Towards Interpretable Neural Networks: An Exact Transformation to Multi-Class Multivariate Decision Trees

Duy T. Nguyen[a,*], Kathryn E. Kasmarik[a], Hussein A. Abbass[a]

[a]*Trusted Autonomy Laboratory, School of Engineering and Information Technology, University of New South Wales - Canberra, Canberra 2600, Australia*

Abstract

Artificial neural networks (ANNs) are commonly labelled as black-boxes, lacking interpretability. This hinders human understanding of ANNs' behaviors. A need exists to generate a meaningful sequential logic for the production of a specific output. Decision trees exhibit better interpretability and expressive power due to their representation language and the existence of efficient algorithms to generate rules. Growing a decision tree based on the available data could produce larger than necessary trees or trees that do not generalise well. In this paper, we introduce two novel multivariate decision tree (MDT) algorithms for rule extraction from an ANN: an Exact-Convertible Decision Tree (EC-DT) and an Extended C-Net algorithm to transform a neural network with Rectified Linear Unit activation functions into a representative tree which can be used to extract multivariate rules for reasoning. While the EC-DT translates the ANN in a layer-wise manner to represent exactly the decision boundaries implicitly learned by the hidden layers of the network, the Extended C-Net inherits the decompositional approach from EC-DT and combines with a C5 tree learning algorithm to construct the decision rules. The results suggest that while EC-DT is superior in preserving the structure and the accuracy of ANN, Extended C-Net generates the most compact and highly effective trees from ANN. Both proposed MDT algorithms generate rules including combinations of multiple attributes for precise interpretation of

*Corresponding author
Email address: tung.nguyen@student.adfa.edu.au (Duy T. Nguyen)

decision-making processes.

Keywords: Explainable Artificial Intelligence, Rule Extraction, Multivariate Decision Tree, Artificial Neural Network.

---

1. Introduction

Symbolic AI technologies adopt interpretable and explainable representation languages with sufficient expressive power for a human to understand the system's behaviours [1, 2, 3, 4]. Nevertheless, symbolic systems that rely on deductive logic lack the ability to adapt to changes in the environment and context, require excessive knowledge of the problem domain in advance, and often end up with a hard-to-maintain knowledge base.

Inductive learning affords a machine the ability to form and update its own internal representation using experiences. Data labelled with ground truth, user feedback or self-guidance using predesigned similarity metrics guide the three basic forms of learning: supervised, reinforcement and unsupervised, respectively. Early machine learning and statistical inferencing algorithms required significant feature engineering efforts by the human designer. This effort substantially decreased with the arrival of deep learning algorithms [5, 6, 7]. With the possibility for implementing a neural network (NN) on the hardware, and ability to leverage the vectorisation available in graphics processing units (GPUs) in software, NNs offer significant advantages when it comes to speed of learning and ability to handle big data. Recent advances in black-box models such as deep neural networks have shown significant abilities to learn and even surpass human-level performance in some tasks [8, 9, 10]. One of the major disadvantages normally associated with NNs is their lack of interpretability; they are labelled as black-boxes. The highly distributed nature of their implicit capture of knowledge and their ability to approximate highly non-linear functions using many local units and different layers of transformations, complicates the interpretability of ANNs. In the absence of appropriate representations to interpret what a ANN learns, social integration, human acceptance, verification against requirements or previous knowledge bases, and performance assurance are some of the main challenges to the use of to ANN in prac-

tical applications, especially in safety-critical environments. Addressing the drawback of these opaque systems to enable seamless and safe interactions between humans and machines can be attained by developing new technologies for explainable models without sacrificing the performance of the AI models.

Interpretability is required throughout the learning process. Although poorly defined in literature, the term 'interpretation' can be simply described as a process involving the transformation of a type of information into another, which provides a better understanding for a specific group of users. Different studies refer to two categories of the subject including algorithmic interpretability and model interpretability [11]. The former is related to the methods to investigate the mechanism and help debug the system by revealing the knowledge of the algorithm's structure and its system of parameters. The latter one concerns only the mapping model between the inputs and outputs. In this paper, the 'interpretability' term refers to the first category. For a model that has not learnt the task well-enough, interpretability could shed light on which part of the model is lagging behind. For a model that generalises well, interpretability sheds light on the rationale behind the model's behaviour. Among the techniques to interpret the hidden layers of the neural networks, decision trees [12, 13, 14, 15, 16] have been used to learn the relationship between the input and output of a learned neural network. This provides a means to extract and represent the implicit knowledge in the network in an interpretable form.

In this paper, we propose two multivariate decision tree algorithms called the Extended C-Net and EC-DT algorithms. Extended C-Net learns the relationship between the last hidden layer and the outputs, then infers the input-output relationships by back-projections guided by weights from ANNs. EC-DT constructs the rule sets layer-wise based on the activation of the hidden nodes to find an exact representation of ANNs in tree forms. The performance of the ANN is used as the baseline. We use three performance indicators: task performance (accuracy or error of classification), rule compactness (model size), and interpretability (model interpretability).

The contributions of the paper are three-fold. First, the paper introduces novel multivariate tree algorithms as decompositional rule extraction techniques for inspecting the ANNs with a focus on continuous input and Rectified Linear Function (ReLU)

activation function. Secondly, a deep investigation is conducted on the relation between rule set compactness and complexity, and the complexity of the data space to address the question of which situations a simple black-box (pedagogical) rule extraction technique are more favored, taking into account the compromise between the generalization capability and transparency. Last, but not least, the paper also discusses some methods to transform the decision rules into a more interpretable type of representation that bridges the gap between the mathematical interpretation into common-sense explanation.

The organization of this paper is as follows. Section 2 reviews the literature including previous decision tree approaches and multivariate decision trees (MDTs) used to extract rules from ANNs. Section 3 introduces the EC-DT algorithm, which can convert a ANN to representative rules which preserve 100% performance of the ANN. Section 4 describes the Extended C-Net algorithm to learn the decision rules from neural network models with back-projection techniques. Experimental setup and the set of metrics we use to assess the performance and the interpretability of our approach are then presented in Section 5. We then discuss our results and corresponding analysis in Section 6 and conclude the paper in Section 7.

2. Background

In this section, we present a brief introduction of the ANN. Then a short review of recent literature on neural network interpretation methods is introduced, following by the requirements and various types of rule extraction techniques for ANNs. Finally, we focus on several multi-variate decision tree methods to extract interpretable representations from neural networks.

2.1. Artificial Neural Network

An artificial neural network (ANN) is a structure that contains multiple computing units, often arranged in layers, with various connecting configuration among them. Each computing unit is associated with an activation function for linear or non-linear mapping of its input to output. An ANN is a function approximator that can transform

input data into desired output by the units' properties and the weights associated with the interconnections among them.

Here we introduce some notations of a trained densely feed-forward ANN with K hidden layers:

- I and M are the number of input and output units respectively.
- K is the number of hidden layers.
- $J_1, J_2, ..., J_k, ..., J_K$ are the number of units in each hidden layer.
- $\mathbf{X^t} = (x_1^t, x_2^t, ..., x_I^t)$ is the input fed to the input layer of the network.
- $Y^t \in \mathbf{Y}$ is the output class of the corresponding input.
- $w_{ij_1}, w_{j_1j_2}, ...w_{j_Km}$ are the weights of the links from input unit i to hidden unit j_1 at hidden layer 1, hidden unit j_1 at hidden layer 1 to hidden unit j_2 at hidden layer 2,..., and hidden unit j_K to output unit m respectively.
- $H_{j_1}^1(\mathbf{X^t}) = \sigma_{j_1}^1(\sum_{i=1}^{I} w_{ij_1}x_i^t + \beta_{j_1}^1), j = 1...J_1$ is the output of the hidden unit j_1 at hidden layer 1. $\sigma(.)$ is the activation function.
- $H_{j_k}^k(\mathbf{X^t}) = \sigma_{j_k}^k(\sum_{j_{k-1}=1}^{J_{k-1}} w_{j_{k-1}j_k}H_{j_{k-1}}^{k-1}(\mathbf{X^t}) + \beta_{j_k}^k)$, $j_k = 1...J_k$ is the output of the hidden unit j_k at hidden layer k, where $1 < k < K$.
- $Y_m(\mathbf{X^t}) = \sigma_m(\sum_{j_K=1}^{J_K} w_{j_Km}H_{j_K}^K(\mathbf{X^t}) + \beta_{j_m}^Y)$ is the output of unit m in the output layer of the neural network.
- $\mathbf{O}^t$ is the class output of the neural network corresponding to the pattern of neural network output $\mathbf{Y}(\mathbf{X^t}) = \{Y_1(\mathbf{X^t}, ...Y_K(\mathbf{X^t}\}$.

The ANN can be trained with a back-propagation algorithm, a learning process that modifies the weights of the ANN according to the errors between the outputs of output neurons and the target values, and back-propagated errors to previous layers. The weights of the ANN is refined until a satisfactory level of performance is achieved.

2.2. *Interpreting Black-box Models*

ANNs have demonstrated a significant social impact due to their universal function approximation properties, robustness, very large scale implementation characteristic, generalization abilities, and success in many applications. However, due to their black-box nature, they are not as widely acceptable by humans when compared to classic rule-based systems that rely on symbolic representations [17, 18]. For opaque models like ANNs, there is a need to explain their decision-making processes, to be transparent without sacrificing their predictive power.

Currently, the explainable artificial intelligence (xAI) domain calls for solutions to overcome the opaqueness of ANNs to improve their reliability and trustworthiness when they are used in decision support engines and expert systems. Approaches in xAI might alleviate the problems of knowledge extraction in these black-box systems, provide the systems with explanation and reasoning abilities, facilitate the verification and validation of the model, and inspect and diagnose the sources of erroneous interferences [19, 20]. These abilities could enhance the utility, transparency, and explainability of ANN in safety critical applications [21, 22, 23].

Recent literature for explaining deep neural networks focuses on two approaches: deep visualization and rule extraction methods. Deep visualization produces saliency maps from the activation convolutional layers of the Convolutional Neural Networks (CNNs) whose activated areas serve as the regions with highest correlation to the models' decisions or the objects that need to be identified in the input images [24, 25, 26]. Those method can be further extended to include an explanation structure, such as a Recurrent Neural Network (RNN), for both visual and language description [27]. These techniques are currently most suited for machine vision applications such as image classification and object detection.

The latter approach extracts the rules governing the mapping between the inputs and outputs of the models without modifying the original operation of the networks [28]. These rule extraction algorithms are flexible and some are general and can apply to any black-box model. They generate rules that can be translated easily into a human-understandable language. Extracting knowledge from neural networks has been practiced in many applications such as classifying products in industries [29] as well as

business analysis [15]. We will expand on these algorithms below.

2.3. Rule Extraction from Artificial Neural Networks

The usefulness of a rule extraction algorithm depends on multiple criteria. Taha et al. [20] listed different aspects that need to be considered when designing a rule extraction system from ANNs including:

- **Level of detail**: The presentation of information in the explanation based on hypotheses of the system.
- **Comprehensiveness:** The fidelity of rules to represent the knowledge within the black-box system.
- **Comprehensibility:** The property of rule set to identify knowledge of a model's processes. It can be represented by the number of extracted rules and premises in each rule expression.
- **Transparency:** The ability of the rules to be easily inspected in order to inform explanations or decisions.
- **Generalization:** The performance of the extracted rule set on new, unprecedentedly observed data.
- **Mobility:** The ability to apply the rule extraction algorithm to different network architectures.
- **Adaptability:** The ability to modify the set of extracted rules when the networks are updated after a further training session.
- **Theory refinement:** The ability to overcome restrictions due to missing data or inaccurate domain knowledge.
- **Robustness:** The insensitivity of the extracted rules to the noise in the training data.

- **Computational complexity:** Demand of computational resources for extracting the set of rules and the execution of inference relative to the size and number of attributes in the dataset.

- **Scalability**: The ability of the rule extraction algorithm to scale corresponding to the change of problem complexity and network structure.

These criteria serve as a guideline to design evaluation metrics of rule extraction models. In this paper, the criteria of *comprehensiveness*, *comprehensibility*, *transparency*, *generalization*, and *adaptability* with different rule extraction techniques are investigated.

