On the Performance and Robustness of Linear Model U-Trees in Mimic Learning

Matthew Green
Department of Computer Science
Johns Hopkins University
Baltimore, MD 21218
mgree164@jhu.edu

John W. Sheppard
Gianforte School of Computing
Montana State University
Bozeman, MT 59717
john.sheppard@montana.edu

***Abstract*—The Linear Model U-Tree (LMUT) has been used to increase the interpretability of Deep Reinforcement Learning (DRL) agents by mimicking behavior in terms of Q-value predictions and gameplay. In this paper, we consider two extensions to LMUT. First, we evaluate the impact of prepruning and bottom-up postpruning on LMUT and find that while prepruning has a mixed to negligible impact on performance, postpruning brings its Q-value predictions closer in line with the DRL agent, increasing the effectiveness of its influence on DRL interpretability. Second, we find evidence that LMUT gameplay typically more closely matches that of the DRL agent it learns to mimic when the DRL agent policy is more robust to noise, even after controlling for the performance of the DRL agent on the underlying task. This indicates that LMUT efficacy is driven in part by the robustness of the DRL policy.**

***Index Terms*—Linear Model U-Tree, Deep Reinforcement Learning, Mimic Learning, Tree Pruning**

I. Introduction

Deep Reinforcement Learning (DRL) [1] expands the state, action, and reward space on which a reinforcement learning agent can efficiently learn an optimal policy by using a deep neural network (DNN) to approximate expected cumulative rewards. By incorporating a DNN, researchers have been able to identify solutions to complex reinforcement learning tasks [2] but often at the expense of interpretability, raising potential issues in usefulness, trust, ethics, safety, and accountability [3]. Thus, there is a strong motivation to help make DNN predictions more interpretable.

To address these interpretability concerns, Liu *et al.* introduced a model that learns to mimic the predictions of a DRL agent in a more interpretable fashion using Linear Model U-Trees (LMUT) [4]. They train LMUT under two conditions: Experience Training (where LMUT and DRL train simultaneously) and Active Play (where LMUT trains after DRL completes its training). They judge the performance of LMUT in terms of how well it can match the DRL Q-value predictions (fidelity) and how well it can perform the original task (gameplay). They find that LMUT achieves competitive fidelity to DRL, and while DRL outperforms on gameplay, LMUT achieves strong performance while offering greater interpretability. They suggest that future research consider the impact of pruning on performance and interpretability.

A. Research Questions and Hypotheses

Here we pose two research questions. First, what impact do prepruning and postpruning have on the performance of LMUT trained to mimic the behavior of a DRL agent? Performance will be measured in terms of both fidelity and gameplay. Prepruning will take the form proposed by Liu *et al.* [4]: new node splits occur only when the differences between Q-value predictions and DRL Q-values at the original leaf exceed a minimum error threshold. A form of bottom-up postpruning is implemented where, after building the LMUT, we gather validation data and test whether any combination of leaves experience greater error than their common ancestor. If so, the leaves are pruned.

We hypothesize that prepruning will not impact performance while postpruning will increase performance on both measures, which is consistent with prior work on pruning and decision tree learning [5]. We tested this hypothesis using data gathered after training. Performance was compared across all combinations of pruning techniques using bootstrapping.

The second research question asks to what extent LMUT performance is influenced by the robustness of the policy learned by the DRL agent. We define robustness according to the theory of robust control, whereby a more robust policy is one that performs better when noise is introduced [6]. To measure robustness, for each reinforcement learning task, we trained two DRL agents over a different number of episodes but with the same target gameplay performance. We then tested robustness by introducing noise into the observations during testing and measured the amount of performance degradation. We then trained a separate LMUT agent on each DRL agent and evaluated the LMUT's gameplay performance.

We hypothesize that the LMUT agent that mimics the more robust DRL agent will outperform the agent that mimics the less robust agent in gameplay, given the policy mimicked by the former should be more capable of handling noise that may be introduced through the LMUT training process. Once again, performance was compared using bootstrapping.

In addressing these questions, we considered LMUT learning under only Active Play, as this condition performed better than Experience Training in [4]. It also needed less memory and required no access to the the DRL model itself.

B. Motivation

This work is motivated by the goal to improve the interpretability of the DRL agent behavior based on the LMUT mimic model. DRL has exhibited strong performance on a variety of games [1], and the technology is potentially applicable in impactful areas such as autonomous driving and healthcare. But the lack of interpretability serves as an obstacle to DRL's deployment in such real-world applications [7].

Furthermore we wish to understand the drivers of LMUT performance relative to DRL better. If as we hypothesize LMUT performance is related to the robustness of the DRL agent's learned policy, this has important implications for when LMUT action interpretations can serve as reliable proxies for action interpretations of the DRL agent.

