Imitation learning from neural networks with continuous action spaces using regression trees

Tymon Cichocki
Responsible Professor: Anna Lukina
Supervisor: Daniël Vos
EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 22, 2025

Name of the student: Tymon Cichocki
Final project course: CSE3000 Research Project
Thesis committee: Anna Lukina, Daniël Vos, Luciano Cavalcante Siebert

Abstract

Reinforcement learning models are being utilised in a wide range of industries where even minor mistakes can have severe consequences. For safety reasons, it is important that a human expert can verify the decision-making process of a model. This is where interpretable reinforcement learning proves its importance. This research is focused on training decision tree policies with a limited size and evaluating them on continuous action space environments. For that, a DAgger algorithm is used with appropriate modifications to account for the continuous setting. The results demonstrate that small decision trees can replicate the high-performing neural network policies (e.g., TD3), achieving close to benchmark scores. Therefore, it is possible to explain the complex model's behaviour with much more understandable structures.

1 Introduction

Reinforcement learning is an area of machine learning that focuses on training agents that would perform optimal decisions in a specified environment. It is used in a wide variety of fields such as robotics, self-driving vehicles, game playing agents (such as AlphaGo). Deep reinforcement learning algorithms such as PPO[14], DDPG[7], and TD3[5] are known to perform well in the reinforcement learning environments [6] and are easily modifiable to work with both discrete and continuous action spaces. This makes them a popular choice to use as a baselines.

However, such flexibility also introduces problems. For example, neural networks are uninterpretable to a human. This means that it would be difficult for an expert to understand the reasoning behind the model. Therefore, it would also be difficult to gain additional insights and decision making intuition for a given environment. Because of that, experts cannot verify the correctness of a model: whether the agent is basing its actions on relevant features or how it would react to unexpected conditions. Such limitations cause safety problems, which might be a serious concern in environments such as healthcare robots or autonomous vehicles. The solution for the interpretability problem might be learning a decision tree policy, with a limited number of nodes. Such a policy would be easily verifiable by a human, but it can perform worse than the neural network policy.

This research aims to learn interpretable decision tree regressor models with a DAGGER algorithm [13], which would imitate the neural network policy in a given environment. That is a form of imitation learning, where we treat the neural network as an expert policy and learn model to closely match the expert's actions. The goal of the research is to check whether such a decision tree policy does provide comparable results to its expert policy (neural network). The aim can be summarised as the following research question:

Does using DAgger with a decision tree regressor provide comparable results to the neural network approach for continuous action space problems?

The following subquestions will help to motivate the answer:

- *How does increasing the number of DAGGER iterations impact the results?*
- *How does limiting the decision tree nodes affect performance?*
- *How does observation space dimensionality impact the trained policy results?*

In other words, experiments would investigate how limiting maximum depth, changing environment dimensionality, and increasing the size of the training dataset influence the performance and interpretability of a policy.

The structure of the paper is as follows: Section 2 describes the problem and motivation for the research. Section 3 mentions methodology by introducing the DAgger algorithm and relevant definitions. Sections 4 and 5 provide a detailed description of the experiments performed, including a setup and discussion of the results. The Responsible Research section focuses on the reproducibility of the results and ethical concerns. Then, the future work section discusses what aspects can be improved and researched next. Finally, the conclusion section summarises the paper.

2 Background

With growing interest in reinforcement learning, safety has become one of the most important metrics in assessing the model's performance. It is because such models are a popular choice in areas where making a mistake can have serious consequences. Those environments include self-driving cars, healthcare robots, and air traffic control. As mentioned in the research by Osbert Bastani *et al.* [1], all those cases require agents to satisfy some safety properties (e.g. dealing with unexpected human behaviour) to prove that it is possible to use them in real-world situations.

Interpretable reinforcement learning aim is to train models that can be easily explained. By doing so, a human expert can understand the decision making process and therefore also assess the safety of a model. Decision trees are a common choice for Interpretable reinforcement learning models. The research by Rok Piltaver *et al.* [8] demonstrates that the size of a tree (number of leaves) is one of the most important parameters in a tree's comprehensibility. Because of that, this research also uses decision trees with a limited number of leaves.