Various categories of rule extraction algorithms are reported in the literature. One of the earliest classification frameworks was based on different features that rule extraction methods exhibit including: (1) expressive power of the extracted rules, (2) transparency of the rule extraction method, (3) usage of specialized training scheme, (4) quality of the extracted rules, and (5) computational complexity of the techniques [19]. However, this categorization system seems complex due to the overlap between its elements, such as the strong dependency between expressive power and quality of extracted rules. Another taxonomy, called *Input-Network-Training-Output-Extraction-Knowledge*, divided clearly the techniques based on modules of the classification frameworks [30]. Following this type of taxonomy, it is simpler for system designers to select or design the rule extraction algorithms with clear requirements for each component.

In this paper, we will follow Hruschka et al. [31]'s classification due to its popularity and wide acceptance. They divide rule extraction techniques into three categories: **pedagogical (black-box rule extraction)**, **decompositional (link rule extraction)**, and **eclectic (hybrid)** techniques.

Pedagogical approaches are data-driven and only find the direct mapping between the inputs and outputs of the ANN using some machine learning techniques. This set of methods does not reach inside the black-box to find the real links within the neural networks. Quinlan's C4.5 [32] is one of the most popular algorithms for building a tree representation that utilizes a discrimination process over different data attributes to maximize the information gain ratio. The C4.5 decision tree is commonly used for ex-

tracting rules from neural networks. Decision Detection by Rule Extraction (DEDEC) is another rule extraction technique that ranks the input attributes of the input data relative to the outputs of the ANN [33]. These rankings are then used to cluster the input space and produce a set of binary rules describing the relationships between data attributes in each cluster and the ANN's outputs. Other notable examples of black-box rule extraction approaches are BRAINNE [34], Rule-extraction-as-learning [35], TREPAN [36] and BIO-RE [20]. The strengths of these approaches are they offer a fast, simple rule set with high transparency and scalability. However, the extracted rules might not be comprehensive and or able to generalize to the test data in various domains.

Decompositional rule extraction techniques consider the links between layers of an ANN (the weights and activation at each hidden and output nodes) to compose the rules. These approaches generally describe more accurately the input-output relationship than *pedagogical* approaches. Most techniques in this class consist of two main stages including searching for the weighted sum of the input links for the activation of each hidden node and then producing a rule with inputs as premises. Typical examples of this class of techniques are SUBSET [37], KT [38], NeuroRule [39], NeuroLinear [40], rule extraction by successive regularization [41], and Greedy Rule Generation [42]. Towell et al [37] developed a well-known rule-extraction algorithm called MofN that can address the limits of SUBSET in terms of binary inputs, scalability, and repetition of extracted rules and achieve higher fidelity compared to some other black-box and link rule extraction methods. RuleNet [43] and RULEX [44] are decompositional techniques that were specialized for ANNs with localized hidden units. While RULEX extracts rules from a Constrained Error Back-Propagation (CEBP) network whose hidden nodes are localized in a bounded area of the training samples, RuleNet extracts binary rules from a mixture of experts trained on a localized ANN. The extracted rules using these approaches are much more comprehensive, but complicated if the number of attributes or input nodes of the ANN is large. The decompositional approaches face some challenges in the transparency, computational complexity, and scalability when applying for large networks.

In this paper, we compare the generalization capability, comprehensiveness, and

transparency of three rule extraction methods. C5.0 decision trees (Pedagogical) is selected as a baseline method. We proposed an algorithm to exactly convert a neural network into a decision tree (Decompositional) and an Extended C-Net algorithm, an *eclectic/hybrid* method combining the strengths of both methods above.

2.4. Multi-variate Decision Trees

Decision Trees are interpretable representations able to approximate the underlying functions that the ANNs represent. Using Decision Trees [45, 46] to approximate the input-output relationship of a neural network is a popular practice because of the ease when converting a decision tree to a set of simple rules.

Univariate decision trees are limited to produce a set of rules each composed of multiple constraints considering a single data attribute at a time. They rely on axis-parallel decision hyperplanes to approximate the decision boundary, which results in a huge model when the decision boundary is oblique and/or highly nonlinear [47]. These drawbacks limit their capability to produce succinct explanations.

A multivariate decision tree can generate a rule expression in terms of a combination of multiple data attributes as inputs. OC1 [48] is one of the earliest to build a multivariate decision tree. OC1 searches for optimal set of weights on all data attributes and tries different weighted sums of input attributes to find the best local decision boundaries. Sok et al. [49] modify a univariate alternating decision tree algorithm into a multivariate one by proposing three approaches to weight multiple attributes and replace the base of univariate conditions by combinations of multivariate ones. The resultant trees demonstrate significantly better accuracy while maintaining acceptable comprehensiveness in comparison with their univariate counterparts and ensembles of univariate decision trees. A PCA-partitioned multivariate decision trees algorithm is proposed to solve the multi-label classification problems in large scale datasets [50]. The multivariate trees, constructed by using the maximum eigenvalue of PCA to choose the weights of each variable in combination, produce a high accuracy.

In the next sections, we describe our proposed multivariate decision trees built with the decompositional rule extraction and hybrid techniques to address the problems of binary and multi-class classification problems with a focus on continuous input data.

3. Conversion of an ANN to a Multivariate Decision Tree Using EC-DT Algorithm

In this section, we propose a multivariate tree algorithm that transforms the ANN structure into an equivalent decision tree. The algorithm is designed to be complete and sound; that is, it preserves 100% performance of ANNs by maintaining the generalizability of the network while providing an interpretable representation. This method is a *decompositional extraction* approach.

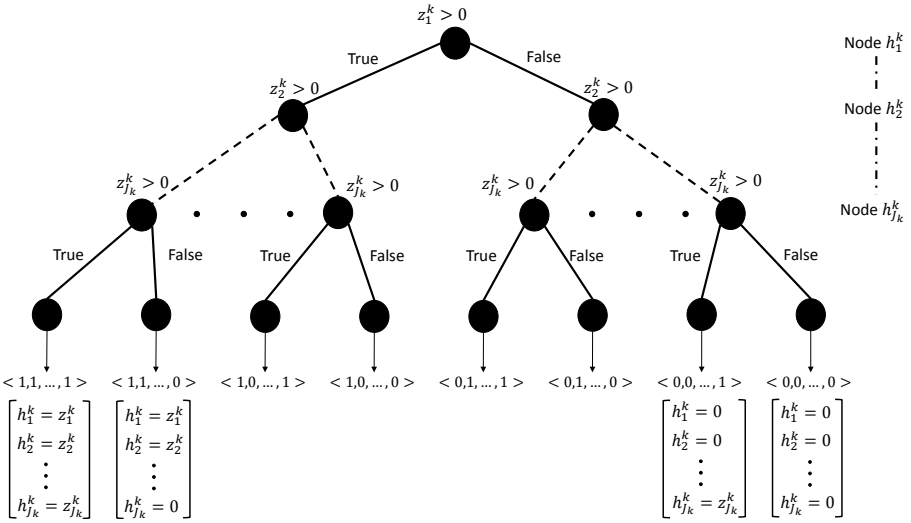


Figure 1: Binary EC-DT representing the output of a hidden layer with ReLU activation function.

In our study, the activation functions of hidden layers are Rectified Linear Function (ReLU). The ReLU function has the form:

$$\sigma(f) = \begin{cases} 0, & \text{for } f \leq 0 \\ f, & \text{for } f > 0 \end{cases} \tag{1}$$

For ReLU, a node is considered activated if its output is greater than 0. We might construct a decision tree to represent the constraint by which the activation of each node in the hidden layers is decided. Denote z_{j_k} as the value of hidden node j_k in hidden layer k before applying the activation function, that is

$$z_{j_k}^k = \sum_{j_{k-1}=1}^{J_{k-1}} w_{j_{k-1}j_k} H_{j_{k-1}}^{k-1} + \beta_{j_k}^k \tag{2}$$

For nodes in the first hidden layer, we have alternatively

$$z_{j_1}^1 = \sum_{i=1}^{I} w_{ij_1} x_i + \beta_{j_1}^1 \tag{3}$$

Therefore, we can represent the overall neural network by a binary tree with $2^{\sum_{l=1} KJ_l}$ leaves for a network with K hidden layers. The size of the tree grows exponentially as the network size grows; a drawback we will fix below.

Due to the nature of ReLU activation as described in Equation 1, the output of this activation function is either 0 or a weighted sum of the input to the neural node, which we called the excited or activated node. Thus, each hidden layer of the neural network can be transformed into a binary decision tree with each tree stage deciding the activation of the corresponding node in the hidden layer based on the constraint of whether or not the value before the activation function is greater than 0.

Given a hidden layer of J_K nodes with ReLU activation, the direct translation of this layer to a binary tree can be illustrated in Figure 1. The tree can be considered a multi-target tree with the output being a binary vector of the active nodes (value of 1) and disabled node (value of 0). In the case that a node is activated (satisfying constraint for TRUE branch), the real output of the node after activation is the same as its value prior activation: $h_{J_k}^k = z_{J_k}^k$. That is, we can alternatively replace this activation array by a vector of regression values for each node based on weights.

Algorithm 1 illustrates the steps to build an EC-DT from a Neural Network with ReLU activation function. Each leaf node in the produced tree has a list of true or false branches that starts from the root node all the way to the leaf. As a result, we can produce a list of ANN layers' activations $S = \{S^1, S^2, ...S^K\}$, where S^k is an array representing the activations of hidden nodes in layer k of the ANN. The set of rules can be extracted from the EC-DT tree by converting every leaf node in the tree into constraints given list of hidden nodes'activations, the weights matrices, and biases matrices from ANN (see Algorithm 2). The values of weights and biases of disabled nodes do not contribute to the constraints of the next neural layer. Equivalently, we set the weights of connections from the disable nodes of current layer to zero and then compute the updated weights and biases matrices representing the linear combination

Algorithm 1: EC-DT Tree Generation Algorithm

Input : Number of nodes in hidden layers $\{J_1, J_2, ..., J_K\}$

Output: Decision tree's set of nodes $\mathcal{T}$

Initialize: $\mathcal{T} \leftarrow \varnothing$, current set of parent nodes $\mathcal{P} \leftarrow \varnothing$, current set of child nodes $\mathcal{C} \leftarrow \varnothing$, node tuple $q^{ID_\tau}(id, hidden\_layer, hidden\_node\_id, parent\_id, branch, leaf)$

Set tree node id $ID_\tau = 1$

Set hidden layer $k = 1$.

Set hidden node id $ID_k = 1$.

Create root node $q^r(ID_\tau, k, ID_k, None, None, leaf = False)$.

Add q^r to $\mathcal{T}$, add $q^r.id$ to $\mathcal{P}$.

$ID_\tau \leftarrow 2$; $ID_k \leftarrow 2$.

while $k \leq K$ **do**

  while $ID_k \leq J_k$ **do**

    Set $\mathcal{E}$ the number of elements in $\mathcal{P}$.

    for $i = 1$ **to** $\mathcal{E}$ **do**

      if $k = K$ *and* $ID_k = J_K$ **then**

        `// leaves nodes`

        Create node ; `// true branch`

        $q_1^{ID_\tau}(ID_\tau, k, ID_k, ID_{p_i \in \mathcal{P}}, 1, True)$

        Create node ; `// false branch`

        $q_0^{ID_\tau}(ID_\tau, k, ID_k, ID_{p_i \in \mathcal{P}}, 0, True)$

        Find set of branches $S_1^{ID_\tau}$ and $S_0^{ID_\tau}$ leading to $q_1^{ID_\tau}$ and $q_0^{ID_\tau}$ by tracing back to root.

        $q_1^{ID_\tau}.value \leftarrow S_1^{ID_\tau}$; $q_0^{ID_\tau}.value \leftarrow S_0^{ID_\tau}$

        $ID_\tau \leftarrow ID_\tau + 1$

      else

        Create node ; `// true branch`

        $q_1^{ID_\tau}(ID_\tau, k, ID_k, ID_{p_i \in \mathcal{P}}, 1, False)$

        Create node ; `// false branch`

        $q_0^{ID_\tau}(ID_\tau, k, ID_k, ID_{p_i \in \mathcal{P}}, 0, False)$

        $ID_\tau \leftarrow ID_\tau + 1$

      end

      Add nodes to $\mathcal{T}$.

