Transition Matrix Representation of Trees with Transposed Convolutions

Jaemin Yoo* Lee Sael†

Abstract

How can we effectively find the best structures in tree models? Tree models have been favored over complex black box models in domains where interpretability is crucial for making irreversible decisions. However, searching for a tree structure that gives the best balance between the performance and the interpretability remains a challenging task. In this paper, we propose TART (Transition Matrix Representation with Transposed Convolutions), our novel generalized tree representation for optimal structural search. TART represents a tree model with a series of transposed convolutions that boost the speed of inference by avoiding the creation of transition matrices. As a result, TART allows one to search for the best tree structure with a few design parameters, achieving higher classification accuracy than those of baseline models in feature-based datasets.

1 Introduction

Tree models [1] have been favored over complex black-box models in domains where interpretability is a crucial factor for making reliable decisions, such as in biological and medical fields [4, 13], where decisions make irreversible effects. The main advantage of tree models over other classifiers is that their decision processes are understandable without post-processing methods [22, 24] that explain approximate reasons for decisions.

Recent works improve the performance of tree models by adopting complex decision functions [9, 31, 32] or utilize tree-structured decisions as a component of large black-box models to gain in interpretability [15, 23, 25]. However, although these approaches have fundamental similarities in dealing with trees, there is no unified way to generalize and represent them by a single framework. This makes one resort to manually search for the best tree structure only among a few feasible choices, losing the opportunity to improve in performance.

In this work, we propose TART (Transition Matrix Representation with Transposed Convolutions), a novel framework for generalizing tree models with a unifying view. TART characterizes a tree model as a sequence of linear transformations whose transition matrices are determined by input features. The unified representation of trees allows us a) to effectively characterize and categorize existing models and b) to perform a systematic search over possible tree structures. TART also utilizes transposed convolutions to avoid the generation of large transition matrices during inference. This optimization improves the speed of training and inference especially in trees with large depth.

We perform extensive experiments on 121 feature-based datasets and show that TART outperforms existing classifiers with reasonable choices of the tree structure. We also provide detailed guidelines on the design choices of TART by thorough comparisons between different combinations of parameters.

Our contributions are summarized as follows:

- **General representation.** We propose TART, a general and efficient tree representation that gives a unifying view of existing tree models.
- **Categorization and characterization.** We analyze existing tree and non-tree classifiers based on the generalizability of TART.
- **Ablation study.** We undergo extensive ablation study on 121 tabular datasets to analyze the effects of the design parameters of TART.
- **Efficiency.** TART speeds up the inference of tree models up to 36.3 times based on the utilization of transposed convolution operations.

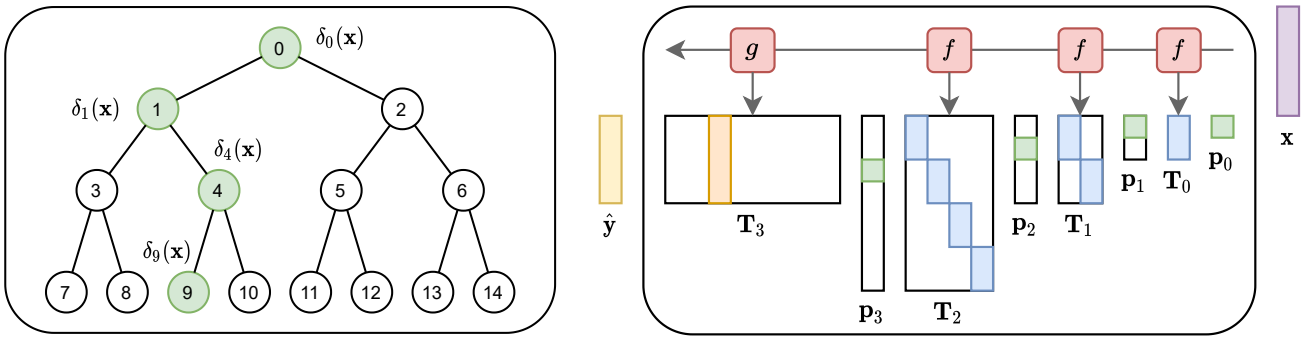
The rest of this paper is organized as follows. We review related works in Section 2 and propose TART in Section 3. We discuss how TART generalizes existing classifiers in Section 4. We present experimental results in Section 5 and conclude at Section 6. Symbols used in this paper are summarized as Table 1. Our code is available at https://github.com/leesael/TART.

2 Related Works

Decision trees (DT) propagate input data from the root to the leaf nodes through tree-structured layers without updating their representations [1]. This process is considered inherently interpretable, but typical DTs often show poor performance due to the low generalizability to unseen test data. We list three types of related works that focus on improving the accuracy of DTs.

Tree models with linear decisions. Soft decision trees (SDT) [12] are characteristic in that the internal

*Seoul National University (jaeminyoo@snu.ac.kr).
†Corresponding author. Ajou University (sael@ajou.ac.kr).



(a) A traditional view of a tree model.

(b) A TART view of a tree model.

Figure 1: The illustration of a binary tree model by (a) the traditional node-and-branch view and (b) the view of our TART utilizing transition matrices. The traditional view treats the prediction $\hat{\mathbf{y}}$ as a sequence of decisions δ_0, δ_1, δ_4, and δ_9, while the view taken by our TART treats the model as a series of linear transformations where each matrix $\mathbf{T}_d$ represents the transition at each depth $d \in \{0, 1, 2, 3\}$ from the root to the leaf nodes.

Table 1: Symbols frequently used in this paper.

