Interpretable AI Agent Through Nonlinear Decision Trees for Lane Change Problem

Abhiroop Ghosh, Yashesh Dhebar, Ritam Guha, and Kalyanmoy Deb
Michigan State University
East Lansing, USA
{ghoshab1, dhebarya, guharita, kdeb}@msu.edu

Subramanya Nageshrao, Ling Zhu, Eric Tseng, and Dimitar Filev
Ford Motor Company
Dearborn, USA
{snageshr, lzhu40, htseng, dflev}@ford.com

COIN Report Number 2021016

Abstract—The recent years have witnessed a surge in application of deep neural networks (DNNs) and reinforcement learning (RL) methods to various autonomous control systems and game playing problems. While they are capable of learning from real-world data and produce adequate actions to various state conditions, their internal complexity does not allow an easy way to provide an explanation for their actions. In this paper, we generate state-action pair data from a trained DNN/RL system and employ a previously proposed nonlinear decision tree (NLDT) framework to decipher hidden simplistic rule sets that interpret the working of DNN/RL systems. The complexity of the rule sets are controllable by the user. In essence, the inherent bi-level optimization procedure that finds the NLDTs is capable of reducing the complexities of the state-action logic to a minimalist and interpretable level. Demonstrating the working principle of the NLDT method to a revised mountain car control problem, this paper applies the methodology to the lane changing problem involving six critical cars in front and rear in left, middle, and right lanes of a pilot car. NLDTs are derived to have simplistic relationships of 12 decision variables involving relative distances and velocities of the six critical cars. The derived analytical decision rules are then simplified further by using a symbolic analysis tool to provide English-like interpretation of the lane change problem. This study makes a scratch to the issue of interpretability of modern machine learning based tools and it now deserves further attention and applications to make the overall approach more integrated and effective.

Index Terms—Decision trees, bi-level optimization, machine learning, reinforcement learning, autonomous vehicles.

I. INTRODUCTION

In the field of artificial intelligence and machine learning, reinforcement learning focuses on deriving an AI controller (or agent) to control an object or multiple objects in a given environment. Usually, artificial neural networks (ANN) are popularly used as AI agents for reinforcement learning problems. An ANN agent processes the state of the object being controlled and suggests the action to take. While the ANNs, particularly DNNs (Deep Neural Networks), have been able to successfully execute different control tasks, the main drawback lies in their lack of interpretability. In this work, we try to decipher the black-box logic being executed by the ANN agent to control the object. We use a customized evolutionary algorithm to generate an interpretable AI (IAI) which assumes the framework of a non-linear decision tree (NLDT). It aims to explain complex control rules through a humanly-interpretable, hierarchically defined rule-based system. The complexity of the control task depends on the dimensionality and type of the state space and the action space.

In recent years, DNNs have shown exceptional progress in solving various classification and regression problems. This motivated researchers to use DNNs for solving complex control tasks in a reinforcement learning framework [1]. Researchers have demonstrated the effectiveness of such DNNs. However, the inner workings of DNNs are still beyond human comprehension. The lack of interpretability of such complex systems may lead to various issues:

• Firstly, a well-understood system has more chances of being deployed in an actual working scenario. An incomprehensible system is prone to maintenance, reliability issues, thereby requiring vast experimentation and presence of team-expertise to be acceptable for deployment.

• The most fundamental problem with the lack of interpretability is that it is not possible to extract the reasoning behind the outputs of the complex systems. For example, a DNN deployed in an yield classification environment, might have a high accuracy, but it does not articulate the reasons behind the yield being bad. At the end, it does not help to show what should be corrected in the manufacturing process to reduce the amount of bad yields. An interpretable system will allow us to grasp the underlying reasoning embedded inside the black box classifier and help in correcting any manufacturing errors.

In the paper, we have attempted to extract interpretable rules from a complex DNN/RL system by trying to replicate its inner workings using a set of simple mathematical rules. As a part of the process, we have collected the DNN/RL system’s decisions in various scenarios (states) to form a dataset. An NLDT is then trained using this data, which is able
to reproduce the agent’s decisions with significant accuracy. While the agent’s system is really complex and difficult to understand, the NLDT is easier to explain.

In the remainder of this paper, we first present a brief description of the search of NLDTs using a bi-level evolutionary optimization procedure in Section II. The NLDT-based interpretable AI agent is applied to a simple mountain car problem with two state and three action variables in Section III. Thereafter, we investigate its applicability over a pilot car’s lane change problem using the dataset provided by Ford Motor Company for autonomous highway driving using deep reinforcement learning in a simulated environment [2] in Section IV. This control task contains 12 state and three action variables. Finally, the conclusions are drawn in Section VI.