      Add nodes IDs to $\mathcal{C}$.

    end

    $\mathcal{P} \leftarrow \mathcal{C}$; $\mathcal{C} \leftarrow \varnothing$.

    $ID_k \leftarrow ID_k + 1$.

  end

  $k \leftarrow k + 1$.

end

return $\mathcal{T}$.

between the input variables and the outputs value of next neural layer.

We present a multiplexer problem with binary input as an example to illustrate how our proposed EC-DT algorithm can convert a neural network into a decision tree. Figure 2 introduces a gate that takes two binary inputs X_1 and X_2 and produces a function $Y = XOR(X_1, X2)$. Assume that we have a neural network with weights as shown in Figure 2 and zero biases to represent this function. The EC-DT algorithm converts the neural network into a binary tree. Each layer in the tree includes nodes which represent the constraint for the test of whether a corresponding unit in the ANN is activated or not. The paths from root to leaf represent combinations of activation of hidden units. This results in four cases (with one impossible case) of output value Y. The constraints in each layer and the consequences at the leaves form the rule set of the neural network.

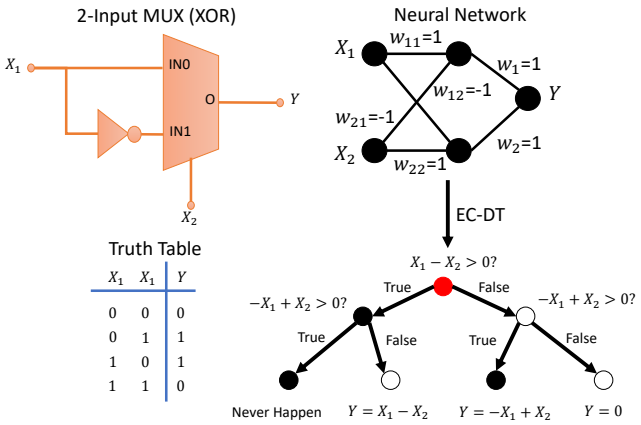


Figure 2: A XOR gate example with binary inputs that can be represented by a neural network which is converted into an EC-DT.

4. Extended C-Net: An approach to learn multivariate decision trees from a neural network

Decision trees have been used with neural networks to build an interpretable model that explains the decision-making process of neural-networks. The C-Net algorithm, proposed by Abbass et al [51], is one of those early algorithms which uses a univari-

Algorithm 2: EC-DT Rule Extraction Algorithm (for a leaf)

Input : A leaf node with its list of branches corresponding to activations of ANN'hidden layers $S = \{S^1, S^2, ...S^K\}$, number of nodes in hidden layers $\{J_1, J_2, ..., J_K\}$, a set of weights matrices of ANN $\mathcal{W} = \{\mathcal{W}^{I1}, \mathcal{W}^{12}, ..., \mathcal{W}^{(K-1)K}, \mathcal{W}^{KY}\}$, and a set of biases matrices of ANN $\mathcal{B} = \{\mathcal{B}^1, \mathcal{B}^2, ..., \mathcal{B}^K, \mathcal{B}^Y\}$

Output: A rule/set of constraints and consequences $\mathcal{R}$

Set $\mathcal{R} \leftarrow \varnothing$.

for $k = 1$ **to** K **do**

  for $s = 1$ **to** J_k **do**

    if $S^k(s) = 0$ **then**

      Convert matrix form $X\mathcal{W}^{Ik}_{*,s} + \mathcal{B}^{Ik}(s) > 0$ into linear inequation form.

      Elements in s^{th} row of $\mathcal{W}^{k(k+1)}$ are set to 0.

    else

      Convert matrix form $X\mathcal{W}^{Ik}_{*,s} + \mathcal{B}^{Ik}(s) \leq 0$ into linear inequation form.

    end

    Add inequation to $\mathcal{R}$ as constraint.

  end

  if $k = K$ **then**

    Compute $\mathcal{W}^{IY} = \mathcal{W}^{IK}\mathcal{W}^{KY}$.

    Compute $\mathcal{B}^{IY} = \mathcal{B}^{IK}\mathcal{W}^{KY} + \mathcal{B}^{Y}$.

  else

    Compute $\mathcal{W}^{I(k+1)} = \mathcal{W}^{Ik}\mathcal{W}^{(k(k+1)}$.

    Compute $\mathcal{B}^{I(k+1)} = \mathcal{B}^{Ik}\mathcal{W}^{k(k+1)} + \mathcal{B}^{k+1}$.

  end

end

Convert matrix form $\mathcal{Y} = X\mathcal{W}^{IY} + \mathcal{B}^{IY}$ to equations.

Add equations to $\mathcal{R}$ as consequences.

return $\mathcal{R}$.

ate decision tree (UDT) to generate a multivariate decision tree (MDT) from neural networks. In this paper, we propose a modification of the algorithm into an extended version of C-Net to extract the rules from ANNs. We describe the original algorithm and the modification below, where the Extended C-Net Algorithm is adopted to the specific use of the ReLU activation functions in the hidden layers.

After the ANN is trained, new data set are introduced and the outputs of the last hidden layer are computed. We may split the new data set into training and testing sets for the purpose of training the decision tree. Therefore from a set of training and test data, denoted as $\langle \mathbf{X}_{training}, \mathbf{O}_{training} \rangle$ and $\langle \mathbf{X}_{testing}, \mathbf{O}_{testing} \rangle$ respectively, we can compute the mapping between the last hidden output layer and the output, denoted as $\langle \mathbf{H}^K_{training}, \mathbf{O}_{training} \rangle$ and $\langle \mathbf{H}^K_{testing}, \mathbf{O}_{testing} \rangle$.

We use the data representing the relationship between the last hidden layer and the output of the network to train a C5 decision tree whose algorithm adapts an entropic information gain ratio for branch-splitting criterion and has been demonstrated to be more accurate, and less memory intensive [52].

4.1. Back Projection of Extended C-Net

After the ANN is trained, we use the output of the last hidden layer $\mathbf{H}^K(\mathbf{X^t})$ and the class prediction $\mathbf{O}^t$ to be the input and output of the UDT layers, respectively. Figure 3 illustrates the Extended C-Net architecture.

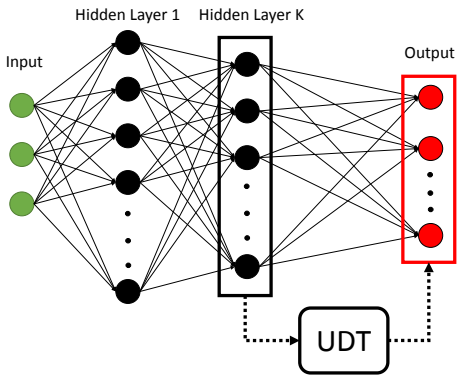


Figure 3: The Extended C-Net framework where C5 algorithm extracts UDT rules between the last hidden layer and the output.

Commonly, a DT can be represented by a set of polyhedrons expressed in the form of linear constraints. Every constraint as learned by the DT has the form of $H_{j_K}(\mathbf{X^t})$ *op* C_{j_K}, in which *op* represents the binary operators $\{\leq, <, =, >, \geq\}$, and C_{j_K} is the numeric threshold of such a constraint on input $H_{j_K}(\mathbf{X^t})$. It is needed to back-project from the output of the neural network to the input of the neural network to obtain a multivariate form of the expression. Extended C-Net adopts the inverse of the used activation function, that is

$$(\sum_{j_{K-1}=1}^{J_{K-1}} w_{j_{K-1}j_K} H_{j_{K-1}}^{K-1}(\mathbf{X^t}) + \beta_{j_K}^K) \quad op. \quad \sigma_K^{-1}(C_{j_K}) \tag{4}$$

In our ANN, the activation functions of hidden layers are ReLU. We chose ReLU due to their linear simplicity and popularity within ANN. Hence the inverse of this function is

$$\sigma^{-1}(f) = \begin{cases} f, & \text{for } f > 0 \\ \eta, & \text{for } f \leq 0 \end{cases} \tag{5}$$

where η is an arbitrary negative number. ReLU is partially invertible; hence, it is a challenge to find the mathematical expression relating the output back to the input. Some of the nodes might not be activated resulting in a value of zero, which then might not appear in the mathematical combination expression for nodes of the next level. The expression is dynamic at the output depending on the value of inputs.

The input-output mapping is represented by the following expression in case of two hidden layers activated by ReLU, in which the outputs of the second hidden layer are fed as inputs to the UDT:

$$\sigma(\sum_{j_1=1}^{J_1} w_{j_1 j_2} \sigma(\sum_{i=1}^{I} w_{i j_1} x_i + \beta_{j_1}^1) + \beta_{j_2}^2) \quad op. \quad C_{j_2} \tag{6}$$

A node is considered active if its output is greater than 0. We could rewrite the non-linear constraint into one linear constraint on the activation function level, and a second linear constraint on an index of the former. We note that a constraint on the index is mathematically too complex, but we will compensate that by using this constraint as a filter.

$$\sum_{j_1} \sum_{i=1}^{I} w_{j_1 j_2} w_{i j_1} x_i + w_{j_1 j_2} \beta_{j_1}^1 + \beta_{j_2}^2 \quad op. \quad C_{j_2} \tag{7}$$

rewritten as

$$\sum_{j_1}\sum_{i=1}^{I} w_{j_1 j_2} w_{i j_1} x_i \quad op. \quad C_{j_2} - (w_{j_1 j_2}\beta_{j_1}^1) + \beta_{j_2}^2 \tag{8}$$

with additional constraints:

$\forall j_1$ satisfying $\sum_{i=1}^{I} w_{i j_1} x_i + \beta_{j_1}^1 > 0$

The multivariate decision tree Extended C-Net is a rewrite of the ANN into interpretable logic using nodes/conditions and branches/information-flows. Each rule induced by each leaf of the resultant Extended C-Net tree can be traversed back to weights of the network to deduce the input-target relationships as represented by the network and its weights. The Extended C-Net algorithm takes advantage of the EC-DT tree generation algorithm, but with leaves produced earlier at layer $K-1$ and target values of those leaves are linear combinations of input X as hidden nodes' outputs at layer K. The procedures of Extended C-Net algorithm can be described as follows:

- **Step 1:** Feed inputs to ANN and compute the values at final hidden layer's nodes $H_{j_K}^K$.
- **Step 2:** Use $H_{j_K}^K$ values and labels of instances to train a UDT (C5 decision trees) and extract rules from UDT tree.
- **Step 3:** Build a decision tree using algorithm S1 (Supplementary Document, Section S.1) until hidden layer $K-1$.
- **Step 4:** Use Algorithm 3 to extract final set of rules for every leaf.

Due to the use of both a data-driven method to build UDT tree and a decompositional method to build the decision tree that representing the constraints between inputs and the last hidden layers, Extended C-Net algorithm belongs to *eclectic/hybrid* approach. An example of how the Extended C-Net algorithm works can be found in Section S.1 in Supplementary document.

Algorithm 3: Extended C-Net Rule Extraction Algorithm (for a leaf)

Input : A leaf node with its list of branches corresponding to activations of ANN'hidden layers $S = \{S^1, S^2, ...S^{(K-1)}\}$, number of nodes in hidden layers $\{J_1, J_2, ..., J_{(K-1)}\}$, a set of weights matrices of ANN $\mathcal{W} = \{\mathcal{W}^{I1}, \mathcal{W}^{12}, ..., \mathcal{W}^{(K-1)K}\}$, a set of biases matrices of ANN $\mathcal{B} = \{\mathcal{B}^1, \mathcal{B}^2, ..., \mathcal{B}^K\}$, and a set of constraints from UDT's leaf in form: $\mathbf{H^K_{j_K}}$ **op** $\mathbf{C_{j_K}}$ $\forall j_K \in \Upsilon$; $\Upsilon \subseteq \{1, 2, ..., J_K\}$

Output: A rule/set of constraints and consequences $\mathcal{R}$

Set $\mathcal{R} \leftarrow \varnothing$.

for $k = 1$ **to** $K - 1$ **do**

  for $s = 1$ **to** J_k **do**

    if $S^k(s) = 0$ **then**

      Convert matrix form $X\mathcal{W}^{Ik}_{*,s} + \mathcal{B}^{Ik}(s) > 0$ into linear inequation form.