Symbol	Description
$\mathbf{T}_d$	Transition matrix at layer d
$\mathbf{p}_d$	Assignment vector at layer d
f	Internal decision function
g	Leaf classifier function
h	Leaf-combining function
D	Tree depth
W	Window size of convolutions
S	Stride of convolutions
H	Number of layers in f
L	Number of layers in g

decisions are made by logistic classifiers that utilize all elements of each feature vector. The logistic classifiers allow differentiable updates of the parameters in SDTs through backpropagation. SDTs have been studied and used widely due to their simplicity and generalizability [9, 11, 17, 31]. Deep neural decision trees [30] extend DTs into multi-branched trees by splitting each example directly into multiple bins using a set of learnable thresholds. These models have an interpretable nature due to the linearity of decision and leaf functions.

Tree models on learned representations. Recent works have utilized deep neural networks to provide the ability of representation learning to tree models. They first learn a better representation of each example using a complex black-box model and use the learned representation as input to tree models instead of the raw features. One popular approach is to use abstract representations generated from convolutional neural networks [26, 25, 29] or multilayer perceptrons [3]. Such approaches make higher accuracy than those of linear tree models, however, they provide interpretability only on top of the abstract representations. Thus, the direct relationship between the raw features and predictions is unclear due to the nonlinear feature extraction.

Tree models for data categorization. Another approach to combine DTs with deep neural networks is to categorize raw examples by hierarchical decisions before feeding them into black-box classifiers [20]. Recent works improve the accuracy of deep neural networks by inserting hierarchical decisions as differentiable operations into a deep neural network, instead of building a complete tree model [19, 18, 2, 28]. These approaches take advantage of DTs with respect to data clustering, rather than focusing on making interpretable decisions, to improve the decision boundaries learned by black-box learners while minimizing the complexity.

In this work, we focus on generalizing and improving complete tree structures that do not change the input features, which often have associated context information that is useful for interpretation.

3 Proposed Method

We propose TART (Transition Matrix Representation with Transposed Convolutions), a unified approach to represent tree models with a series of transition matrices efficiently with transposed convolutions. Figure 1 shows the transition matrix view of a binary tree, on which our TART is based. Algorithm 1 summarizes the decision process of TART for an input feature vector $\mathbf{x}$, which we explain in detail in Section 3.3.

3.1 Transition Matrix Representation

We introduce the transition matrix representation of a tree. We first define a transition matrix in Definition 1 and describe its properties in Lemmas 3.1 and 3.2.

Algorithm 1 TART

Input: Feature vector $\mathbf{x}$
Output: Prediction $\hat{\mathbf{y}}$
Parameter: Tree depth D, internal decision function f, leaf classifier g, and leaf-combining function h

for each $d \in [0, D)$ **do**
  $N_d \leftarrow$ Get the number of nodes at layer d
  $\mathbf{B}_d \leftarrow \text{Stack}(\{f(\mathbf{x}; \theta_{di}) \mid i \in [1, N_d]\})$
end for
$\mathbf{p}_0 \leftarrow \mathbf{1}$ ▷ Vector of length 1
$\mathbf{p}_D \leftarrow \mathbf{B}_D * (\mathbf{B}_{D-1} * \cdots * (\mathbf{B}_1 * \mathbf{p}_0))$ ▷ Alg. 2
$\hat{\mathbf{y}} \leftarrow h(\mathbf{p}_D, \{g(\mathbf{x}; \theta_i)\}_{i=1,\cdots,N_D})$ ▷ Eq. (3.4) or (3.5)

DEFINITION 1. *A rectangular matrix* $\mathbf{T}$ *is a transition matrix if* $\mathbf{T} \geq 0$ *and* $\sum_i T_{ij} = 1$ *for all* j. *We represent the set of all possible transition matrices as* $\mathcal{P}$.

Following from Definition 1, every probability vector $\mathbf{p}$ such that $\mathbf{p} \geq 0$ and $\sum_i p_i = 1$ satisfies $\mathbf{p} \in \mathcal{P}$, since it can be thought of as a matrix of size $|\mathbf{p}| \times 1$.

LEMMA 3.1. *Given a matrix* $\mathbf{T} \in \mathcal{P}$ *of size* $l \times m$ *and a vector* $\mathbf{p} \in \mathcal{P}$ *of length* m, $\mathbf{Tp} \in \mathcal{P}$.

Proof. Let $\mathbf{q} = \mathbf{Tp}$. Then, the following holds:

$$\sum_i q_i = \sum_i \sum_j T_{ij} p_j = \sum_j p_j \sum_i T_{ij} = 1.$$

Thus, the resulting $\mathbf{q}$ is a probability vector. ∎

LEMMA 3.2. *Given two matrices* $\mathbf{T} \in \mathcal{P}$ *and* $\mathbf{U} \in \mathcal{P}$ *of sizes* $l \times m$ *and* $m \times n$, *respectively,* $\mathbf{TU} \in \mathcal{P}$.

Proof. Let $\mathbf{V} = \mathbf{TU}$. Then, for every j,

$$\sum_i V_{ij} = \sum_i \sum_k T_{ik} U_{kj} = \sum_k U_{kj} \sum_i T_{ik} = 1.$$

Thus, the resulting $\mathbf{V}$ is a transition matrix. ∎

Given an input feature $\mathbf{x}$, the soft down spread of $\mathbf{x}$ from the root to leaves is represented as a set $\{\mathbf{p}_d\}_d$ of assignment vectors, where $\mathbf{p}_d \in \mathcal{P}$ is for each layer d. Each node in a layer d computes a decision probability for passing $\mathbf{x}$ to its child node k based on a decision function $f(\mathbf{x})_k$ that sums to one over all ks. This process can be understood as the multiplication of a transition matrix $\mathbf{T}_d \in \mathcal{P}$ and the assignment vector $\mathbf{p}_d$, where $\mathbf{T}_d$ is generated from applying f to all nodes in layer d and combining their outputs. Based on this, we define the *transition matrix representation* as Definition 2.