II. NONLINEAR DECISION TREE (NLDT)

In this section, we discuss the general structure of the NLDT [3] used to generate an interpretable AI for control problems involving two and more discrete actions. An NLDT is a decision tree except that every node (say, the i-th one) uses a nonlinear rule \( f_i(x) \) to split into two child nodes. Here \( x \) is the state variable vector of size \( d \). If \( f_i(x) \leq 0 \), then the data moves to the left child node, else it moves to the right child node. Each rule is constructed with \( n_p \) power law functions \( \{B_i\} \), as follows:

\[
f_i(x) = \sum_{j=1}^{n_p} w_{ij} B_{ij} + \theta_i, \tag{1}
\]

where weights \( w_{ij} \) and bias \( \theta_i \) can take any real value within \([-1, 1]\). The complexity of the rule can be controlled by simply setting a suitable \( n_p \). In this study, we use \( n_p = 2 \) to 4 (at most). The power law functions are also controllable and are represented as follows:

\[
B_{ij} = \prod_{k=1}^{d} x_k^{b_{ijk}}, \tag{2}
\]

where the exponents \( b_{ijk} \) are restricted to take a positive or negative integer. In this study, we restrict them to take a value from the set \( E = \{-2, -1, 0, 1, 2\} \). To be able to classify a sandwiched data set, a modulus operator is also included in the split function as an option: \( f_i(x) = \left| \sum_{j=1}^{n_p} w_{ij} B_{ij} \right| + \theta_i \).

The depth of the NLDT is also a control parameter, which restricts the total number of split functions needed to classify the entire data set. Starting from the root node, all intermediate non-leaf nodes contain one split function. Every leaf node is associated with a specific action. Thus, every state variable vector \( x \) is supplied to the root node and after traversing through the NLDT, a leaf node will be reached which suggests a specific action.

The leaf node action is selected by computing the proportions of data points of different classes present in the node and choosing the class with the maximum proportion. The NLDT presented in Figure 1 can be used to describe the class assignment process. Node 0 contains all the data points where Class 0, 1 and 2 have 4,045, 537 and 5,418 samples, respectively. The proportions of data points of any class present in a leaf node is calculated with respect to these values and then the proportions are normalized for the node. For a particular leaf node \((k)\), the class assignment is performed according to the following equation:

\[
C^k = \arg\max_{i \in \{1, n_c\}} \tilde{p}_i^k, \tag{3}
\]

\[
\tilde{p}_i^k = \frac{p_i^k}{\sum_{i=1}^{n_c} p_i^k},
\]

where \( p_i^k = \frac{n_i^k}{n_c^k} \). \( n_c \) is the total number of classes, \( k \) is the node number, \( n_i^k \) is the number of instances of class \( i \) in the \( k \)th node, \( n_c^0 \) is the number of instances of class \( i \) in the root node. This process of assignment for the NLDT of Figure 1 is clearly demonstrated in Table I.

TABLE I: Class assignment procedure for the NLDT shown in Figure 1. The maximum normalized proportions are marked bold for each leaf node.

<table>
<thead>
<tr>
<th>Leaf</th>
<th>Class</th>
<th>#Samples</th>
<th>( \tilde{p}_i^k )</th>
<th>( \hat{p}_i^k )</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>3,375</td>
<td>0.83</td>
<td>0.81</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>54</td>
<td>0.10</td>
<td>0.10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>492</td>
<td>0.09</td>
<td>0.09</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>227</td>
<td>0.06</td>
<td>0.08</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>33</td>
<td>0.06</td>
<td>0.08</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3,327</td>
<td>0.61</td>
<td>0.84</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>445</td>
<td>0.11</td>
<td>0.09</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>450</td>
<td>0.84</td>
<td>0.67</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1,599</td>
<td>0.30</td>
<td>0.24</td>
<td></td>
</tr>
</tbody>
</table>

---

Similar to [3], we use a bi-level algorithm to derive the split rule power laws \((w \text{ and } B)\) and biases \( \Theta = [\theta_1, \theta_2, \ldots, \theta_{n_{action}} - 1] \). Next, we discuss upper and lower-level optimization problems for our bi-level approach in finding the split rules.