We chose to focus on LMUT because it is "the first work that extends interpretable mimic learning to Reinforcement Learning [4]," and because it achieved strong performance in fidelity and gameplay compared to the baseline mimic models tested. With mimic learning, the goal is to use the interpretable predictions of one model as a proxy for the less interpretable predictions of another. One necessary condition for mimic learning to be effective is that the two models behave similarly.

Our goal is to measure the extent to which different pruning techniques are able to align the behavior of the LMUT agent with that of the DRL agent, as well as to determine whether the robustness of the DRL policy influences the similarity between DRL and LMUT gameplay. If our pruning hypothesis is validated and employing postpruning brings about the best performance alignment, the interpretations associated with the postpruned LMUT will be more effective as proxy interpretations for the DRL agent. If our robustness hypothesis is validated and the more robust DRL policy results in a better aligned LMUT gameplay, future practitioners will have a better understanding of when LMUT interpretations will be able to serve as reliable proxies for DRL interpretations.

C. Contributions

Our contributions can be summarized as follows.

1) We evaluated the impact of prepruning and our proposed postpruning technique on the fidelity and gameplay of LMUT trained to mimic DRL agents on three puzzles. We found that prepruning had a mixed to negligible effect according to both measures, and that postpruning had a positive to negligible effect on fidelity with a mixed to negligible effect on gameplay.
2) We evaluated the impact of the robustness of the DRL policy on LMUT gameplay and found that most of the time, LMUT gameplay increased with the robustness of the DRL policy. The robustness of the DRL policy played a significant role in the efficacy of LMUT gameplay in certain circumstances, as evidenced by the results in two puzzles.

The remainder of this paper is organized as follows. Section II summarizes related work in decision tree pruning, mimic learning, and robust control. Section III describes pertinent background information on reinforcement learning and LMUT learning. Section IV presents our approach for testing our hypotheses. Section V presents the results of our experiments, followed by a discussion of those results in Section VI. Section VII presents areas for future research, and Section VIII presents our concluding remarks.

II. Related Work

A. Decision Tree Pruning

Quinlan evaluated the impact of different postpruning techniques on the interpretability and accuracy of decision trees [8]. The goal was to make the decision tree smaller, and thereby more interpretable, without significantly impairing its accuracy on the underlying classification task. After applying three pruning techniques to six datasets across various domains, Quinlan found that, while the size of the tree was reduced, typically the accuracy of predictions on unseen data also improved. The improvement in accuracy was likely due to the fact that decision trees can overfit the data they are trained on, and by pruning leaves that underperform on predictions made on held-out test data, one can help make the tree generalize better. Schaffer argued that pruning is not inherently beneficial to decision tree accuracy, and that instead the question of whether pruning can be successfully applied depends on if it is correct to assume that simpler trees will generalize better for a particular problem [9]. Due to the novelty of LMUT, we cannot rely on existing work to determine whether the resulting decision trees have a propensity to overfit the data, and our hypothesis that postpruning will enhance performance is premised instead on the finding that such an effect has been found when working with decision trees more generally.

Leiva *et al.* introduced the Minimum Surfeit and Inaccuracy (MSI) algorithm for growing a decision tree with prepruning, where the choice over whether or not to preprune depends on the Kolmogorov complexity [10]. This approach factors in both the inaccuracy and complexity of the tree. They found that applying MSI resulted in reduced computational complexity and a smaller tree when compared to benchmark decision tree algorithms, and that while accuracy did suffer, the impact was not substantial. We are testing the impact of prepruning on LMUT, yet in light of findings such as with MSI, we do not expect performance to be enhanced.

B. Mimic Learning

Mimic learning is a knowledge distillation process whereby one aims to mimic the predictions of a complex and potentially difficult to interpret model with another more interpretable one. Che *et al.* used Gradient Boosting Trees to increase the interpretability of deep learning models applied in the healthcare domain for making phenotype predictions [11]. Liu *et al.* developed LMUT to mimic the predictions of DRL agents so that the decision process is easier to interpret [4]. With mimic learning, the assumption is that the closer the performance of the mimic model is to the original, the more applicable the corresponding interpretations are. We aim to

evaluate the impact of prepruning, postpruning, and DRL robustness on the performance gap between the LMUT agent and the DRL agent it mimics.

C. Robust Control

Robust control is a subset of control theory that focuses on how to design a system to withstand uncertainty, for example as a consequence of noise in the environment. With robust control, the more noise a system can withstand while functioning properly, the more robust that system is. Dorato presents an overview of various approaches to quantifying and solving robust control problems [12].

The notion from robust control that robustness is associated with successful system functioning in the presence of noise translates well to the reinforcement learning domain. If two agents $\mathcal{A}$ and $\mathcal{B}$ perform the same on a task under perfect conditions, but $\mathcal{A}$ performs better than $\mathcal{B}$ after noise is added to the environment, then by robust control theory, $\mathcal{A}$'s learned policy is more robust. We hypothesized that the more robust DRL policy would have a demonstrably positive impact on LMUT gameplay performance because the policy LMUT is learning to mimic would be better able to withstand noise that may be added through the LMUT training process.