DEFINITION 2. *The transition matrix representation of a tree classifier* $\mathcal{M}$ *is given as*

$$\mathcal{M}(\mathbf{x}) = \mathbf{T}_D \cdots \mathbf{T}_1 \mathbf{T}_0 \mathbf{p}_0, \tag{3.1}$$

Algorithm 2 TConv

Input: Local transition matrix $\mathbf{B}_d$ of size $W \times N_d$ and arrival probability $\mathbf{p}_d$ at layer d
Output: Arrival probability $\mathbf{p}_{d+1}$ of layer $d+1$
Parameter: Stride S

$\mathbf{p}_{d+1} \leftarrow \mathbf{0}$ ▷ Initialize the output
$j \leftarrow 0$ ▷ Starting index of an output node
for each $i \in [1, N_d]$ **do**
  $p_{d+1,j:j+W} \leftarrow p_{d+1,j:j+W} + p_{d,i} \mathbf{b}_{d,i}$
  $j \leftarrow j + S$
end for

where $\mathbf{p}_0 = 1$ *is the arrival probability to the root node, and* D *is the tree depth.* $\mathbf{T}_d \in \mathcal{P}$ *is the transition matrix at layer* d, *generated by a decision function* f *as*

$$T_{dji} = f(\mathbf{x}; \theta_{di})_j, \tag{3.2}$$

where T_{dji} *refers to the* (j, i)*-th element of* $\mathbf{T}_d$, *and* θ_{di} *is the set of parameters for node* i *at layer* d.

LEMMA 3.3. $\mathcal{M}(\mathbf{x}) \in \mathcal{P}$ *for any* $\mathbf{x}$.

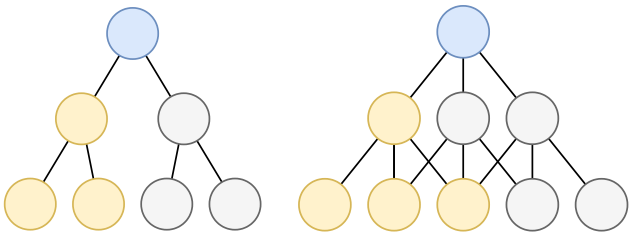
Proof. $\mathcal{M}$ is a series of liner transformations done with transition matrices. Since $\mathbf{p}_0 \in \mathcal{P}$ in Equation (3.1), the lemma is proved due to Lemma 3.1. ∎

Figure 1 visualizes a binary tree by the traditional view and by the transition matrix representation. Figure 1a treats the model as a series of independent decisions following the path of $\mathbf{x}$, while Figure 1b represents the model as a series of linear transformations. We denote the decision function of the last layer by g, since it is defined differently from the internal decision function f in many tree models. For example, in decision trees, g is a fixed one-hot vector, while f is a decision function that takes $\mathbf{x}$ as an input. $\mathbf{T}_D \in \mathcal{P}$ is still satisfied with a different g if we assume a classification task.

The figure also indicates that the nonzero elements of each transition matrix determine the tree shape. For example, the transition matrices of a binary tree (shown in Figure 1b) have nonzero values at the block-diagonal positions. Any tree structure can be represented based on the positions of the nonzero elements in transition matrices that derive from diverse decision function f. We present in Section 4 the structural generalization of TART for representing existing classifiers.

3.2 Optimization by Transposed Convolutions

The transition matrices allow TART to represent general tree structures. However, generating the transition matrix $\mathbf{T}_d$ for every layer d requires a heavy computation, e.g., size for T_d is $2^{d+1} \times 2^d$ in a binary tree model. The



(a) TConv ($W = S = 2$) (b) TConv ($W = 3, S = 1$)

Figure 2: Comparison between tree structures determined by the values of W and S. Intersections between adjacent decisions occur when $S < W$.

overall complexity is $O(2^{2D-1})$ in a binary tree of depth D, which is infeasible with large D.

We propose to utilize transposed convolutions [8] in the formation of tree structures to avoid the generation of complete transition matrices in TART. A transposed convolution maps each input node into multiple output nodes by sliding a small kernel. Thus, the transposed convolution can be applied to spread input data to child nodes in a tree structure. In the rest of this paper, we denote a transposed convolution as TConv for brevity.

Specifically, TConv is utilized in TART as follows. We are given the arrival probability $\mathbf{p}_d$ of layer d and a decision function f. Then, we create a local transition matrix $\mathbf{B}_d \in \mathbb{R}^{W \times N_d}$ by stacking the outputs of f for all nodes in layer d, where N_d is the number of nodes and W is the number of children that each node connects to. $\mathbf{B}_d$ is then spread out to the assignment vector $\mathbf{p}_{d+1}$ of the next layer by the transposed convolution. In typical n-way trees, $\mathbf{B}_d$ is n^{d-1} times smaller than $\mathbf{T}_d$, allowing us to save extensive time and space in computation.

TConv is then applied to $\mathbf{B}_d$ as described in Algorithm 2. It generates the new arrival probability $\mathbf{p}_{d+1}$ without explicitly generating $\mathbf{T}_d$, given two parameters W and S that determine the shape of the tree. The kernel slides from the leftmost node in $\mathbf{p}_d$ to the rightmost one, generating $\mathbf{p}_{d+1}$, which is $\in \mathcal{P}$ by Lemma 3.1, since $\mathbf{B}_d$ is a transition matrix generated from f.