A. Upper level optimization

The upper level operates on the search space of discrete exponents of the power laws \( B_{ij} \). The optimization formulation of the overall algorithm to obtain a split rule \( f_i \) is as shown in Figure 1. The maximum normalized proportions are marked bold for each leaf node.
Minimize \( F_U(\mathbf{B}_i, \mathbf{w}_i^*, \theta_i^*) \), 
Subject to \((\mathbf{w}_i^*, \theta_i^*) \in \text{argmin} \left\{ F_L(\mathbf{w}_i, \theta_i) | \mathbf{B}_i \right\} \)
\[
F_L(\mathbf{w}_i, \theta_i) | \mathbf{B}_i \leq \tau_f, \\
-1 \leq w_{ij} \leq 1 \quad \forall j, \\
\theta_i \in [-1, 1], \\
b_{ijk} \in E, \quad \forall j, k.
\]

The upper level objective function \( F_U \) represents the complexity of the tree structure equation \( f_i(x) \), obtained by computing the number of leaf nodes \( n_{leaf} \).

where \( n_{leaf} \) indicates number of nodes resulting from 

\[ F_L = \sum_{j=1}^{N_j} \frac{I_j}{N} \]

where \( n_{children} \) indicates number of children resulting from 

\[ I_j = 1 - \frac{n_{children}}{N} \]

where \( N_j \) is the total number of data points in node \( j \) with action value \( a \) and \( N \) is the total number of data points in the node. We use the real-parameter GA [5] to conduct a reliable search on variables \( w_i \) and \( \theta_i \) to minimize the lower level objective function \( F_L \).

C. Pruning and tree simplification

The lower level optimization minimizes the impurity of resulting child nodes to enhance the split quality. Split quality \( F_L \) is expressed as the weighted sum of impurities of resulting child nodes as shown below:

\[
\text{Node Impurity} \quad I_j = 1 - \frac{n_{children}}{N} \left( \frac{N_j}{N} \right) \]

where \( N_j \) is the total number of data points in node \( j \) with action value \( a \) and \( N \) is the total number of data points in the node. We use the real-parameter GA [5] to conduct a reliable high-level search on variables \( w_i \) and \( \theta_i \) to minimize the lower level objective function \( F_L \).

B. Lower level optimization

The lower level searches for continuous variables \( w_i \) and \( \theta_i \). Lower level optimization minimizes the impurity of resulting child nodes to enhance the split quality. Split quality \( F_L \) is expressed as the weighted sum of impurities of resulting child nodes as shown below:

\[
F_L = \sum_{j=1}^{n_{children}} \frac{N_j}{N} I_j,
\]

where \( n_{children} \) indicates number of children resulting from the split, \( N_j/N \) indicates fraction of points going to \( j \)-th child after the split, and \( I_j \) indicates the impurity of child node \( j \).

For this problem, we are only working with binary NLDTs.

\[
C. \text{Pruning and tree simplification}
\]

The resulting tree is fairly complex with many split nodes. Then, we simplify this tree by removing redundant splits through a pruning process [3]. Lower depth trees obtained thus far are relatively simple and also have better generalizability.

D. Open-loop and Closed-loop Training

In the open-loop training, every \((i\text{-th})\) node in the NLDT is optimized one at a time using the labeled state-action dataset. Optimization starts at the root node and continues until the prescribed depth is reached. This allows for a fast NLDT search. Since this process has no feedback loop, we refer this as NLDT\(_{OL}\) (NLDT in open-loop).

In order to ensure a better closed-loop performance, we take a already-trained NLDT\(_{OL}\) and re-optimize the weights \((\mathbf{w})\) and biases \((\mathbf{\Theta})\) of all rules in the NLDT\(_{OL}\) simultaneously using a real-parameter GA [5]. Note that the power laws \((\mathbf{B})\) obtained from NLDT\(_{OL}\) are kept the same. The impurity (lower level objective) and rule complexity (upper level objective) objective functions are replaced by a closed-loop cumulative reward function. Applying such a reinforcement learning procedure from scratch would have required a large computation time. Instead, a labeled training dataset is used to find an optimal NLDT architecture, followed by a closed-loop optimization procedure. This results in a fine-tuned tree, referred to as NLDT* [6]. The difference between NLDT\(_{OL}\) and NLDT* is illustrated in Figure 2. The optimization formulation for closed-loop training is shown in the following:

\[
\text{Maximize} \quad F_{OL}(\mathbf{W}, \mathbf{\Theta}) = \sum_{i=1}^{M} R_i(\mathbf{W}, \mathbf{\Theta}),
\]

where \( n_r \) is the number of rules in the NLDT\(_{OL}\) and \( M \) is the total number of episodes, which is a simulation of the NLDT from start to end of the control experiment. \( R_i \) is the total reward for the \( i \)-th episode.