Imitation learning algorithms such as SMILe [11], VIPER [1], DAgger [13], and AggreVaTe [12] have proven to be effective for training the reinforcement learning policies. However, among these, only Osbert Bastani *et al.* [1] addresses the interpretability of resulting policies. Furthermore, existing research rarely focuses on continuous action space environments. It is a serious concern, since many of the previously mentioned scenarios involve continuous actions. This highlights the importance of studying interpretable reinforcement learning in continuous action space settings.

3 Methodology

DAgger is an iterative algorithm for training deterministic policies. It was firstly introduced by Ross et. al. [13] and it was proven to achieve results comparable to state-of-the-art algorithms such as SMILe [11] and SEARN [4]. DAgger is a form of online imitation learning algorithm. Which means that it needs an expert policy that it is trying to replicate. In this research, the TD3 [5] model is used as an expert policy and Gymnasium environments [2] described in Section 4.1 are used for evaluation.

The DAgger can be summarised as follows, at each iteration we train the policy on the dataset of (state, action) pairs, where actions are given by the expert, but the states we visit depend on the current policy. That is, we start with an empty dataset D and sample k trajectories, using an expert policy. Then, we add those to the dataset D and learn a decision tree based on those states and expert actions. After initialisation, we then repeat the process, but now the set of trajectories T_i is sampled using the expert policy π^* with

$D \leftarrow \varnothing$
$N \leftarrow$ number of DAgger iterations
$T \leftarrow$ number of trajectory rollouts
for i in N **do**
    $\pi_i = \beta_i \pi^* + (1 - \beta_i)\hat{\pi}_i$
    for j in T **do**
        $T_i^j \leftarrow$ sample trajectory with π_i
        For each state s in T_i^j add $(s,\pi^*(s))$ to D_i
    end for
    $D \leftarrow D \cup D_i$
    Learn $\hat{\pi}_{i+1}$ on D
end for
return best $\hat{\pi}_i$ on validation

Figure 1: DAgger algorithm used to train the decision tree policies

probability β_i and the latest trained policy π_i otherwise. The pseudo code for this algorithm is shown in Figure 1.

Beta parameter can be treated as the learning rate. The practice shows that setting $\beta_i = I(i = 1)$ performs best [13], but also an exponential decay formula such as $\beta_i = p^i$ for $p < 1$ can be used. For the sake of simplicity, we decided to use the first version in the experiments.

Since the aim of the research is to train an interpretable policy that would work for continuous action space environments, a decision tree regressor was chosen as a suitable solution. To keep the policy interpretable and to avoid overfitting, the maximum number of leaves was chosen for those trees, which is explained in more detail in Section 5.

4 Experiments Setup

This section describes the experiments conducted and explains the setup required to run them. Additionally, it discusses the environments in which the policies were evaluated.

4.1 MuJoCo environments

Because of its popularity, OpenAI Gymnasium API [2] was used in the research. Since one of the reasons to have interpretable models is safety in robotics, real-life physics simulation environments (MuJoCo) were chosen. Those three environments are described in the table 1

In the **Inverted Pendulum**, the goal is to keep the pole on the cart upright. The agent is allowed to move the cart left or right with any force between -3 and 3. Reward is the number of timesteps that the agent managed to keep the pole in a standing position. The angle between the pole and the cart should be within $\frac{\pi}{2} \pm 0.2$ radians. Once this angle gets outside of the given range, the pole falls and the simulation ends. The observation space consists of: cart position and velocity, angle between the pole and the cart, and angular velocity of the pole. Default number of steps (so also the maximal reward) is a thousand, but to better distinguish policies, in this research, 10,000 steps were used. Implementation of the TD3[5] model from the stable-baselines3 [10] achieves this perfect score with a standard deviation of 0 on 100 validation samples.