The window size W and the stride S of convolutions are two parameters that determine the shape of a tree. The window size W determines the branching factor of trees, e.g., $W = 2$ in binary trees. Large W increases the complexity of the decision function f but decreases the tree depth required to make the same number of leaf nodes. Thus, the value of W makes a tradeoff between the width and depth, and its optimal value depends on the property of f and the characteristic of the dataset. The stride S determines the number of nodes that are skipped between convolution operations. Branches have no shared children if $S = W$, since a node slides by the width of the previous decision. If $S < W$, a node slides less than the width of the previous decision, making a child node take inputs from multiple parents.

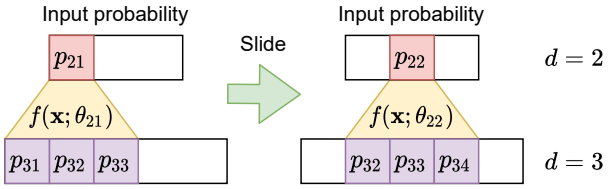


Figure 3: TConv at work between depth 2 and 3 in the tree of Figure 2b. The kernel has the width $W = 3$ and slides by $S = 1$ from the left to the right.

Figure 2 compares two structures of trees based on the values of W and S. Figure 2a depicts the structure of a typical binary tree, where each node is connected to two child nodes without intersections. Figure 2b shows a 3-way tree, where each node has three children. There are two child nodes shared between adjacent decisions since $W - S = 2$. Figure 3 is an illustration of TConv in the tree of Figure 2b, when a convolution kernel slides from node 1 to node 2 at layer 2.

3.3 Training and Inference We describe how to train TART and how to make its predictions. We train TART in an end-to-end fashion, updating all parameters by gradient-based optimization. The objective function is defined as the sum of all loss values from leaf nodes weighted by the arrival probability $\mathbf{p}_D$:

$$\mathcal{L}(\mathbf{x}, \mathbf{y}) = \sum_{u=1}^{N_D} p_D(u) l(g(\mathbf{x}; \theta_u), \mathbf{y}), \tag{3.3}$$

where g is the leaf classifier parameterized with θ_u, $p_D(u)$ is the arrival probability for u, and $l(\hat{\mathbf{y}}, \mathbf{y})$ is the cross entropy function. The cross entropy $l(\hat{\mathbf{y}}, \mathbf{y})$ is defined as $-\sum_{v \in \mathcal{S}} y(v) \log \hat{y}(v)$, where $\mathcal{S}$ is the set of target classes, $\hat{\mathbf{y}}$ is the prediction, and $\mathbf{y}$ is the true one-hot label vector.

There are two ways to make a decision after TART is trained: a) making a weighted average of predictions from the leaf nodes by $\mathbf{p}_D$, and b) choosing the leaf node that gives the largest arrival probability. We call these two choices *multi-leaf selection* and *single-leaf selection*, respectively. The multi-node selection produces higher accuracy in general, resembling ensemble learning, while the single-leaf selection is better for interpretability as a single leaf node participates in each prediction.

Multi-leaf selection. The prediction with the multi-leaf selection is defined as follows:

$$\mathcal{M}(\mathbf{x}_i) = \sum_{u=1}^{N_D} p_D(u) g(\mathbf{x}_i; \theta_u). \tag{3.4}$$

Table 2: Representation of existing binary tree models as TART. f and g refer to the internal and leaf decision function, respectively. Details are in Section 4.1.

Model	$f(\mathbf{x};\theta_i)$	$g(\mathbf{x};\theta_j)$
DT [1]	$\mathbb{I}(s_i(\mathbf{1}_i^\top \mathbf{x} - b_i) > 0)$	Onehot(θ_j)
SDT [12]	$\sigma(\mathbf{w}_i^\top \mathbf{x} + b_i)$	Categorical(θ_j)
NDF [3]	MLP$_i$(rand$(\mathbf{x})$)	Categorical(θ_j)
DNDF [15]	CNN$(\mathbf{x}; i)$	Categorical(θ_j)
NRF [23]	CNN$(\mathbf{x}; i, \text{depth}(i))$	Gaussian(θ_j)

In this way, a decision process resembles the weighted ensemble of weak classifiers, which are the leaf nodes in our case. A model can make accurate predictions even though the representation power of each classifier is not sufficient, due to the effect of ensemble learning.

Single-leaf selection. The prediction with the single-leaf selection is defined as follows:

$$\mathcal{M}(\mathbf{x}_i) = g(\mathbf{x}_i; \theta_{u^*}), \tag{3.5}$$

where $u^* = \arg\max_u p_D(u)$ is the leaf node that makes the largest arrival probability among all leaves. In this way, the ability to split examples to proper leaves plays a crucial role for achieving high accuracy.

Overall algorithm. The decision process of TART is summarized as Algorithm 1. In lines 1 to 4, it runs the decision for every internal node and stacks the results of decisions at each layer. The local transition matrix $\mathbf{B}_d$ of each layer d is used to run inference through the transposed convolution operations in line 6, where $*$ is the TConv function of Algorithm 2. The predictions of leaf nodes are combined in line 7 by the leaf-combining function h, based on the arrival probability $\mathbf{p}_D$.

4 Further Analysis

We characterize and categorize existing classifiers based on the generalized representation of TART. We also present three promising combinations of design parameters of TART that have different advantages.

4.1 Generalizability We study the generalizability of TART in binary trees and general classifiers.

Representation of binary trees. Existing tree models have different characteristics but share a similar tree structure. Such models differ in the choice of decision functions f and g working at the internal layers and the leaf layer, respectively. We show in Table 2 how TART represents different tree models with the choice of f and g. We set the structural parameters W and S to 2, since all these models have the binary tree structure.