III. MOUNTAIN CAR PROBLEM

The mountain car control system problem, first suggested in [7], is widely used for testing AI methods [8]–[10]. The goal is to get an under-powered car to reach a goal location at the top of the mountain. In this paper, a revised version of the mountain car problem used in [6] is considered. The car is underpowered by reducing the acceleration \((\alpha)\) from \(10^{-3}\) to \(6 \times 10^{-4}\) units, almost half the original value. The purpose of this revision is to ensure that both the left and right applied forces are just adequate to ensure the car reaches its goal. There are 3 actions \((A_t)\) at time \( t \) available to the controller: decelerate at a constant value \(-a\) along the positive x-axis (Class 0), do nothing (Class 1), accelerate at a constant value \(a\) along the positive x-axis (Class 2). The state vector \( \mathbf{S}_t = (x_t, v_t) \) at time \( t \) has two variables: \( x_t \), denoting the position, and \( v_t \), describing the velocity, both along the x-axis. Any control policy \( \pi(\mathbf{S}) \) has to reach the goal within 200 time steps with the starting position close to the bottom of the mountain.

A black-box controller is initially trained using the SARSA algorithm [11]. This will serve as the oracle with a policy \( \pi_{\text{oracle}} \). It returns an action \( A_t \in [0, 1, 2] \) based on the current state \( \mathbf{S}_t = (x_t, v_t) \in \mathbb{R}^2 \). The training dataset required for NLDT\(_{OL}\) is generated by supplying a random initial state \( \mathbf{S}_0 \) and following the actions suggested by \( \pi_{\text{oracle}} \) for 200 time steps or when the car reaches its goal, whichever is earlier. Each such simulation constitutes an episode. Simulations are run until 10,000 state-action pairs are obtained. There are 4,045, 537 and 5,418 state vectors corresponding to actions.
Fig. 2: Open-loop versus closed-loop training (taken from [6]).

0, 1 and 2, respectively. A significant difference from the previous work [6] is the imbalanced dataset. Action 2 has 5,418 points, almost 10 times that of Action 1. The state-action combinations obtained are shown in Figure 3a. The points are color-coded, with green, orange and blue representing actions 0, 1 and 2, respectively. It is to be noted that there exists a lot of overlap between different actions in the state-space, which requires the oracle to devise complex non-linear policies, hampering its interpretability.

The state variables $S_t = (x_t, v_t)$ in the training dataset are normalized to obtain the transformed state variables $\hat{S}_t = (\hat{x}_t, \hat{v}_t) \in [1, 2]$. This is to enable the possibility of negative powers appearing in the NLDT rules. The normalized dataset is used to train an NLDT$\text{OL}$, followed by closed-loop training (NLDT$^*$). This results in an interpretable AI policy ($\pi_{NLDT}$) which can be expressed in the form of simple mathematical functions. The trained NLDT$^*$ is shown in Figure 1 with Table II showing the mathematical relations corresponding to each node.

**TABLE II: NLDT$^*$ rules for mountain car problem.**

<table>
<thead>
<tr>
<th>Node</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$-1.00\hat{v}_t^{-0.67} + 0.67$</td>
</tr>
<tr>
<td>4</td>
<td>$-1.00\hat{x}_t^{-1}\hat{v}_t^{-1} + 0.17$</td>
</tr>
</tbody>
</table>

Based on Figure 1 and Table II, we can express the simplified policy ($\pi_{NLDT}$) as follows:
- If $0.67 - \frac{\hat{v}_t}{\hat{v}_t} \leq 0$, decelerate (Action 0). Otherwise, go to the next step.
- If $-\frac{1}{\hat{x}_t\hat{v}_t} + 0.17 \leq 0$, accelerate (Action 2). Otherwise, do nothing (Action 1).

The state-action plot in Figure 3b shows the effect of the interpretable policy $\pi_{NLDT}$. The two curves in black show the decision boundaries defined by the relations in Table II. These boundaries, constituting the classifier, are much more simple and smooth compared to that of the oracle. The blue, orange and green shaded areas represent the state variable pairs that would result in actions 0, 1 and 2, respectively, according to $\pi_{NLDT}$. Interestingly, even though Action 1 (in orange) has the least number of data points supporting it, the NLDT$^*$ is still able to find a smooth decision boundary which classifies many of its points. This can be attributed to the proportion-based class assignment method described by Equation 3. Due to the approximations involved, many state-action pairs do not match the oracle. Thus, the overall training accuracy is only 82%.