      Elements in s^{th} row of $\mathcal{W}^{k(k+1)}$ are set to 0.

    else

      Convert matrix form $X\mathcal{W}^{Ik}_{*,s} + \mathcal{B}^{Ik}(s) \leq 0$ into linear inequation form.

    end

    Add inequation to $\mathcal{R}$ as constraint.

  end

  if $k = K - 1$ **then**

    Compute $\mathcal{W}^{IK} = \mathcal{W}^{I(K-1)}\mathcal{W}^{(K-1)K}$.

    Compute $\mathcal{B}^{IK} = \mathcal{B}^{I(K-1)}\mathcal{W}^{(K-1)K} + \mathcal{B}^K$.

  else

    Compute $\mathcal{W}^{I(k+1)} = \mathcal{W}^{Ik}\mathcal{W}^{(k(k+1)}$.

    Compute $\mathcal{B}^{I(k+1)} = \mathcal{B}^{Ik}\mathcal{W}^{k(k+1)} + \mathcal{B}^{k+1}$.

  end

end

Convert matrix form $H^K = X\mathcal{W}^{IK} + \mathcal{B}^{IK}$ to linear equations.

for $j_K \in \Upsilon$ **do**

  Add the following constraint to $\mathcal{R}$:

  $w_{1(j_K)}x_1 + w_{2(j_K)}x_2 + ... + w_{M(j_K)}x_M$ *op.* $(C_{j_K} - \mathcal{B}^{IK}_{j_K})$.

end

Add UDT rule's consequence to $\mathcal{R}$ as consequence.

return $\mathcal{R}$.

5. Experiments

This section starts by describing three datasets on which we conduct experiments. The datasets describe increasingly complex problem spaces.

5.1. Synthetic Dataset

We start by constructing a feedforward network for binary classification with two input attributes. An artificial dataset is generated with polynomial relationships between the attributes. The polynomial relationship varies in complexity from one dataset to another to control the level of non-linearity in the decision boundary. A UDT may need to grow arbitrarily large before it is able to approximate a highly nonlinear function properly. We will discuss the generalization and misclassification tolerance of decision trees and our method in Section 6. The classification problem is for the ANN to estimate the classes with the data distributed as

$$y = \begin{cases} 1, & \text{for } x_1^2 + x_2^2 \geq 5 \\ 0, & \text{for } x_1^2 + x_2^2 < 5 \end{cases} \tag{9}$$

The artificial data attributes, x_1 and x_2, are sampled uniformly within the interval of $[0, \sqrt{6}]$. Data are also sampled subject to the constraint $4 \leq x_1^2 + x_2^2 \leq 6$ so that the samples concentrate around the decision boundary. As the level of non-linearity increases, the concentration of the data increases the probability of an error due to a coarse approximation of the decision boundary.

In this problem (called the P2 problem), 10 data sets are randomly sampled each with 5000 data points. Each set is then split into training and test subsets according to the ratio 80:20. Every training subset is shuffled 10 times. Each shuffled subset is used to train a ANN. A trained model is tested with the corresponding test subset. In total, the number of trials are $10 \times 10 = 100$ models.

The ANNs use two hidden layers with ReLU activation functions and one output unit with a sigmoid function. The number of nodes for each hidden layers is 5. The network is trained with a learning rate of 0.001, 500 epochs, and mini-batch size of 128. After a network is fully trained, decision trees using C5, Extended C-Net and EC-DT are generated to represent the network. The trees are then pruned with a pessimistic

pruning algorithm identical to the one used for the C5 algorithm described in Quinlan's work [53].

5.2. Benchmarking Datasets: UCI Data

We further apply our proposed methodology on benchmark datasets obtained from the UCI repositories [54]. Some properties of every data set used in this study can be found in Table S.4 (Supplementary Document, Section S.2). In each problem, one ANN is built with 2 hidden layers each with 5 hidden nodes. The number of outputs of the ANN for binary classification is 1 with sigmoid activation functions while the number for multi-class classification is the number of output classes with softmax activation functions. The training and testing schemes are similar to the process used for the P2 problem.

5.3. Autonomous Control Problem: Shepherding Task

The shepherding problem is the problem of using a sheepdog to collect sheep within a paddock and herd them toward a target position. Strömbom et al [55] investigated a model that represents the interactions between the sheep and the shepherd based on simple rules. This problem motivates a form of control method for an autonomous swarm system. For example, these interactions can be used as a mode of control for unmanned aerial vehicles (UAVs) and unmanned ground vehicles (UGVs), for search-and-rescue, surveillance, and environmental management studies [56, 57, 58, 59].

A shepherding agent uses two main forces to characterise its movement, called collecting and driving forces. The interactions among the sheep, and between the sheep and the sheepdog are described in Section S.3 (Supplementary Document). The parameters of the simulation environment are described in Table S.5 (Supplementary Document). For this problem, we implement a reinforcement learning model called Deep Q-Network to train two neural network models that produce the collecting (shepherding-c) and driving (shepherding-d) behaviours respectively. Each NN includes 2 hidden layers each with 5 nodes. The number of inputs are 4 representing the observed states of the shepherding agent, which are described in detail in Section S.4 (Supplementary Document).The number of outputs are 4 corresponding to 4 directions of movement

(north, south, east, and west). The detailed training parameters of the reinforcement learning algorithm and the training and testing procedure are listed in Table S.6 and Section S.4 (Supplementary Document). For each behaviour, we train 10 NN models, from each 10 rule models are extracted with three tree extraction models.

The shepherding task cannot be evaluated by using the same metrics as for the classification problems. Therefore we introduce the *fidelity*, the *success rate*, and the *number of steps* to complete the task to compare the performance of different rule extraction algorithms.

Later in Section 6.4, we transform the set of rules representing the neural network into an interpretable visualization to improve the transparency of the decision making processes.

5.4. Performance Metrics

To assess the performance of our decision tree approach and the baseline Extended C-Net and C5 algorithms, we introduce three types of metrics:

- **Performance:**
  - *Accuracy (Classification)*: In classification problems, we first measure the accuracy of the neural network for classification on test data. The extracted decision trees are also tested for their prediction accuracy on the same test set. This metric reflects the accuracy of the DT algorithm when approximating the function learned by the ANN.
  - *Fidelity* (Shepherding task): In a shepherding problem, after extracting the rules from the trained neural networks with three extraction algorithms, we sample random data whose dimensions are constrained within the limits defined by the size of the environment to represent possible states that an agent may encounter. The ratio of the number of matches between the outputs of a rule model and the outputs of the original neural network over the total number of samples is the *fidelity* of the such rule model. A fidelity of 1 or 100% means a perfect conversion from the neural network into rules.

  - *Success rate* and *number of steps* (Shepherding task): After the rule model are extracted. They are evaluated with 100 random test cases to acquire the task success rate in percentage. The number of steps to successfully perform the task indicates the efficiency of a model.

- **Compactness**: the capability of the algorithm to represent information with the smallest model size. For decision trees, this could be measured by the number of leaves or size of the tree. In this paper, we also examine the average number of constraints within one leaf. One tree may be constructed with a low number of leaf nodes. However, the final complexity of rule interpretation also depends on the number of constraints under each leaf.

- **Rule interpretation**: We also investigated the decision boundaries generated by the decision trees. This could be implemented by first converting the trees into sets of rules, and second visualizing the hyper-planes corresponding to the constraints in rules on the data space.

6. Results and Discussion

6.1. Performance

The predictive accuracy of the 100 base neural networks and the corresponding rule models extracted from those networks for the P2 problem are shown in Table 1. Among all rule extraction methods for ANNs, the EC-DT can maintain exactly the same performance of ANNs as expected. For the Pruned C5 and Extended C-Net trees, we can observe the decrease of accuracy compared to the original performance of ANNs, by approximately 0.1% and 1.8% respectively. Extended C-Net preserves the performance of ANN better when compared to the baseline C5 extraction method. It is important to emphasize that all transformations are deterministic, thus, variations of performance from the original ANN are not due to any stochastic variations.

The classification accuracy for UCI datasets follow a similar trend where EC-DT captures the accuracy of ANNs perfectly. Table 2 summarizes the mean and standard deviations of the predictive accuracy of the decision trees extracted from 100 different trained ANNs on each problem. In a majority of data sets, the null hypothesis of

Table 1: Accuracy of C5, Extended C-Net, and EC-DT models extracted from ANNs for the P2 problem.

ANN Accuracy ($\mu \pm \sigma\%$)	C5 ($\mu \pm \sigma\%$)	Extended C-Net ($\mu \pm \sigma\%$)	EC-DT ($\mu \pm \sigma\%$)
93.81±5.45	91.32±10.17	92.02±10.32	93.81±5.45

the significant difference between the accuracy of EC-DT and the other methods is rejected with significance level of 0.05 or 0.01 (ANOVA and two-sample t-test). For the Extended C-Net algorithm, the accuracy of the generated multivariate trees are higher compared to the performance of C5 trees in seven datasets (*banknote*, *wdbc*, *balance*, *new-thyroid*, *wine*, *wifi*, and *satimage*) by 1-5% in average, and are equivalent to EC-DT in some cases (*banknote* and *wifi*). The performance of Extended C-Net is equivalent to C5 on 7 other datasets (*skin*, *occupancy*, *climate*, *ionosphere*, *wall-following-2*, *segment*, and *ecoli*). The uses of multivariate constraints between attributes of the input data can better form the necessary decision hyperplanes for classifying the output. Another interesting observation is that an increase in the number of attributes and the number of samples of datasets contributes to the deterioration of C5 and Extended C-Net's accuracy.

Regarding shepherding problems, Table 3 summarizes the mean and standard deviations of the fidelity of 100 decision trees (for each method) extracted from 10 trained ANNs for collecting behaviour and 10 trained ANNs for driving behaviour. The results suggest that the EC-DT algorithm can generate rule models with 100% fidelity. Therefore the EC-DT tree models can reproduce perfectly the performance of the ANNs for both shepherding sub-tasks.

Although the fidelities of Extended C-Net and C5 trees are only 1.7% and 7.3% lower than that of EC-DT trees in average, the performance are significantly different when comparing those models to one another on 100 test scenarios where both collecting and driving behaviours are combined together. EC-DT achieves 100% success rate when performing the combined task compared to only 70% and 98% of C5 and

Table 2: Accuracy of pruned trees generated from C5, Extended C-Net and EC-DT algorithms for UCI benchmarks.

Data set	**NN Accuracy** ($\mu \pm \sigma\%$)	**Rule Extraction Algorithm for NN**		
		C5 ($\mu \pm \sigma\%$)	**Extended C-Net** ($\mu \pm \sigma\%$)	**EC-DT** ($\mu \pm \sigma\%$)
skin	**94.43±8.93***	86.77±21.80	86.79±21.81	**94.43±8.93***
banknote	**99.84±0.49**	98.16±0.82	**99.74±0.56**	**99.84±0.49**
occupancy	98.86±0.15	98.85±0.15	98.86±0.16	98.86±0.15
wdbc	**94.57±1.88***	92.42±2.46	93.61±2.18	**94.57±1.88***
climate	**91.01±3.02**†	84.55±14.75	84.59±15.29	**91.01±3.02**†
ionosphere	**93.11±3.38**†	91.64±6.64	89.10±6.68	**93.11±3.38**†
balance	**95.12±3.22***	77.95±3.33	92.30±3.59	**95.12±3.22***
new-thyroid	**97.12±2.98**†	90.53±7.69	95.42±7.41	**97.12±2.98**†
wine	**97.13±2.29***	92.78±5.30	95.25±3.59	**97.13±2.29***
wall-following-2	**97.82±2.84**†	95.78±9.06	95.30±8.92	**97.82±2.84**†
wall-following-4	97.59±1.90	97.30±6.06	96.45±5.94	97.59±1.90
wifi	**97.61±1.17**	96.99±0.88	**97.38±1.21**	**97.61±1.17**
dermatology	**95.34±3.70**	**95.05±3.40**	93.85±3.54	**95.34±3.70**
satimage	**89.21±1.01***	85.05±1.15	86.64±1.05	**89.21±1.01***
segment	**95.44±1.35**	94.65±1.60	94.10±1.60	**95.44±1.35**
ecoli	**89.66±4.85***	80.55±5.06	80.39±5.53	**89.66±4.85***

Figures in **bold** are the best among methods.