Name	Observations	Action space	Description
Inverted Pendulum	4	1	*Keep the pole on the cart upright for as long as possible*
Swimmer	8	2	*Move the worm forward by applying torque to the joints*
Hopper	11	3	*Agent has to jump forward as far as possible in a predefined number of timestamps*

Table 1: Environments used in the research

Swimmer is an environment where the agent controls a worm like robot that tries to move forward by applying torque on its rotors. At each timestep agent is getting a reward which directly corresponds with how much forward it was able to move since the last measurement. Observation space consists of the angle and angular velocity of the three main segments of a robot (which are the tip, first, and second rotors), and the position of the robot (in x and y axis). The possible actions are the torque applied to the first and the second rotor. Environment terminates after a thousand timesteps, and the final reward is the sum of all received rewards throughout the simulation. The TD3 [5] model trained with stable-baselines3 [10] and RL-zoo3 [9] achieves a score of 354, which is considered a good score in this environment.

Hopper shares some similarities with both previous environments. Hopper is a robot that resembles a human leg and moves by jumping. The goal is to jump as far as possible in a fixed number of timesteps. Hopper consists of 4 parts: torso, thigh, shin, and foot, and the agent controls it by applying torque to the joints. Simulation terminates either after reaching a thousand timesteps or when the environment status is unhealthy. The second scenario usually occurs when the robot has fallen, which is checked by two factors: the angle between the torso and the ground is bigger than 0.2 radians, or the height of the robot is less than 0.7m (it has collapsed). Expert policy, same as before, TD3 [5] model trained with the stable-baselines3 [10] and RL-zoo3 [9], achieved a score of 3766 points. The observations are: z coordinate of the torso, angles of the four body parts together with their angular velocities, and x and z velocities of the torso.

4.2 Experiments conducted

There are three research sub-questions, which were formulated in Section 1. To answer them I decided to do the following experiments. For the first sub-question *How does increasing the number of DAgger iterations impact the results* fixed size (max number of leaves) decision tree policy was trained with DAgger[13]. Since it is an online learning algorithm, it is possible to evaluate the current policy at each iteration.

The next two research sub-questions are *How does limiting the decision tree nodes affect performance* and *How does observation space dimensionality impact the trained policy results*. They were answered by training decision tree policies with different sizes on different environments and comparing them with the expert policy (which was treated as a benchmark). Results for this experiment are discussed in sub-section 5.2

It is important to note the DAgger modification that was introduced in the experiments. In the original implementation, as can be seen in the Figure 1, each iteration lasts a fixed number of rollouts. However, for some of the environments, such as the Inverted Pendulum

or Hopper, the number of steps might differ significantly between the runs. This is because simulation can be terminated when the environment has an unhealthy status (in the case of the Inverted Pendulum it means that the pole has fallen off the cart). Therefore, the size of a training dataset grows inconsistently among the iteration, which has a direct impact on the learning curve plot. For bad policies, the performance will stay low longer, because the model will get much less samples from the expert to fix its mistakes. To counter this issue, each DAgger iteration is now terminated after a fixed number of new samples have been added to the training dataset. Such an approach guarantees that the iterations are much more similar to each other (e.g., they complete in the same amount of time). Moreover, it is now also possible to estimate the training dataset size based on the iteration number (the default value is 5000 new samples per iteration).

5 Results

The section starts with the difficulties encountered during the experiments in environments that check for healthiness (Inverted Pendulum and Hopper). Afterwards, it shows the dependency of the policy size on the performance.

5.1 DAgger in Early-Termination envrionments

The first experiment is to train a decision tree with a fixed maximum amount of leaf nodes and analyse the learning curve of such a policy. The aim was to determine how the number of iterations (affecting the size of a training dataset) influences the model's performance. Results for Inverted Pendulum and Hopper environments are shown in Figure 2 and for the Swimmer environment in Figure 3.

First observation is that the Inverted Pendulum plot shows significant performance drops, has high variance and no general trend, which is not typical for the standard machine learning models. The same behaviour can be observed for the Hopper. In contrast, the Swimmer shows only one drop occurring after the first iteration, which may be due to the small training dataset. Because of this unpredictability, it was decided to investigate the problem.