The closed-loop performances of the trained SARSA agent,
NLDT\textsubscript{OL} and NLDT\textsuperscript{*} were evaluated over 1,000 episodes with 200 time steps each. The SARS\textsubscript{A} agent was able to get the car to its goal 100% of the time. NLDT\textsubscript{OL} was able to reach the goal 96.5% of the time, whereas NLDT\textsuperscript{*} was able to do the same 99.2% of the time. Thus, despite a slightly low training accuracy, NLDT\textsuperscript{*} is able to closely approximate the performance of the oracle, with much simpler policies. Next, we apply the NLDT approach to a vehicle lane change problem, which has more state variables and is much more complex than the mountain car problem.

### IV. Lane Change Problem

The lane change problem was originally proposed in [2] where a double deep Q-network (DDQN) [12], [13], combined with a short horizon safety controller, was trained to safely control an autonomous ego vehicle (EV) on simulated highway traffic. There are three lanes on the highway and the EV is free to use any lane. There are 12 state variables and 3 possible actions, as shown in Tables III and IV, respectively.

#### TABLE III: State variable description. All quantities are measured relative to the EV car.

<table>
<thead>
<tr>
<th>Variable $x_i$</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_0$</td>
<td>Front left vehicle longitudinal distance</td>
</tr>
<tr>
<td>$x_1$</td>
<td>Front left vehicle velocity</td>
</tr>
<tr>
<td>$x_2$</td>
<td>Front center vehicle longitudinal distance</td>
</tr>
<tr>
<td>$x_3$</td>
<td>Front center vehicle velocity</td>
</tr>
<tr>
<td>$x_4$</td>
<td>Front right vehicle longitudinal distance</td>
</tr>
<tr>
<td>$x_5$</td>
<td>Front right vehicle velocity</td>
</tr>
<tr>
<td>$x_6$</td>
<td>Rear left vehicle longitudinal distance</td>
</tr>
<tr>
<td>$x_7$</td>
<td>Rear left vehicle velocity</td>
</tr>
<tr>
<td>$x_8$</td>
<td>Rear center vehicle longitudinal distance</td>
</tr>
<tr>
<td>$x_9$</td>
<td>Rear center vehicle velocity</td>
</tr>
<tr>
<td>$x_{10}$</td>
<td>Rear right vehicle longitudinal distance</td>
</tr>
<tr>
<td>$x_{11}$</td>
<td>Rear right vehicle velocity</td>
</tr>
</tbody>
</table>

#### TABLE IV: List of possible EV actions.

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Stay on current lane</td>
</tr>
<tr>
<td>1</td>
<td>Change to the left lane</td>
</tr>
<tr>
<td>2</td>
<td>Change to the right lane</td>
</tr>
</tbody>
</table>

A proprietary traffic simulator used in [2] has been provided by Ford Motor Company. This acts as the RL environment which is used to train a DDQN. A fully-trained DDQN can navigate the car safely without any collisions and minimal safety triggers [2]. However, the underlying operating principles of a trained DDQN, or any deep neural network for that matter, is often very poorly understood, necessitating a black box treatment. This has significant real-world implications. A poorly understood autonomous vehicle (AV) controller will give rise to concerns about their safety and effectiveness which may slow down the adoption of AV technology. Some interpretable rules which approximately describe the operation of the DDQN-based controller will help in reducing such concerns. Training an NLDT to mimic the behavior of a DDQN will provide a set of non-linear functions describing the mapping between the state variables and the action taken.

Training dataset for NLDT\textsubscript{OL} is obtained by connecting the DDQN and the simulation environment into a feedback loop. Starting from a random state $(S_0)$ denoting the initial state of the EV with respect to the surrounding cars, the DDQN can suggest an action $A_t$ which updates the state $(S_t)$ of the EV as well as the other cars on the road. This process is repeated for 200 time steps, thus constituting one episode. 1200 such episodes are repeated with different initial states and traffic conditions. For every time step $(t)$ in each episode which does not result in a collision or a safety trigger, the $(S_t, a_t)$ pairs are collected. To ensure a balanced dataset, undersampling is performed, resulting in a total of 21,000 points distributed equally among the 3 actions.