† significantly better than its counterparts at significant level of 0.05.

* significantly better than its counterparts at significant level of 0.01.

Extended C-Net respectively. As illustrated in Table 4, the average number of steps in successful mission using EC-DT method is also lowest. Even though the number of steps of rule models extracted using Extended C-Net is better than the figures of C5, both methods show a high variance in the metrics when performing the task with as high as 3000-4000 steps in some cases. This instability and low efficiency are due to the lower fidelity of C5 and Extended C-Net compared to EC-DT. In control problems like shepherding, an agent is required to output a series of actions in which even a few sub-optimal decisions might dramatically reduce overall performance.

Table 3: Fidelity of pruned trees generated from C5, Extended C-Net and EC-DT algorithms for shepherding problem.

Sub-task	**Rule Extraction Algorithm for NN**		
	C5	**Extended C-Net**	**EC-DT**
	$(\mu \pm \sigma\%)$	$(\mu \pm \sigma\%)$	$(\mu \pm \sigma\%)$
shepherding-c	94.16±0.16	99.45±0.07	**100.00±0.00***
shepherding-d	92.67±0.12	98.34±0.10	**100.00±0.00***

Figures in **bold** are the best among methods.

* significantly better than its counterparts at significant level of 0.01.

Table 4: Performance of pruned trees generated from C5, Extended C-Net and EC-DT algorithms for combined shepherding test runs.

Rule extraction algorithm	**Test metrics**	
	Success rate (%)	**Number of steps in successful missions**
C5	70	608.96±638.93
Extended C-Net	98	542.66±468.40
EC-DT	100	399.46±251.60

6.2. Compactness

The analysis of the sizes of the decision trees extracted from the neural networks provides us the information on the compactness achieved by the proposed methods compared to the basic C5 rule extraction algorithm. As mention earlier, the number of leaves in a tree is one measure of compactness; however, the interpretability of rules converted from the decision trees is also influenced by the number of constraints that forms each rule. A higher number of constraints in each rule can make the interpretation of it becoming more complex, and vice versa.

For the P2 problem, it can be demonstrated from the data in Table 5 that the number of rules or leaves extracted by Extended C-Net or EC-DT are significantly lower than the number of rules extracted by C5 algorithm. Extended C-Net among all methods achieves the lowest number of leaves on average (significant at the level) in the P2 problem. The number of leaves in C5 method is 3-5 times higher than the Extended C-Net, which implies that a much greater number of decision hyperplanes is used for the classification problem.

The constraints that one can find on average at each leaf of the C5 trees, on the other hand, is much lower than the others. With less than 2 constraints per leaf, the null hypothesis of the difference between the number of constraints per leaf in C5 and Extended C-Net is significant (ANOVA and two-sample t-test in both one and two tails with significant level of 0.01). The reason for this comes from the requirement for a number of node activation rules at the first hidden layer of the neural network so that the Extended C-Net can represent the input-output constraints by propagating back to the conditions of the input layer from the output layer. Due to this requirement, given a deeper neural network, the Extended C-Net will cost more constraints, as a trade-off for more accuracy. For the P2 problem, however, the total number of constraints per tree generated from 100 ANNs are lower than C5 which indicates less complex trees (see Table S.7, Section S.5). It is also interesting to note that the mean total number of constraints for EC-DT in one tree is lower than the one in C5, but for an increase in the number of nodes in the network which offers higher accuracy, the number of constraints per leaf increases linearly.

Extended C-Net also achieves the lowest number of leaves (or rules) in most UCI

Table 5: Means and standard deviations of sizes of C5, Extended C-Net, and EC-DT trees to represent ANN behaviour for P2, UCI and shepherding problems.

Data set	C5 (pruned)		Extended C-Net (pruned)		EC-DT (pruned)	
	# leaves (L)	#constraints per leaf (C)	# leaves (L)	#constraints per leaf (C)	# leaves (L)	#constraints per leaf (C)
P2	38.73±11.87	**1.91±0.16***	**11.12±6.00**†	6.69±1.05	13.40±6.35	11.00±0.01
skin	48.25±30.77	**2.59±0.98***	**7.65±8.49***	5.30±2.59	28.93±20.48	11.00±0.02
banknote	12.07±2.36	**2.33±0.10***	**3.91±1.33***	6.46±0.27	45.72±9.71	10.99±0.01
occupancy	2.52±1.35	**1.18±0.41***	**2.05±0.41***	6.01±0.09	13.50±7.36	11.00±0.02
wdbc	8.90±1.58	**2.11±0.25***	**3.08±1.15***	6.28±0.28	11.39±4.52	11.00±0.01
climate	6.92±3.19	**1.95±0.76***	**2.87±1.35***	5.46±1.97	16.96±11.85	11.00±0.01
ionosphere	7.59±1.58	**1.65±0.48***	**2.92±1.28***	6.13±0.79	28.01±10.16	11.00±0.01
balance	18.98±4.37	**2.85±0.27***	**9.19±4.06***	7.26±0.41	20.39±11.09	10.00±0.01
new-thyroid	4.73±0.97	**1.55±0.36***	**2.97±0.36***	6.27±0.77	9.53±3.53	10.00±0.02
wine	4.51±0.50	**1.72±0.22***	**3.00±0.00***	6.48±0.17	27.39±4.84	9.99±0.02
wall-following-2	**9.92±3.99***	**2.08±0.56***	18.87±9.20	7.23±0.98	12.15±4.61	9.00±0.02
wall-following-4	**11.38±4.23***	**2.35±0.45***	21.71±10.39	7.63±0.45	16.50±6.49	9.00±0.01
wifi	13.96±3.22	**2.83±0.30***	**8.09±1.83***	7.15±0.24	14.50±5.56	9.00±0.01
dermatology	8.06±0.87	**2.68±0.20***	**6.29±0.55***	7.36±0.81	27.96±8.19	8.99±0.02
satimage	70.83±6.76	**4.96±0.21***	**38.62±6.93***	8.53±0.21	80.86±18.63	8.99±0.01
segment	**24.20±3.42**	**3.46±0.33***	**23.60±4.47**	8.29±0.30	37.85±11.71	8.99±0.01
ecoli	10.67±2.09	**2.80±0.33***	**8.88±1.96***	7.40±0.30	13.84±6.48	9.00±0.01
shepherding-c	791.30±26.04	4.98±0.15	**136.50±9.56***	**3.69±0.09***	232.00±28.40	9.00±0.01
shepherding-d	1098.00±41.46	5.31±0.17	347.00±18.26	**4.06±0.11***	**282.00±51.28***	9.00±0.01

Figures in **bold** are the best among methods.

† significantly better than its counterparts at significant level of 0.05.

* significantly better its counterparts at significant level of 0.01.

problems (Table 5). The null hypotheses means that differences between the number of leaves generated with Extended C-Net and the ones generated from other methods are rejected at significance level 0.01 for most datasets. The differences between Extended C-Net tree sizes and C5 tree sizes, according to the figures, varies dramatically from around 1 (e.g.*occupancy* and *wine*) up to 40 (e.g. *skin*) depending on the complexity and nonlinearity of the problem. In cases with such large differences, the accuracy of the Extended C-Net are also equivalent or better than the accuracy of C5. This is due to the capability of Extended C-Net to generate multivariate trees for better generalization of the problem which is not easily achieved with a UDT such as C5.

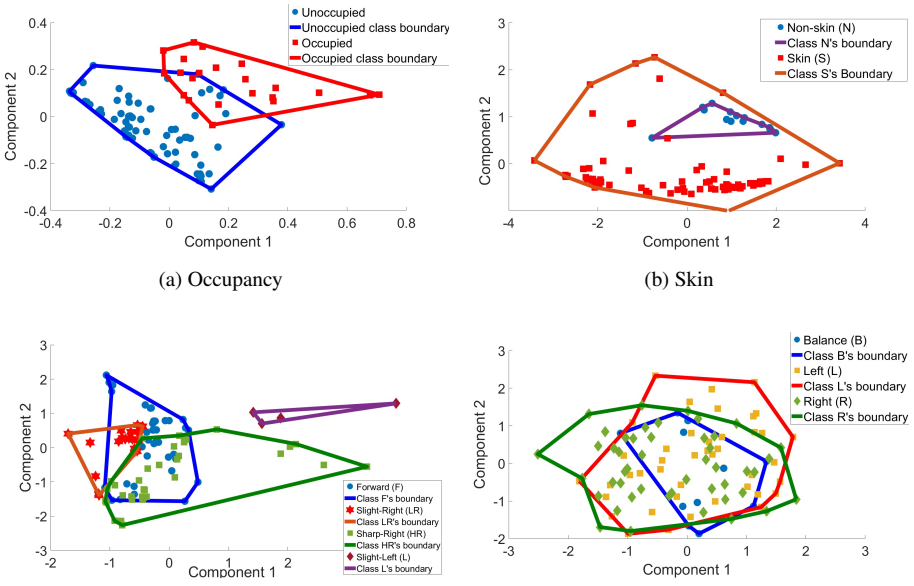
Similar trends of low numbers of constraints per leaf for a C5 tree can be illustrated in the same Table where in most cases the figures are less than 3 constraints per leaf. The low number of constraints per leaf reflects the simplicity of using only some of the most relevant attributes for rule extraction. The use of less attributes cannot achieve much generalization power given the axis-parallel nature of C5 hyperplanes. However, for some problems such as *occupancy*, *wall-following-2*, *wall-following-4*, *dermatology*, *segment*, and *ecoli*, it is noticeable that even with a much lower number of constraints per leaf leading to the lower number of total constraints per tree on average, the C5 algorithm still shows more effectiveness compared to Extended C-Net (though both are not comparable to EC-DT in terms of generalization power). EC-DT generates a comparable size of trees to the C5 ones, but with a huge number of constraints per tree leaf.

For *shepherding-c* and *shepherding-d* problems, C5 trees, however, need an average number of leaves of 3 to 4 times larger than those of EC-DT or Extended C-Net models. This is due to the fact that for autonomous control problems like shepherding, the representative state space can be much more complicated than the problem space of some conventional classification problems. For shepherding problem, the state space can be segmented into a huge number of smaller sub-spaces based on distance or direction between the shepherd and the sheep. Each of those sub-space might include all possible output classes whose distributions overlapping with each other. Therefore, an algorithm that creates axis-parallel decision polytopes like univariate C-5 is not appropriate to model such complex data distribution. Although, EC-DT provides better

performance for rule models in shepherding problems, Extended C-Net generates lowest number of constraints per each rule/leaf, thus models the shepherding behaviours with much significantly lower number of constraints per tree (see Table S.7, Supplementary Document).

6.3. Decision Boundary Complexity

While Extended C-Net provides the most compact trees and lower size of rule sets, C5 provides more interpretability with the lowest number of constraints for each rule. That raises a question of when to use a simple algorithm such as C5 instead of a more complex and comprehensible model like Extended C-Net or EC-DT. In other words, how to decide the best compromise between simplicity versus accuracy. The analysis of the decision boundary sheds some light on this issue.