III. Background

A. Reinforcement Learning

Reinforcement learning is a machine learning framework where an agent learns to maximize expected cumulative rewards through a combination of observing the state of its environment s, interacting with its environment through actions a, and receiving rewards r [13]. The choice of action to take from each state is called a policy π, and reinforcement learning aims to identify the optimal policy π^* associated with maximum expected cumulative rewards. The expected cumulative discounted rewards associated with taking a_t from s_t and following π thereafter is described by the Q-value: $Q(s_t, a_t) = \mathbf{E}_\pi(\sum_{k=0}^{\infty} \gamma^k r_{t+k})$, where γ is a discount rate.

When everything an agent needs to know to estimate the expected cumulative rewards of a particular action a_t from a particular state s_t accurately is known in the current state, the state is described as being Markovian. When all states are Markovian, the reinforcement learning problem follows a Markov decision process (MDP). When working with an MDP, the problem can be modeled mathematically, and when the number of possible states, actions, and rewards in an MDP are finite, after infinite experiences one is guaranteed to find a policy that is optimal [14]. When the state, action, and reward spaces are small, this can often be accomplished in finite time by deriving an optimal value for every possible state-action combination. In situations where the state-action space is large, it is often not feasible (due to computational time and/or resource constraint) to derive optimal values for all state-action combinations. In these instances, function approximation is often employed to approximate the value of state-action pairs, such as through the incorporation of a DNN in DRL [1].

B. Linear Model U-Trees

McCallum introduced the U-Tree, an online reinforcement learning algorithm that explicitly addresses the problem of handling hidden states [15]. With U-Trees, the reinforcement learning agent learns an internal representation of the state-action space by mapping discrete observations to state or state-action values. U-Trees encompass both feature selection and short-term memory, helping manage the dual related problems in reinforcement learning of too much information and not enough information, while at the same time embodying the interpretability feature of decision trees [16].

Uther and Veloso introduced the continuous U-Tree (CUT), an extension of the U-Tree that can be applied to continuous state features by automatically discretizing the input signal [17]. CUT is a binary tree where each node corresponds to an area of the observed state space. Each internal node splits the state space in two, and each leaf node corresponds to a particular discretized internal state. To determine where to introduce new splits, CUT sorts the set of instances at each leaf according to each feature and evaluates the resulting distributions of data points that result from splitting. Once two such distributions are found to be statistically significantly different, a split is introduced. CUT learning uses dynamic programming to solve the MDP at each leaf, which is expensive computationally in both time and space.

Liu *et al.* introduced LMUT to extend CUT by allowing leaf nodes of the resulting binary tree to represent linear functions instead of constants [4]. The resulting tree is more expressive than CUT, allowing closer approximation to a continuous function, while also generating fewer splits, which could make the tree more interpretable [18]. With LMUT, each action has an independent binary tree structure, and each leaf on a given tree structure represents a particular discrete state for the decision process of that particular action. Each leaf makes a Q^{UT}-value prediction. The actions with the highest Q^{UT} values in each state define the policy.

The training process for LMUT with Active Play involves a data gathering phase followed by a node splitting phase. During data gathering, the LMUT learner has three components: an observation of the state-space I, a query function that selects an action a_t from I_t with ϵ-greedy where ϵ decays linearly, and a $\hat{Q}$-value generated by the trained DRL agent for a particular (I_t, a_t). LMUT observes transition T_t, the transition at time t, and identifies the specific leaf N of the tree structure associated with a_t to which I_t belongs. It then adds T_t to the transition set on N and proceeds to gather the data for the next transition, processing a total of B transitions in a single phase, where B represents the batch size. Initially, all tree structures have a single leaf to which every T with associated action in the initial batch is added.

Once the data gathering phase is complete, LMUT proceeds with the node splitting phase. For every leaf N and for every transition in the transition set $\mathbf{T}_N$ at leaf N, it updates the weight vector $\mathbf{w}$ via minibatch Stochastic Gradient Descent

(SGD) over E epochs with step size α according to

$$\begin{aligned} \mathbf{w} &\leftarrow \mathbf{w} + \alpha \nabla_{\mathbf{w}} L(\mathbf{w}), \\ L(\mathbf{w}) &= \sum_{t=1}^{m} 1/2(\hat{Q}_t - Q^{UT}(I_t|\mathbf{w}, a_t))^2, \end{aligned}$$

where m is the total number of transitions in $\mathbf{T}_N$. In the event that SGD is unable to bring about sufficient improvement of $L(\mathbf{w})$ on N, defined as being greater than or equal to a predefined threshold max_err, a new split is added. When the the node is not split, prepruning has occurred. For choosing a split, LMUT sorts the instances according to each feature, temporarily splits the transitions in two at the midpoint of every consecutive pair of sorted values, and computes the weighted average variance in $\hat{Q}$-values of the resulting two groups of transitions. The chosen combination of feature and splitting value is that which has the minimum weighted average variance. Each leaf then inherits the weights and associated transitions of its parent, and these weights are updated with SGD. LMUT then returns to the data gathering process to obtain the next batch of transition data and the process repeats. The hyperparameters E, α, and B are tunable.