#### A. Preprocessing

The training dataset is transformed to a suitable format using two pre-processing steps as mentioned below:

- In a practical sense, whenever a car takes a lane changing decision, it is based on the data from its neighboring lanes. A car in the leftmost lane of a highway, for instance, will not take into account the behavior of a car in the rightmost lane (with $\geq 2$ lane distance) when deciding to do a lane change or not. Hence, making the lane definitions relative to the current lane of the EV will allow the NLDT to learn patterns based on what the driver sees in the immediate vicinity. This transformation does not affect the instances where the car is in the center lane. After the transformation, whichever lane the car is on will be considered as the central lane and the immediate left or right lane will be considered as the left and right lanes, respectively, relative to the current central lane. For the leftmost and rightmost lanes, dummy lanes are created with a car running parallel to the EV at the same speed on their left and right lanes, respectively. It prevents the EV from taking left or right turns on the leftmost or rightmost lane. With this kind of a transformation, any knowledge gained will be much more generalized and applicable to roads with different number of lanes. Figure 4-6 illustrates the basis for the transformed data.

- Finally, for ease of analysis, all variables ($x_i$) are normalized to get $\hat{x}_i \in [1, 2]$ with a low value representing an unsafe scenario for the ego car and a high value denoting a safe situation for the ego car. As the distances and velocities are calculated relative to the ego car, front cars have positive longitudinal distances and rear cars have negative longitudinal distances, whereas the sign of the velocity depends on their respective velocity values. We can observe that when the longitudinal distances for the front cars are small, they represent critical scenarios because they are very close to the ego car. However, in case of rear cars, small negative longitudinal distances represent critical situations. So, after the transformation, smallest positive relative value for front cars and smallest negative relative value for the rear cars’ longitudinal distances are mapped to one. On the other hand, when the relative velocity is positive in case of front cars, it
means that they are moving away which represents a safe situation, but if the relative velocity is positive for the rear cars, it means the rear car is closing in, denoting an unsafe situation for the ego car. Similarly, negative relative velocity is safe with respect to rear cars, but critical with respect to front cars. Taking all these cases into consideration, the critical situations are mapped to lower values and safe situations are mapped to higher values, normalized between 1 and 2. The minimum value is kept at 1 instead of 0 to allow terms with negative powers to appear in the NLDT rules.

V. EXPERIMENTAL RESULTS AND DISCUSSION

This section summarizes the results obtained for the dataset that was transformed according to the procedure shown in Section IV-A. Initially open loop training (NLDT$_{OL}$) is performed, followed by a closed-loop training process (NLDT*). In essence, the NLDT$_{OL}$ serves as the basic architecture which is fine-tuned to get NLDT*. The reward function in the simulation environment provided by Ford Motor Company is used during the closed-loop training. Each episode starts with a random initial state of the ego car and surrounding vehicles. Simulation is performed for a certain number of time-steps with the NLDT suggesting the necessary action. The cumulative reward at the end of each episode is used as the objective function for NLDT*. The parameters for NLDT$_{OL}$ and NLDT* are given in Tables V and VI, respectively. Two cases are considered here. As shown in Table V, Case 1 allows relatively more complex split rules compared to those in Case 2.

### TABLE V: NLDT$_{OL}$ parameter settings.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common</td>
<td>Allowed powers ($b_{ij}$)</td>
<td>[-2, -1, 0, 1, 2]</td>
</tr>
<tr>
<td></td>
<td>Population size</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Pruning threshold ($\tau_1$)</td>
<td>1</td>
</tr>
<tr>
<td>Case 1</td>
<td>Max. depth</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Number of power laws ($n_p$)</td>
<td>3</td>
</tr>
<tr>
<td>Case 2</td>
<td>Max. depth</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Number of power laws ($n_p$)</td>
<td>2</td>
</tr>
</tbody>
</table>

### A. Case 1: Relatively Complex NLDT Search

Figure 7 shows the final NLDT obtained and Table VII shows the corresponding rules. The tree obtained for Case 1 is quite simple and hence, is easier to interpret compared to DDQN. For the root node, it is evident that it seeks to primarily split the dataset into right lane versus non-right lane.