Decision trees (DT) select a single element of each input feature $\mathbf{x}$ by a one-hot vector $\mathbf{1}_i$ and compare it with a learned threshold b_i at each internal node i. Soft decision trees (SDT) improve DTs by performing a soft decision at each branch, which uses all elements of $\mathbf{x}$ as a linear separator using the logistic sigmoid function σ. The weight vector $\mathbf{w}_i$ is learned for each node i. Their decision processes are naturally interpretable, since the decision functions are linear with respect to $\mathbf{x}$.

The remaining models use nonlinear decision functions. Neural decision forests (NDF) utilize a randomized multilayer perceptron (MLP) as a decision function. Deep neural decision forests (DNDF) use a single convolutional neural network (CNN) for all decisions, changing only the last fully-connected layer. Neural regression forests (NRF) and their variants use hierarchical CNNs having different numbers of convolutions [25, 26]. All of these models use deep neural networks as their decision functions to improve representation power.

Categorization of general classifiers. We utilize the framework of TART to categorize and characterize existing classifiers. We assume that deep neural networks having a nonlinear activation function are used for both f and g. Then, we introduce three design parameters of TART as the main variables: tree depth D, the number H of layers in f, and the number L of layers in g. The result of categorization is given as Table 3.

A classifier is a single expert having no tree structure if $D = 0$. In this case, logistic regression (LR) and MLPs are distinguished by the value of L. If $H = 0$, no internal decisions are made even with $D > 1$, meaning that all examples are equally split into all leaf nodes. In this case, a classifier makes a prediction by computing the simple average of predictions as an ensemble model.

Table 3: Classifier models represented by TART with three design parameters: tree depth D, the number H of layers in f, and the number L of layers in g.

Models	D	H	L
Logistic regression	$D = 0$	-	$L = 1$
Multilayer perceptrons [21]	$D = 0$	-	$L > 1$
Simple ensembles of experts	$D > 0$	$H = 0$	Any L
Trees of type 1 [1, 12]	$D > 0$	$H = 1$	$L = 0$
Trees of type 2 [15, 23]	$D > 0$	$H > 1$	$L = 1$
Trees of type 3 [19, 20]	$D > 0$	$H = 1$	$L > 1$

Table 4: Promising tree structures of TART that have different properties. Details are in Section 4.2.

Model	W	S	D	H	L	Property
TART-A	2	2	6	1	1	Strong in small data
TART-B	2	2	2	1	4	Strong in large data
TART-C	3	2	3	1	2	Best balance

Table 5: The information of 121 datasets divided into three groups by the number of examples, which include 9, 37, and 75 datasets, respectively.[1]

Group	Examples		Features	Labels
	Min	Max	Avg ± Std	Avg ± Std
Large	10,992	130,064	19.0 ± 15.8	8.2 ± 8.5
Mid	1,000	8,124	40.2 ± 48.4	12.2 ± 26.6
Small	10	990	24.4 ± 37.9	4.1 ± 3.7
All	10	130,064	28.8 ± 40.8	6.9 ± 15.5

[1] http://persoal.citius.usc.es/manuel.fernandez.delgado/papers/jmlr

A classifier splits given examples by learnable decisions only if $D > 0$ and $H > 0$, becoming a tree model whose structure represents a decision path.

The characteristic of a tree classifier is determined by the values of H and L. Models having $H = 1$ and $L = 0$ split given examples by linear decisions into leaf classifiers that return fixed predictions. Thus, they are the simplest tree models that focus on interpretability. Models with $H > 1$ focus on the ability to split examples by utilizing a nonlinear decision function at the internal nodes, while those with $H = 1$ and $L > 1$ use a simple decision rule but focus on the leaf classifiers.

4.2 Promising Tree Structures Based on the categorization of existing models, we propose three promising structures of Tart consisting of different values of parameters. Table 4 summarizes the structures, which we call Tart-A, Tart-B, and Tart-C, respectively. We assume decision functions f and g as multilayer perceptrons with H and L layers, respectively, as in Table 3. We set $H = 1$ in this case, because we have found that $H > 1$ makes a tree model easily overfit to training data without a clear advantage in our datasets.

Linear leaves (Tart-A). A tree model with linear decision functions gives clear interpretability. Tart-A is characterized by an abundant number of leaf nodes each of which makes a linear decision boundary for the examples that have arrived through internal decisions. Tart-A performs the best in small datasets, where the nonlinearity is not essential for acquiring high accuracy. On the other hand, the linearity of Tart-A allows one to avoid overfitting in such small datasets, resulting in improving accuracy in unseen test data.

Nonlinear leaves (Tart-B). The linearity requires us to use a sufficient number of leaf nodes to make high accuracy. On the other hand, we can bound the number of leaves if we increase the capacity of each leaf node. This turns our model into a small ensemble of nonlinear classifiers, where any leaf selection scheme can be used with a different advantage: the single-leaf selection has better interpretability of decisions, while the multi-leaf selection improves performance. Still, we focus on only the single-leaf selection, as our primary goal of utilizing tree models is to make clear interpretability.

Table 6: Classification accuracy of Tart and baseline models. MLP-l represents an MLP having l layers. Our three Tart models show the best accuracy in different groups of datasets, based on their characteristics.

