Journal of Machine Learning Research 27 (2026) 1-31 Submitted 8/25; Revised 4/26; Published 5/26

Approximation-Free Differentiable Oblique Decision Trees

Subrat Prasad Panda SUBRATPR001@E.NTU.EDU.SG
College of Computing and Data Science
Nanyang Technological University
Singapore

Blaise Genest BLAISE.GENEST@CNRSATCREATE.SG
CNRS@CREATE
Singapore
IPAL, CNRS
France

Arvind Easwaran ARVINDE@NTU.EDU.SG
College of Computing and Data Science
Nanyang Technological University
Singapore

Editor: Honglak Lee

Abstract

Decision Trees (DTs) are widely used in safety-critical domains such as medical diagnosis, valued for their interpretability and effectiveness on tabular data. However, training accurate oblique DTs is challenging due to complex optimization landscapes and overfitting risks, particularly in regression. Recent advances have introduced differentiable formulations that enable gradient-based training and joint optimization of decision boundaries and leaf regressors. Yet, existing approaches typically rely on approximations, either through probabilistic softening of boundaries (soft DTs) or quantized gradients such as the Straight-Through Estimator (STE). To overcome these limitations, we propose `DTSemNet`, a novel, semantically equivalent, and invertible representation of hard oblique DTs as neural networks. `DTSemNet` enables end-to-end training with standard gradient descent, eliminating the need for approximations in both classification and regression. While classification aligns naturally with this formulation, regression remains challenging due to the joint optimization of internal nodes and leaf regressors. To address this, we analyze the limitations of STE and introduce an annealed Top-k method that provides accurate gradient signals without approximation. Extensive experiments on classification and regression benchmarks show that `DTSemNet`-trained oblique DTs outperform state-of-the-art differentiable DTs. Furthermore, we demonstrate that `DTSemNet` can serve as programmatic DT policies in reinforcement learning environments, thereby broadening their applicability.

Keywords: Oblique Decision Tree, Regression Trees, Differentiable Trees, Programmatic Policies, Programmatic Reinforcement Learning

1. Introduction

Decision Trees (DTs) are widely used across various domains, including high-stakes applications such as medical diagnostics (Chen and Asch, 2017). Recent studies (Grinsztajn et al., 2022) highlight that DTs often outperform neural networks (NNs) on tabular data, owing

to their strong inductive bias for modeling non-smooth functions. Despite their advantages, learning optimal DTs remains a challenging task. The search space grows combinatorially with the number of features and decision points, which makes learning an optimal DT NP-hard (Laurent and Rivest, 1976).

Existing methods for learning DTs can be broadly grouped into non-gradient-based and gradient-based approaches. Non-gradient methods include greedy algorithms like CART (Breiman et al., 1984), global optimization via MIP (Bennett and Blue, 1996) or evolutionary algorithms (Costa et al., 2024), and non-greedy methods like TAO (Carreira-Perpinán and Tavallali, 2018; Zharmagambetov and Carreira-Perpinan, 2020). These methods either yield suboptimal trees (Zantedeschi et al., 2021; Karthikeyan et al., 2022) or scale poorly with tree complexity (Bertsimas and Dunn, 2017; Karthikeyan et al., 2022). In contrast, gradient-based methods make DTs differentiable, enabling end-to-end training via backpropagation (Hazimeh et al., 2020; Yang et al., 2018; Zantedeschi et al., 2021). These have shown improved efficiency and accuracy for both classification and regression tasks (Karthikeyan et al., 2022; Paleja et al., 2022; Marton et al., 2024).

To use gradient descent at the training stage on DTs, most previous works adopt "soft" decisions (e.g., using the `Sigmoid` activation function) to make the tree differentiable (Lee and Jaakkola, 2019; Biau et al., 2019; Zantedeschi et al., 2021; Hazimeh et al., 2020). However, the resulting tree does not provide "hard" decisions but "soft", that is, probabilistic ones. Hard DTs can be derived from soft DTs through hardening, albeit at the cost of accuracy. Some recent studies, such as Dense Gradient Trees (DGT) (Karthikeyan et al., 2022), Interpretable Continuous Control Trees (ICCT) (Paleja et al., 2022), and GradTree (Marton et al., 2024, 2025), have introduced an alternative approach for obtaining hard DTs by using an approximation during backpropagation to compute gradients with Straight-Through Estimators (STEs) (Bengio et al., 2013; Hubara et al., 2016). This approximation hampers DT training as the *evaluation function* computing the output during forward propagation and the *optimizing function* computing gradient during backward propagation are different. This is especially true in large datasets or in Reinforcement Learning (RL) tasks, where errors may accumulate over many training steps.

In this work, we focus on producing an approximation-free architecture, that is an architecture where the *evaluation* and the *optimizing* functions are the same.[1] We propose a novel encoding of oblique DTs as NNs, namely the `DTSemNet` architecture, which overcomes the aforementioned shortcomings. It uses `ReLU` activation functions and linear operations, making it differentiable and enabling the use of vanilla gradient descent to train the DT. The proposed encoding is semantically equivalent to a hard oblique DT, with each decision (weight) in the DT corresponding one-to-one to a trainable weight in the NN, while all other weights in the architecture are fixed (non-trainable). `DTSemNet` can be used in both classification and regression settings. In classification, it selects the class with the highest output using a standard `argmax` operation, and we show in Theorem 1 that, for any given input, the output of `DTSemNet` is identical to that of the DT. In regression, each leaf of the DT is associated with a regressor, and `DTSemNet` selects the appropriate leaf regressor, which is then used to compute the final output.

1. This was not the case for the regression task in the conference version (Panda et al., 2024a)

Training DTs for regression is more challenging than in the classification setting, as it requires jointly learning the parameters of DT (acting as the router) so that it selects the appropriate leaf, along with the parameters of the corresponding leaf regressor. Following prior work, DGT and ICCT, Panda et al. (2024a) used `DTSemNet` with a single STE during training to select both the leaf and its corresponding output. However, STE introduces approximation due to biased gradient updates to the internal node parameters, caused by a *mismatch between the forward and backward propagation objectives*. This mismatch gives rise to two key issues that degrade performance: (i) internal node parameters are updated based on the outputs of all regressors at each step, which destabilizes the tree structure and reduces the accuracy of routing samples to the correct leaves; and (ii) only a single regressor is updated per step, so if a sample is later routed to a different leaf, the corresponding regressor (being untrained for that sample) incurs a high loss. The tree therefore tends to retain the previously used leaf, since its loss is comparatively lower and reverting requires fewer updates than training a new regressor. This, in turn, results in repeated assignment of samples to the same leaf and underutilization of many leaves. Together, these issues substantially impair the performance of DTs trained with STE.

To address the above limitations in regression, this work introduces a method inspired by the Top-k selection mechanism widely used in Mixture-of-Experts (MoE) models. In this mechanism, a routing network selects a subset of k experts, and their outputs are combined to produce the final result. Indeed, DT regression can be viewed similarly, with the DT acting as a Top-1 router (in the form of a classifier) that selects a single expert (leaf), which is then used to compute the final output. However, training a DT (represented by `DTSemNet`) with Top-1 selection does not provide informative gradients. In the MoE setting, it is common to use a fixed $k \geq 2$ (often $k = 4$ or $k = 5$), which has been shown to outperform STE-based approaches (Muqeeth et al., 2024). Motivated by this, our method starts with $k \geq 2$ when training the router `DTSemNet` and gradually anneals it to $k = 1$, yielding the hard DT required for `DTSemNet`-regression. This annealing strategy provides more stable and targeted updates by adjusting the parameters of the internal nodes of `DTSemNet` based only on the outputs of the selected regressors. Using the formal definition of Top-k, we show two key differences in its gradient updates compared to STE. First, in Top-k, the top k regressors are updated, whereas STE updates only a single regressor. Second, the parameters of the DT are updated using only the top k selected regressors, in contrast to STE, which updates based on all regressors. Because Top-k trains multiple regressors for each sample during training, it enables neighboring regressors to adapt when the router DT shifts a sample from one leaf to another. By gradually annealing k from $k \geq 2$ to $k = 1$, our approach ensures that at each step $k \in \{K, \ldots, 1\}$ there is *no approximation*, since the forward and backward passes share the same semantics. In particular, for any fixed k, the same function is used to compute the output in forward propagation (evaluation) and the gradients in backward propagation (optimization), thereby maintaining an approximation-free training process throughout. This preserves a consistent training objective for Top-k and thereby improves DT regression performance, unlike STE, which suffers from a mismatch between forward and backward propagation. From another perspective, the Top-k annealing process can be regarded as a smart initialization of step k, using the parameters from a previous step $k' > k$.

We experimented with `DTSemNet` on various benchmarks for both classification and regression tasks, comparing its performance against different methods for producing hard DTs: DGT (Karthikeyan et al., 2022) for gradient descent-based approaches, TAO (Carreira-Perpinán and Tavallali, 2018; Zharmagambetov and Carreira-Perpinan, 2020) for non-greedy algorithms, CRO-DT (Costa et al., 2024) for evolutionary algorithms, and CART as the standard greedy algorithm. The classification and regression tasks primarily involve tabular datasets, except for MNIST, a small image dataset. The number of features ranges from 4 to 780 (with MNIST as an outlier), and the number of classes ranges from 2 to 26. For classification, `DTSemNet` achieves the best performance on every dataset, demonstrating the effectiveness of our proposed unapproximated methodology. For regression, `DTSemNet` outperforms all methods on all datasets except two, for which it produces DTs almost as accurate as the best ones ($< 3\%$ and $< 1\%$ from the best). Additionally, the training time for gradient-based DTs is significantly shorter than that of competing methods.

Finally, we demonstrate the deployment of `DTSemNet` as a programmatic DT policy in RL environments with both discrete and continuous action spaces, achieved by simply replacing the NN. This aligns with the growing interest in Differentiable Programmatic RL to represent policies as programs (Verma et al., 2018; Qiu and Zhu, 2021). Gradient-based DT training is particularly well-suited for RL, where training without gradients is challenging due to the absence of fixed datasets. A common workaround is to train an NN and then distill it into a DT using greedy strategies such as CART, which performs well only in simple RL tasks with low-dimensional, discrete action spaces (Bastani et al., 2018). In contrast, gradient-based DTs can directly learn policies for RL tasks and match the efficiency of NNs (Karthikeyan et al., 2022; Paleja et al., 2022; Marton et al., 2025). `DTSemNet` can replace NNs in standard RL pipelines (e.g., Stable-Baselines3 (Raffin et al., 2021) or CleanRL (Huang et al., 2022)). On discrete-action benchmarks (up to 10 actions, 32 input dimensions), `DTSemNet` matches NN performance and significantly outperforms other DT-based methods trained using gradient descent or imitation learning. For continuous actions with limited input dimensions, we modify the Soft Actor-Critic (SAC) (Haarnoja et al., 2018) training loop to use Top-k methods for DT training. `DTSemNet` Top-k outperforms competitors in continuous control, though architecture choice is less critical in this setting.

Main contributions are as follows:

- We introduce the `DTSemNet` architecture, which we prove to be semantically equivalent to DTs in Theorem 1.

- Learning `DTSemNet` for classification with standard gradient descent corresponds exactly to learning a DT, without resorting to any approximations, unlike competing methods. Experimentally, this results in the most efficient DTs for classification tasks in every standard benchmark we experimented with.

- We develop an annealed Top-k approach for using `DTSemNet` in regression tasks that avoids approximation and overcomes the biased gradients and structural instability caused by STE approximation. This procedure provides a stable and accurate differentiable `DTSemNet`, achieving the best performance on almost all benchmarks and coming within $< 3\%$ of the best on the remaining benchmarks.

- We explain how to use `DTSemNet` as a programmatic DT policy in the RL setting, again producing experimentally the most efficient DT policies, for both continuous action spaces using `DTSemNet`-regression; and by a large margin for discrete actions using `DTSemNet`-classification.

This paper is an extended version of our conference publication (Panda et al., 2024a), offering substantial advances in the regression setting through a novel, approximation-free annealed Top-k approach, introduced in Section 3.4. In this new section, we formally define Top-k and systematically compare its gradient updates with those of the STE-based method, thereby addressing the inherent limitations of STE-based approximation that constrained our earlier work (Panda et al., 2024a). Experimental results, presented in Section 4.1 with twice as many benchmarks as in (Panda et al., 2024a), demonstrate the clear superiority of Top-k for `DTSemNet`-based regression across all benchmark datasets. While `DTSemNet` STE was less accurate than TAO in 60% of the benchmarks (by 5% on average, worst case by 30%), `DTSemNet` Top-k was more accurate than TAO in 80% of the benchmarks (by 3% on average, best case by 6%). These findings are further strengthened by an ablation study that analyzes which component of our annealed Top-k process has the greatest impact on training stability and performance compared to using the STE approximation. Finally, Section 4.3 extends our contribution to the RL setting, where we show the effectiveness of Top-k vs. STE in continuous-action environments. Our source code for the introduced Top-k approach is publicly available on GitHub under the branch "`topk`", which succeeds the earlier version of our code: `https://github.com/CPS-research-group/dtsemnet/tree/topk`.

We organize the paper as follows. Section 2 reviews related work. Section 3 introduces the `DTSemNet` architecture and its application to regression, classification, and RL tasks. Section 4 presents extensive empirical evaluations across these benchmarks. Finally, Section 5 concludes the paper with a summary of contributions and future research directions.

2. Related Work

Non-Gradient-Based DT Training. Historically, DT training techniques have not relied on gradient descent. CART (Breiman et al., 1984) (and its extensions) is a well-known method for training *axis-aligned* DTs by recursively splitting the dataset at each node based on selected features using metrics such as entropy or Gini impurity. For learning *oblique* DTs in this manner, methods such as Oblique Classifier 1 (OC1) (Murthy et al., 1993) and GUIDE (Loh, 2014) have been proposed, but they typically yield suboptimal performance because learning oblique DTs is considerably more challenging than learning axis-aligned ones. TAO is currently a state-of-the-art method for oblique DT learning, improving DTs obtained from CART (or random DTs of a given depth) by alternately fine-tuning (using gradients but not end-to-end) node parameters at specific depths (Carreira-Perpinán and Tavallali, 2018; Zharmagambetov and Carreira-Perpinan, 2020).