Performance in the Inverted Pendulum environment is measured by the number of timesteps the agent manages to keep the pole upright on the cart. At any point when the pole has an angle of more than 0.2 radians (around 11 degrees), the status of the environment is marked as unhealthy and the simulation stops. A similar early termination scenario might occur in the Hopper environment. Once a robot collapses on the ground (which might be caused by a bad landing), the episode ends, and the agent cannot get any more rewards. Such a setup makes some states much more important than others. For instance, in the Inverted Pendulum, states with almost falling pole are crucial for a policy performance, since the agent would not have time to recover (simulation stops). However, a decision tree policy that optimises the mean squared error of the predictions, treats all the samples equally, which means that those critical states are not emphasised during the training. Possible solutions to this problem are described in Section 7.

Figure 4 shows two policies for the Inverted Pendulum, one that achieves a perfect score of 10,000 and one that appeared a few iterations later with a score of 597. We can see from the structure of those trees that the policies are very similar and both of them got comparably low mean squared error on the training dataset. However, since DAgger uses last policy (high performing one) to sample the trajectories, it does not reach critical states. It is because the policy achieving a perfect score manages to keep the pole upright

throughout the whole simulation. Therefore, the critical states might be underrepresented, and the policy trained on that dataset would make an incorrect prediction when they are encountered. Another observation that supports this reasoning is that the agent gets back to high performance after each drop. Since a bad-performing policy takes over the trajectory sampling task, it visits many more states that lead to an early termination, thus equalising the importance of critical states in the dataset. Important to note is the fact that DAgger returns best best-performing policy on the validation set and not the last one found. Which means that once a policy gets a high score, subsequent low scores would not affect the final result.

Looking at both Figures 2 and 3, we can see that the learning process highly depends on the training environment. Despite those differences, increasing the training dataset size does help to find better policies. For example, in Figure 2 for both Hopper and Inverted Pendulum, there is a significant difference in the best policies found after 5, 10, and 20 iterations. For the Swimmer environment, shown in Figure 3, the improvement is smaller but still present. Moreover, results for the Inverted Pendulum and Swimmer demonstrate that once a policy that closely replicates an expert has been found, there is no need to search further. The chance of improving is negligible which can be observed at Figure 3 where the performance plot stays flat after reaching a score of about 340.

5.2 Impact of the tree size on the performance

Second set of experiments was meant to measure how the number of leaf nodes would affect the policy results. For that, many policies with different leaf limits have been trained. The results for the Hopper environment are shown in Figure 5, for the Inverted Pendulum in Figure 6, and for the Swimmer in Figure 7. The red line indicates the expert performance. The light blue region indicates the standard deviation of the results on the validation dataset of a hundred samples.

Firstly, Figure 5 demonstrates that it was not possible to replicate an expert policy with a decision tree. An expert got a score of 3,766, while the trees with a small size (fewer than 50 leaves) did not get a 1,000. Even uninterpretable trees with a thousand leaves were only able to achieve a score of around 3,000. Such a bad performance may be caused by the observation space dimensionality, as the Hopper had the most observations among all the testing environments. However, early termination might have also increased the difficulty of training the model. Thus, interpretable policies for environments with more observations may exist.

Results for the Inverted Pendulum and the Swimmer environments, despite the differences between them, described in Sections 3 and 5.1, share many similarities. First of all, trees with fewer than 10 leaves do not perform well, and in both cases minimal tree that manages to replicate the expert is at 12 leaves. Moreover, increasing the tree size does in general increase the policy score, which is also true for the Hopper.