Unlike the U-Tree, neither CUT nor LMUT incorporate short-term memory; therefore, both may suffer when working with tasks involving hidden state. The tasks we evaluate in this paper do not incorporate hidden state, so the lack of short-term memory will not pose a problem in our case.

IV. Approach and Experimental Design

A. Environment

For running our experiments, we trained DRL agents using Deep Q-Networks [1] to learn to play Cart Pole, Mountain Car, and Acrobat, three puzzles whose environments are simulated with the OpenAI Gym toolkit [19].

The goal of Cart Pole is to learn which of two actions—push left or right—to take in order to get a pole to stay upright on top of a cart. Cart Pole has a four-dimensional observation space—the position and velocity of the cart, and the angle and angular velocity of the pole—and earns a reward of $+1$ for each step other than when the poll falls down for a reward of 0. An episode ends after 500 steps or when the pole falls.

The goal of Mountain Car is to learn which of three actions—accelerate left, right, or do not accelerate—to take in order to get a car to reach the goal at the top of a mountain. Mountain Car has a two-dimensional observation space—the position and velocity of the car—and incurs a cost of -1 for each step or a reward of 0 at the goal. An episode ends after 200 steps or when the goal is obtained.

The goal of Acrobat is to learn which of three actions—apply -1, 0, or $+1$ torque—to take in order to get a chain to reach a target height. Acrobat's chain has two rotational joints and a six-dimensional observation space—cosine, sine, and the velocity of each joint angle—and incurs a cost of -1 for each step or a reward of 0 at the goal. An episode ends after 500 steps or when the goal is obtained.

B. DQN Training

Recall that we hypothesized that LMUT gameplay performance will improve as the policy learned by the DRL agent grows more robust, even after controlling for performance of the DRL agent on the underlying task. In order to generate two DRL agent policies with different degrees of robustness, we need to train our DRL agents such that the policies will grow more robust to noise over time. To do this, we use Averaged DQN [20], an extension of DQN where Q-value predictions are made using the average of the previous K learned weights. We set $K = 10$ because the work by [20] indicates that this results in a more stable training trajectory than the original DQN, as well as relative to other well known DQN extensions such as Double-DQN. Setting $K = 10$ appears to be consistent with hitting a point of diminishing returns in that work.

For all three puzzles, we trained DQN networks with 2 hidden layers and 128 hidden nodes in each layer. For learning the weights of the network, we used the AdamW optimizer with AMSGrad and a weight decay of 0.01. For ϵ (the exploration rate), we employed a dynamic ϵ-decay strategy whereby ϵ starts at 1 for every state in the puzzle and reduces by 0.001 for each visit to a particular state, with a minimum ϵ of 0.05. We set the batch size in the replay monitor to 128 and the decay factor γ to 0.99 for all three puzzles. During training, we evaluated performance after every 100 episodes using 100 test episodes where we fix ϵ to 0 and record the average reward. Based on several preliminary test runs, we predefined an average reward target of 500 for Cart Pole, -150 for Mountain Car, and -80 for Acrobat. Once the reward target is matched or exceeded, we save the model and continue to train in increments of 100 episodes until the model has trained for at least 1000 more episodes and the reward target is matched or exceeded, at which point we save a second version of the model. We set an initial learning rate of 0.0005, and for Cart Pole, we decayed this by 50% after each set of test episodes for which target performance was met or exceeded, leaving the learning rate unchanged otherwise, in order to reduce the model's tendency to overfit when learning this puzzle.

C. LMUT Training

To train the LMUT agent, we used the process described in Section III-B, with a few modifications. First, we noticed that when a new feature split is determined and the new leaf nodes inherit the weights of the parent node, this resulted in very poor performance. This makes sense, as these weights were derived using transition data, some of which were subsequently determined to be a better fit with a different part of the feature space. As a result, we decided to reset the weights to small positive random values and then perform SGD with only the transition data assigned to the new leaf node. This resulted in much better performance but did require setting an additional hyperparameter: a minimum number of instances required to evaluate for a potential split, as too few instances would result in poor SGD performance. We set this minimum to twenty times the number of weights (where the

TABLE I: Tuned and chosen LMUT hyperparameters (max_err = 0.00 implies no prepruning).

Problem	Training Episodes	max_err	B	α	E
Cart Pole	1200	0.066	1000	0.005	50
—	2200	0.019	1000	0.005	100
—	1200	0.000	1000	0.005	50
—	2200	0.000	1000	0.005	100
Mountain Car	2300	1950	500	0.01	100
—	3400	3926	500	0.01	100
—	2300	0.000	500	0.01	100
—	3400	0.000	500	0.01	100
Acrobat	1000	0.324	1000	0.01	100
—	2000	0.437	1000	0.005	50
—	1000	0.000	1000	0.005	50
—	2000	0.000	1000	0.005	100

number of weights is one plus the number of observation space features). With more time, we would have liked to tune this hyperparameter.