Concerning MIP formulations (Bennett and Blue, 1996; Bertsimas and Dunn, 2017) or Global EA-based search approaches, such as CRO-DT (Costa et al., 2024), they learn DTs by searching over various structures of DTs but at high computational costs, which is impractical for large DTs and datasets. The proposed `DTSemNet` overcomes these challenges by using gradient-descent to lower training time, which we confirm by comparing with

CRO-DT (Costa et al., 2024), which proposes matrix encoding of (oblique) DTs to speed up training compared to previous EA-based methods, and produces axis-aligned DTs.

Gradient-Based DT Training. Lately, several works have proposed approximating DTs as soft DTs to enable gradient descent for learning, where decision nodes typically use the `Sigmoid` function (Zantedeschi et al., 2021; Hazimeh et al., 2020; Yang et al., 2018; Frosst and Hinton, 2017; Tanno et al., 2019; Biau et al., 2019; Silva et al., 2020; Ding et al., 2021; Qiu and Zhu, 2021; Gupta et al., 2015; Kontschieder et al., 2015; Bulò and Kontschieder, 2014; Wan et al., 2020). *Hardening* soft DTs, that is, transforming them into hard DTs by discretizing the probabilities induces severe inaccuracies (Paleja et al., 2022). More closely related to our work, DGT (Karthikeyan et al., 2022) represents (oblique) DTs as an NN-architecture using the (non-differentiable) sign activation function, resorting to *quantized* gradient descent to learn it, leveraging principles from training binarized NNs using STE (Hubara et al., 2016). Specifically, during forward propagation, nodes utilize a 0-1 step function, whereas, during backward propagation, nodes employ a piecewise linear function or some differentiable approximation (see Karthikeyan et al., 2022). Similarly, ICCT (Paleja et al., 2022) and GradTree (Marton et al., 2024) learns (axis-aligned) DTs using NNs with the `Sigmoid` activation function and STEs. In all these works, the hard DTs that are produced are (slightly) different from the DT (soft DT or using STE), which is being optimized by gradient descent. In contrast, the `DTSemNet` architecture using ReLU activation functions allows standard gradient descent to be performed, and the output DT from `DTSemNet`-classification is exactly the same as the function optimized by gradient-descent, without approximation. Experiments confirm that it is more accurate in practice, significantly so for classification tasks.

Top-k Routing. DT regression bears strong similarity to MoE architectures (Shazeer et al., 2017; Muqeeth et al., 2024), where each regressor at a leaf can be viewed as an expert and the DT acts as a routing network that selects which leaf regressor (expert) to use, with the regressor's output serving as the expert output. While standard MoE models typically use a Top-k selection mechanism with a fixed $k \geq 2$, DT regression requires a Top-1 selection to choose a single leaf regressor for a given input. However, Top-1 selection produces flat gradients, making it difficult to train the DT effectively. Although Switch Transformer (Fedus et al., 2021) also uses Top-1 expert routing, it scales the expert output by the routing probability, which is incompatible with DT regression where the regression output must come directly from the selected leaf regressor. To address these challenges, we instead use an annealing strategy that starts with $k \geq 2$ to provide meaningful gradient signals for learning DT parameters, which is consistent with prior findings that Top-k selection yields better gradient signals than STE-based approximations (Muqeeth et al., 2024), and gradually reduces k to 1 to obtain a hard regression DT. Finally, while REINFORCE (Muqeeth et al., 2024) offers an alternative for discrete expert selection, its high variance and weak learning signals make it ineffective in supervised regression settings.

Training DTs as Programmatic Policies in RL. Hard or soft DT policies can be obtained via imitation learning (Abbeel and Ng, 2004), i.e., learning from expert policies, usually pretrained NNs (Bastani et al., 2018; Jhunjhunwala et al., 2020; Liu et al., 2019; Bewley and Lawry, 2021; Roth et al., 2019; Verma et al., 2018, 2019; Coppens et al., 2019). For instance, VIPER (Bastani et al., 2018) imitates a Q-network (or policy network)

by creating a dataset from collected samples and trains a DT using CART, with sample weightage assigned based on Q-values. In contrast, `DTSemNet` directly learns a hard oblique DT in RL (using Proximal Policy Optimization (PPO) (Schulman et al., 2017) or SAC). Other works, such as ProLoNet (Silva and Gombolay, 2021), that learn *soft* DTs using the RL framework, with the objective of initializing weights from expert humans. In contrast, `DTSemNet` learns a *hard* DT. ICCT (Paleja et al., 2022) and GradTree (Marton et al., 2025) proposed an STE-based approach to learn axis-aligned DTs using gradient descent. By comparison, we can handle oblique trees, which are more expressive and accurate than axis-aligned DTs, especially for discrete actions.

3. Differentiable Oblique Decision Trees

Notations: We use lowercase letters (e.g., y) for scalars, bold lowercase (e.g., $\boldsymbol{x}$) for vectors, and bold uppercase (e.g., $\mathbf{A}$) for matrices or higher-order tensors. Blackboard bold (e.g., $\mathbb{R}$) is used for sets and calligraphic font (e.g., $\mathcal{L}$) denotes functions. A dataset is represented as $\mathbb{D} = (\boldsymbol{x}_i, y_i)_{i=1}^{N}$, where inputs $\boldsymbol{x}_i \in \mathbb{R}^d$ have d features and labels $y_i \in \mathbb{R}$. The loss function $\mathcal{L}(\hat{y}, y)$ measures the discrepancy between prediction $\hat{y}$ and ground truth y. The dot product of vectors $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^d$ is denoted by $\langle \boldsymbol{u}, \boldsymbol{v} \rangle$.

3.1 Oblique Decision Trees

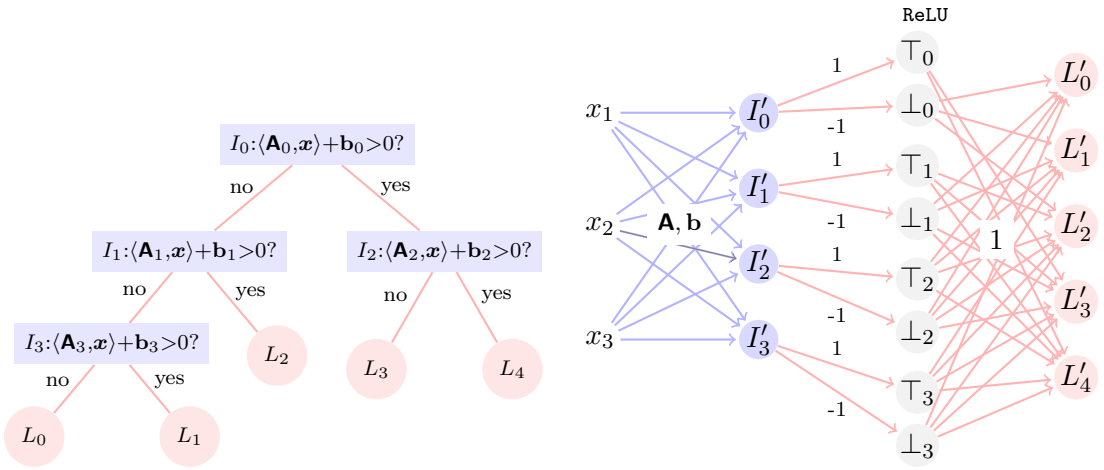
An *oblique decision tree* $\mathcal{T}$ comprises m internal nodes $\mathbb{I} = \{I_0, \ldots, I_{m-1}\}$ and n leaf nodes $\mathbb{L} = \{L_0, \ldots, L_{n-1}\}$. Each internal node performs a binary test (true or false) based on an oblique hyperplane, defined as a linear combination of input features, as illustrated in Fig. 1a. Given an input-output pair $(\boldsymbol{x}, y) \sim \mathbb{D}$, the routing function $\mathcal{T}(\boldsymbol{x}; \mathbf{A}, \boldsymbol{b}) : \mathbb{R}^d \to \mathbb{L}$ deterministically maps $\boldsymbol{x}$ to a leaf through a sequence of binary tests. Specifically, at node $I_j \in \mathbb{I}$, the test $\langle \mathbf{A}_j, \boldsymbol{x} \rangle + \boldsymbol{b_j} > 0$ determines whether to proceed to the right child (true) or the left child (false), ignoring ties[2], where $\mathbf{A}_j \in \mathbb{R}^d$ and $\boldsymbol{b_j} \in \mathbb{R}$ are parameters at that internal node. Once a leaf $L_\ell = \mathcal{T}(\boldsymbol{x})$ is reached, a prediction is made based on the assigned class label or model (e.g., regressor) associated with that leaf, parametrized by $\boldsymbol{\Theta}$, which collectively denotes parameters of all leaves. The training objective is to optimize $(\mathbf{A}, \boldsymbol{b}, \boldsymbol{\Theta})$ by minimizing the expected loss: $\texttt{min}_{\mathbf{A}, \boldsymbol{b}, \boldsymbol{\Theta}} \, \mathbb{E}_{(\boldsymbol{x}, y) \sim \mathbb{D}} \left[\mathcal{L}(\hat{y}, y) \right]$.

3.2 `DTSemNet` Encoding

Finding suitable values for $\mathbf{A}$ and $\boldsymbol{b}$ to make the DT $\mathcal{T}$ accurate is challenging. We explain in the following how to find such values using standard gradient descent algorithms. For that, we encode the DT $\mathcal{T}$ as a NN architecture `DTSemNet` $\mathcal{N}_\mathcal{T} : \mathbb{R}^d \to \mathbb{R}^n$, which is semantically equivalent with DT $\mathcal{T}$, and which can be converted back to an equivalent DT after learning. It is important to note that the `DTSemNet` $\mathcal{N}_\mathcal{T}$ outputs a vector of dimension n equal to the number of leaves in DT $\mathcal{T}$. We now describe the `DTSemNet` architecture, illustrated in Fig. 1b:

Input Layer. d nodes, one for each dimension of the input $\boldsymbol{x} \in \mathbb{R}^d$.

2. We assume inputs almost surely do not lie exactly on decision boundaries.



(a) Decision Tree $\mathcal{T}$ (b) `DTSemNet` $\mathcal{N}_{\mathcal{T}}$

Figure 1: (a) DT with 4 internal nodes (I_0–I_3) and 5 leaves (L_0–L_4) and (b) `DTSemNet` $\mathcal{N}_{\mathcal{T}}$ corresponding to DT $\mathcal{T}$ in (a) with $\boldsymbol{x} \in \mathbb{R}^3$.

Second Layer (Linear Layer). m linear nodes $(I'_0, \ldots, I'_{m-1})$, each corresponding to an internal node $(I_0, \ldots, I_{m-1})$ of the DT $\mathcal{T}$ and computing $I'_i(\boldsymbol{x}) = \langle \mathbf{A}_i, \boldsymbol{x} \rangle + b_i$.

Third Layer (Branching Activation). $2m$ nodes, two per internal node:

$$\top_i(\boldsymbol{x}) := \texttt{ReLU}(I'_i(\boldsymbol{x})), \qquad \bot_i(\boldsymbol{x}) := \texttt{ReLU}(-I'_i(\boldsymbol{x})).$$

Exactly one of $\top_i(\boldsymbol{x})$ or $\bot_i(\boldsymbol{x})$ is positive (except on measure-zero boundaries) and $\top_i(\boldsymbol{x}) + \bot_i(\boldsymbol{x}) = |I'_i(\boldsymbol{x})|$.

Last Layer (Logits Output). A linear layer with fixed (non-trainable) weights and n nodes $(L'_0, \ldots, L'_{n-1})$, each corresponding to a leaf $(L_0, \ldots, L_{n-1})$ of the DT $\mathcal{T}$, is defined as

$$L'_j(\boldsymbol{x}) = \sum_{i=0}^{m-1} \Big(\delta^{\top}_{i,j} \top_i(\boldsymbol{x}) + \delta^{\bot}_{i,j} \bot_i(\boldsymbol{x}) \Big), \qquad \delta^{\top}_{i,j}, \delta^{\bot}_{i,j} \in \{0, 1\}.$$

The coefficients encode the tree structure: $\delta^{\top}_{i,j} = 0$ if L_j is a left descendant of I_i and 1 otherwise; $\delta^{\bot}_{i,j} = 0$ if L_j is a right descendant of I_i and 1 otherwise; and if L_j is not a descendant of I_i, then $\delta^{\top}_{i,j} = \delta^{\bot}_{i,j} = 1$.

Theorem 1 *Given a DT $\mathcal{T}$ and its equivalent NN representation `DTSemNet` $\mathcal{N}_{\mathcal{T}}$ (used as a classifier), the output node (from L'_is) with the maximum value in $\mathcal{N}_{\mathcal{T}}$ corresponds to the same leaf selected by $\mathcal{T}$ for any input $\boldsymbol{x}$. That is, $\forall\, \boldsymbol{x} \in \mathbb{R}^d$, $\texttt{argmax}\, \mathcal{N}_{\mathcal{T}}(\boldsymbol{x}) = \mathcal{T}(\boldsymbol{x})$.*

Proof Fix $\boldsymbol{x} \in \mathbb{R}^d$ and let $L_\ell = \mathcal{T}(\boldsymbol{x})$ be the leaf selected by the DT. Note that L_ℓ has all its associated decisions satisfied, whereas for any other leaf $L_j \neq L_\ell$, at least one associated decision is violated. Recall the construction of the `DTSemNet` $\mathcal{N}_{\mathcal{T}}$, for each internal node I'_i, the branching activations are $\top_i(\boldsymbol{x}) = \texttt{ReLU}(I'_i(\boldsymbol{x}))$ and $\bot_i(\boldsymbol{x}) = \texttt{ReLU}(-I'_i(\boldsymbol{x}))$, where exactly one of $\top_i(\boldsymbol{x})$ or $\bot_i(\boldsymbol{x})$ is strictly positive (ties occur only on a measure-zero set), and each leaf output is given by $L'_j(\boldsymbol{x}) = \sum_{i=0}^{m-1} \big(\delta^{\top}_{i,j} \top_i(\boldsymbol{x}) + \delta^{\bot}_{i,j} \bot_i(\boldsymbol{x}) \big)$.