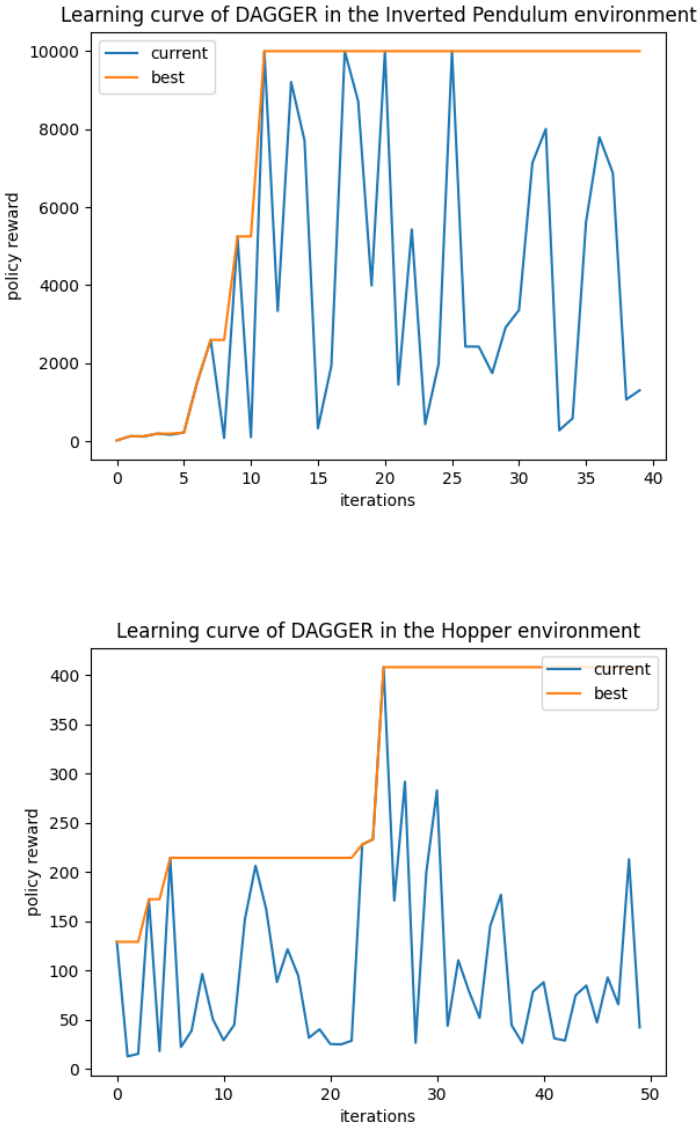


Figure 2: Learning curve of a decision tree in the Inverted Pendulum (upper plot) and Hopper (lower plot) environments. The curves show significant performance drops, caused by the early termination.

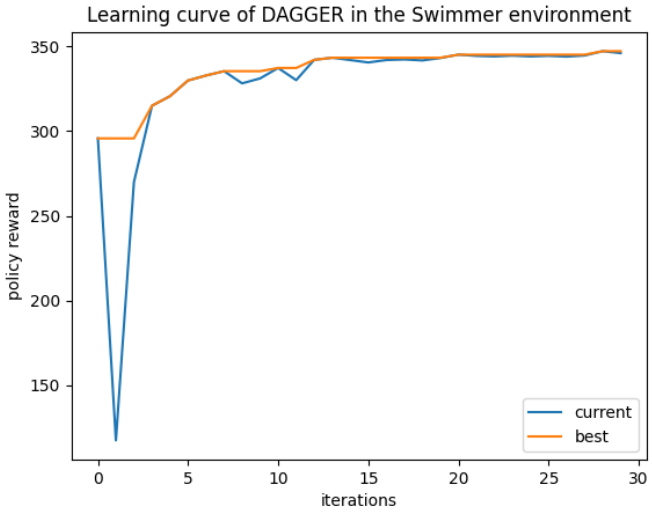


Figure 3: Learning curve of a decision tree in the Swimmer environment. The curve does not show consistent performance drops because the environment does not have the early termination property.