(a) Occupancy

(b) Skin

(c) Wall following (4 inputs)

(d) Balance

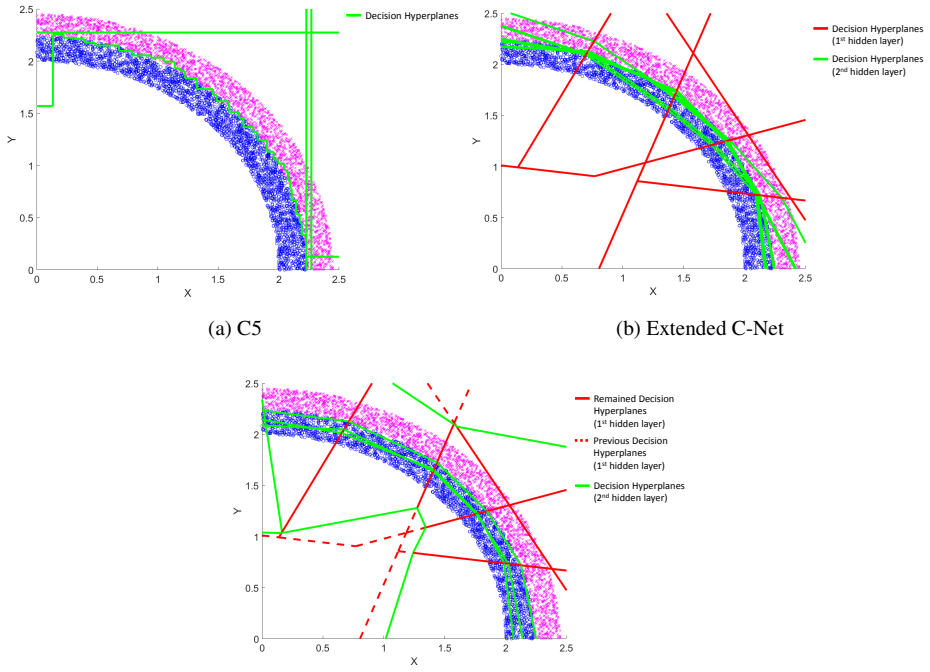
Figure 4: Two PCA components with highest variances of different datasets.

For analyzing the complexity of the data spaces and class distributions, we apply Principle Component Analysis (PCA) transformation on each data set and visualize the two largest components with the largest variance. Figure 4 illustrates the distribution

of classes according to two chosen principle components in two binary classification problem (*occupancy* and *skin*) and classification problem with more than two target labels (*wall-following-4* and *balance*). It can be seen that the data classes in the problem of *occupancy* and *wall-following-4* are more linearly separable than the other problems where the class coverage strongly overlaps with one another. Due to this linear-separability, it is feasible to use a simple tree with low number of leaves and constraints under each leaf to represent the classification rules and hence achieve a higher fidelity with the performance of neural networks. The visualization of PCA components of different classes in remaining data sets can be found in Section S.6 (Supplementary document).

According to the figures in Table 2 and Table 5, the accuracy of linearly-separable problems such as *occupancy* and *wall-following-4* are not significantly different or at least as close to the EC-DT as the Extended C-Net algorithm, while the number of leaves and the number of constraints under each leaf on average is equivalent or much lower than the ones of Extended C-Net. In these cases, the use of C5 for extracting rules from the neural networks is more appropriate as it achieves acceptable performance with better simplicity. On the other hand, for more complex problems with non-linear separable properties, it is less accurate when using C5 to extract the rules. In the *skin* problem, to achieve around 86% accuracy, the C5 trees have to use up to nearly 50 leaves each with more than 2.5 rules on average. As another example, the C5 generates approximately 19 rules, each includes around 3 constraints, to only achieve less than 80% accuracy on average in *balance* problem. For the same problem, the Extended C-Net reaches more than 92% accuracy with half the number of rules of C5, but with more than 7 constraints in each rule. Therefore, it is favorable to use C5 with simple rule sets for linearly-separable datasets with low dimensions while using Extended C-Net or EC-DT if one favors higher accuracy and understanding of correlation among a large number of attributes of input space.

The synthetic problem P2 is representative for the set of highly nonlinear problems. To address this non-linearity, the simple C5 rule extraction algorithm has to generate a huge number of axis-parallel hyperplanes joining together to create a highly complex decision boundary as illustrated in Figure 5a. On the contrary, Extended C-Net ex-



(a) C5

(b) Extended C-Net

(c) EC-DT

Figure 5: Decision hyperplanes extracted from pruned C5, Extended C-Net, and EC-DT transformed from a ANN model for P2 problem.

tracts the rules from ANN by considering the constraints for hidden nodes activation. Therefore, the set of rules from the first hidden layer tries to cluster the data space into sub-regions demonstrated by the area surrounded by the red lines (Figure 5b), while the second layer forms the constraints that are hyperplanes which at this stage directly separate two classes. In EC-DT, the tree representation of the true process of the ANN, the rules extracted from the first hidden layer resemblance the same clustering method as implemented by Extended C-Net. However, the set of rules extracted from the second hidden layer takes two roles at the same time where some nodes attempt to further shrink different data sub-regions and the others directly get involved in creating decision hyper-planes between two classes (see Figure 5c). The Extended C-Net, due to the employment of the similar rule structure at early layers of the EC-DT, exhibits a closer data processing to the ANN.

6.4. Interpretation of Rules

In this paper, we present the figures on tree size as a compactness metrics to analyze the global interpretability of the methodologies in the previous subsection. In this section, the instance-based interpretation of a rule is considered so that we can have a comprehensive view of how the explanation of a neural network as a black-box model can be generated.

Given an instance in a dataset, an explanation can be generated from constraints which are included from an activated leaf that the instance falls under. We provide the rule of one leaf in a tree so that we can evaluate the complexity and correctness of the constraints. We analyze the decision boundaries for the *skin segmentation* dataset, which are a collection of samples extracted randomly from RGB images in FERET and PAL databases. These databases contain a variety of images of people with different characteristics such as age, race, and genders. Given an area in an RGB image, with three attributes of Green (G), Red (R), and Blue (B) values ranging between 0 and 255, one rule to identify that this area is human skin from a C5 tree can be found below:

IF:

- $B > 92$
- $G \leq 157$
- $R > 231$

THEN: This is *skin*

In case of Extended C-Net the constructed explanation for one leaf can be displayed as below:

IF:

- $(-0.47 * B) + (1.51 * G) + (0.04 * R) > -5.23$
- $(0.28 * B) + (0.35 * G) + (-0.47 * R) > -4.00$
- $(0.69 * B) + (-0.49 * G) + (0.24 * R) > -5.68$
- $(0.30 * B) + (-0.66 * G) + (0.32 * R) > 4.22$
- $(0.53 * B) + (-0.18 * G) + (-0.29 * R) > -10.14$
- $(2.75 * B) + (-1.82 * G) + (-0.81 * R) > -5.37$

THEN: This is *skin*

Meanwhile, EC-DT shows a more complex explanation in exchange for the highest accuracy with the highest number of constraints:

IF:

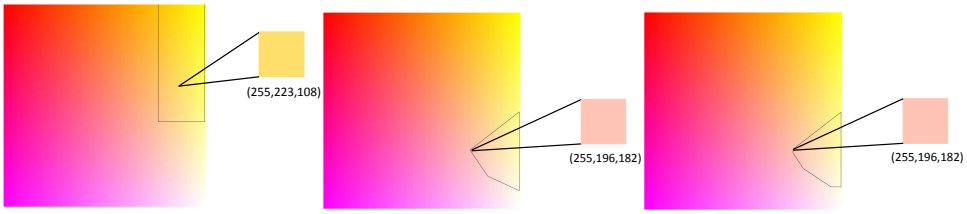
- $(-0.47 * B) + (1.51 * G) + (0.04 * R) > -5.23$
- $(0.28 * B) + (0.35 * G) + (-0.47 * R) > -4.00$
- $(0.69 * B) + (-0.49 * G) + (0.24 * R) > -5.68$
- $(0.30 * B) + (-0.66 * G) + (0.32 * R) > 4.22$
- $(0.53 * B) + (-0.18 * G) + (-0.29 * R) > -10.14$
- $(-0.32 * B) + (0.50 * G) + (-0.10 * R) > 11.20$
- $(-0.38 * B) + (0.56 * G) + (-0.49 * R) \leq -1.52$
- $(-0.64 * B) + (0.27 * G) + (0.29 * R) > 7.15$
- $(-0.03 * B) + (4.52 * G) + (-4.14 * R) \leq -36.45$
- $(2.75 * B) + (-1.82 * G) + (-0.81 * R) \leq -7.76$
- $(3.32 * B) + (-1.80 * G) + (-1.27 * R) + (46.98457) > 0$

THEN: This is *skin*

In the cases of Extended C-Net and EC-DT, the rules involve a combination between all attributes. The attributes with higher weights have more influence on the final decisions than those with lower weights. The positive/negative signs of the weights emphasize the contribution of the attributes towards positive or negative classes in the problem.

Despite the fact that the exhibition of rules extracted from Extended C-Net and EC-DT trees provides a more complete explanation of the decision, the complexity from the number and structure of the constraints within the rules are extremely higher than the C5 constraints. The rule as a result becomes less interpretable. Nevertheless, a way forward on this issue might come from an additional technique to transform the mode of explanation depending on problem.

In the *skin segmentation* problem, one might find it more appropriate to transform the rule constraints into visualization of the RGB colour maps and a sample of data



(a) C5 (b) Extended C-Net (c) EC-DT

Figure 6: The RGB colormaps with bounded regions representing corresponding rules from different tree models.

needs to be explained can be also projected in the map. For example, given the red value $R = 255$, we can construct the colour map for the rules above as illustrated in Figure 6.

With this type of visualization, it is also interesting to note that the decision regions created by rules from C5 are axis-parallel rectangles which are simple, but less precise than the polygons and the shrunk polygons created by Extended C-Net and EC-DT respectively.

In previous literature regarding skin segmentation from images, a branch in computer vision, many studies [60, 61] have introduced the explicitly defined skin region approach, which defines fixed conditions for RGB or YUV color ranges, to discriminate between skin and non-skin areas. While this method is simple, fast, and highly interpretable, it faces a significant challenge in achieving a competitive level of accuracy. Nevertheless, the method is still reliable to use as an initial screening method for skin detection [62]. Many machine learning methods, including neural networks, have been used to enhance the detection rate in this problem. However, their interpretability is lower than that of the simple explicitly defined skin region approach. The translation of the rules extracted from a ANN into a visualizable representation might improve the transparency of the decision making processes. Our transformation of rules into colormaps shows similar utility to the visualization of color ranges in the literature [63].

As another example, we convert the rules extracted from C5, Extended C-Net and EC-DT trees into a visual explanation of the decision-making processes of the neural network as illustrated in Figure 7 (driving behaviour) and Figure 8 (collecting be-

haviour). The direction of the movement force of the shepherd is originally modelled as a function of the direction between the sheep' global centre of mass (GCM) and the shepherd, and the direction between the target/furthest sheep and the sheep' GCM in previous literature [55]. The visualization provides a map of decisions with regards to the input states of relative directions of interest.

Similar to the *skin detection* problem, the decision regions created by rules from C5 are also axis-parallel rectangles. This offers a relatively easy way to illustrate general relationships between variables that result in the outputs of the model. Nevertheless, this visualization of the model behaviour is much less precise than the oblique polygons created by Extended C-Net and EC-DT. Figure 7 illustrates the decision polytopes of shepherd's driving behaviour models extracted from the driving neural network in regards with two dimensions of the input states, which are the direction of the sheep GCM relative to the shepherd agent (x-axis) and the direction of the target in relative to the sheep GCM (y-axis), both are represented by angles (in degree) of corresponding directional vectors in the Unit Circle. The driving behaviours of the shepherd represented by the outputs of the polytopes generated by Extended C-Net and EC-DT rules are highly similar to each other. However, the EC-DT rules are better learned evidenced by the higher number of divisions of the region characterised by coordinates $x \in (-150, -150)$ and $y \in (-150, -50)$ in the state space where Extended C-Net rule model, in fact, overlooks. The observed variations above contribute to the differences of the fidelity of the models to the original neural network.