For the selected leaf L_ℓ, the coefficients $(\delta^{\top}_{i,\ell}, \delta^{\perp}_{i,\ell})$ ensure that the strictly positive branching activation is included for each path node, while both activations are included for off-path nodes. Hence,

$$L'_\ell(\boldsymbol{x}) = \sum_{i=0}^{m-1} \big(\top_i(\boldsymbol{x}) + \perp_i(\boldsymbol{x})\big).$$

Now consider any other leaf $L_j \neq L_\ell$. Since its path differs from that of L_ℓ, there exists an internal node $I'_{i^\star}$ where the decision required by L_j differs from the actual decision at that node. If $I'_{i^\star}(\boldsymbol{x}) > 0$, the actual decision is right while L_j requires left, so $\delta^{\top}_{i^\star,j} = 0$ and the strictly positive term $\top_{i^\star}(\boldsymbol{x})$ is omitted. If $I'_{i^\star}(\boldsymbol{x}) < 0$, the actual decision is left while L_j requires right, so $\delta^{\perp}_{i^\star,j} = 0$ and the strictly positive term $\perp_{i^\star}(\boldsymbol{x})$ is omitted. For all $i \neq i^\star$, the contributions coincide with those in $L'_\ell(\boldsymbol{x})$. Therefore,

$$L'_j(\boldsymbol{x}) = L'_\ell(\boldsymbol{x}) - \begin{cases} \top_{i^\star}(\boldsymbol{x}), & I'_{i^\star}(\boldsymbol{x}) > 0, \\ \perp_{i^\star}(\boldsymbol{x}), & I'_{i^\star}(\boldsymbol{x}) < 0, \end{cases}$$

and hence $L'_j(\boldsymbol{x}) < L'_\ell(\boldsymbol{x})$. Thus $L'_\ell(\boldsymbol{x})$ is the unique maximum among $\{L'_0(\boldsymbol{x}), \ldots, L'_{n-1}(\boldsymbol{x})\}$, i.e., $\texttt{argmax}\,\mathcal{N}_{\mathcal{T}}(\boldsymbol{x}) = L_\ell = \mathcal{T}(\boldsymbol{x})$. ■

Informally, for a given input $\boldsymbol{x}$, the selected leaf $\ell = \mathcal{T}(\boldsymbol{x})$ in the DT corresponds to the index of the output node with the maximum value in the output vector $\boldsymbol{z} = \mathcal{N}_{\mathcal{T}}(\boldsymbol{x})$, analogous to selecting a class label from the output of a NN. The `DTSemNet` is differentiable due to its use of `ReLU` and linear activations. In the following sections, we describe how `DTSemNet` can be used in classification and regression tasks without relying on approximations, and how it can be applied in RL setting as programmatic policies.

3.3 `DTSemNet` Classification:

In the *classification* setting, each leaf $\ell \in \mathbb{L}$ is assigned a class label $\Theta_\ell \in \mathbb{C}$, where $\mathbb{C} = \{C_1, \ldots, C_c\}$ denotes the set of c possible classes, and $\boldsymbol{\Theta} = \{\Theta_\ell\}_{\ell=0}^{n-1}$ represents the class assignments for all leaves. The prediction is given by $\hat{y} = \Theta_{\mathcal{T}(\boldsymbol{x})}$.

We now describe how the `DTSemNet` architecture implements this classification procedure. In `DTSemNet`, the tree is encoded as a NN in which each leaf $L_\ell \in \mathbb{L}$ corresponds to an output neuron L'_ℓ. Since multiple leaves may share the same class label, we introduce an additional output layer with c nodes $\{C_1, \ldots, C_c\}$, one for each class. Each output node C_i receives input from all leaf nodes ℓ such that $\Theta_\ell = C_i$. The connection weight from L_ℓ to C_i is set to 1 if $\Theta_\ell = C_i$, and 0 otherwise. A `MaxPool` operation is applied at each C_i to select the maximum activation among its inputs. By Theorem 1, if $\ell = \mathcal{T}(\boldsymbol{x})$ for input $\boldsymbol{x}$, then L'_ℓ of `DTSemNet` has the highest output. Consequently, the class node $C_i = \Theta_\ell$ also receives the highest input, yielding the same prediction as $\mathcal{T}$ and preserving semantic equivalence.

3.4 `DTSemNet` Regression:

Traditional regression trees, such as CART (Breiman et al., 1984) and DGT (Karthikeyan et al., 2022), assign to each leaf ℓ a scalar parameter $\alpha_\ell \in \mathbb{R}$. In CART, α_ℓ typically represents the average of training labels reaching leaf $\ell = \mathcal{T}(\boldsymbol{x})$, which limits generalization,

even for simple linear functions. In contrast, more expressive regression trees, such as those used in TAO-linear (Zharmagambetov and Carreira-Perpinan, 2020) and ICCT (Paleja et al., 2022), associate a linear regressor with each leaf ℓ, parameterized by a weight vector $\boldsymbol{\theta}_\ell \in \mathbb{R}^d$ and a bias term $\alpha_\ell \in \mathbb{R}$. The collection of linear regressor parameters across all leaves of the DT is represented by the tensor $\boldsymbol{\Theta} = [(\boldsymbol{\theta}_0, \alpha_0), \ldots, (\boldsymbol{\theta}_{n-1}, \alpha_{n-1})]^\top \in \mathbb{R}^{n\times(d+1)}$, and the prediction for input $\boldsymbol{x}$ is given by $\hat{y} = \langle \boldsymbol{\theta}_{\mathcal{T}(\boldsymbol{x})}, \boldsymbol{x}\rangle + \alpha_{\mathcal{T}(\boldsymbol{x})}$, corresponding to a leaf-specific linear transformation over the input features. DTs with linear regressors actually correspond to a piecewise linear regression, where the different linear pieces are described by the parameters at the leaves, and the switch between pieces by the decisions of the DT. We choose the more expressive model with linear regressors at the leaves for regression tasks with `DTSemNet`.

We now study different methods to learn `DTSemNet` for regression tasks, in order to support aforementioned expressive formulation with linear regressors at each leaf. Mainly, the problem is how to jointly learn (a) the regression parameters $\boldsymbol{\Theta}$ at the leaves of the DT, *and* (b) the decision parameters $(\mathbf{A}, \boldsymbol{b})$ at the internal nodes of the DT that route inputs to the appropriate leaf.

3.4.1 Hardmax Selector with One-Hot Vector

The `DTSemNet` $\mathcal{N}_\mathcal{T}$ (corresponding to DT $\mathcal{T}$) outputs logits $\boldsymbol{z} = \mathcal{N}_\mathcal{T}(\mathbf{x}) \in \mathbb{R}^n$ over the n leaves. We define a selector function $\mathcal{S} : \mathbb{R}^n \to \mathbb{R}^n$ that transforms the logits into a selection vector. The standard approach defines a *hardmax* selector function $\mathcal{S}_\mathcal{H}$ that maps the logit vector $\mathbf{z}$ to a one-hot vector $\boldsymbol{h} = \mathcal{S}_\mathcal{H}(\mathbf{z})$, where $\boldsymbol{h}_i = 1$ iff $i = \texttt{argmax}_j z_j$, and $\boldsymbol{h}_i = 0$ otherwise.

To encode the linear regressors at the leaves of the DT, we define function $\mathcal{R}(\boldsymbol{x}) : \mathbb{R}^d \to \mathbb{R}^n$ that outputs $\mathcal{R}(\boldsymbol{x}) = [\langle \boldsymbol{\theta}_0, \boldsymbol{x}\rangle + \alpha_0, \ldots, \langle \boldsymbol{\theta}_{n-1}, \boldsymbol{x}\rangle + \alpha_{n-1}]^\top$. The final prediction is then $\hat{y} = \langle \boldsymbol{h}, \mathcal{R}(\boldsymbol{x})\rangle$,which corresponds to selecting leaf (one-hot vector) and applying its associated linear regressor (regressor value corresponding to the one-hot index) and is equivalent to $\langle \boldsymbol{\theta}_{\mathcal{T}(\boldsymbol{x})}, \boldsymbol{x}\rangle + \alpha_{\mathcal{T}(\boldsymbol{x})}$. That is, in the forward propagation phase, the one-hot vector based on the *hardmax* selector incurs *no approximation*. We now turn to the gradient derivation used in the backward propagation phase.

3.4.2 Gradient Derivation with a One-Hot Vector

We summarize the previously-described forward propagation with one-hot vector $\boldsymbol{h}$, producing scalar $\hat{y}$ (the regression output), as:

$$\boldsymbol{x} \xrightarrow{\mathcal{N}_\mathcal{T}} \boldsymbol{z} \xrightarrow{\mathcal{S}_\mathcal{H}} \boldsymbol{h} \xrightarrow{\langle \boldsymbol{h}, \boldsymbol{r}\rangle} \boldsymbol{h}^\top \cdot \mathcal{R}(\boldsymbol{x}) = \hat{y}$$

Given a loss $\mathcal{L}(\hat{y}, y)$, and denoting $\boldsymbol{r} = \mathcal{R}(\boldsymbol{x})$, the gradients with respect to the parameters (weights) $\mathbf{A}, \boldsymbol{b}, \boldsymbol{\Theta}$ is computed using the chain rule:

$$\nabla_\mathbf{A}\mathcal{L} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \boldsymbol{h}} \cdot \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} \cdot \frac{\partial \boldsymbol{z}}{\partial \mathbf{A}}, \quad \nabla_\boldsymbol{b}\mathcal{L} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \boldsymbol{h}} \cdot \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} \cdot \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}, \quad \nabla_\boldsymbol{\Theta}\mathcal{L} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \boldsymbol{r}} \cdot \frac{\partial \boldsymbol{r}}{\partial \boldsymbol{\Theta}}$$

The gradient updates to the regressor parameters $\boldsymbol{\Theta}$ are given by $\nabla_\boldsymbol{\Theta}\mathcal{L} = \mathcal{L}'(\hat{y}, y) \cdot \boldsymbol{h} \cdot \boldsymbol{x}^\top$. Note that, because of the one-hot vector $\boldsymbol{h}$, only the parameters $\boldsymbol{\theta}_i$ (and α_i) corresponding to the selected leaf receive gradient updates: only the leaf regressor selected by $\boldsymbol{h}$ is updated.

The gradient update for the (routing in the tree) parameters $\mathbf{A}$ follows the chain rule: $\nabla_{\mathbf{A}}\mathcal{L} = \mathcal{L}'(\hat{y}, y) \cdot \boldsymbol{r}^\top \cdot \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} \cdot \frac{\partial \boldsymbol{z}}{\partial \mathbf{A}}$ (and similarly for $\boldsymbol{b}$). Since `DTSemNet` is differentiable, $\frac{\partial \boldsymbol{z}}{\partial \mathbf{A}}$ exists. However, when using hardmax selector $\mathcal{S}_{\mathcal{H}}$, the term $\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}$ is zero almost everywhere, making the gradient non-informative.

To address this issue, the standard technique is to rely on STE (Bengio et al., 2013; Hubara et al., 2016) methods to enable gradient flow, e.g. in ICCT (Paleja et al., 2022), DGT (Karthikeyan et al., 2022) and GradTree (Marton et al., 2024), as well as in the earlier conference version of our paper (Panda et al., 2024a). In the following, we elaborate on the STE mechanism.

3.4.3 Straight-Through Estimator (STE)

STE (Bengio et al., 2013) enables gradient-based optimization through the hardmax selector function $\boldsymbol{h} = \mathcal{S}_{\mathcal{H}}(\boldsymbol{z})$ which is non-differentiable. It does so by keeping the original function unchanged during the forward pass, while approximating its Jacobian during the backward pass as $\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} \approx \mathbf{I}$ (identity function). This approximation allows gradients to flow through the discrete selector, resulting in the approximate gradient with respect to the internal tree parameters $\mathbf{A}$ (similarly for $\boldsymbol{b}$) as $\nabla_{\mathbf{A}}\mathcal{L} \approx \mathcal{L}'(\hat{y}, y) \cdot \boldsymbol{r}^\top \cdot \frac{\partial \boldsymbol{z}}{\partial \mathbf{A}}$. Note that the gradient updates are based on the outputs of *all* regressors $\boldsymbol{r}$, and will flow through *all* the leaves and then back to *all* the internal decision parameters in the tree (changing all these values), including those very far from impacting the actual (accurate) forward pass. While gradient flow back from leaves close to (e.g., one switch in decision away) the selected leaf may make sense to enable switching of decisions, flowing through all of them seems inappropriate. In any case, STE introduces an objective mismatch due to the forward-backward propagation discrepancy, driving the model toward suboptimal local minima and causing training instability (Yin et al., 2019), which in turn hurts the accuracy of learning the parameters of the DT regressor.