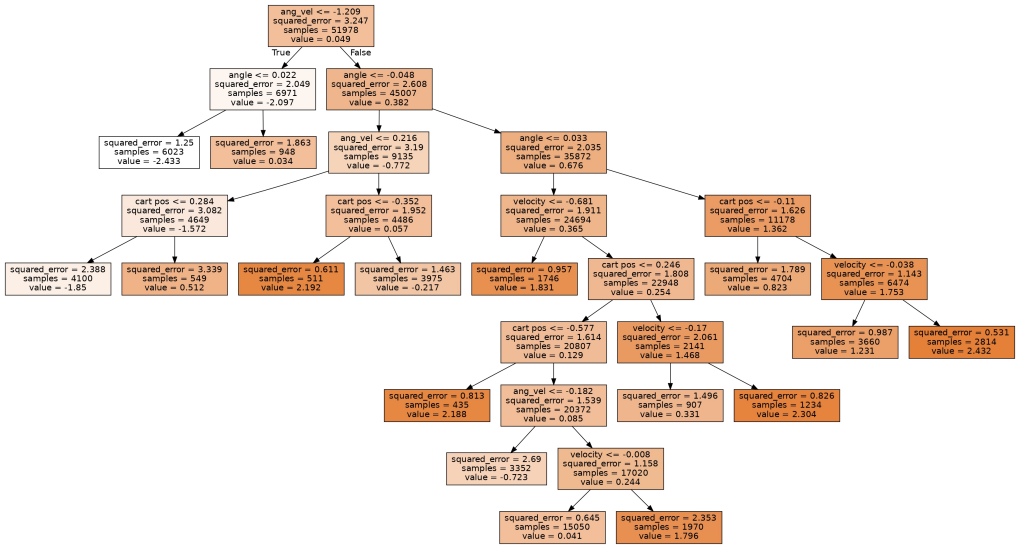
It can be concluded that despite the Swimmer environment having twice as many observations and actions as the Inverted Pendulum environment, a comparable tree size is needed to replicate the expert policy. Moreover, those trees, because of their small size are possible to interpret. Figure 4 shows examples of trees with 15 leaves produced by the DAgger for the Inverted Pendulum. Such conclusions, combined with the findings of Section 5.1, prove that using DAgger with the decision tree regressor as a policy can provide comparable results to the neural network approach. Although for the more complex environments, with bigger observation space dimensionality (such as Hopper), it might not be possible.

6 Responsible Research

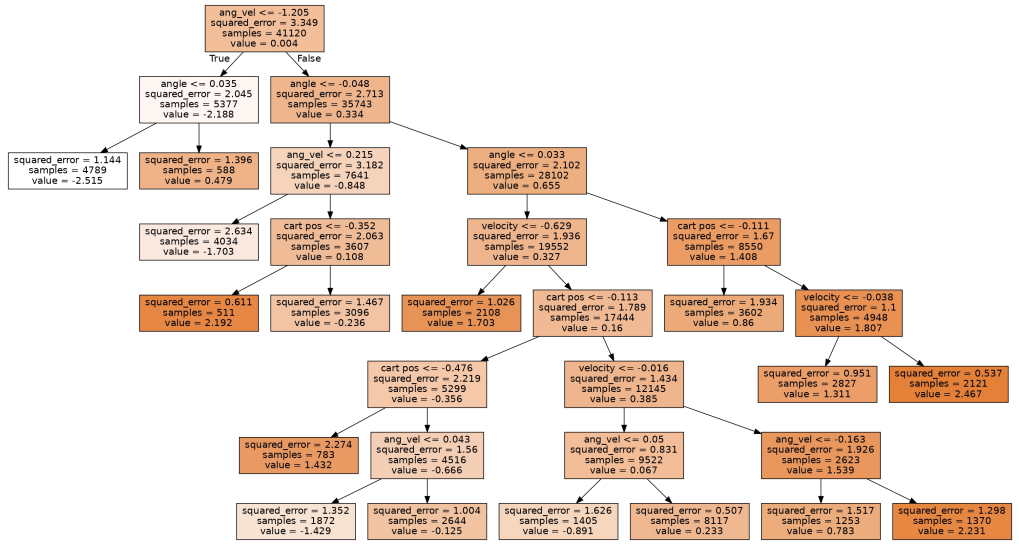
This section focuses on the steps that were taken to ensure the reproducibility of the results. Additionally, it highlights the potential dangers associated with using the proposed approach to train interpretable reinforcement learning models.

6.1 Reproducibility

One of the main challenges in every research is to ensure the reproducibility of the results. All frameworks used in the research are publicly available. Those include: Open AI Gym[2], stable-baselines3[10], RL-zoo3 [9]. Details about their usage can be found in Section 3. Moreover, code written for this research is also published as a GitHub repository [3]. All sources of randomness (including environment, decision tree construction, and number generators) were seeded to enable producing the same results as those presented in the paper.



(a) Poorly performing policy



(b) Well performing policy

Figure 4: Comparison of the two decision tree policies produced by the DAgger. The lower one is performing significantly better than the higher one, despite having a similar structure. More orange colour in the nodes indicates higher impurity.