Model	Large	Medium	Small
DT	88.3±0.1	76.3±0.2	71.9±0.7
LR	79.1±0.1	80.8±0.2	75.8±0.3
SVM-lin	77.7±0.1	79.0±0.2	74.9±0.5
SVM-rbf	87.6±0.0	81.1±0.1	**77.0±0.2**
MLP-1	78.7±0.1	78.9±0.3	73.4±0.4
MLP-2	87.8±0.1	83.0±0.4	76.5±0.4
MLP-4	91.8±0.1	83.0±0.2	76.8±0.2
MLP-8	91.5±0.1	82.5±0.3	76.0±0.5
MLP-16	85.3±0.9	78.3±0.2	75.1±0.6
Tart-A	88.2±0.2	82.6±0.2	**77.0±0.6**
Tart-B	**92.1±0.1**	82.7±0.4	76.0±0.3
Tart-C	89.6±0.4	**83.1±0.2**	76.3±0.1

Three-way decisions (Tart-C). Tart-C focuses on the balance between Tart-A and Tart-B. Three-way branches with $W = 3$ make each internal decision richer than in binary trees. Still, it makes the width of a tree increases much faster with the tree depth than in binary trees. Thus, we make intersections between decisions by setting $S = 2$ to bound the tree width while utilizing the rich decisions. The choices of other parameters such as D and L are in between those of Tart-A and Tart-B. The chosen structure is similar to Figure 2b, except that Tart-C slides the kernel by two instead of one.

5 Experiments

We compare our Tart with existing tree and non-tree classifiers by experiments on feature-based data, where tree models have been adopted actively.

Datasets. We use 121 feature-based datasets taken from UCI Machine Learning Repository [7], which were used as a benchmark in [6, 21]. Table 5 summarizes the information of our datasets, which are categorized into three groups by the number of examples. We follow the experimental setup of [21] including the data split and feature preprocessing. In all experiments, we run each model four times with different random seeds and report the average and standard deviation.

Baselines. We include the following baseline classifiers in our experiments, which have been used widely for feature-based datasets: logistic regression (LR), de-

Table 7: Accuracy of TART when the linear leaf nodes are adopted. Models with multi-leaf selection perform better than single-leaf models in most cases, and both models show higher accuracy with larger D.

Leaves	D	L	Large	Medium	Small
Multi	2	1	84.6±0.2	81.4±0.3	75.3±0.5
Multi	4	1	86.7±0.1	82.1±0.3	76.2±0.2
Multi	6	1	88.2±0.2	82.6±0.2	**77.0±0.6**
Multi	8	1	**89.1±0.1**	**82.9±0.4**	76.5±0.6
Single	2	1	84.4±0.2	81.1±0.3	74.7±0.5
Single	4	1	86.4±0.1	81.6±0.4	75.1±0.4
Single	6	1	87.8±0.2	82.0±0.3	75.7±0.6
Single	8	1	88.6±0.1	82.2±0.3	74.6±0.6

Table 8: Accuracy of TART when nonlinear leaf nodes are adopted. They are specialized for large datasets and achieve higher accuracy than those of MLPs (Table 6) or TART models with linear leaves (Table 7).

Leaves	D	L	Large	Medium	Small
Single	2	2	87.4±0.1	**82.7±0.3**	76.0±0.2
Single	4	2	89.0±0.1	82.6±0.5	**76.1±0.4**
Single	6	2	90.0±0.1	82.6±0.4	76.0±0.4
Single	8	2	90.7±0.0	82.6±0.4	75.4±0.3
Single	2	4	92.1±0.1	**82.7±0.6**	76.0±0.3
Single	4	4	**92.3±0.1**	82.2±0.4	75.7±0.3
Single	6	4	92.1±0.1	81.9±0.1	75.6±0.4
Single	8	4	91.9±0.1	82.0±0.2	75.4±0.3

cision trees (DT), and support vector machines (SVM) with the linear and RBF kernels. We also include multi-layer perceptrons (MLP) as a strong competitor, whose structure is taken from a previous work that studied our UCI datasets [21]: 100 units at each hidden layer, the ELU activation function [5], He-initialization [10], and dropout of probability 0.15 [27]. The training of MLPs follows the same process as our TART.

Hyperparameters. We adopt an MLP with the same ELU activation and dropout of probability 0.15 as the decision functions f and g of TART. We train TART and MLPs based on the Adam optimizer [14] with the initial learning rate 0.005. The batch size is set to 1024, which is large enough to load most datasets by a single batch. We ran all of our experiments on a workstation having GTX 1080 Ti, based on PyTorch. We use classification accuracy as a metric to evaluate all classifiers.

5.1 Classification Accuracy We compare the accuracy of TART and baseline models in Table 6. Our TART models show the highest accuracy in general, with their strengths in different groups of datasets.

DTs show the lowest accuracy in the medium and small datasets, since they easily overfit to training data. MLPs and SVM with the RBF kernel perform the best among the baselines due to the nonlinearity of decisions. MLP-1 works in a similar way to LR, but its accuracy is lower than those of LR and SVM-lin. This is because the stochastic training of MLPs does not guarantee the global optimum of parameters. The accuracy of MLPs depends heavily on the number of layers, indicating the sensitivity to the choice of hyperparameters.

Our three TART models show the best accuracy in different groups of datasets. TART-A works the best in small datasets since it consists of linear leaf nodes each of which has a limited capacity, minimizing the risk of overfitting. TART-B achieves the best accuracy in large datasets by combining multiple nonlinear leaves based on tree decisions, each of which has the same structure as MLP-4. TART-C is a balance between TART-A and TART-B, resulting in the best accuracy for medium-sized datasets among all TART models and baselines.

5.2 Structural Search The flexibility of our TART allows us to easily search for a suitable structure by the choice of its design parameters. We categorize possible options of parameters into three groups that correspond to TART-A, TART-B, and TART-C, respectively.

Linear leaves (TART-A). Table 7 performs an ablation study for TART-A by changing the depth D and the leaf selection function h. All these models use linear leaf nodes to maximize the interpretability, which is the main strength of TART-A. Multi-leaf models perform better than single-leaf models in general, because they make up for the limited capacity of leaf nodes by combining multiple nodes for each prediction. Still, single-leaf models work better than the linear baselines such as LR or SVM-lin, since they choose a suitable classifier for each example following the tree structure. It is also notable that both multi- and single-leaf models perform better with larger D, without showing a significant drop of its accuracy unlike MLPs of Table 6.