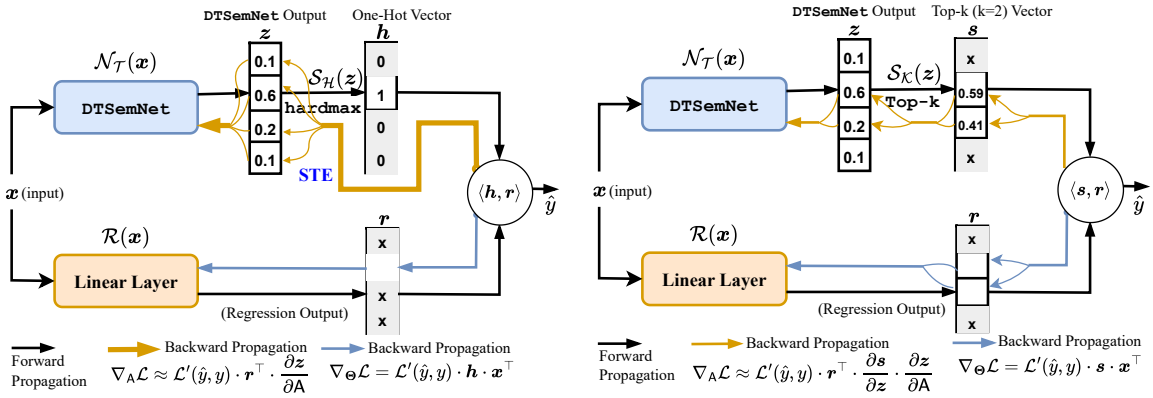
The brighter side is that a single STE call (and thus approximation) is necessary when used in `DTSemNet` (such as in the earlier conference version of this paper (Panda et al., 2024a)), at the output layer. For comparison, methods such as DGT and ICCT (Karthikeyan et al., 2022; Paleja et al., 2022) rely on STE at every internal node for both classification and regression, introducing further approximations that impact learning.

3.4.4 Top-k Selection

Conceptually, the DT part of `DTSemNet` for regression tasks aims at selecting the leaf/regressor, analogous to expert selection in the MoE framework, where accurate routing is crucial (Shazeer et al., 2017; Muqeeth et al., 2024). In the MoE setting, STE can be used, but it frequently fails to learn an effective routing or classification network (Muqeeth et al., 2024), which impairs the correct routing of input samples to their corresponding experts, ultimately resulting in suboptimal performance. In the MoE context, Top-k selectors , $k \geq 2$, are usually preferred (Shazeer et al., 2017).

For $k \geq 1$, the Top-k selector function $\mathcal{S}_k$ combines a masking operation, which retains only the k largest logits $\boldsymbol{z}^{(k)}$ of $\boldsymbol{z}$ (setting the rest to $-\infty$), with a temperature-scaled `Softmax` that normalizes the masked output to sum to 1, giving rise to Top-k selector $\mathcal{S}_k$. Formally, we define $\mathcal{S}_k(\boldsymbol{z}) := \sigma_\tau(\boldsymbol{z}^{(k)})$, where $\texttt{Softmax}\ \sigma_\tau(\boldsymbol{z})_i = \exp(\boldsymbol{z}_i/\tau)/\sum_{j=1}^{n}\exp(\boldsymbol{z}_j/\tau)$, $\tau > 0$ is the temperature parameter that controls the `Softmax` sharpness. Note that the

`Softmax` output for $-\infty$ is zero, so $\sigma_\tau(\boldsymbol{z}^{(k)})$ behaves as if only the Top-k entries are input to the function.



(a) `DTSemNet` Regression with STE

(b) `DTSemNet` Regression with Top-k

Figure 2: Architecture for extending `DTSemNet` to regression tasks. In (a), the thick line (orange) for backpropagation indicates that `DTSemNet` is updated using all regressor outputs $\boldsymbol{r}$, whereas in (b), only the selected leaf(s) is used for the update.

When used in `DTSemNet` for regression tasks, we replace the one-hot vector $\boldsymbol{h}$ by $\mathcal{S}_k(\boldsymbol{z}) = \boldsymbol{s}$, the final output is thus the weighted sum over the k leaf-specific regressors: $\hat{y} = \boldsymbol{s}^\top \mathcal{R}(\boldsymbol{x})$. For $k = 1$, the forward propagation yields the exact expected hard DT as a regressor, as $\boldsymbol{s} = \boldsymbol{h}$ the one hot-vector. For $k \geq 2$, the forward propagation is a weighted sum between the results of several (k) regressors at the leaves. The differentiability of the temperature-scaled `Softmax` in $\mathcal{S}_k$ allows gradients to flow through $\boldsymbol{s}$, while the Top-k masking operation acts as a fixed selection mechanism that preserves the computational graph thereby allowing informative gradients with $\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}$. That is, there is no mismatch, and thus no approximation, between the forward propagation of Top-k with the backward propagation of Top-k.

We compare in Fig. 2 the backpropagation of gradients with Top-2 against that with STE. There are two main differences between Top-k ($k \geq 2$) and STE. First, in STE, only a single regressor is updated (the one selected by the one-hot vector). In Top-2, two regressors will be updated. This may be beneficial, e.g. in the case that the current routing is inaccurate, and some internal choices send the input to the wrong regressor (which will often be the case at the beginning of the learning process): then the correct regressor will also get updated with the target value, and a switch of internal choices will be more likely to happen in the future as the other regressor will be more accurate on these inputs. Second, in STE, the outputs of *all* leaf regressors influence the gradient updates of *all* the internal node parameters of the `DTSemNet`, which is extreme as nodes very far from the current decision will get updated. By comparison, Top-k routing will update the decision nodes in the path from the root to one of the Top-k leaves. This will leave the decision nodes far from the actual decision for input $\boldsymbol{x}$ unchanged by the gradient, while changing a few nearby ones to make a later switch of decision more likely (if it improves the gradient). Additionally, the gradient of the regressor parameters $\boldsymbol{\Theta}$ is influenced by a weighted combination of the Top-k leaf predictions. This soft combination distributes gradient signals across multiple

Algorithm 1 `DTSemNet` Regression with Annealed Top-k Selector

1: **Input:** Initial parameters $(\mathbf{A}^0, \boldsymbol{b}^0, \boldsymbol{\Theta}^0)$, learning rate η, schedule $\mathcal{K}(t)$ specifying k for Top-k at each epoch t, dataset $(\boldsymbol{x}, y) \sim \mathbb{D}$, and total epochs T
2: **for** $t = 0, 1, \ldots, T$ **do**
3: **if** $k \geq 2$ **then** ▷ **Joint training** of DT and regressors
4: Selector output $\boldsymbol{s} = \mathcal{S}_{\mathcal{K}(t)}(\mathcal{N}_{\mathcal{T}}(\boldsymbol{x}))$
5: Regressor output $\hat{y} = \langle \boldsymbol{s}, \boldsymbol{r} \rangle$ and compute loss $\mathcal{L}(\hat{y}, y)$
6: Update parameters: $(\mathbf{A}^t, \boldsymbol{b}^t, \boldsymbol{\Theta}^t) \leftarrow (\mathbf{A}^{t-1}, \boldsymbol{b}^{t-1}, \boldsymbol{\Theta}^{t-1}) - \eta \nabla \mathcal{L}$
7: **else** ▷ Top-1: **Fine-Tuning** of regressors
8: **if** $t < T_f$ **then** Fine-tune $\boldsymbol{\Theta}$ with *augmented samples* routed to the leaf
9: **else** Fine-tune $\boldsymbol{\Theta}$ with *actual samples* routed to the leaf
10: **end if**
11: **end for**
12: **Return:** Final parameters $(\mathbf{A}^T, \boldsymbol{b}^T, \boldsymbol{\Theta}^T)$

leaf nodes, enabling more stable training and accurate gradient flow to both the `DTSemNet` and regressors, in contrast to STE-based methods, which propagate inaccurate gradients to internal nodes of `DTSemNet`.

3.4.5 Annealing from Top-k to Top-1

We are interested to learn a Top-1 selector, as $\boldsymbol{s}$ reduces to the one-hot vector $\boldsymbol{h}$, and only one regressor $\boldsymbol{\theta}_\ell$ is called per input, the one selected by the DT, which is the expected semantics of `DTSemNet` for regression tasks. However, we cannot learn Top-1 directly because the gradient is constant and non-informative during backpropagation. Instead, we propose Algorithm 1, which has two phases of training. It begins training `DTSemNet` with Top-k (first phase), $k \geq 2$ (e.g., $k = 4$), softly routing each input to multiple leaves (lines 3–6). This promotes broader exploration and richer gradient signals. During training, we gradually anneal k to a smaller value according to a schedule $\mathcal{K}(t)$, where t is the epoch number. This sharpens the selection and induces sparser routing. The temperature parameter τ in `Softmax` further controls the sharpness of the output, where lower values make the selector more discrete, while higher values promote uniform distributions. At the end of the first training phase, we begin the second phase with $k = 1$ (lines 7–9), where only the leaf regressors are further updated. The internal node parameters are frozen because gradients no longer propagate through `DTSemNet`. This is fine as the internal node parameters have been trained before using Top-k, in the steps where $k \geq 2$. For fine-tuning the regressor, we use a two-stage process. In the first stage, we train with *augmented samples* to combat *overfitting* and provide a strong initialization for the next stage. Specifically, when the training epoch is less than T_f, we route augmented samples to each leaf, that is, samples for which the leaf is either the first or second choice. This augmentation includes boundary samples near transition regions, thereby capturing training points not originally routed to the leaf but likely close to testing samples that will be. Training on this augmented dataset reduces overfitting and prepares the regressor with a better initialization. In the second

(and final) stage, we fine-tune using only the *actual* samples routed to each leaf, thereby fitting the regressor more precisely to the given dataset.

This two-phase (for $k \geq 2$ and $k = 1$) training strategy balances exploratory learning ($k \geq 2$) during structure formation with precise fitting (leaf fine-tuning) of regressors in the final ($k = 1$) phase. The Top-k' step can be seen as a precise initialization of Top-k step with $k < k'$. At each step, *no approximation* is used, as the *evaluation function* used to compute the output in the forward pass matches semantically with the *optimization function* used to compute the gradients in the backward pass. This contrasts with STE, where the optimization function is a smoothened approximation of the evaluation function, creating a semantic mismatch.

3.5 `DTSemNet` for RL:

To tackle RL tasks, we use standard Deep RL frameworks (e.g., StableBaselines3 (Raffin et al., 2021) and CleanRL (Huang et al., 2022)), simply replacing the NN with the `DTSemNet` architecture as a programmatic policy and running gradient descent within the RL framework. For environments with discrete action spaces, we use PPO (Schulman et al., 2017) with a `DTSemNet`-classification model, assigning one class per action. For environments with continuous action spaces, we use SAC (Haarnoja et al., 2018) with a `DTSemNet`-regression model, assigning one linear regressor per action dimension (e.g., two for a continuous lunar lander: one for horizontal and one for vertical engines). Compared with the earlier conference version (Panda et al., 2024a), we compare STE vs Top-k for RL with continuous action space. To this end, we modify the SAC (Haarnoja et al., 2018) training loop as follows. During sample collection the `DTSemNet` model always uses Top-1, which serves as the output policy. At each update step the collected samples are used to train the model in three phases. First, training is performed with Top-k selection for a fixed number of gradient steps. This is followed by training with augmented samples, and finally by Top-1 selection for half as many gradient steps, consistent with the procedure outlined for `DTSemNet`-regression.

3.6 Comparison with Other Encodings of DTs into NNs

DGT (Karthikeyan et al., 2022) encodes oblique decision trees using the `Sign` activation at each layer. Because `Sign` is non-differentiable (unlike the `ReLU` activation used in `DTSemNet`), DGT resorts to a *quantized* gradient descent strategy, with an STE approximation applied at every node. In regression tasks, DGT assigns a scalar to each leaf, in contrast to the linear regressors used in `DTSemNet`-regression. The ICCT architecture (Paleja et al., 2022) targets axis-aligned DTs for RL tasks. Each leaf is associated with the product of edge weights along its path, computed in logarithmic space to avoid explicit multiplications. A `Sigmoid` activation is applied at each decision node, yielding a soft DT approximation. During RL training, this soft DT is repeatedly "crispified" into a hard (axis-aligned) DT. To backpropagate through the resulting non-differentiable Heaviside step function, ICCT again relies on STE approximation, which `DTSemNet`-classification avoids. For `DTSemNet`-regression, our earlier work (Panda et al., 2024a) used a single STE at the output layer, which still introduced approximation. In this work, we propose a Top-k approach that eliminates this reliance, thereby offering a comprehensive framework that avoids approximation in both classification and regression settings.

4. Experimental Evaluation

In this section, we evaluate the performance of `DTSemNet`, comparing it against competing methods for learning hard DTs. We begin with supervised learning setups, using a collection of benchmark datasets for both multi-class classification and regression. On these datasets, we compare test accuracy against state-of-the-art (SOTA) methods: TAO (Carreira-Perpinán and Tavallali, 2018; Zharmagambetov and Carreira-Perpinan, 2020), a non-greedy approach for learning oblique DTs; DGT (Karthikeyan et al., 2022), a gradient descent-based method also for oblique DTs; CRO-DT (Costa et al., 2024), a global search-based method for axis-aligned DTs; and CART, a standard greedy algorithm for axis-aligned DTs.

We first discuss the performance of the `DTSemNet` Top-k approach on regression datasets, which constitutes a major contribution of this paper beyond our earlier work on `DTSemNet` (Panda et al., 2024b). For regression, we also analyze the effect of using Top-k selection versus the STE on the learning dynamics using a synthetic dataset.

We then present results on classification datasets, along with training time comparisons. Note that training time is reported only for benchmarks where such data is available. TAO could not be evaluated in our setting due to the lack of publicly available implementation. To further understand the learning behavior of `DTSemNet` compared to DGT, we also analyze their loss landscapes following the approach of Li et al. (2018), along with a discussion on the generalization gap based on the difference between training and test accuracies.

Finally, we extend the evaluation to RL environments, considering both discrete and continuous action spaces. We compare `DTSemNet` with DGT (both learning oblique DT policies via gradient descent), ICCT (learning axis-aligned DT policies via gradient descent), and VIPER, which learns axis-aligned DT policies via imitation learning from a NN policy trained with Deep RL. The performance of the underlying NN policy is also reported as a baseline.