The collecting behaviours of the shepherding problem using the collecting network are modelled by three rule extraction algorithms. These decision polytopes corresponding to the behaviours based on two input dimensions are visualized in Figure 8. In this visualization, we can observe the same interpretability and precision characteristics of the rule models as mentioned with rule models for driving behaviour above. The behaviours of the Extended C-Net and EC-DT models are similar to each other except a small region at coordinates $x \in (-20, 10)$ (the sheep GCM is around north of the shepherd) and $y \in (50, 180)$ (the fursthest sheep is between northeast and west relative to the sheep GCM). Figure 9 demonstrates an example state of such situation. The optimal behaviour for this case is to move *north* so that the positions of the shepherd,

the furthest sheep, and the sheep GCM are alligned, which helps maximise the effectiveness of the sheep collection behaviour. The EC-DT shows a higher fidelity to the original neural network and provides a better performance by choosing to move *north* in this situation while the Extended C-Net shows a less effective solution by selecting *south* direction. Such lower fidelity of the Extended C-Net rules accounts for the poorer performance represented by the success rate and the number of steps to complete the task in the shepherding problem compared to that of EC-DT model despite its better rule compactness.

Our transformation of rules into maps of regions and vectors helps alleviate the lack of transparency of the mathematical forms of rules extracted from the neural networks. It also delivers a mode of representation, where not only the fidelity is maintained but higher complexity of the rules set from Extended C-Net or EC-DT are lessened to the same level of the one from C5.

7. Conclusion

In this paper, we propose two novel multivariate decision tree frameworks which can generate interpretable rules for explaining the operation of neural networks. The first framework is a modification of Extended C-Net algorithm into a Extended C-Net which can learn the relationship between the last layer of a ANN and the output to back-project and extract the multivariate constraints on input as a set of highly accurate rules. The second is an algorithm called EC-DT that can directly translate the ANN layer-wise and build the set of rules with 100% fidelity to the ANN.

EC-DT offers the best accuracy when it perfectly preserves the performance of the ANNs, but with the cost of a larger number of rules and number of constraints in each rule. It is understandable as the high number of rules are to capture the generalization that the ANN offers. Compared to a traditional approach of generating trees with a baseline C5 algorithm, Extended C-Net in general can better maintain the accuracy of the ANN while achieving the most compact trees, implying a smaller number of rules in use.

However, the use of simple versus complex models results in the trade-off between

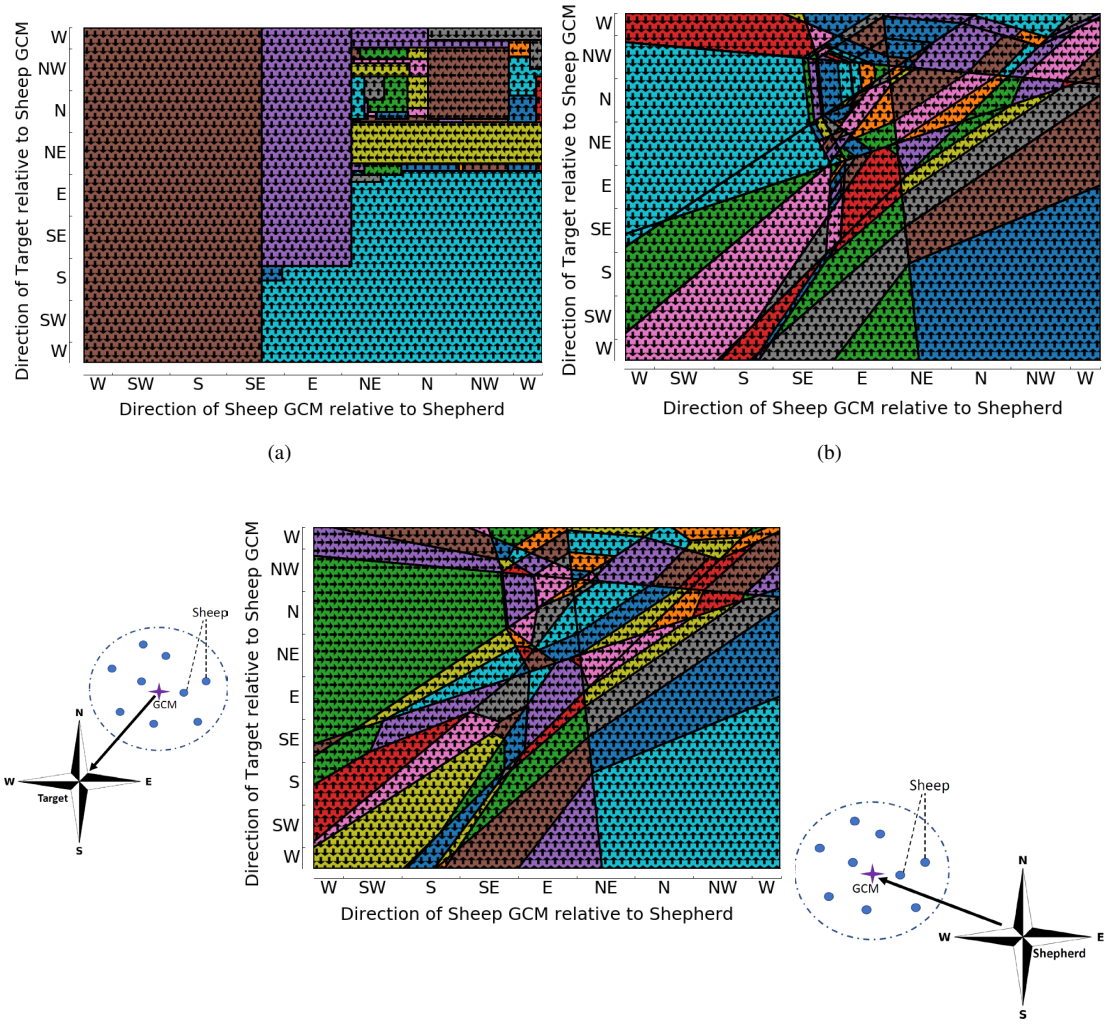
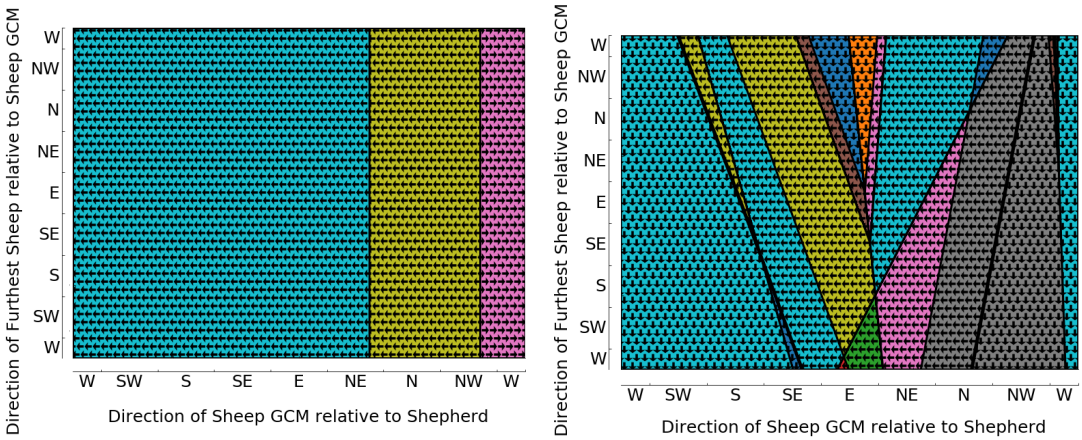


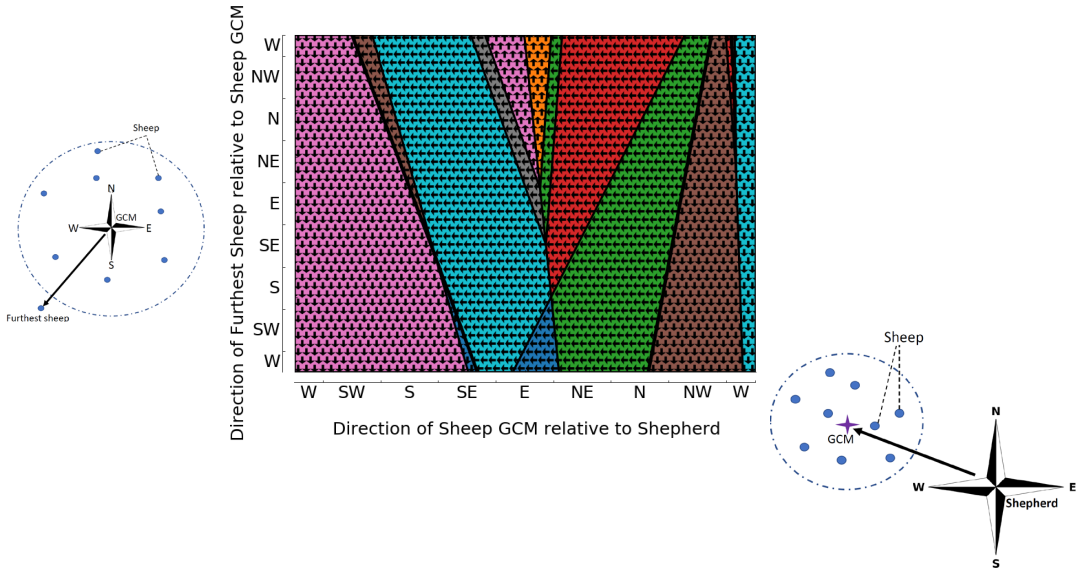
Figure 7: The decision polytopes characterizing the driving behaviour of the shepherd with (a) C5, (b) Extended C-Net, and (b) EC-DT model. The distance between the shepherd and sheep' GCM, and the distance between the sheep' GCM and the target are 50 and 100 metres respectively. The directions of arrows in the polytopes are output directions of the shepherd.

the simplicity and interpretability against the accuracy and precision. To decide on which model to use, one should consider the complexity of the problem space. For linear-separable classification problems, a simple C5 can achieve similar results to



(a)

(b)



(c)

Figure 8: The decision polytopes characterizing the collecting behaviour of the shepherd with (a) C5, (b) Extended C-Net, and (b) EC-DT model. The distance between the shepherd and sheep' GCM, and the distance between the sheep' GCM and the furthest sheep are 40 and 40 metres respectively. The directions of arrows in the polytopes are output directions of the shepherd.

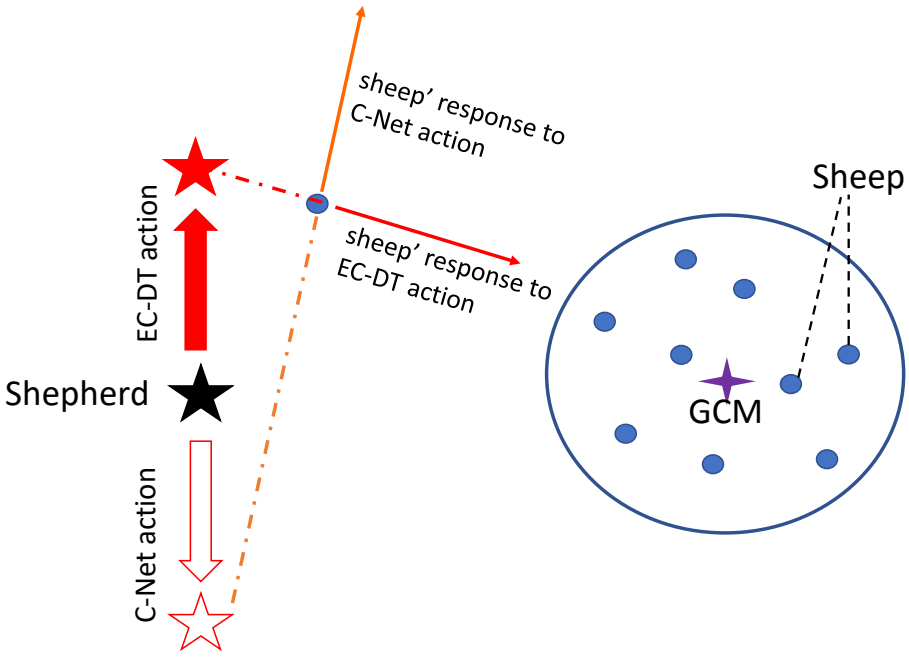


Figure 9: The actions of the shepherd according to Extended C-Net and EC-DT models and the reaction of the sheep in collecting task.

ANN with a very low and simple set of rules. For highly nonlinear problems where a large number of attributes are involved, EC-DT and Extended C-Net exhibit significantly higher accuracy than a simple C5. In general, in the situation where the priority is accuracy, the EC-DT can be used, while in cases where the balance between accuracy and interpretability is required, Extended C-Net is favored.