For LMUT training, Liu *et al.* [4] did not define how they chose what maximum error threshold max_err to use to determine whether a given leaf warrants splitting. We chose the following approach: first, we measured the median reward per episode (MRPE) with the fully trained DRL agent on 10 iterations over 20 episodes each. Next, we set $cutoff$ as the worst MRPE over the 10 iterations. We then continued to measure MRPE over 20 episodes while adding to the $\hat{Q}$-values a small amount of uniformly random noise centered on zero, where the degree of noise was defined by the span of the noise distribution range. Specifically, the degree of noise added began at 1 and increased by factors of 10. Once MRPE falls below $cutoff$, we divide the most recent degree of noise by 10 and increase the degree of noise in tenths until performance again drops below the cutoff. The degree of noise beyond which MRPE drops below $cutoff$ the second time was then used as the bounds for creating a uniform distribution of 100 randomly generated values that are each squared. The average of these 100 squared values then became our max_err threshold below which we decided to not split the leaf. The pseudocode for calculating max_err is given in Algorithm 1.

We tuned three hyperparameters using grid search: $B \in \{100, 500\}$, $\alpha \in \{0.005, 0.01\}$, and $E \in \{50, 100\}$. The batch sizes B were chosen to balance the number of instances needed to perform SGD against the number of batches needed for producing new feature splits. The learning rates α were chosen to balance speed against stability in SGD. The epochs E were chosen to balance having enough iterations for effective learning against the risk of overfitting. The resulting hyperparameters derived from tuning are shown in Table I.

We defined the more robust DRL policy as being that which continues to perform well in the face of a greater amount of noise. The magnitude of max_err is therefore a measure for the robustness of the DRL learned policy. By this measure, according to Table I, the longer trained DRL agents for Mountain Car and Acrobat learned a more robust policy as expected, but the reverse was true for Cart Pole.

Algorithm 1 Calculate max_err

$max\_err \leftarrow 0$
$cutoff \leftarrow \infty$
$n \leftarrow 20$
$err \leftarrow 0$
for $iteration = 1$ to 10 **do**
  $MRPE \leftarrow$ GetMRPE(n, err)
  if $MRPE < cutoff$ **then**
    $cutoff \leftarrow MRPE$
  end if
end for
$err \leftarrow 1$
while True **do**
  $MRPE \leftarrow$ GetMRPE(n, err)
  if $MRPE < cutoff$ **then**
    break
  end if
  $err \leftarrow err \times 10$
end while
$err \leftarrow err \div 10$
$cur\_err \leftarrow err$
while True **do**
  $cur\_err \leftarrow cur\_err + err \div 10$
  $MRPE \leftarrow$ GetMRPE(n, cur_err)
  if $MRPE < cutoff$ **then**
    $err \leftarrow cur\_err - err \div 10$
    break
  end if
end while
for $iteration = 1$ to 100 **do**
  $max\_err \leftarrow max\_err + (err \times (\text{Rand}(0, 1) - 0.5))^2$
end for
return $max\_err \div 100$

D. Tree Postpruning

For postpruning the LMUT trees, we carry out the following: during training, we maintain the state of the linear model at each node, even if the node is later split. After training completes, we drop all original transition data and assemble a new set of transition data in the same fashion as during training, only now we assign each new transition to each node encountered from root to leaf according to the feature splits. We then evaluate the performance at each node defined as $\sum_{t=1}^{m}(\hat{Q}_t - Q_t^{UT})^2$, where m is the number of new transitions at the given node. Next we compare the cumulative performance of each set of descendant leaves to their common ancestor, starting with the youngest common ancestor and moving recursively towards the root node. In the event that the common ancestor outperforms its descendant leaves, the leaves are pruned, the common ancestor becomes the new leaf, and the process continues until the root node is evaluated against all leaves. This process is carried out for each LMUT tree. The pseudocode for postpruning a given LMUT tree is displayed in Algorithm 2.

Algorithm 2 Postprune an LMUT Tree

Definitions:

- $self$: The current considered node in the LMUT tree.
- $self.children$: A list of child nodes of the current node.
- $self.error$: $\sum_{t=1}^{m}(\hat{Q}_t - Q_t^{UT})^2$, where m is the number of new transitions at the current node.