We implemented `DTSemNet` and conducted all experiments using Python and PyTorch. Our testing platform has 8 CPU cores (AMD 75F3, Zen 3 architecture), 128 GB of RAM, and a 2 GB GPU (NVIDIA Quadro P620). The supplementary material (Panda et al., 2024b) provides additional results and details regarding datasets, train-test splits, hyperparameters, etc. Our source code is available on GitHub under the new branch "`topk`": `https://github.com/CPS-research-group/dtsemnet/tree/topk`.

4.1 Regression Tasks

We begin by evaluating the Root Mean Squared Error (RMSE) performance of the proposed regression methods on standard tabular benchmarks. Following this, we conduct an ablation study to investigate the effect of using Top-k routing in `DTSemNet` on learning dynamics in comparison to STE by focusing on parameter updates and overall performance. For this, we design a controlled synthetic regression task to isolate and examine how STE introduces estimation errors during the optimization of decision parameters, which in turn impairs routing accuracy and ultimately degrades regression performance.

Table 1: Average RMSE results (lower is better) on regression tasks, reported as mean ± standard deviation over 10 runs. The number of features N_f and the number of training samples N_s are provided for each dataset. DGT-linear is a modified version of DGT (Karthikeyan et al., 2022) that we implemented to incorporate regressors at the leaves. The datasets are taken from (Zharmagambetov and Carreira-Perpinan, 2020). For `DTSemNet` Top-k, `DTSemNet` STE, DGT-linear, and TAO-linear, the tree height is fixed following (Zharmagambetov and Carreira-Perpinan, 2020), whereas CART does not use a fixed height. CART results are taken from (Karthikeyan et al., 2022), and TAO-linear results from (Zharmagambetov and Carreira-Perpinan, 2020). The ***Avg* % *distance to best*** is computed by dividing each model RMSE by the best RMSE achieved on that dataset, then averaging these normalized values across all datasets (lower distance is better).

Dataset	N_f, N_s	DT Height	`DTSemNet` Top-k	`DTSemNet` STE	DGT-Linear	TAO-Linear	CART
Abalone	10, 2004	5	2.13 ± 0.01	2.14 ± 0.03	2.14 ± 0.03	$\mathbf{2.07 \pm 0.01}$	2.29 ± 0.034
Comp-Active	21, 3932	5	$\mathbf{2.48 \pm 0.03}$	2.65 ± 0.18	2.65 ± 0.15	2.58 ± 0.02	3.35 ± 0.221
Ailerons	40, 5723	5	$\mathbf{1.65 \pm 0.01}$	1.66 ± 0.01	1.67 ± 0.017	1.74 ± 0.01	2.01 ± 0.00
YearPred	90, 370972	6	$\mathbf{8.99 \pm 0.01}$	$\mathbf{8.99 \pm 0.01}$	9.02 ± 0.025	9.08 ± 0.03	9.69 ± 0.00
CTSlice	384, 34240	5	$\mathbf{1.09 \pm 0.04}$	1.45 ± 0.12	1.78 ± 0.25	1.16 ± 0.02	5.78 ± 0.224
Avg* % *distance to best ↓			**0.6**%	8.8%	15%	3.4%	101%

4.1.1 Performance on Regression Benchmark

We first evaluate regression performance (RMSE) on tabular benchmarks by considering five regression datasets from (Zharmagambetov and Carreira-Perpinan, 2020), along with ten additional datasets from (Grinsztajn et al., 2022; Karthikeyan et al., 2022), for which TAO results and code are not publicly available (we requested access; no code was available at the time of submission). While TAO-linear (Zharmagambetov and Carreira-Perpinan, 2020) generates DT with regressors at the leaves, similarly as `DTSemNet`, allowing them to generalize, DGT (Karthikeyan et al., 2022) only learns scalars at the leaves, similarly as CART, and thus is less efficient (and cannot generalize), needing deeper DTs to get acceptable accuracy. For a fair comparison, we implemented DGT-Linear, a modification of DGT using regressors at the leaves, in the same way as in `DTSemNet` and in ICCT (Paleja et al., 2022). Results for the original DGT with scalars at the leaves can be found in (Panda et al., 2024a), and they are not better than DGT-Linear in any case, as expected. Following (Zharmagambetov and Carreira-Perpinan, 2020), in Tables 1 and 2 we fix the height to the same (small) value when evaluating all architectures with regressors at the leaves, namely `DTSemNet` Top-k, `DTSemNet` STE, DGT-Linear, and TAO-Linear. CART is allowed to produce DTs as deep as needed, since it generates non-oblique, axis-aligned trees with scalars at the leaves, and therefore requires deep trees to achieve reasonable performance. For `DTSemNet` Top-k, we empirically set the annealing schedule to $(4 \rightarrow 1)$ with temperature $\tau = 0.5$ across all datasets.

`DTSemNet` Top-k demonstrates the best performance on almost all datasets compared to all other models, except in two cases: On the PDB-Bind dataset (Table 2), Top-k is very slightly outperformed ($< 1\%$) by STE in PDBBind, within the standard deviation (± 0.01). On this test, the very limited number of leaves (four) does not discriminate much

between STE and Top-k. For Abalone (Table 1), `DTSemNet` Top-k ranks second, $< 3\%$ away from TAO-Linear. Apart from these, `DTSemNet` Top-k achieves the best performance on all other datasets and is statistically significantly better than all competitors across datasets, as confirmed by the Wilcoxon signed-rank test at the 0.05 significance level. It outperforms the state of the art TAO-Linear on 4 out of 5 benchmarks, significantly so for 3 of them: Comp-Active (4%), Ailerons (5%), and CTSlice ($> 6\%$). On average, it produces DTs 3% more accurate than TAO-Linear for regression tasks. This is a significant improvement compared to `DTSemNet` STE, which was underperforming TAO-Linear in 3 out of 5 cases, including 25% less accurate on CTSlice. Overall, `DTSemNet` Top-k shows an average RMSE advantage over STE of approximately 4% across all datasets, with a notable 33% improvement on CTSlice and 15% on Pol. The main reason is that STE underutilizes available leaves, concentrating data on fewer leaves, as illustrated in Fig. 3. We will experimentally explain the reason why Top-k learns more balanced leaves utilization than STE in Section 4.1.2. Concerning DGT-Linear, `DTSemNet` Top-k triples the lead (at 8%) that `DTSemNet` STE already have over DGT-Linear, with a substantial 60% improvement on the CTSlice dataset. The historical method from CART produces particularly inaccurate DTs ($> 50\%$ less accurate on average), in particular 5 times less accurate for CTSlice. Overall, `DTSemNet` Top-k ranks very high across all datasets, highlighting the benefits of avoiding approximations in differentiable trees.

4.1.2 Ablation Studies to Understand Training with Top-k vs. STE

We now perform an ablation study to understand which difference(s) between STE and Top-k training (updating via the gradient of *one* regressor in STE vs. k regressors in Top-k; updating via the gradient of all paths from all leaves/regressors to the DT root node in

Table 2: Average RMSE results (lower is better) on regression tasks, reported as mean $\pm$ standard deviation over 10 runs. The number of features N_f and the number of training samples N_s are provided for each dataset. The datasets are from (Grinsztajn et al., 2022; Karthikeyan et al., 2022), for which TAO-linear results are unavailable. For `DTSemNet` Top-k, `DTSemNet` STE, and DGT-Linear, the tree height is fixed using the best-performing height from $\{3, 4, 5, 6\}$, whereas CART does not use a fixed height.

Dataset	N_f, N_s	Height	`DTSemNet` Top-k	`DTSemNet` STE	DGT-Linear	CART
Medical	5, 114145	5	$\mathbf{0.84 \pm 0.01}$	0.85 ± 0.02	0.85 ± 0.07	1.46 ± 1.40
Sulfur	6, 7056	5	$\mathbf{0.34 \pm 0.01}$	0.35 ± 0.01	0.35 ± 0.01	0.43 ± 0.03
Bike Sharing	6, 12165	6	$\mathbf{1.15 \pm 0.01}$	1.18 ± 0.01	1.22 ± 0.02	1.22 ± 0.04
Houses	8, 14448	5	$\mathbf{0.28 \pm 0.00}$	$\mathbf{0.28 \pm 0.00}$	$\mathbf{0.28 \pm 0.01}$	0.45 ± 0.01
Wine Quality	11, 4547	5	$\mathbf{0.69 \pm 0.005}$	0.70 ± 0.004	0.70 ± 0.005	0.78 ± 0.01
Elevator	16, 11619	4	$\mathbf{0.20 \pm 0.00}$	0.21 ± 0.00	0.21 ± 0.00	0.54 ± 0.01
Pol	26, 10500	5	$\mathbf{6.61 \pm 0.34}$	7.62 ± 1.38	8.03 ± 0.95	11.02 ± 1.05
Superconduct	79, 14884	5	$\mathbf{1.36 \pm 0.02}$	1.40 ± 0.01	1.47 ± 0.03	1.51 ± 0.01
Microsoft	136, 578729	5	$\mathbf{0.77 \pm 0.00}$	$\mathbf{0.77 \pm 0.00}$	$\mathbf{0.77 \pm 0.00}$	$\mathbf{0.77 \pm 0.00}$
PDBBind	2052, 9013	2	1.34 ± 0.01	$\mathbf{1.33 \pm 0.01}$	1.34 ± 0.01	1.55 ± 0.00
Avg % distance to best $\downarrow$			**0.1**%	3.1%	4.7%	50%

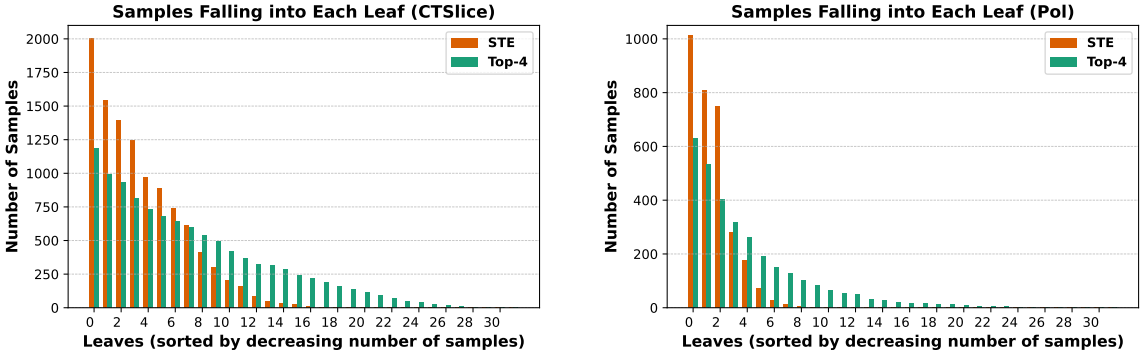
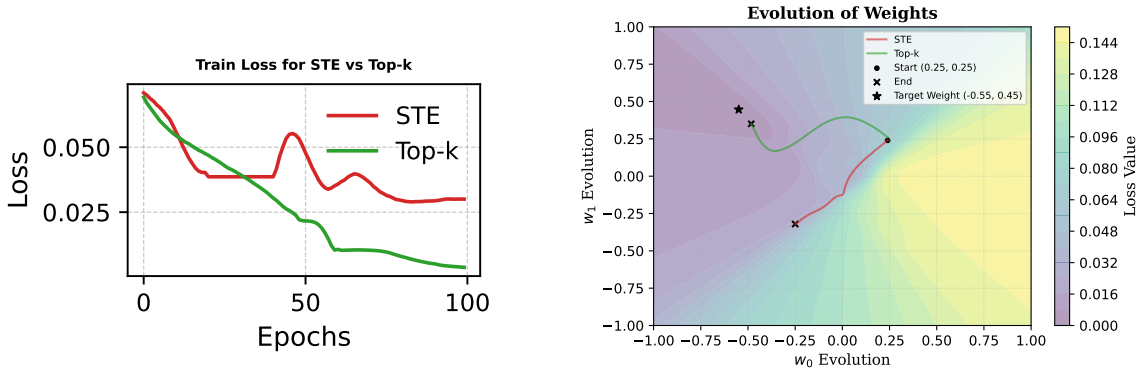


Figure 3: Sample distribution across leaves for the CTSlice and Pol datasets, showing that Top-k ($k = 4$) achieves more balanced leaf utilization. Results are averaged over 10 seeds.

STE vs. paths from k leaves/regressors to the DT root node in Top-k) is responsible for producing significantly more accurate DTs. We perform two distinct ablation studies:

(a) To understand the difference between updating, during backpropagation, the internal node parameters based on all leaves/regressors (in STE) vs. only the top k leaves/regressors, we design a synthetic test where only the root is updated (while other nodes and regressors are frozen). This allows us to work in low parametric dimensions (2), which can be represented graphically; and

(b) To understand the impact of updating one vs. top k regressors, we cannot symmetrically freeze the DT, because training k regressors is only meaningful in the early phase of learning (to enable switching in decisions), when routing in the DT is uncertain. Instead, we consider a benchmark that highlights differences in final leaves utilization between STE and Top-k (see Fig. 3). We then study how this balance evolves over time from a similar



(a) Train Loss

(b) Evolution of weights $\boldsymbol{w}$

Figure 4: Comparison of training dynamics between `DTSemNet` with Top-k routing ($k = 2$) and STE in a `DTSemNet`-regression model of height 5, using a synthetic regression dataset generated by a teacher DT. The student network is trained to imitate the teacher.

initialization, comparing STE, Top-k, and a hybrid Top-k/Reg-1 method, which behaves similarly to Top-k except that it updates only the top-1 regressor during backpropagation, as in STE.

(a) Training the Weight of the DT Root Node with Fixed Regressors.