The weakness of the more complex models such as Extended C-Net and EC-DT comes from the large number of multivariate constraints for each rule, where the form of C5 constraints is very simple. The plain display of the mathematical conditions as an explanation might lower the transparency. Therefore, a suitable transformation of the representation of rules to some explanation mode that reduces the number of dimensions can be employed to overcome the issue. The visualization of decision hyperplanes that is introduced in this paper in a specific problem of *skin segmentation* and *shepherding control task* are examples for an effective explanation interface for

instance-based interpretation.

In future work, the interpretability of rules extracted from our proposed EC-DT and Extended C-Net algorithms will be investigated on more problems. The suitability of the extracted knowledge may contribute to new pieces of knowledge to different human experts, which would call for a subject-matter expert-evaluation of the extracted knowledge.

References

References

[1] E. H. Shortliffe, B. G. Buchanan, A model of inexact reasoning in medicine, Rule-based expert systems (1984) 233–262.

[2] W. L. Johnson, Agents that learn to explain themselves., in: AAAI, 1994, pp. 1257–1263.

[3] W. Swartout, C. Paris, J. Moore, Explanations in knowledge systems: Design for explainable expert systems, IEEE Expert 6 (3) (1991) 58–64.

[4] M. Van Lent, W. Fisher, M. Mancuso, An explainable artificial intelligence system for small-unit tactical behavior, in: Proceedings of the National Conference on Artificial Intelligence, Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2004, pp. 900–907.

[5] Y. Wen, K. Zhang, Z. Li, Y. Qiao, A discriminative feature learning approach for deep face recognition, in: European Conference on Computer Vision, Springer, 2016, pp. 499–515.

[6] S. Wu, Y.-C. Chen, X. Li, A.-C. Wu, J.-J. You, W.-S. Zheng, An enhanced deep feature representation for person re-identification, in: Applications of Computer Vision (WACV), 2016 IEEE Winter Conference on, IEEE, 2016, pp. 1–8.

[7] R. Salakhutdinov, J. B. Tenenbaum, A. Torralba, Learning with hierarchical-deep models, IEEE transactions on pattern analysis and machine intelligence 35 (8) (2013) 1958–1971.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al., Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529.

[9] P. S. Churchland, T. J. Sejnowski, T. A. Poggio, The computational brain, MIT press, 2016.

[10] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu, Asynchronous methods for deep reinforcement learning, in: International conference on machine learning, 2016, pp. 1928–1937.

[11] A. J. Hepworth, D. P. Baxter, A. Hussein, K. J. Yaxley, E. Debie, H. A. Abbass, Human-swarm-teaming transparency and trust architecture, IEEE/CAA Journal of Automatica Sinica.

[12] K. Tsujino, S. Nishida, Implementation and refinement of decision trees using neural networks for hybrid knowledge acquisition, Artificial Intelligence in Engineering 9 (4) (1995) 265–276.

[13] M. W. Craven, Extracting comprehensible models from trained neural networks, Tech. rep., University of Wisconsin-Madison Department of Computer Sciences (1996).

[14] G. P. J. Schmitz, C. Aldrich, F. S. Gouws, Ann-dt: an algorithm for extraction of decision trees from artificial neural networks, IEEE Transactions on Neural Networks 10 (6) (1999) 1392–1401. doi:10.1109/72.809084.

[15] Y. Hayashi, M.-H. Hsieh, R. Setiono, Understanding consumer heterogeneity: A business intelligence application of neural networks, Knowledge-Based Systems 23 (8) (2010) 856–863.

[16] M. Chakraborty, S. K. Biswas, B. Purkayastha, Rule extraction from neural network using input data ranges recursively, New Generation Computing 37 (1) (2019) 67–96.

[17] E. W. Saad, D. C. Wunsch II, Neural network explanation using inversion, Neural networks 20 (1) (2007) 78–93.

[18] J. A. Alexander, M. Mozer, Template-based procedures for neural network interpretation, Neural Networks 12 (3) (1999) 479–498.

[19] R. Andrews, J. Diederich, A. B. Tickle, Survey and critique of techniques for extracting rules from trained artificial neural networks, Knowledge-based systems 8 (6) (1995) 373–389.

[20] I. A. Taha, J. Ghosh, Symbolic interpretation of artificial neural networks, IEEE Transactions on knowledge and data engineering 11 (3) (1999) 448–463.

[21] D. Gunning, Explainable artificial intelligence (xai), Defense Advanced Research Projects Agency (DARPA), nd Web 2.

[22] A. Adadi, M. Berrada, Peeking inside the black-box: A survey on explainable artificial intelligence (xai), IEEE Access 6 (2018) 52138–52160.

[23] W. Samek, Explainable AI: Interpreting, explaining and visualizing deep learning, Vol. 11700, Springer Nature, 2019.

[24] C. Gan, N. Wang, Y. Yang, D.-Y. Yeung, A. G. Hauptmann, Devnet: A deep event network for multimedia event detection and evidence recounting, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 2568–2577.

[25] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, D. Batra, Grad-CAM: Visual explanations from deep networks via gradient-based localization, in: Proceedings of the IEEE international conference on computer vision, 2017, pp. 618–626.

[26] A. Chattopadhay, A. Sarkar, P. Howlader, V. N. Balasubramanian, Grad-CAM++: Generalized gradient-based visual explanations for deep convolutional networks, in: 2018 IEEE Winter Conference on Applications of Computer Vision (WACV), 2018, pp. 839–847. doi:10.1109/WACV.2018.00097.

[27] L. A. Hendricks, Z. Akata, M. Rohrbach, J. Donahue, B. Schiele, T. Darrell, Generating visual explanations, in: European Conference on Computer Vision, Springer, 2016, pp. 3–19.

[28] M. T. Ribeiro, S. Singh, C. Guestrin, Why should I trust you?: Explaining the predictions of any classifier, in: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, ACM, 2016, pp. 1135–1144.

[29] A. Amin, A novel classification model for cotton yarn quality based on trained neural network using genetic algorithm, Knowledge-Based Systems 39 (2013) 124–132.

[30] A. Gupta, S. Park, S. M. Lam, Generalized analytic rule extraction for feedforward neural networks, IEEE transactions on knowledge and data engineering 11 (6) (1999) 985–991.

[31] E. R. Hruschka, N. F. Ebecken, Extracting rules from multilayer perceptrons in classification problems: A clustering-based approach, Neurocomputing 70 (1-3) (2006) 384–397.

[32] J. R. Quinlan, Generating production rules from decision trees., in: ijcai, Vol. 87, Citeseer, 1987, pp. 304–307.

[33] A. B. Tickle, M. Orlowski, J. Diederich, DEDEC: A methodology for extracting rules from trained artificial neural networks, Neurocomputing Research Centre, Queensland University of Technology, 1996.

[34] S. Sestito, Automated knowledge acquisition of rules with continuously valued attributes, in: Proceedings of the 12th international conference on expert systems and their applications, 1992, 1992.

[35] M. W. Craven, J. W. Shavlik, Using sampling and queries to extract rules from trained neural networks, in: Machine learning proceedings 1994, Elsevier, 1994, pp. 37–45.

[36] M. Craven, J. W. Shavlik, Extracting tree-structured representations of trained networks, in: Advances in neural information processing systems, 1996, pp. 24–30.

[37] G. G. Towell, J. W. Shavlik, Extracting refined rules from knowledge-based neural networks, Machine learning 13 (1) (1993) 71–101.

[38] L. Fu, Rule generation from neural networks, IEEE Transactions on Systems, Man, and Cybernetics 24 (8) (1994) 1114–1124.

[39] R. Setiono, H. Liu, Symbolic representation of neural networks, Computer 29 (3) (1996) 71–77.

[40] R. Setiono, H. Liu, Neurolinear: From neural networks to oblique decision rules, Neurocomputing 17 (1) (1997) 1–24.

[41] M. Ishikawa, Rule extraction by successive regularization, Neural Networks 13 (10) (2000) 1171–1183.

[42] K. Odajima, Y. Hayashi, G. Tianxia, R. Setiono, Greedy rule generation from discrete data and its use in neural network rule extraction, Neural Networks 21 (7) (2008) 1020–1028.

[43] C. McMillan, M. C. Mozer, P. Smolensky, The connectionist scientist game: rule extraction and refinement in a neural network, in: Proceedings of the 13th Annual Conference of the Cognitive Science Society, 1991, pp. 424–430.

[44] R. Andrews, S. Geva, Rule extraction from local cluster neural nets, Neurocomputing 47 (1-4) (2002) 1–20.

[45] O. Boz, Extracting decision trees from trained neural networks, in: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2002, pp. 456–461.

[46] U. Johansson, L. Niklasson, Evolving decision trees using oracle guides, in: 2009 IEEE Symposium on Computational Intelligence and Data Mining, IEEE, 2009, pp. 238–244.

[47] C. E. Brodley, P. E. Utgoff, Multivariate decision trees, Machine learning 19 (1) (1995) 45–77.

[48] S. K. Murthy, S. Kasif, S. Salzberg, A system for induction of oblique decision trees, Journal of artificial intelligence research 2 (1994) 1–32.

[49] H. K. Sok, M. P.-L. Ooi, Y. C. Kuang, S. Demidenko, Multivariate alternating decision trees, Pattern Recognition 50 (2016) 195–209.

[50] F. Wang, Q. Wang, F. Nie, W. Yu, R. Wang, Efficient tree classifiers for large scale datasets, Neurocomputing 284 (2018) 70–79.

[51] H. A. Abbass, M. Towsey, G. Finn, C-Net: A method for generating non-deterministic and dynamic multivariate decision trees, Knowledge and Information Systems 3 (2) (2001) 184–197.

[52] J. R. Quinlan, Data mining tools see5 and c5. 0, http://www. rulequest. com/see5-info. html.

[53] R. Quinlan, C5. 0: An informal tutorial (1998).

[54] D. Dua, C. Graff, UCI machine learning repository (2017).
URL http://archive.ics.uci.edu/ml

[55] D. Strömbom, R. P. Mann, A. M. Wilson, S. Hailes, A. J. Morton, D. J. Sumpter, A. J. King, Solving the shepherding problem: heuristics for herding autonomous, interacting agents, Journal of the royal society interface 11 (100) (2014) 20140719.

[56] J. Chen, X. Zhang, B. Xin, H. Fang, Coordination between unmanned aerial and ground vehicles: A taxonomy and optimization perspective, IEEE Transactions on Cybernetics 46 (4) (2016) 959–972. doi:10.1109/TCYB.2015.2418337.

[57] S. Minaeian, J. Liu, Y. Son, Vision-based target detection and localization via a team of cooperative UAV and UGVs, IEEE Transactions on Systems, Man,

and Cybernetics: Systems 46 (7) (2016) 1005–1016. doi:10.1109/TSMC.2015.2491878.

[58] N. Mathew, S. L. Smith, S. L. Waslander, Planning paths for package delivery in heterogeneous multirobot teams, IEEE Transactions on Automation Science and Engineering 12 (4) (2015) 1298–1308.

[59] N. Mathews, A. L. Christensen, A. Stranieri, A. Scheidler, M. Dorigo, Supervised morphogenesis: Exploiting morphological flexibility of self-assembling multirobot systems through cooperation with aerial robots, Robotics and autonomous systems 112 (2019) 154–167.

[60] J. Kovac, P. Peer, F. Solina, Human skin color clustering for face detection, Vol. 2, IEEE, 2003.

[61] V. Vezhnevets, V. Sazonov, A. Andreeva, A survey on pixel-based skin color detection techniques, in: Proc. Graphicon, Vol. 3, Moscow, Russia, 2003, pp. 85–92.

[62] Z. Li, L. Xue, F. Tan, Face detection in complex background based on skin color features and improved adaboost algorithms, in: 2010 IEEE International Conference on Progress in Informatics and Computing, Vol. 2, IEEE, 2010, pp. 723–727.

[63] S. A. Naji, R. Zainuddin, H. A. Jalab, Skin segmentation based on multi pixel color clustering models, Digital Signal Processing 22 (6) (2012) 933–940.