### TABLE VII: NLDT* rules for Case 1.

<table>
<thead>
<tr>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 $-0.30x_{10}^{-1} + 0.13x_{10}^{-1}x_1 - x_2 - x_4 - 0.59x_2^2 + 1.00$</td>
</tr>
<tr>
<td>1 $0.06x_0x_4 - 0.04x_{10}^{-1}x_1 - 0.34x_{10}^{-1}x_2 - 1.00 - 0.12$</td>
</tr>
<tr>
<td>3 $-0.16x_0x_2 - 0.17x_{10}^{-1}x_2x_9 + 1.00$</td>
</tr>
</tbody>
</table>
change actions. Around 87% of the Class 2 data points are classified into Node 6 along with \( \approx 23\% \) of Class 0 data points and \( \approx 12\% \) of Class 1 points. Thus, the root node equation shown in Table VII needs to be satisfied for the NLDT to suggest a right lane change. On analyzing the terms present in the equation, we can see that it considers the front left car’s distance and velocity \((\hat{x}_0, \dot{\hat{x}}_0)\) and velocity \((\hat{x}_1, \dot{\hat{x}}_1)\) variables have a negative exponent. So, in order for the overall value to be positive, those variables will have to take a low value, meaning any left lane change actions will be unsafe. Similarly, front center car’s distance \((\hat{x}_2)\) needs to be small, which results in the lane change being considered in the first place. Lastly the front right car’s distance \((\hat{x}_4)\) and rear right car’s distance \((\hat{x}_{10})\) need to be high, which is absolutely necessary for a safe right lane change action. From a human driver’s perspective, this type of reasoning makes sense. It is interesting that from the driving data, such intuitive rules are possible to be extracted by our bi-level optimization based NLDT procedure.

To evaluate the closed-loop performances of the DDQN, NLDT\(_{OL}\) and NLDT\(^*\), 100 episodes were simulated using each of the agents as the decision makers. Each episode had 200 time steps and the corresponding cumulative rewards were recorded. Since the rewards assigned by the environment are mostly penalties for safety overrides, they are usually negative. The objective function used during the NLDT\(^*\) training is the negative of the reward function since we have framed the optimization procedure as a minimization problem. The average and median objective values for each agent per episode are shown in Table VIII. The table also shows the number of collisions along with the number of times the safety controller has prevented the agent from performing an unsafe action. From Table VIII, it is evident that DDQN is the best performing agent with the lowest median objective value. It also resulted in the least number of safety overrides. NLDT\(^*\), while not being the best, is not far behind. NLDT\(_{OL}\) is the worst performer, and it is clear here that NLDT\(^*\) plays an important role in improving the overall performance from NLDT\(_{OL}\). None of the agents caused any collision, which can be attributed to the presence of the safety controller.

### B. Case 2: Relatively Simple NLDT Search

This experiment is equivalent to the previous case, the difference being that the maximum allowable depth as well as the number of power law terms has been reduced to 2. This is a special case designed to demonstrate the complexity-accuracy trade-off associated with such interpretable system design. The trained NLDT\(^*\) is shown in Figure 8 and the corresponding rules are shown in Table IX.

**TABLE VIII: Closed-loop performance over 100 episodes with 200 time steps each for DDQN and NLDTs for both cases.**

<table>
<thead>
<tr>
<th>Agent</th>
<th>Mean</th>
<th>Median</th>
<th>Safety overrides</th>
<th>Collisions</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDQN</td>
<td>92.12</td>
<td>80.94</td>
<td>681</td>
<td>0</td>
</tr>
<tr>
<td>Case 1 NLDT(^*) Results</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NLDT(_{OL})</td>
<td>134.64 ± 36.16</td>
<td>100.98 ± 57.34</td>
<td>134.44</td>
<td>98.88</td>
</tr>
<tr>
<td>NLDT(^*)</td>
<td>76%</td>
<td>36%</td>
<td>681</td>
<td>0</td>
</tr>
<tr>
<td>Case 2 NLDT(^*) Results</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NLDT(_{OL})</td>
<td>144.98 ± 22.35</td>
<td>125.92 ± 18.63</td>
<td>142.13</td>
<td>110.15</td>
</tr>
<tr>
<td>NLDT(^*)</td>
<td>57.34</td>
<td>98.88</td>
<td>692</td>
<td>0</td>
</tr>
</tbody>
</table>

From Figure 8, it is evident that the NLDT\(^*\) classified the right lane change instances well (\(\approx 86.7\%\) accuracy), but does not perform well on the others, resulting in \(\approx 36\%\) accuracy for maintain lane and \(\approx 56\%\) accuracy for left lane change instances. This can be attributed to the restrictive NLDT parameters which do not allow the formation of complex rules. Since right lane change action has a high accuracy, it will be used for the symbolic analysis in the subsequent section. The closed-loop performance of Case 2 is described in Table VIII. The performance for both NLDT\(_{OL}\) and NLDT\(^*\) is noticeably worse for Case 2 than that of Case 1. This shows that reduced rule complexity or greater interpretability comes with a performance tradeoff.