The random DT serves as the teacher model, providing the training dataset (pairs of inputs and regression values to reach) that the student models will try to imitate. The student models (STE and Top-k) use a `DTSemNet` with the same architecture as the teacher DT (height 5, input $\boldsymbol{x} \in \mathbb{R}^2$), but with some weights missing that need to be trained, while the others are fixed to the teacher's weights and frozen. To ensure fairness, we initialize both STE and Top-k with the same `DTSemNet` (i.e., the missing weights are initialized in the same way). The dataset generated from the teacher comprises 30K samples, obtained by querying the teacher model with inputs uniformly sampled from the domain $[-1, 1]^2$.

All weights from the teacher network are copied to the student network, except for the weights at the root node of the DT, denoted by $\boldsymbol{w}$. Specifically, the teacher uses $\boldsymbol{w} = (-0.55, 0.45)$, while the student is initialized with $\boldsymbol{w} = (0.25, 0.25)$. The objective is for the student to learn the target $\boldsymbol{w}$ of the teacher, with all other parameters fixed and a learning rate of 0.01. Recall that the one-hot vector representing the semantics of `DTSemNet`, equivalent to Top-k with $k = 1$, has no gradient backpropagated to internal nodes. To perform gradient descent on internal nodes, one must either rely on STE to create a gradient or on Top-k with $k \geq 2$. We compare learning with STE and Top-k ($k = 2$) here, while evaluation is performed using the standard one-hot/Top-1 semantics of `DTSemNet`. As shown in Fig. 4, training with the STE converges to a suboptimal local minimum. In contrast, Top-k routing leads to more stable and consistent progress toward the target weights. This confirms the claim that the mismatch between the forward and backward objectives in STE leads to inaccurate gradient updates to the internal weights of the DT, ultimately hampering performance. On the other hand, there is no such problem when using Top-2 for learning the decisions but interpreting them in a Top-1 fashion.

(b) Evolution of Leaves Balance with STE, Top-k, and Top-k/Reg-1.

We analyze the sample distribution across leaves, i.e., the number of samples routed to each leaf, using regression benchmark datasets. For two datasets where there is a significant performance gap between STE and Top-k ($\geq 10\%$), the distributions are shown in Fig. 3. We observe that Top-k ($k = 4$) achieves a more balanced utilization of regressors, following a long-tailed distribution, whereas STE underutilizes many leaves.

To further investigate this phenomenon (underutilization of leaves in STE versus balanced utilization in Top-k) we examine in Fig. 5 the evolution of sample distributions across leaves over training epochs on the CTSlice dataset, where the performance gap is particularly pronounced ($\geq 30\%$).

At the beginning of training, the leaf distributions of all three models (STE, Top-4/Reg-1, and Top-4) are similar. As training progresses, Top-k is observed to use leaves in a more balanced manner.

Our hypothesis is that Top-k, by updating multiple regressors during training, adapts more effectively to changes in decision boundaries, resulting in more balanced regressor utilization. To test this hypothesis, we modify Top-k so that only one regressor is updated

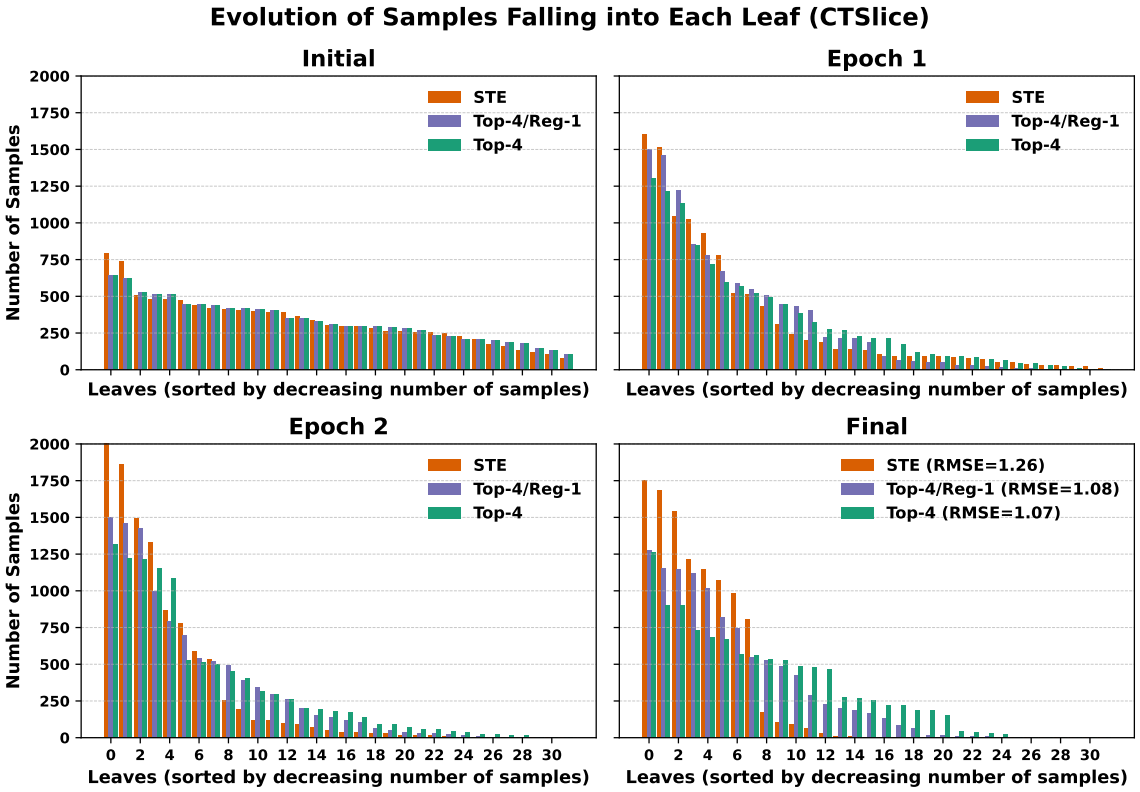


Figure 5: Sample distribution across leaves / regressors for CTSlice at the beginning of training, at epochs 1 and 2, and after training for a single seed. Initially, the sample distribution across leaves is similar for all models. Progressively, `DTSemNet` with STE shows underutilization of leaves/regressors, whereas Top-k ($k = 4$) achieves more balanced utilization, and Top-k/Reg-1 being in-between.

per sample (as in STE), which we call Top-4/Reg-1. As shown in Fig. 5, STE and Top-4/Reg-1 utilize fewer regressors than Top-4 from Epoch 1 onwards.

After the first epoch, Top-4/Reg-1 uses far fewer leaves than Top-4: Top-4 utilizes 22 regressors with more than 100 samples, whereas Top-4/Reg-1 utilizes 16 regressors. This confirms that updating a single regressor has a significant impact on the balance of leaf distribution.

Interestingly, at the end of learning, Top-4 utilizes 20 regressors with more than 180 samples, compared with 15 regressors for Top-4/Reg-1 and only 8 regressors for STE, an inversion compared with Epoch 1 between Top-4/Reg-1 and STE. On top of the hypothesis we made and confirmed (that updating several regressors helps balance the regressors), there is an even stronger factor, acting over the long term, that explains why STE has a very narrow use of regressors. Because STE updates the internal nodes in the DT based on all the regressors (this is the only difference with Top-4/Reg-1, and we already proved in (a) that it was detrimental to learning a good DT), it prefers to use fewer regressors (≈ 8) compared to Top-4/Reg-1 to alleviate this factor.

In summary, updating DT parameters with all regressors, as in STE, has the most harmful effect, which is further compounded when updates are restricted to a single regressor. Top-k avoids both issues, resulting in better performance.

4.2 Classification Tasks

For classification tasks, we evaluate the performance of `DTSemNet` by comparing its test accuracy against other DT training approaches. We report results for both concise and dense trees, noting that concise DTs are generally more challenging to train effectively (since they have fewer choices of oblique hyperplanes to partition the input space). Prior work on CRO-DT (Costa et al., 2024) reports results only for relatively concise trees (height ≤ 4). For deeper trees, CRO-DT exhibits a substantial performance drop, and since we were unable to configure the publicly available implementation to handle such depths reliably, we omit its results. We then report the training time of the different DT learning methods to assess training efficiency. For ablation studies, we analyze `DTSemNet` and DGT to study the impact of the STE approximation on learning. Then, we discuss the performance of `DTSemNet` compared to soft DTs.

Results on Concise DTs. We now turn to the 14 classification tabular datasets used in CRO-DT (Costa et al., 2024). Global search methods such as CRO-DT (Costa et al., 2024) are efficient only for small DTs (here, up to depth 4, i.e., 32 nodes). We sort the benchmarks by the number of features, since handling more features is increasingly difficult, particularly for small DTs. Table 3 reports the (average) score over 100 DTs trained with different seeds for the most accurate height (up to 4). Across all benchmarks, `DTSemNet` consistently produces the most accurate DTs, with the largest improvement observed on

Table 3: Percentage accuracy (higher is better) on **classification tasks** with the most accurate height ≤ 4, as tested in Costa et al. (2024). For each dataset, we provide the number of features N_f, classes N_c and training samples N_s. Averaged accuracy $\pm$ std is reported over 100 runs. The best-performing height is reported in parentheses for each architecture. We run the `DTSemNet` and DGT experiments, while the results for TAO, CART and CRO-DT are copied from (Costa et al., 2024).

Dataset	N_f, N_c, N_s	`DTSemNet`	DGT	TAO	CART	CRO-DT
Balance Scale	4, 3,625	$\mathbf{90.2 \pm 2.2}$ (2)	88.6 ± 1.7 (4)	77.4 ± 3.1 (4)	74.9 ± 3.6 (4)	77.8 ± 3.0 (4)
Banknote Auth	4, 4, 1372	$\mathbf{99.8 \pm 0.4}$ (3)	$\mathbf{99.8 \pm 0.4}$ (3)	96.6 ± 1.3 (4)	93.6 ± 2.2 (4)	95.2 ± 2.2 (4)
Blood Transfusion	4, 2,784	$\mathbf{78.5 \pm 1.7}$ (2)	78.3 ± 2.4 (4)	76.9 ± 2.1 (3)	77.1 ± 1.8 (4)	76.1 ± 1.9 (3)
Acute Inflam. 1	6, 2, 120	$\mathbf{100 \pm 0.0}$ (2)	$\mathbf{100 \pm 0.0}$ (4)	99.7 ± 1.2 (4)	$\mathbf{100 \pm 0.0}$ (3)	$\mathbf{100 \pm 0.0}$ (2)
Acute Inflam. 2	6, 2, 120	$\mathbf{100 \pm 0.0}$ (2)	$\mathbf{100 \pm 0.0}$ (4)	99.0 ± 2.6 (2)	99.0 ± 2.6 (2)	$\mathbf{100 \pm 0.0}$ (2)
Car Evaluation	6, 4, 1728	$\mathbf{93.3 \pm 2.2}$ (4)	92.1 ± 2.4 (4)	84.5 ± 1.5 (4)	84.3 ± 1.4 (4)	86.1 ± 1.3 (4)
Breast Cancer	9, 2, 683	$\mathbf{97.2 \pm 1.3}$ (2)	$\mathbf{97.2 \pm 1.2}$ (2)	94.7 ± 1.6 (3)	94.7 ± 1.7 (3)	95.5 ± 1.8 (2)
Avila Bible	10, 12, 10430	$\mathbf{62.2 \pm 1.4}$ (4)	59.7 ± 1.8 (4)	55.8 ± 0.8 (4)	54.0 ± 1.3 (4)	59.6 ± 0.7 (4)
Wine Quality Red	11, 6, 1599	$\mathbf{58.6 \pm 2.2}$ (3)	56.6 ± 1.4 (4)	56.9 ± 2.5 (4)	55.9 ± 2.3 (4)	55.8 ± 2.2 (2)
Wine Quality White	11, 7, 4898	$\mathbf{53.5 \pm 1.4}$ (4)	52.1 ± 1.6 (4)	52.3 ± 1.4 (4)	52.0 ± 1.3 (4)	51.4 ± 1.2 (2)
Dry Bean	16, 7, 13611	$\mathbf{91.4 \pm 0.5}$ (4)	89.0 ± 1.6 (4)	83.2 ± 1.5 (4)	80.5 ± 1.9 (4)	77.9 ± 4.7 (4)
Climate Crashes	18, 2, 540	$\mathbf{92.9 \pm 1.4}$ (2)	92.4 ± 2.4 (3)	90.6 ± 2.2 (3)	91.8 ± 1.8 (4)	91.5 ± 2.0 (2)
Conn. Sonar	60, 2, 208	$\mathbf{82.1 \pm 5.1}$ (4)	80.8 ± 5.3 (4)	70.9 ± 5.8 (4)	70.6 ± 6.6 (4)	71.7 ± 6.7 (4)
Optical Recognition	64, 10, 3823	$\mathbf{93.3 \pm 1.0}$ (4)	91.9 ± 1.0 (4)	64.6 ± 6.5 (4)	53.2 ± 3.2 (4)	65.2 ± 2.0 (4)
Avg % distance to best $\downarrow$		0.0%	1.4%	7.4%	9.2%	7.3%

Dry Beans, where the classification error decreases from 11% to 8.6%. DGT ranks second overall, except on the two wine quality benchmarks, where TAO outperforms it. Typically, DGT comes close to `DTSemNet`, which is expected given their similarity in design, though the use of approximations (STEs, quantized gradient descent) makes DGT perform on average 0.85% worse than `DTSemNet`. Notably, this gap increases to 1.3% on the four benchmarks with the most features (the most challenging cases). Moreover, DGT often requires larger trees than `DTSemNet` to achieve its best results (in 6 out of 14 benchmarks). When compared with non–gradient-based methods, the advantage of `DTSemNet` is striking: 5.5% on average, rising to 12% on the four benchmarks with the most features. The largest margin is observed in *Optical Recognition*, where the classification error drops from 34.8% to only 6.7%.