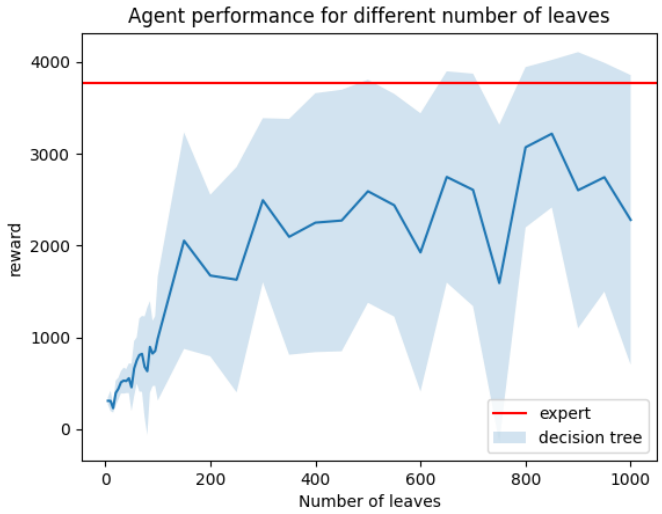


Figure 5: Agent performance for trees with different numbers of leaves in the Hopper environment. Trained policy does not achieve results comparable to an expert, even with the big decision trees.

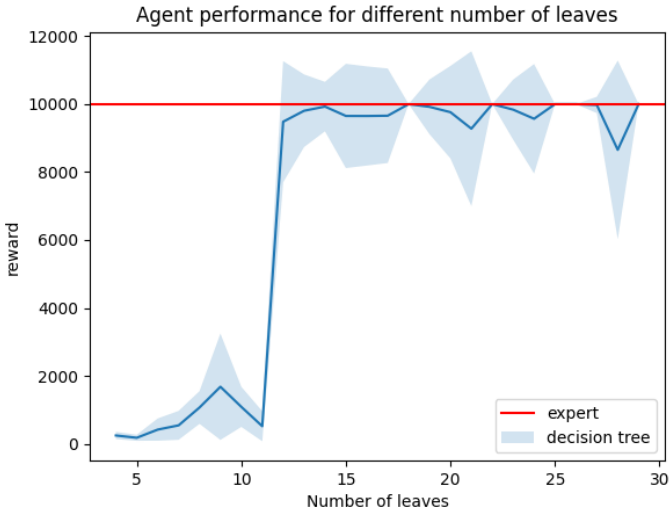


Figure 6: Agent performance for trees with different numbers of leaves in the Inverted Pendulum environment. Small decision trees achieve an expert level of performance.

Figure 7: Agent performance for trees with different numbers of leaves in the Swimmer environment. Small decision trees achieve an expert level of performance.

6.2 Ethical concerns

Another issue that has to be addressed in this research is ethical concerns. As described earlier, one of the use cases is environments where an agent's mistake can have severe consequences. For example, an autonomous driving model in an unexpected situation should act in a way that ensures the passengers' safety.

However, Section 5.1 describes the problems that might occur in such environments. That is, during the training process, crucial states (in the Inverted Pendulum case, the states where the pole was almost falling off the cart) might be underrepresented, thus resulting in a much worse-performing policy. Such models are unacceptable to use in real-life situations where a human life can depend on them.

Moreover, the research also highlighted the issue with the interpretability of the models. Decision trees that have almost identical structure (an example of such trees can be found in the Figure 4) can yield different performance scores. Thus, a human expert reviewing the final model is prone to make an error and classify a dangerous policy as safe to use. Such problems show that interpretability cannot be the only factor to prove the policy's safety.

7 Future Work

One of the main problems encountered during this research, was sudden performance drops in the training process. Such behaviour was only observed for the environments with the possibility of early termination (Inverted Pendulum and Hopper). The main reason for this is that the agent was unable to replicate the expert policy correctly in crucial states (those that lead to early termination). A detailed explanation can be found in Section 5.1. Unpredictability in the learning curve (no visible trend) makes it harder to estimate the number of iterations needed to train the model.

The primary area for improvement would be to emphasise the importance of the states that have a bigger impact on the performance score. To do so, usually the Q function is used. Examples of such algorithms are AggreVaTe [12] and VIPER [1], which are both DAgger [13] modifications.