Nonlinear leaves (TART-B). Adopting nonlinear leaf nodes requires us to choose the single-leaf selection scheme for interpretability. Table 8 compares the performance of TART when $L > 1$, changing the tree depth D from 2 to 8, as an ablation study for TART-B. Trees with $D \geq 4$ and $L = 4$ achieve the best accuracy in the large datasets compared to MLPs (in Table 6) and trees with linear leaf nodes (in Table 7). Models with $L = 2$ work well in the medium and small datasets but show limited performance in the large datasets.

The result implies that the representation power of each leaf classifier is an important factor for achieving

Table 9: Accuracy of TART with multi-way decisions and the single-leaf selection. These models show similar accuracy with the choice of parameters.

D	W	L	**Large**	**Medium**	**Small**
1	3	2	88.5±0.1	83.1±0.3	**76.5±0.7**
3	3	2	89.6±0.1	83.1±0.4	76.3±0.4
5	3	2	90.7±0.1	82.9±0.3	76.2±0.1
7	3	2	**91.2±0.1**	82.8±0.4	75.8±0.3
1	3	2	88.9±0.2	**83.3±0.1**	75.7±0.9
1	7	2	89.5±0.1	**83.3±0.2**	75.6±0.5
1	15	2	90.1±0.1	83.1±0.3	75.4±0.9
1	31	2	90.4±0.1	83.2±0.3	75.5±0.4

high accuracy in large datasets. At the same time, the split of data examples through tree-structured decisions is effective for improving the performance of classification avoiding overfitting. This is shown well in Table 6, where a significant drop of accuracy is observed when a large number of layers are adopted for MLPs.

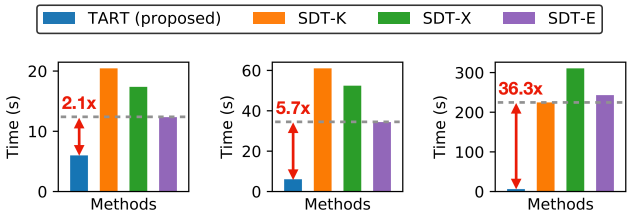
Multi-way decisions (TART-C). Table 9 performs an ablation study for TART-C, comparing models with multi-way decisions. We set the stride S of transposed convolutions to 2 while changing the tree depth D and the window size W. If $W = 3$, the number of leaves at each model of depth D is $2^{D+1} - 1$. Thus, the first four models in Table 9 have the same number of leaf nodes as the last four models in the table, respectively.

We observe the effect of branching intersections by comparing the models in Tables 8 and 9. The first four models in Table 9 work better than the first four models in Table 8, even though they have fewer leaves, taking advantage of intersecting branches. On the other hand, it is observed from the last four models of Table 9 that the ability to split data to the leaf nodes is limited when $D = 1$, even with large W, due to the limited capacity that a single decision function can have.

5.3 Efficiency A notable advantage of TART is the speedup from existing implementations of tree models due to the efficient computation of transposed convolutions. We compare TART with three public implementations of soft decision trees (SDT) [9], which is a special case of TART with $H = 1$, $L = 0$, $W = S = 2$, and the single-leaf selection. We call the baselines SDT-K, SDT-X, and SDT-E, respectively, following the first letters of their repository names.[1][2][3] TART and the all baselines are implemented based on the PyTorch framework.

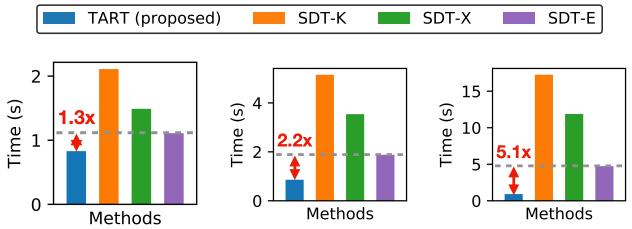
We make all methods have the same structure and decision function, changing the tree depth D from 8 to 12. We use the MNIST dataset [16] in this experiment to be on par with other baselines methods. We consider each 28×28 image as a 768-dimensional vector with no structural information [9]. The training set has 60,000 examples, while the test set has 10,000 examples. We use a single GPU of GTX 1080 Ti and set the batch size to 1024 as in the other experiments.

[1]https://github.com/kimhc6028/soft-decision-tree
[2]https://github.com/xuyxu/Soft-Decision-Tree
[3]https://github.com/endymion64/SoftDecisionTree



(a) $D = 8$. (b) $D = 10$. (c) $D = 12$.

Figure 4: Training time of soft decision trees (SDT) by different implementations. TART achieves the shortest training time due to its efficiency.



(a) $D = 8$. (b) $D = 10$. (c) $D = 12$.

Figure 5: Inference time of soft decision trees (SDT) by different implementations. TART achieves the shortest inference time due to its efficiency.

Figure 4 compares the training time of methods for a single epoch, while Figure 5 shows the inference time in the test data. In both experiments, our TART consistently improves the speed of existing implementations. TART achieves the speedup of up to 36.3× and 5.1× in the training and inference, respectively, compared to the best competitors. This is because the baselines treat a tree model as a set of independent decisions, while TART treats it as a sequence of linear transformations with the efficiency of transposed convolutions.

6 Conclusion

We propose TART (Transition Matrix Representation with Transposed Convolutions), our novel approach to represent tree models as a series of stochastic decisions efficiently with transposed convolutions. TART generalizes the structures of different tree models only with a few design parameters. The generalized representation

allows us to systematically search for the best structure for each dataset. We also present three promising combinations of structural parameters that can be applied to small, medium, and large datasets, respectively. Our extensive experiments on 121 datasets show that TART achieves the highest accuracy compared to existing classifiers. At the same time, the optimization with transposed convolutions improves the speed of training and inference up to 36.3 and 5.1 times, respectively.