Results on Denser DTs. We then consider the 8 classification tasks originally introduced in TAO (Carreira-Perpinán and Tavallali, 2018) and later used in (Karthikeyan et al., 2022). All of these involve tabular datasets, except for MNIST, which is a small image dataset. We follow the fixed tree heights specified in (Carreira-Perpinán and Tavallali, 2018) (and reused in (Karthikeyan et al., 2022)). The benchmarks are sorted by tree height, which serves as a good indicator of task complexity. Table 4 reports the (average) accuracy over 10 DTs trained with different seeds. Consistent with the results in Table 3 for shallower DTs, `DTSemNet` achieves the highest accuracy across all benchmarks with deeper DTs as well. With deeper trees, TAO improves and ranks second in 5 tasks, compared to 3 for DGT. On average, `DTSemNet` outperforms TAO by 1%, with the gap increasing to 1.4% on the 4 benchmarks with deeper trees. The largest differences are observed in *Letter*, where the classification error decreases from 12.6% to 10.8%, and in *Pendigits*, from 3.9% to 3%. Relative to DGT, `DTSemNet` achieves an average improvement of 1.4%, which grows to 2.1% on the four benchmarks with deeper trees, larger than the margin observed in smaller trees from Table 3. The most pronounced improvements occur in *Letter*, with error reduced from 13.9% to 10.8%, and in *MNIST*, from 6% to 3.9%. Finally, a Friedman–Nemenyi test

Table 4: Percentage accuracy (higher is better) on **classification tasks** for the datasets reported in (Carreira-Perpinán and Tavallali, 2018). For each dataset, we provide the number of features N_f, classes N_c, and training samples N_s. All methods use the tree height fixed in (Karthikeyan et al., 2022) (except CART, which has no predefined height). Averaged accuracy ± std is reported over 10 runs. The results for DGT and CART are from (Karthikeyan et al., 2022), and the results of TAO are from (Carreira-Perpinán and Tavallali, 2018).

Dataset	N_f, N_c, N_s	Height	`DTSemNet`	DGT	TAO	CART
Protein	357, 3, 14895	4	$\mathbf{68.60 \pm 0.22}$	67.80 ± 0.40	68.41 ± 0.27	57.53 ± 0.00
SatImages	36, 6, 3104	6	$\mathbf{87.55 \pm 0.59}$	86.64 ± 0.95	87.41 ± 0.33	84.18 ± 0.30
Segment	19, 7, 1478	8	$\mathbf{96.10 \pm 0.53}$	95.86 ± 1.16	95.01 ± 0.86	94.23 ± 0.86
Pendigits	16, 10, 5995	8	$\mathbf{97.02 \pm 0.32}$	96.36 ± 0.25	96.08 ± 0.34	89.94 ± 0.34
Connect4	126, 3, 43236	8	$\mathbf{82.03 \pm 0.39}$	79.52 ± 0.24	81.21 ± 0.25	74.03 ± 0.60
MNIST	780, 10, 48000	8	$\mathbf{96.16 \pm 0.14}$	94.00 ± 0.36	95.05 ± 0.16	85.59 ± 0.06
SensIT	100, 3, 63058	10	$\mathbf{84.29 \pm 0.11}$	83.67 ± 0.23	82.52 ± 0.15	78.31 ± 0.00
Letter	16, 26, 10500	10	$\mathbf{89.19 \pm 0.29}$	86.13 ± 0.72	87.41 ± 0.41	70.13 ± 0.08
Avg %	***dist. to best*** ↓		0.0%	1.6%	1.1%	9.8%

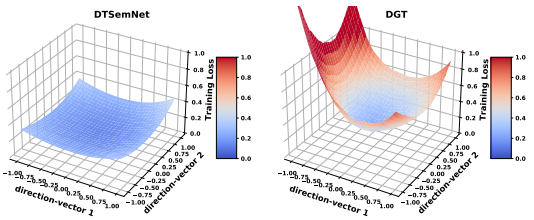
across all classification benchmarks at a significance level of 0.05 confirms that `DTSemNet` is significantly better than all other methods. The average ranks are: 1.11 for `DTSemNet`, 2.25 for DGT, 2.86 for TAO, 3.60 for CRO-DT, and 3.77 for CART.

Training Time. We report in Table 5 the training time for the architecture on MNIST, as (Carreira-Perpinán and Tavallali, 2018) reports this number for TAO. We train other architectures on comparable computing configurations. We provide the accuracy number obtained for reference. We also report the training time for the simpler DryBean, for comparison sake, except for TAO, for which this is not available. First, CRO-DT for MNIST takes much longer to train (number of generations set to the default 4k) than other architectures while having very low accuracy numbers ($< 60\%$ instead of $> 90\%$), the reason why we do not consider it for these benchmarks with deeper DTs (indeed, (Costa et al., 2024) does not report CRO-DT results for these benchmarks). The training times of DGT and `DTSemNet` are almost identical due to their architectural similarities. As expected, training with gradient-based methods is observed to be significantly faster than non-gradient learning methods.

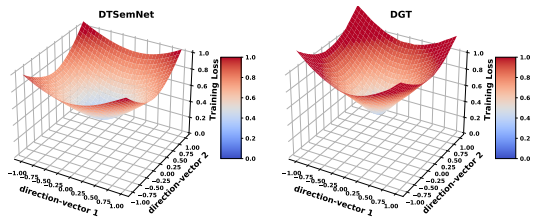
Table 5: Training times in seconds (lower is better), and (avg accuracy % (higher is better)). The MNIST training time from the non-publicly available TAO is quoted from (Carreira-Perpinán and Tavallali, 2018), while we train other architecture on a similar compute configuration. The tree height is 8 for MNIST and 4 for DryBean.

Dataset	`DTSemNet`	DGT	TAO	CRO-DT
MNIST	306 (96.1)	288 (94.0)	1200 (95.0)	4659 (58.2)
DryBean	4.4 (91.4)	3.8 (89.0)	NA (83.2)	1300 (77.9)

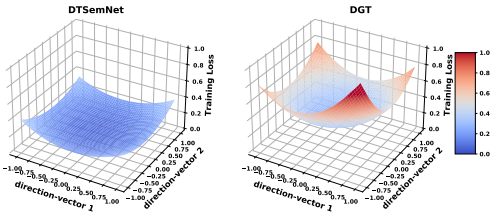
Comparision with DGT. To better understand the difference in performance due to the difference in architecture used by `DTSemNet` and DGT, we resort to two difference studies. First, the loss landscape (Li et al., 2018) (see Fig. 6) to understand how easy learning is for a given architecture; and generalizability potential, by considering the difference between train and test accuracies (in Table 6). We restrict to the 4 datasets from Table 3 for which there is a significant difference ($> 1\%$) in accuracy on the test data, namely SatImages, Connect4, MNIST and Letter. For both studies, we rerun the algorithms on the datasets, implying a small deviation in test accuracy with the results from Table 3. The training times are reported in brackets in Table 6. Fig. 6 shows the loss landscape around the final trained parameters along two random vector directions of training parameters. A flatter loss landscape indicates that changing the trained parameters does not significantly impact the (trained) accuracy, leaving flexibility to adjust the parameters to accommodate updates during training without affecting the current accuracy. On the contrary, a very steep loss landscape means that parameters cannot be changed without worsening the loss, so they must be tuned very precisely. In Fig. 6, the loss landscapes of SatImages and MNIST are much flatter for `DTSemNet` than for DGT, while those of Connect4 and Letter are less flat. The fact that `DTSemNet` is easier to train than DGT on MNIST and SatImages results in better training accuracy (Table 6). This largely explains the better *test* accuracy; generalizability is not improved, as computed by the difference in accuracy between train and test. The case is similar for Letter: although the loss landscape is not flat, there is still



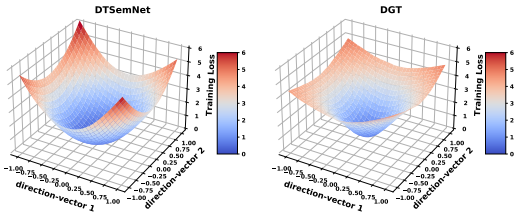
(a) SatImages



(b) Connect4



(c) MNIST



(d) Letter

Figure 6: The loss landscape of `DTSemNet` and DGT for different datasets. A flatter landscape allows flexible parameter updates without affecting training loss.

Table 6: Performance comparison of `DTSemNet` and DGT in terms of train and test accuracy. Training time is provided in brackets. The results for DGT are obtained on our machine.

Dataset	Train Accuracy		Test Accuracy		Difference: (Train Acc. – Test Acc.)	
	`DTSemNet`	DGT	`DTSemNet`	DGT	`DTSemNet`	DGT
SatImages	$\mathbf{95.30 \pm 0.60}$ (42s)	92.22 ± 0.80 (25s)	$\mathbf{87.55 \pm 0.59}$	86.08 ± 0.98	7.75	6.14
Connect4	$\mathbf{85.86 \pm 0.35}$ (352s)	84.78 ± 0.36 (598s)	$\mathbf{82.03 \pm 0.39}$	80.06 ± 0.30	3.83	4.72
MNIST	$\mathbf{98.78 \pm 0.06}$ (306s)	96.93 ± 0.21 (288s)	$\mathbf{96.16 \pm 0.14}$	94.37 ± 0.22	2.62	2.56
Letter	$\mathbf{95.04 \pm 0.91}$ (381s)	89.75 ± 0.66 (282s)	$\mathbf{89.19 \pm 0.29}$	84.52 ± 0.48	5.85	5.23

sufficient parameter freedom for `DTSemNet` to achieve high training accuracy (95%), much better than DGT. The last case is Connect4: here, the loss landscape is equally not flat for `DTSemNet` and DGT, and the training accuracy is only average, around 85%. This time, `DTSemNet` generalizability is superior to DGT, accounting for half of the 2% improvement in test accuracy.

Comparison with Adaptive Neural Tree. We compare the performance of `DTSemNet`, a hard oblique DT, with Adaptive Neural Tree (ANT) (Tanno et al., 2019). While ANT has a DT strucutre, it uses NNs at decision nodes, whereas `DTSemNet` uses linear expressions. This is a major difference, making ANT more suitable at analyzing complex shapes (e.g. MNIST benchmark). Further, ANT usually produces soft (probabilistic) DTs, although it could be configured to produce hard DT (but still with NN at decision nodes), namely ANT-Hard which we compare with. While NNs increase node capacity, they negate the advantage of simple linear expressions, which are particularly appreciated for their simplicity, interpretability and amenability to verification. In particular, for MNIST, the average number of decision nodes in the learned tree is 2, which is basically two convolutional layers (at internal nodes) followed by a 2-layer ReLU-activated classification head (at leaf nodes), explaining the similar accuracy for ANT-Soft and ANT-Hard. As shown in Table 7, `DTSemNet` compares favorably with ANT-Hard, except for MNIST for the reason described

Table 7: Performance comparison with ANT from Tanno et al. (2019). Test accuracy (training time) is reported for `DTSemNet`, ANT-Hard, and ANT-Soft averaged over 10 seeds. ANT-Soft is a soft DT, while ANT-Hard is obtained by hardening ANT-Soft.

Dataset	`DTSemNet`	ANT-Hard	ANT-Soft
Protein	68.60 ± 0.22 (43s)	$\mathbf{68.81 \pm 0.07}$ (12s)	68.81 ± 0.07
SatImages	$\mathbf{87.55 \pm 0.59}$ (41s)	83.73 ± 13.53 (19s)	88.15 ± 0.58
Segment	$\mathbf{96.10 \pm 0.53}$ (3s)	91.83 ± 12.74 (20s)	96.32 ± 1.12
Pendigits	$\mathbf{97.02 \pm 0.32}$ (114s)	95.58 ± 3.05 (101s)	96.36 ± 0.69
Connect4	$\mathbf{82.03 \pm 0.39}$ (352s)	80.94 ± 1.92 (613s)	83.01 ± 0.33
MNIST	96.16 ± 0.14 (306s)	$\mathbf{98.14 \pm 0.08}$ (538s)	98.14 ± 0.13
SensIT	$\mathbf{84.29 \pm 0.11}$ (1200s)	76.36 ± 10.32 (1233s)	84.20 ± 0.27
Letter	$\mathbf{89.19 \pm 0.29}$ (381s)	87.92 ± 3.10 (588s)	91.09 ± 0.95
Avg % dist. to best $\downarrow$	0.3%	3.0%	

above. Still, `DTSemNet` is overall more accurate than ANT-Hard (2.7% on average), despite the limited capacity. We provide numbers for ANT-Soft for reference: the performance of `DTSemNet` is close to ANT-Soft, despite its lower capacity, hard DT and simpler structure.

4.3 RL Tasks:

We evaluate our approach on a range of RL environments. For discrete-action tasks with limited features (≤ 32), we consider CartPole (4 features, 2 actions), Acrobot (6 features, 3 actions), LunarLander (8 features, 4 actions), and FindandDestroyZerglings (32 features, 30 actions) (Vinyals et al., 2017). For continuous-action tasks, we include continuous LunarLander (8 features, 2 action dimensions) and BipedalWalker (24 features, 4 actions). For all tasks, we fix the DT height across architectures and compare against Deep RL baselines with NNs matched in the number of learned parameters. We train 5 policies per environment using different seeds, applying PPO (Schulman et al., 2017) for discrete actions and SAC (Haarnoja et al., 2018) for continuous actions, and evaluate each on 100 test seeds to compute mean rewards, reported in Table 8. For `DTSemNet` Top-k, we adopt an annealing

Table 8: Average rewards (higher is better) on **RL tasks** $\pm$ std over 5 policies generated with different learning seeds. The policies are evaluated over 100 episodes. We report the number N_f of features and N_a of actions for each environment. The first four environments have discrete actions, and the bottom two continuous actions, for which we test both the original (*scalar*) version of DGT and DGT-linear (*linear*). For the 5 first environments, the *Height* is fixed for all architectures. For Bipedal Walker, DGT(-linear) performed much better with deeper DTs, while ICCT performed much better with shallower DTs (`DTSemNet` was not much impacted by the height). For Bipedal Walker, the heights yielding the best rewards for DGT(-linear) and ICCT are shown in parentheses.