Function Prune($self$):
if length($self.children$) $= 0$ **then**
  return $self.error$
end if
$desc\_error \leftarrow 0$
for $child$ **in** $self.children$ **do**
  $desc\_error \leftarrow desc\_error$ + $child$.Prune()
end for
if $self.error < desc\_error$ **then**
  $self.children \leftarrow$ []
  return $self.error$
end if
return $desc\_error$

For each DRL agent, we trained an LMUT with and without prepruning (to omit prepruning, we simply set max_err equal to zero), and then evaluated each of these LMUT with and without postpruning. We compared each of these four combinations using fidelity and gameplay. We trained the initial LMUT with 30K consecutive transitions, then applied postpruning using another 30K transitions. We ran this entire process for five iterations. To determine the fidelity performance, we averaged Q^{UT} on 10K additional transitions across each iteration for each pruning configuration and recorded the absolute error at each transition t (i.e., $AE = |\hat{Q}_t - Q_t^{UT\_Avg}|$). We report the median absolute error and 95% confidence interval for each pruning combination. For determining the gameplay performance, we had each iteration of each pruning condition play each puzzle for 100 episodes, averaged the results by condition, and reported the associated MRPE and 95% confidence intervals. We also reported the average number of leaves in each condition across all five iterations.

E. Impact of DRL Robustness

As part of the pruning evaluation process, we gathered performance metrics for LMUT agents under the various pruning conditions trained on both more robust and less robust DRL agents, where the degree of robustness is defined by max_err. To determine whether differences in the robustness of the DRL policies impact the gameplay performance of the corresponding LMUT agents, we compared the gameplay performance across the two DRL robustness conditions when holding the pruning technique constant.

V. Results

Table II shows a summary of the performance of our LMUT agents, DRL agents, and an agent acting at random across the three puzzles. For LMUT, performance is given in terms of fidelity and gameplay, as well as the average number of leaves, across each pruning condition and each DRL condition (more and less robust). For DRL and the random agent, we report only the gameplay performance.

When comparing the more robust LMUT agent to the less robust LMUT agent, we see that gameplay improved significantly (i.e., with no overlap in confidence intervals) across all four pruning methods with Cart Pole and Mountain Car, and in two out of four pruning methods with Acrobat.

In considering the impact on gameplay and fidelity from pruning, we note first that implementing prepruning appears to have had no noticeable impact on the number of leaves for Cart Pole and Acrobat; although it did reduce the number of leaves in Mountain Car. Interestingly, with Mountain Car, prepruning led to a significant increase in median absolute error under the less robust condition, a significant decrease under the more robust condition, and did not have a significant impact on gameplay in either condition. As such, we conclude that prepruning had a mixed to negligible impact on LMUT performance across these three puzzles.

Postpruning led to a noticeable reduction in the number of leaves across all three puzzles and was associated with either a negligible change or a significant improvement in fidelity across the three puzzles. With gameplay, postpruning led to a significant improvement in performance for the more robust Cart Pole condition and in one of two circumstances when working with the more robust Acrobat condition. On the other hand, it resulted in significantly worse performance when working with the less robust Acrobat condition and in one of two circumstances when working with the less robust Cart Pole condition. Finally, it had no significant impact under either condition for Mountain Car.

VI. Discussion

We hypothesized that an LMUT agent trained with a more robust DRL policy would perform better in terms of gameplay than one trained with a less robust DRL policy, even if the two DRL agents performed the same. The results of our experiments provide evidence to support this from both Cart Pole and Mountain Car, where the LMUT agent trained on the less robust DRL policy was unable to outperform an agent acting randomly, but the LMUT agent trained on the more robust DRL policy performed significantly better than random, despite the fact that the two DRL agents performed either the same (cf. Cart Pole) or highly similar (cf. Mountain Car).

The findings from Acrobat are less uniformly supportive of our hypothesis, where gameplay performance increased under the more robust condition relative to the less robust condition when postpruning was applied, but the impact was mixed when postpruning was not applied. Furthermore, we found that the second best gameplay performance across all Acrobat training conditions took place under the less robust policy when no pruning was applied. Therefore, based on our findings, the relationship between the robustness of the DRL policy and the LMUT gameplay performance holds in most, but not all, circumstances. We note that Acrobat had the smallest relative difference in max_err across the three

TABLE II: Agent performance. 'Less (More) Robust LMUT' indicates the LMUT agent trained on the less (more) robust DRL agent. 'Pre' and 'Post' indicates prepruning and postpruning, respectively. 'Fidelity' is the median absolute error in average Q-value prediction over five iterations. Gameplay is the MRPE of the average gameplay performance across five iterations over 100 episodes each.