Acknowledgments

Publication of this article has been funded by the Basic Science Research Program through the National Research Foundation of Korea (2018R1A5A1060031).

References

[1] L. BREIMAN, J. H. FRIEDMAN, R. A. OLSHEN, AND C. J. STONE, *Classification and Regression Trees*, 1984.

[2] C. BRUST AND J. DENZLER, *Integrating domain knowledge: Using hierarchies to improve deep classifiers*, in ACPR, 2019.

[3] S. R. BULÒ AND P. KONTSCHIEDER, *Neural decision forests for semantic image labelling*, in CVPR, 2014.

[4] D. CHE, Q. LIU, K. RASHEED, AND X. TAO, *Decision tree and ensemble learning algorithms with their applications in bioinformatics*, Software tools and algorithms for biological systems, (2011).

[5] D. CLEVERT, T. UNTERTHINER, AND S. HOCHREITER, *Fast and accurate deep network learning by exponential linear units (elus)*, in ICLR, 2016.

[6] M. F. DELGADO, E. CERNADAS, S. BARRO, AND D. G. AMORIM, *Do we need hundreds of classifiers to solve real world classification problems?*, J. Mach. Learn. Res., 15 (2014).

[7] D. DUA AND C. GRAFF, *UCI machine learning repository*, 2017.

[8] V. DUMOULIN AND F. VISIN, *A guide to convolution arithmetic for deep learning*, arXiv, (2016).

[9] N. FROSST AND G. E. HINTON, *Distilling a neural network into a soft decision tree*, in CEx@AI*IA, 2017.

[10] K. HE, X. ZHANG, S. REN, AND J. SUN, *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification*, in ICCV, 2015.

[11] O. IRSOY AND E. ALPAYDIN, *Autoencoder trees*, in ACML, vol. 45 of JMLR Workshop and Conference Proceedings, 2015.

[12] O. IRSOY, O. T. YILDIZ, AND E. ALPAYDIN, *Soft decision trees*, in ICPR, 2012.

[13] A. JALALI, D. J. LICHT, AND C. NATARAJ, *Application of decision tree in the prediction of periventricular leukomalacia (PVL) occurrence in neonates after heart surgery*, in EMBC, 2012.

[14] D. P. KINGMA AND J. BA, *Adam: A method for stochastic optimization*, in ICLR, 2015.

[15] P. KONTSCHIEDER, M. FITERAU, A. CRIMINISI, AND S. R. BULÒ, *Deep neural decision forests*, in ICCV, 2015.

[16] Y. LECUN, L. BOTTOU, Y. BENGIO, AND P. HAFFNER, *Gradient-based learning applied to document recognition*, Proceedings of the IEEE, 86 (1998), pp. 2278–2324.

[17] A. R. LINERO AND Y. YANG, *Bayesian regression tree ensembles that adapt to smoothness and sparsity*, Journal of the Royal Statistical Society: Series B (Statistical Methodology), 80 (2018).

[18] M. MCGILL AND P. PERONA, *Deciding how to decide: Dynamic routing in artificial neural networks*, in ICML, 2017.

[19] C. MURDOCK, Z. LI, H. ZHOU, AND T. DUERIG, *Blockout: Dynamic model selection for hierarchical deep networks*, in CVPR, 2016.

[20] V. N. MURTHY, V. SINGH, T. CHEN, R. MANMATHA, AND D. COMANICIU, *Deep decision network for multi-class image classification*, in CVPR, 2016.

[21] M. OLSON, A. J. WYNER, AND R. BERK, *Modern neural networks generalize on small data sets*, in NeurIPS, 2018.

[22] M. T. RIBEIRO, S. SINGH, AND C. GUESTRIN, *"why should I trust you?": Explaining the predictions of any classifier*, ACM, 2016.

[23] A. ROY AND S. TODOROVIC, *Monocular depth estimation using neural regression forest*, in CVPR, 2016.

[24] R. R. SELVARAJU, M. COGSWELL, A. DAS, R. VEDANTAM, D. PARIKH, AND D. BATRA, *Grad-cam: Visual explanations from deep networks via gradient-based localization*, Int. J. Comput. Vis., 128 (2020).

[25] W. SHEN, Y. GUO, Y. WANG, K. ZHAO, B. WANG, AND A. L. YUILLE, *Deep regression forests for age estimation*, in CVPR, 2018.

[26] W. SHEN, K. ZHAO, Y. GUO, AND A. L. YUILLE, *Label distribution learning forests*, in NIPS, 2017.

[27] N. SRIVASTAVA, G. E. HINTON, A. KRIZHEVSKY, I. SUTSKEVER, AND R. SALAKHUTDINOV, *Dropout: a simple way to prevent neural networks from overfitting*, J. Mach. Learn. Res., 15 (2014).

[28] R. TANNO, K. ARULKUMARAN, D. C. ALEXANDER, A. CRIMINISI, AND A. V. NORI, *Adaptive neural trees*, in ICML, 2019.

[29] A. WAN, L. DUNLAP, D. HO, J. YIN, S. LEE, H. JIN, S. PETRYK, S. A. BARGAL, AND J. E. GONZALEZ, *NBDT: neural-backed decision trees*, arXiv, (2020).

[30] Y. YANG, I. G. MORILLO, AND T. M. HOSPEDALES, *Deep neural decision trees*, ICML Workshop, (2018).

[31] J. YOO AND L. SAEL, *EDiT: interpreting ensemble models via compact soft decision trees*, in ICDM, 2019.

[32] ———, *Gaussian soft decision trees for interpretable feature-based classification*, in PAKDD, 2021.