Environments	N_f, N_a	*Height*	`DTSemNet`	Deep RL	DGT	ICCT	VIPER
CartPole	4, 2	4	$\mathbf{500 \pm 0}$	$\mathbf{500 \pm 0}$	$\mathbf{500 \pm 0}$	496 ± 0.3	499.95 ± 0.05
Acrobot	6, 3	4	$\mathbf{-82.5 \pm 1.05}$	-84 ± 0.84	-83.1 ± 1.88	-88.6 ± 1.77	-83.92 ± 1.59
LunarLander	8, 4	5	$\mathbf{252.5 \pm 3.9}$	245 ± 14.5	183.6 ± 14.6	-85 ± 16.3	86.73 ± 7.93
Zerglings	32, 30	6	$\mathbf{15.54 \pm 2.07}$	10.47 ± 0.23	8.21 ± 1.03	9.40 ± 1.10	10.61 ± 0.46
Cont. LunarLander	8, 2 dim.	4	STE: 277.24 ± 2.09 Top-k: $\mathbf{282.49 \pm 3.46}$	276.12 ± 1.45	*scalar*: 131.92 ± 51.49 *linear*: 267.9 ± 9.37	255.57 ± 4.19	NA
Bipedal Walker	24, 4 dim.	7	STE: 314.98 ± 3.35 Top-k: $\mathbf{325.35 \pm 0.79}$	315.3 ± 6.91	*scalar*: 78.33 ± 57.19 (8) *linear*: 244.5 ± 61.84 (8)	301.34 ± 3.09 (6)	NA

schedule for k from $4 \rightarrow 1$, where each SAC training step begins with Top-4 for a specified number of gradient updates, followed by Top-1 with augmented samples, and concludes with Top-1 on actual samples. At the end of each training step, this yields a `DTSemNet` Top-1 model corresponding to a hard DT, which is then used for interaction with the environment.

`DTSemNet` demonstrates competitive performance with Deep RL on environments with limited state dimensions while consistently generating the most efficient DT policies. The performance advantage increases with environment complexity: tied for first on CartPole, leading by several percent on Acrobot, achieving substantial leads on discrete LunarLander ($\geq 28\%$) and Zerglings ($\geq 39\%$) compared to other DT architectures. While VIPER achieves comparable performance on LunarLander, it requires significantly larger trees (> 1300 leaves vs 32 for our method).

Continuous action spaces prove easier to handle than discrete cases, with all regressor-based architectures achieving higher rewards and showing more consistent performance across methods. This confirms that architecture choice is less critical when policies can use linear actuators rather than fixed discrete actions. The approximations used by DGT and ICCT, particularly the multiple uses of STEs, lead to reduced rewards. While `DTSemNet` with a single STE shows some improvement, STE approximations still degrade performance. By avoiding these approximations, `DTSemNet` with Top-k maintains superior performance across environments, outperforming NN baselines by about 2% on Continuous Lunar Lander and 3% on Bipedal Walker, thereby demonstrating the advantage of approximation-free training of DTs.

5. Conclusion

We proposed `DTSemNet`, a differentiable architecture that is semantically equivalent to oblique DTs and can be trained end-to-end with standard gradient descent. For classification, this enables an exact, approximation-free procedure. For regression, we introduced an annealed Top-k approach that overcomes the limitations of STE approximation, which suffers from biased gradients due to objective mismatch and instability from inconsistent parameter updates. By retaining forward and backward semantics during training, the annealed Top-k approach ensures approximation-free learning. Together, these contributions make `DTSemNet` the first architecture to achieve approximation-free training of oblique DTs in both classification and regression. Our experiments show that `DTSemNet` consistently produces more accurate trees than competing differentiable approaches. In classification, `DTSemNet` reduces error by more than 10% on harder benchmarks, establishing a clear margin over state-of-the-art methods. In regression, it achieves an improvement of over 3% (on average) compared to baselines such as TAO-Linear, with substantially larger gains ($> 10\%$) on challenging datasets. At the same time, compared to non-gradient-based methodologies (such as greedy, non-greedy, and global search approaches) for learning DTs, gradient-based methods are significantly faster. Finally, we demonstrated the application of `DTSemNet` as a programmatic policy in RL, achieving performance comparable to or better than NN policies across both discrete and continuous action spaces. These results show that approximation-free differentiable trees combine the interpretability of symbolic models with the scalability of gradient-based training, opening promising directions for deploying interpretable yet high-performing models in safety-critical domains.

Limitations: `DTSemNet` is a DT, making it unsuitable for high-dimensional inputs like images, where DTs struggle with complex shapes and require many leaves, negating their benefits.

Future Work: For `DTSemNet`, the choice of the height of the DT is treated as a hyperparameter, similar to the choice of the number of layers in an NN, in contrast to methods (e.g. CART (Breiman et al., 1984)) that grow trees height. For future work, we will consider developing differentiable methods for adaptive growth and pruning.

Acknowledgments

This research was conducted as part of the DesCartes program and was supported by the National Research Foundation, Prime Minister's Office, Singapore, under the Campus for Research Excellence and Technological Enterprise (CREATE) program. This research/project is also supported by the National Research Foundation, Singapore and DSO National Laboratories under the AI Singapore Programme (AISG Award No: AISG2-RP-2020-017). The second author is partly supported by ANR-23-PEIA-0006 SAIF. The computational work for this research was partially performed using resources provided by the NSCC, Singapore.

References

Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1, 2004.

Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. *Advances in neural information processing systems*, 31, 2018.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Kristin P Bennett and Jennifer A Blue. Optimal decision trees. *Rensselaer Polytechnic Institute Math Report*, 214(24):128, 1996.

Dimitris Bertsimas and Jack Dunn. Optimal classification trees. *Machine Learning*, 106: 1039–1082, 2017.

Tom Bewley and Jonathan Lawry. Tripletree: A versatile interpretable representation of black box agents and their environments. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35(13), pages 11415–11422, 2021.

Gérard Biau, Erwan Scornet, and Johannes Welbl. Neural random forests. *Sankhya A*, 81 (2):347–386, 2019.

Leo Breiman, Jerome Friedman, Richard Olshen, and Charles Stone. Cart. *Classification and regression trees*, 1984.

Samuel Bulò and Peter Kontschieder. Neural decision forests for semantic image labelling. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 81–88, 2014. doi: 10.1109/CVPR.2014.18.

Miguel A Carreira-Perpinán and Pooya Tavallali. Alternating optimization of decision trees, with application to learning sparse oblique trees. *Advances in neural information processing systems*, 31, 2018.

Jonathan H Chen and Steven M Asch. Machine learning and prediction in medicine—beyond the peak of inflated expectations. *The New England journal of medicine*, 376(26):2507, 2017.

Youri Coppens, Kyriakos Efthymiadis, Tom Lenaerts, Ann Nowé, Tim Miller, Rosina Weber, and Daniele Magazzeni. Distilling deep reinforcement learning policies in soft decision trees. In *Proceedings of the IJCAI 2019 workshop on explainable artificial intelligence*, pages 1–6, 2019.

Vinícius G. Costa, Sancho Salcedo-Sanz, and Carlos E. Pedreira. Efficient evolution of decision trees via fully matrix-based fitness evaluation. *Applied Soft Computing*, 150: 111045, 2024. ISSN 1568-4946. doi: https://doi.org/10.1016/j.asoc.2023.111045.

Zihan Ding, Pablo Hernandez-Leal, Gavin Weiguang Ding, Changjian Li, and Ruitong Huang. Cdt: Cascading decision trees for explainable reinforcement learning, 2021.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity, 2021.

Nicholas Frosst and Geoffrey Hinton. Distilling a neural network into a soft decision tree. *arXiv preprint arXiv:1711.09784*, 2017.

Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? *Advances in neural information processing systems*, 35:507–520, 2022.

Ujjwal Das Gupta, Erik Talvitie, and Michael Bowling. Policy tree: Adaptive representation for policy gradient. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, page 2547–2553. AAAI Press, 2015. ISBN 0262511290.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

Hussein Hazimeh, Natalia Ponomareva, Petros Mol, Zhenyu Tan, and Rahul Mazumder. The tree ensemble layer: Differentiability meets conditional computation. In *International Conference on Machine Learning*, pages 4138–4148. PMLR, 2020.

Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274): 1–18, 2022. URL `http://jmlr.org/papers/v23/21-1342.html`.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. *Advances in neural information processing systems*, 29, 2016.

Aman Jhunjhunwala, Jaeyoung Lee, Sean Sedwards, Vahdat Abdelzad, and Krzysztof Czarnecki. Improved policy extraction via online q-value distillation. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2020. doi: 10.1109/IJCNN48605.2020.9207648.

Ajaykrishna Karthikeyan, Naman Jain, Nagarajan Natarajan, and Prateek Jain. Learning accurate decision trees with bandit feedback via quantized gradient descent. *Transactions on Machine Learning Research*, 2022.

Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, and Samuel Rota Bulò. Deep neural decision forests. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1467–1475, 2015. doi: 10.1109/ICCV.2015.172.

Hyafil Laurent and Ronald L Rivest. Constructing optimal binary decision trees is np-complete. *Information processing letters*, 5(1):15–17, 1976.

Guang-He Lee and Tommi S Jaakkola. Oblique decision trees from derivatives of relu networks. *arXiv preprint arXiv:1909.13488*, 2019.

Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. *Advances in neural information processing systems*, 31, 2018.

Guiliang Liu, Oliver Schulte, Wang Zhu, and Qingcan Li. Toward interpretable deep reinforcement learning with linear model u-trees. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2018, Dublin, Ireland, September 10–14, 2018, Proceedings, Part II 18*, pages 414–429. Springer, 2019.

Wei-Yin Loh. Fifty years of classification and regression trees. *International Statistical Review*, 82(3):329–348, 2014.

Sascha Marton, Stefan Lüdtke, Christian Bartelt, and Heiner Stuckenschmidt. Gradtree: Learning axis-aligned decision trees with gradient descent. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pages 14323–14331, 2024.

Sascha Marton, Tim Grams, Florian Vogt, Stefan Lüdtke, Christian Bartelt, and Heiner Stuckenschmidt. Mitigating information loss in tree-based reinforcement learning via direct optimization. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=qpXctF2aLZ`.

Mohammed Muqeeth, Haokun Liu, and Colin Raffel. Soft merging of experts with adaptive routing. *Transactions on Machine Learning Research*, 2024.

Sreerama K Murthy, Simon Kasif, Steven Salzberg, and Richard Beigel. Oc1: A randomized algorithm for building oblique decision trees. In *Proceedings of AAAI*, volume 93, pages 322–327. Citeseer, 1993.

Rohan R. Paleja, Yaru Niu, Andrew Silva, Chace Ritchie, Sugju Choi, and Matthew C. Gombolay. Learning interpretable, high-performing policies for continuous control problems. In *Robotics: Science and Systems*, 2022.

Subrat Prasad Panda, Blaise Genest, Arvind Easwaran, and Ponnuthurai Nagaratnam Suganthan. *Vanilla Gradient Descent for Oblique Decision Trees*. IOS Press, October 2024a. ISBN 9781643685489. doi: 10.3233/faia240607. URL http://dx.doi.org/10.3233/FAIA240607.

Subrat Prasad Panda, Blaise Genest, Arvind Easwaran, and Ponnuthurai Nagaratnam Suganthan. Vanilla gradient descent for oblique decision trees. *arXiv preprint arXiv:2408.09135*, 2024b. Full version of this paper.

Wenjie Qiu and He Zhu. Programmatic reinforcement learning without oracles. In *International Conference on Learning Representations*, 2021.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

Aaron M Roth, Nicholay Topin, Pooyan Jamshidi, and Manuela Veloso. Conservative q-improvement: Reinforcement learning for an interpretable decision-tree policy. *arXiv preprint arXiv:1907.01180*, 2019.

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

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Andrew Silva and Matthew Gombolay. Encoding human domain knowledge to warm start reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35 (6):5042–5050, 2021.

Andrew Silva, Matthew Gombolay, Taylor Killian, Ivan Jimenez, and Sung-Hyun Son. Optimization methods for interpretable differentiable decision trees applied to reinforcement learning. In *International conference on artificial intelligence and statistics*, pages 1855–1865. PMLR, 2020.

Ryutaro Tanno, Kai Arulkumaran, Daniel Alexander, Antonio Criminisi, and Aditya Nori. Adaptive neural trees. In *International Conference on Machine Learning*, pages 6166–6175. PMLR, 2019.

Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. In *International Conference on Machine Learning*, pages 5045–5054. PMLR, 2018.

Abhinav Verma, Hoang Le, Yisong Yue, and Swarat Chaudhuri. Imitation-projected programmatic reinforcement learning. *Advances in Neural Information Processing Systems*, 32, 2019.

Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, et al. Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017.

Alvin Wan, Lisa Dunlap, Daniel Ho, Jihan Yin, Scott Lee, Henry Jin, Suzanne Petryk, Sarah Adel Bargal, and Joseph E Gonzalez. Nbdt: neural-backed decision trees. *arXiv preprint arXiv:2004.00221*, 2020.

Yongxin Yang, Irene Garcia Morillo, and Timothy M Hospedales. Deep neural decision trees. *arXiv preprint arXiv:1806.06988*, 2018.

Penghang Yin, Jiancheng Lyu, Shuai Zhang, Stanley Osher, Yingyong Qi, and Jack Xin. Understanding straight-through estimator in training activation quantized neural nets. In *International Conference on Learning Representations*, 2019.

Valentina Zantedeschi, Matt Kusner, and Vlad Niculae. Learning binary decision trees by argmin differentiation. In *International Conference on Machine Learning*, pages 12298–12309. PMLR, 2021.

Arman Zharmagambetov and Miguel Carreira-Perpinan. Smaller, more accurate regression forests using tree alternating optimization. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pages 11398–11408. PMLR, 2020.