment rather than solely supervised training on episode recordings. This finetuning would be similar to the active play setting performed in [Liu *et al.*, 2019].

In addition, one drawback of soft decision trees is that these come with a predetermined depth which has to be manually chosen beforehand. Ideally, we could opt for adaptively growing trees, as recently proposed by [Tanno *et al.*, 2019], where one could determine the degree of compromise the user wants in terms of interpretability and complexity.

7 Conclusion

We have shown preliminary steps in interpreting RL policies directly on action distributions by applying soft decision trees to mimic a deep reinforcement learning policy for the Mario AI benchmark. By generating state-policy samples from a trained PPO agent, we learn hierarchical decision filters on states in the branching nodes and generalized action strategy distributions in the leafs. Visualizing and examining the filters show the considered features along the existing paths in the tree. Applying these filters on state samples result in preliminary insights regarding what elements present in the state observation lead to a certain action distribution. Inspecting the present action distributions in the leafs allows one to gain insights into the adopted strategies of the agent. This approach leads to an increased understanding of the operation of the policy directly linked to elements present in the state observations.

References

[Frosst and Hinton, 2017] Nicholas Frosst and Geoffrey Hinton. Distilling a Neural Network Into a Soft Decision Tree. In Tarek R. Besold and Oliver Kutz, editors, *Proceedings of the First International Workshop on Comprehensibility and Explanation in AI and ML 2017*, volume 2071 of *AI*IA Series at CEUR Workshop Proceedings*. Tarek R. Besold & Oliver Kutz, 2017.

[Hein *et al.*, 2017] Daniel Hein, Alexander Hentschel, Thomas Runkler, and Steffen Udluft. Particle swarm optimization for generating interpretable fuzzy reinforcement learning policies. *Engineering Applications of Artificial Intelligence*, 65:87–98, 2017.

[Hein *et al.*, 2018] Daniel Hein, Steffen Udluft, and Thomas A. Runkler. Interpretable policies for reinforcement learning by genetic programming. *Engineering Applications of Artificial Intelligence*, 76:158–169, 2018.

[Hinton *et al.*, 2015] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the Knowledge in a Neural Network. *arXiv preprint arXiv:1503.02531*, 2015.

[Jordan and Jacobs, 1994] Michael I. Jordan and Robert A. Jacobs. Hierarchical Mixtures of Experts and the EM Algorithm. *Neural Computation*, 6(2):181–214, 1994.

[Karakovskiy and Togelius, 2012] Sergey Karakovskiy and Julian Togelius. The Mario AI Benchmark and Competitions. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):55–67, 2012.

[Kingma and Ba, 2014] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[Liu *et al.*, 2019] Guiliang Liu, Oliver Schulte, Wang Zhu, and Qingcan Li. Toward Interpretable Deep Reinforcement Learning with Linear Model U-Trees. In Michele Berlingerio, Francesco Bonchi, Thomas Gärtner, Neil Hurley, and Georgiana Ifrim, editors, *Machine Learning and Knowledge Discovery in Databases. ECML PKDD 2018*, volume 11052 of *Lecture Notes in Computer Science*, pages 414–429. Springer, Cham, 2019.

[Mnih *et al.*, 2015] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[Rusu *et al.*, 2016] Andrei A. Rusu, Sergio Gomez Colmenarejo, Çaglar Gülçehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations*, 2016.

[Schulman *et al.*, 2017] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[Sutton and Barto, 2018] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning : An Introduction*. MIT Press, 2nd edition, 2018.

[Tanno *et al.*, 2019] Ryutaro Tanno, Kai Arulkumaran, Daniel C. Alexander, Antonio Criminisi, and Aditya Nori. Adaptive Neural Trees. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6166–6175. PMLR, 2019.

[Verma *et al.*, 2018] Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically Interpretable Reinforcement Learning. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5045–5054. PMLR, 2018.

[Williams, 1992] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256, 1992.