Cart Pole					
Agent	Pre	Post	Leaves	Fidelity (95% CI)	Gameplay (95% CI)
Less Robust LMUT	No	No	121.2	3.30 (3.19, 3.41)	19.4 (18.5, 20.0)
Less Robust LMUT	No	Yes	63.8	1.20 (1.17, 1.23)	17.3 (16.6, 19.4)
Less Robust LMUT	Yes	No	122.8	2.99 (2.87, 3.15)	17.3 (16.0, 18.8)
Less Robust LMUT	Yes	Yes	66.0	1.13 (1.09, 1.16)	15.2 (14.2, 15.9)
Less Robust DRL	N/A	N/A	N/A	N/A	500.0 (500.0, 500.0)
More Robust LMUT	No	No	116.8	1.60 (1.55, 1.65)	53.9 (48.9, 56.6)
More Robust LMUT	No	Yes	84.6	1.42 (1.37, 1.47)	62.2 (58.2, 67.5)
More Robust LMUT	Yes	No	116.6	1.31 (1.28, 1.36)	64.4 (60.2, 66.2)
More Robust LMUT	Yes	Yes	77.0	1.34 (1.30, 1.38)	74.9 (72.8, 77.3)
More Robust DRL	N/A	N/A	N/A	N/A	500.0 (500.0, 500.0)
Random	N/A	N/A	N/A	N/A	21.4 (20.8, 23.1)
Mountain Car					
Agent	Pre	Post	Leaves	Fidelity (95% CI)	Gameplay (95% CI)
Less Robust LMUT	No	No	196.2	140.2 (133.4, 147.2)	-200.0 (-200.0, -200.0)
Less Robust LMUT	No	Yes	153.0	130.9 (124.1, 136.9)	-200.0 (-200.0, -200.0)
Less Robust LMUT	Yes	No	122.0	157.5 (149.9, 165.3)	-200.0 (-200.0, -200.0)
Less Robust LMUT	Yes	Yes	94.0	138.4 (132.5, 142.6)	-200.0 (-200.0, -200.0)
Less Robust DRL	N/A	N/A	N/A	N/A	-146.5 (-149, -143.7)
More Robust LMUT	No	No	194.6	497.2 (486.2, 517.4)	-155.1 (-163.0, -153.0)
More Robust LMUT	No	Yes	159.8	479.5 (467.2, 490.8)	-155.5 (-162.2, -153.4)
More Robust LMUT	Yes	No	185.0	463.8 (449.9, 478.9)	-152.8 (-162.0, -152.1)
More Robust LMUT	Yes	Yes	151.2	461.9 (443.0, 472.0)	-153.5 (-163.2, -152.6)
More Robust DRL	N/A	N/A	N/A	N/A	-141.3 (-142.6, -139.8)
Random	N/A	N/A	N/A	N/A	-200.0 (-200.0, -200.0)
Acrobat					
Agent	Pre	Post	Leaves	Fidelity (95% CI)	Gameplay (95% CI)
Less Robust LMUT	No	No	88.8	25.6 (25.1, 26.2)	-306.3 (-315, -297)
Less Robust LMUT	No	Yes	80.4	25.9 (25.3, 26.5)	-329.9 (-334, -322)
Less Robust LMUT	Yes	No	89.4	25.8 (25.2, 26.3)	-343.7 (-352, -338)
Less Robust LMUT	Yes	Yes	74.6	25.3 (24.8, 25.8)	-397.4 (-409, -390)
Less Robust DRL	N/A	N/A	N/A	N/A	-78.0 (-79.6, -76.8)
More Robust LMUT	No	No	89.8	21.2 (20.6, 21.6)	-352.1 (-360, -345)
More Robust LMUT	No	Yes	73.4	20.7 (20.3, 21.2)	-247.0 (-258, -232)
More Robust LMUT	Yes	No	87.2	22.0 (21.5, 22.5)	-333.5 (-341, -327)
More Robust LMUT	Yes	Yes	77.0	21.6 (21.2, 22.2)	-348.3 (-354, -339)
More Robust DRL	N/A	N/A	N/A	N/A	-75.0 (-76.0, -73.2)
Random	N/A	N/A	N/A	N/A	-500.0 (-500, -500)

puzzles. With more time, we would have liked to test a wider range of robustness levels for the DRL policies across all three puzzles to determine whether the relationship between DRL robustness and LMUT gameplay continued to hold across Cart Pole and Mountain Car. We would also like to determine whether the relationship is more consistent with Acrobat at more pronounced differences in robustness.

We hypothesized further that both fidelity and gameplay would improve through postpruning, whereas prepruning would not have a positive impact. Our findings support our hypothesis in the case of prepruning, as we saw no impact on performance in two puzzles and mixed impact in the third. Postpruning had a positive to negligible impact on fidelity, and a mixed impact on gameplay. We conclude that prepruning may not be worth implementing, whereas postpruning should be included in the training process, but that one should test both the unpruned and postpruned models out of sample to determine which to apply on a particular task.

Finally, we highlight that postpruning resulted in a substantial reduction in the number of leaves and, as a consequence, smaller trees. In [4] and [8], the authors assume that smaller trees are inherently more interpretable, which suggests a strong argument in favor of implementing postpruning. We are more reserved in our claims with respect to interpretability but acknowledge the improvement in interpretability is likely.

VII. Future Work

For a mimic model trained to enhance blackbox model interpretability to be effective, the drivers of its predictions must be interpretable because these interpretations serve as a proxy for the interpretations of the blackbox model behavior. Future work might consider the impact of postpruning on LMUT interpretability with user surveys (e.g., see [21]).