In order to perform a right lane change, we can say that the inequations defining nodes 0 and 1 need to be simultaneously satisfied. The pathway from Node 0 to Node 2 can be expressed as a set of rules mentioned in Table IX. Solving this will allow us to determine the exact interactions taking place among the variables.

### C. Further Interpretability Through a Symbolic Analysis

Split rules presented in Table IX are simple, but not simple enough for a human to have a clear comprehension of the lane change decisions suggested in the NLDT\(^*\). For this purpose, we propose the use of a symbolic analysis procedure to discover more simplistic conditions. Here, we use Mathematica version 12.3.1 with a Wolfram Alpha backend [14].

When the two inequalities from Table IX are provided to the Mathematica, it produced the following factors suggesting:

**TABLE IX: NLDT\(^*\) rules for Case 2.**

<table>
<thead>
<tr>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (-1.00x_1^{-2}x_4^{-2} + 0.13)</td>
</tr>
<tr>
<td>1 (-1.00x_2^{-1} + 0.54x_5^{-2} + 0.45)</td>
</tr>
</tbody>
</table>
Equation 11 can be expressed as \( \hat{x}_L \leq \hat{x}_3 \leq 2 \) where \( \hat{x}_L = 3.29 \sqrt{\frac{\hat{x}_2}{20 - 9\hat{x}_2}} \). Substituting Equation 9 to 11, we get \( 1.001 \leq \hat{x}_L \leq 1.979 \). Equations 8-11 can be translated roughly to simple English-like statements. The variables involved are the front left car’s relative velocity (\( \hat{x}_1 \)), front center car distance (\( \hat{x}_2 \)), front right car distance (\( \hat{x}_4 \)) and front right car velocity (\( \hat{x}_5 \)). This shows that representative quantities regarding the state of traffic in all the lanes are considered before deciding to make a right lane change. Thus, in terms of the unscaled state variables, a right lane change is performed when:

1. The front left car’s relative velocity is low and within \([-10, -2.2]\) m/s (implying the left lane change is unsafe).
2. The front center car’s distance lying within a particular range ([1.2, 79] m), but not too high, implying a need for a lane change.
3. The front right car’s distance and relative velocity are high, within [44, 115] m and [\( \hat{x}_L, 10 \)] m/s (meaning the right lane change is safe). \( \hat{x}_L \) is dependent on the front center car’s distance and lies within \([-9.4, 9.8]\) m/s. It is to be noted that \( \hat{x}_L \) is proportional to the front center car distance. It can be interpreted that the faster the front center car is moving the less incentive there is to do a right lane change and hence the minimum threshold for the front right car relative velocity is increased.

Similarly, the left lane change conditions can also be derived from the NLDT. These rules are intuitive, but are arrived at from our NLDT development and rule analysis procedure. It is interesting that such multiple yet simple conditions involving the neighboring car’s location and velocity information (state variable) taken from the DNN’s action specifications can be arrived at through our bi-level optimization based approach. Since they are derived from DNN’s outcome, the rules provide us an interpretable understanding of the working of the DNN, which is otherwise complex to interpret.

VI. CONCLUSIONS

In this paper, we have presented a method to obtain interpretable rules to describe the operation of a deep neural network or reinforcement learning systems trained to operate an autonomous vehicle in simulated highway traffic conditions. The rules are expressed in the form of non-linear inequalities embedded in a decision tree-type structure, referred to as a non-linear decision tree (NLDT). A two-phase training process is performed using a bi-level optimization method – open-loop (NLDT\(_{OL}\)) and closed-loop (NLDT\(_{CL}\)) optimizations.

In order to deal with some of the intrinsic complexities of the dataset, we have proposed a data-preprocessing step. This has resulted in a simple tree with more generic and abstract rules describing the operation of the vehicle. The proposed system is much more interpretable compared to a black-box neural network, and is able to closely approximate the neural network performance. The resulting NLDT structure is controllable to achieve any level of complexity, but the simplicity in its structure comes with a sacrifice in its performance. In the mountain car and the lane change problems, we have found that a reasonably simple NLDT, obtained by our proposed procedure, is capable of finding similar accuracy as the original DNN or RL system. Hence, the derived NLDT can be thought to represent an interpretation of the working of the respective DNN or RL.

ACKNOWLEDGEMENT

Authors acknowledge the support under the Ford-MSU Alliance project.

REFERENCES