AggreVaTe at each rollout follows the current policy π_i until some time $t-1$ (which is randomly chosen), then takes exploration action and evaluates the cost-to-go of that action by following the expert policy from the time $t+1$ until the end of the simulation. Afterwards, it trains a cost-sensitive model, where the weights are the cost-to-go values. By doing so, crucial states have a significantly bigger impact on the resulting policy.

VIPER follows DAgger in the trajectory sampling procedure, but for each of the pairs of state and action, it also calculates the corresponding Q-value. The main difference between the two algorithms is that the VIPER learns the next policy on a smaller subset of the original dataset. In the resampling process, it prioritises states based on their Q-value, so the states where making a mistake matters the most have a higher probability of being chosen.

Both AggreVaTe and VIPER might solve an issue of unpredictable learning curve behaviour for the Inverted Pendulum and Hopper environments. Moreover, the research by Osbert Bastani *et al.* [1] shows that the decision trees produced by VIPER are significantly smaller than the ones created by DAgger, which can result in an interpretable policy for the Hopper environment.

8 Conclusion

The research aimed to study interpretable models in continuous action space environments. For that, we used a DAgger [13] algorithm to train the decision tree policies, which were then evaluated on three MuJoCo [2] environments. Since DAgger is an imitation learning algorithm (that is it tries to replicate an expert policy), the main research question was to answer whether it is possible to achieve performance comparable to an expert policy.

Two of the testing environments had an early termination property - once the status of the environment was marked as unhealthy (e.g., a robot collapsed to the ground), the simulation stopped. We found that in those envrionments, it was common for a DAgger to experience significant performance drops between iterations. That is, after a good-performing policy has been found, it is possible that the next policy would achieve a much lower score. The investigation revealed that the source of the problem was the underrepresentation of critical states in the training dataset. An agent must make a correct decision in states where a mistake can lead to the end of the simulation. Such behaviour is a serious limitation of the DAgger algorithm, as it undermines the safety of the produced policy. DAgger modifications such as AggreVaTe [12] or VIPER [1], which emphasise critical states in the training process, might solve this problem.

Furthermore, results have shown that the dimensionality of the action and observation spaces affected the model's performance. In environments with 4 and 8 observations (1 and 2 actions, respectively), a decision tree with fewer than 20 leaves was able to achieve results comparable to those of an expert. On the other hand, in an environment with 11 observations and 3 actions, even trees with 1,000 leaves were unable to replicate an expert policy.

To summarise, deep neural network models can be replicated with small decision trees. Since a human expert can easily interpret those trees, they can review the decision-making

process and potentially identify unsafe behaviour. Therefore, it is possible to use the presented approach for improving the overall quality of existing models. However, DAgger does not prioritise critical states during training, which might result in a policy making a wrong decision in a crucial situation. Thus, further research is needed on algorithms that address this issue, such as AggreVaTe [12] or VIPER [1].

References

[1] Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[3] Tymon Cichocki. Interpretable RL research project. `https://github.com/G00DC0D3R/Interpretable_RL_Research_Project`, 2025.

[4] Hal Daumé, John Langford, and Daniel Marcu. Search-based structured prediction. *Machine learning*, 75:297–325, 2009.

[5] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1587–1596. PMLR, 10–15 Jul 2018.

[6] Jino Jayan and Lal Priya P. S. Enhancing robotic control: A td3-based approach for planar continuum robots. In *2024 1st International Conference on Trends in Engineering Systems and Technologies (ICTEST)*, pages 1–6, 2024.

[7] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[8] Rok Piltaver, Mitja Luštrek, Matjaž Gams, and Sanda Martinčić-Ipšić. What makes classification trees comprehensible? *Expert Systems with Applications*, 62:333–346, 2016.

[9] Antonin Raffin. Rl baselines3 zoo. `https://github.com/DLR-RM/rl-baselines3-zoo`, 2020.

[10] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[11] Stephane Ross and Drew Bagnell. Efficient reductions for imitation learning. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 661–668, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[12] Stephane Ross and J. Andrew Bagnell. Reinforcement and imitation learning via interactive no-regret learning, 2014.

[13] Stephane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 627–635, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

[14] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.