The inability of LMUT to perform better than a random agent when working with the less robust DRL policy with Cart Pole and Mountain Car, despite achieving Q-value predictions that in most cases were closer to the DRL agent's than those derived from the more robust DRL policy, indicates that the LMUT is pushing the DRL agent into areas of the state-space for which the agent had not established a robust policy. This

presents a potential opportunity for future research: we could use the LMUT policy to train the DRL agent further, focusing the latter to learn Q-values for a wider distribution of the state-space to achieve a more robust DRL policy. Subsequently, we could then train a more closely aligned LMUT for generating stronger associated interpretations of the DRL actions.

We evaluated our hypotheses on three benchmark reinforcement learning puzzles. We recommend that future work extend our experiments to real-world problems in order to gauge to what extent these findings translate to the more complex domains that DRL is used to solve in practical applications.

Finally, we defined DRL robustness according to the metric max_err which was inspired by the concept of robust control in control theory. There may exist other measures of robustness that are applicable to this problem, and we encourage future research to identify and apply these alternative measures to our experiments in order to enhance our understanding of the relationship between LMUT gameplay and the robustness of the mimicked DRL policy.

VIII. Conclusion

We examined the impact of prepruning and postpruning on the fidelity and gameplay of LMUT trained to mimic DRL agents. In light of our findings, we recommend that prepruning be omitted but that postpruning be included as an alternative to compare against the unpruned trees on out-of-sample test data. We also examined the impact of DRL policy robustness on LMUT gameplay and conclude that in most circumstances, LMUT gameplay is related to and serves as an indicator of the robustness of the underlying DRL policy.

Acknowledgments

This work was completed largely as part of the 605.746 Advanced Machine Learning class within the Engineering for Professionals (EP) graduate program at Johns Hopkins University. We thank EP for its ongoing support of student research.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

[2] D. Silver, A. Huang, C. Maddison, *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, 2016.

[3] E. Puiutta and E. M. S. P. Veith, "Explainable reinforcement learning: A survey," in *Machine Learning and Knowledge Extraction*, pp. 77–95, Springer International Publishing, 2020.

[4] G. Liu, O. Schulte, W. Zhu, and Q. Li, "Toward interpretable deep reinforcement learning with linear model u-trees," in *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2018, Dublin, Ireland, September 10–14, 2018, Proceedings, Part II*, (Berlin, Heidelberg), p. 414–429, Springer-Verlag, 2019.

[5] V. G. Costa and C. E. Pedreira, "Recent advances in decision trees: an updated survey," *Artificial Intelligence Review*, vol. 56, no. 6, pp. 4765–4800, 2023.

[6] K. Zhou and J. C. Doyle, *Essentials of Robust Control*. Prentice-Hall, 1998.

[7] C. Glanois, P. Weng, M. Zimmer, L. Dong, T. Yang, J. Hao, and W. Liu, "A survey on interpretable reinforcement learning," *Machine Learning*, pp. 1–44, 04 2024.

[8] J. R. Quinlan, "Simplifying decision trees," *International Journal of Man-Machine Studies*, vol. 27, pp. 221–234, 1987.

[9] C. Schaffer, "Overfitting avoidance as bias," *Machine Learning*, vol. 10, pp. 153–178, 1993.

[10] R. García Leiva, A. Fernández Anta, V. Mancuso, and P. Casari, "A novel hyperparameter-free approach to decision tree construction that avoids overfitting by design," *IEEE Access*, vol. 7, pp. 99978–99987, 2019.

[11] Z. Che, S. Purushotham, R. Khemani, and Y. Liu, "Distilling knowledge from deep networks with applications to healthcare domain." arXiv preprint arXiv:1512.03542, 2015.

[12] P. Dorato, "A historical review of robust control," *IEEE Control Systems Magazine*, vol. 7, no. 2, pp. 44–47, 1987.

[13] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[14] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, pp. 279–292, 1992.

[15] A. K. McCallum, "Learning to use selective attention and short-term memory in sequential tasks," in *Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior (SAB)*, pp. 315–325, 1996.

[16] G. Bonner, "Decision making for health care professionals: use of decision trees within the community mental health setting," *Journal of Advanced Nursing*, vol. 35, no. 3, pp. 349–356, 2001.

[17] W. T. Uther and M. M. Veloso, "Tree based discretization for continuous state space reinforcement learning," in *AAAI/IAAI*, pp. 769–774, 1998.

[18] P. Chaudhuri, M. Huang, W. Loh, and R. Yao, "Piecewise-polynomial regression trees," *Statistica Sinica*, pp. 143–167, 1994.

[19] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym." arXiv preprint arXiv:1606.01540, 2016.

[20] O. Anschel, N. Baram, and N. Shimkin, "Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning," in *International Conference on Machine Learning*, 2016.

[21] O. Bastani, C. Kim, and H. Bastani, "Interpreting blackbox models via model extraction." arXiv preprint arXiv:1705.08504, 2019.