Learning Binary Trees by Argmin Differentiation

Valentina Zantedeschi [1 2] Matt J. Kusner [2] Vlad Niculae [3]

Abstract

We address the problem of learning binary decision trees that partition data for some downstream task. We propose to learn discrete parameters (i.e., for tree traversals and node pruning) and continuous parameters (i.e., for tree split functions and prediction functions) *simultaneously* using argmin differentiation. We do so by sparsely relaxing a mixed-integer program for the discrete parameters, to allow gradients to pass through the program to continuous parameters. We derive customized algorithms to efficiently compute the forward and backward passes. This means that our tree learning procedure can be used as an (implicit) layer in arbitrary deep networks, and can be optimized with arbitrary loss functions. We demonstrate that our approach produces binary trees that are competitive with existing single tree and ensemble approaches, in both supervised and unsupervised settings. Further, apart from greedy approaches (which do not have competitive accuracies), our method is faster to train than all other tree-learning baselines we compare with. The code for reproducing the results is available at `https://github.com/vzantedeschi/LatentTrees`.

1. Introduction

Learning discrete structures from unstructured data is extremely useful for a wide variety of real-world problems (Gilmer et al., 2017; Kool et al., 2018; Yang et al., 2018). One of the most computationally-efficient, easily-visualizable discrete structures that are able to represent complex functions are *binary trees*. For this reason, there has been a massive research effort on how to learn such binary trees since the early days of machine learning (Payne & Meisel, 1977; Breiman et al., 1984; Bennett, 1992; Bennett & Blue, 1996). Learning binary trees has historically been done in one of three ways. The first is via *greedy optimization*, which includes popular decision-tree methods such as classification and regression trees (CART, Breiman et al., 1984) and ID3 trees (Quinlan, 1986), among many others. These methods optimize a splitting criterion for each tree node, based on the data routed to it. The second set of approaches are based on *probabilistic relaxations* (İrsoy et al., 2012; Yang et al., 2018; Hazimeh et al., 2020), optimizing all splitting parameters at once via gradient-based methods, by relaxing hard branching decisions into branching probabilities. The third approach optimizes trees using *mixed-integer programming* (MIP, Bennett, 1992; Bennett & Blue, 1996). This jointly optimizes all continuous and discrete parameters to find globally-optimal trees.[1]

Each of these approaches has their shortcomings. First, greedy optimization is generally suboptimal: tree splitting criteria are even intentionally crafted to be different than the global tree loss, as the global loss may not encourage tree growth (Breiman et al., 1984). Second, probabilistic relaxations: (a) are rarely sparse, so inputs contribute to all branches even if they are projected to hard splits after train-

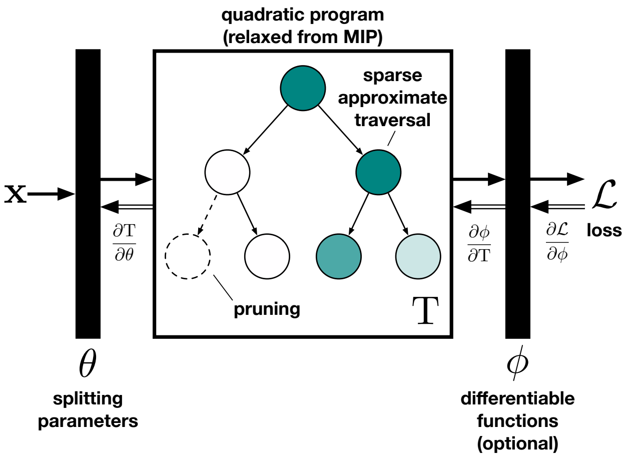


Figure 1: A schematic of the approach.

[1]Inria, Lille - Nord Europe research centre [2]University College London, Centre for Artificial Intelligence [3]Informatics Institute, University of Amsterdam. Correspondence to: V Zantedeschi <vzantedeschi@gmail.com>, M Kusner <m.kusner@ucl.ac.uk>, V Niculae <v.niculae@uva.nl>.

Proceedings of the 38th International Conference on Machine Learning, PMLR 139, 2021. Copyright 2021 by the author(s).

[1]Here we focus on learning single trees instead of tree ensembles; our work easily extends to ensembles.

ing; (b) they do not have principled ways to prune trees, as the distribution over pruned trees is often intractable. Third, MIP approaches, while optimal, are only computationally tractable on training datasets with thousands of inputs (Bertsimas & Dunn, 2017), and do not have well-understood out-of-sample generalization guarantees.

In this paper we present an approach to binary tree learning based on sparse relaxation and argmin differentiation (as depicted in Figure 1). Our main insight is that by quadratically relaxing an MIP that learns the discrete parameters of the tree (input traversal and node pruning), we can differentiate through it to simultaneously learn the continuous parameters of splitting decisions. This allows us to leverage the superior generalization capabilities of stochastic gradient optimization to learn decision splits, and gives a principled approach to learning tree pruning. Further, we derive customized algorithms to compute the forward and backward passes through this program that are much more efficient than generic approaches (Amos & Kolter, 2017; Agrawal et al., 2019a). The resulting learning method can learn trees with any differentiable splitting functions to minimize any differentiable loss, both tailored to the data at hand.

Notation. We denote scalars, vectors, and sets, as x, $\mathbf{x}$, and $\mathcal{X}$, respectively. A binary tree is a set $\mathcal{T}$ containing nodes $t \in \mathcal{T}$ with the additional structure that each node has at most one left child and at most one right child, and each node except the unique root has exactly one parent. We denote the l_2 norm of a vector by $\|\mathbf{x}\| := \left(\sum_i x_i^2\right)^{\frac{1}{2}}$. The unique projection of point $\mathbf{x} \in \mathbb{R}^d$ onto a convex set $C \subset \mathbb{R}^d$ is $\text{Proj}_C(\mathbf{x}) := \arg\min_{\mathbf{y} \in C} \|\mathbf{y} - \mathbf{x}\|$. In particular, projection onto an interval is given by $\text{Proj}_{[a,b]}(x) = \max(a, \min(b, x))$.

2. Related Work

The paradigm of *binary tree learning* has the goal of finding a tree that iteratively splits data into meaningful, informative subgroups, guided by some criterion. Binary tree learning appears in a wide variety of problem settings across machine learning. We briefly review work in two learning settings where latent tree learning plays a key role: 1. *Classification/Regression*; and 2. *Hierarchical clustering*. Due to the generality of our setup (tree learning with arbitrary split functions, pruning, and downstream objective), our approach can be used to learn trees in any of these settings. Finally, we detail how parts of our algorithm are inspired by recent work in isotonic regression.

Classification/Regression. Decision trees for classification and regression have a storied history, with early popular methods that include classification and regression trees (CART, Breiman et al., 1984), ID3 (Quinlan, 1986), and C4.5 (Quinlan, 1993). While powerful, these methods are greedy: they sequentially identify 'best' splits as those which optimize a split-specific score (often different from the global objective). As such, learned trees are likely sub-optimal for the classification/regression task at hand. To address this, Carreira-Perpinán & Tavallali (2018) proposes an alternating algorithm for refining the structure and decisions of a tree so that it is smaller and with reduced error, however still sub-optimal. Another approach is to probabilistically relax the discrete splitting decisions of the tree (İrsoy et al., 2012; Yang et al., 2018; Tanno et al., 2019). This allows the (relaxed) tree to be optimized *w.r.t.* the overall objective using gradient-based techniques, with known generalization benefits (Hardt et al., 2016; Hoffer et al., 2017). Variations on this approach aim at learning tree ensembles termed 'decision forests' (Kontschieder et al., 2015; Lay et al., 2018; Popov et al., 2020; Hazimeh et al., 2020). The downside of the probabilistic relaxation approach is that there is no principled way to prune these trees as inputs pass through all nodes of the tree with some probability. A recent line of work has explored mixed-integer program (MIP) formulations for learning decision trees. Motivated by the billion factor speed-up in MIP in the last 25 years, Rudin & Ertekin (2018) proposed a mathematical programming approach for learning provably optimal decision lists (one-sided decision trees; Letham et al., 2015). This resulted in a line of recent follow-up works extending the problem to binary decision trees (Hu et al., 2019; Lin et al., 2020) by adapting the efficient discrete optimization algorithm (CORELS, Angelino et al., 2017). Related to this line of research, Bertsimas & Dunn (2017) and its follow-up works (Günlük et al., 2021; Aghaei et al., 2019; Verwer & Zhang, 2019; Aghaei et al., 2020) phrased the objective of CART as an MIP that could be solved exactly. Even given this consistent speed-up all these methods are only practical on datasets with at most thousands of inputs (Bertsimas & Dunn, 2017) and with non-continuous features. Recently, Zhu et al. (2020) addressed these tractability concerns principally with a data selection mechanism that preserves information. Still, the out-of-sample generalizability of MIP approaches is not well-studied, unlike stochastic gradient descent learning.

Hierarchical clustering. Compared to standard flat clustering, hierarchical clustering provides a structured organization of unlabeled data in the form of a tree. To learn such a clustering the vast majority of methods are greedy and work in one of two ways: 1. *Agglomerative*: a 'bottom-up' approach that starts each input in its own cluster and iteratively merges clusters; and 2. *Divisive*: a 'top-down' approach that starts with one cluster and recusively splits clusters (Zhang et al., 1997; Widyantoro et al., 2002; Krishnamurthy et al., 2012; Dasgupta, 2016; Kobren et al., 2017; Moseley & Wang, 2017). These methods suffer from similar issues as greedy approaches for classification/regression do: they may be sub-optimal for optimizing the overall tree. Further they are often computationally-expensive due to their sequential nature. Inspired by approaches for classification/regression, recent work has designed probabilistic relaxations for learning hierarchical clusterings via gradient-based methods (Monath et al., 2019).

Our work takes inspiration from both the MIP-based and gradient-based approaches. Specifically, we frame learning the discrete tree parameters as an MIP, which we sparsely relax to allow continuous parameters to be optimized by argmin differentiation methods.

Argmin differentiation. Solving an optimization problem as a differentiable module within a parent problem tackled with gradient-based optimization methods is known as argmin differentiation, an instance of bi-level optimization (Colson et al., 2007; Gould et al., 2016). This situation arises in as diverse scenarios as hyperparameter optimization (Pedregosa, 2016), meta-learning (Rajeswaran et al., 2019), or structured prediction (Stoyanov et al., 2011; Domke, 2013; Niculae et al., 2018a). General algorithms for quadratic (Amos & Kolter, 2017) and disciplined convex programming (Section 7, Amos, 2019; Agrawal et al., 2019a;b) have been given, as well as expressions for more specific cases like isotonic regression (Djolonga & Krause, 2017). Here, by taking advantage of the structure of the decision tree induction problem, we obtain a direct, efficient algorithm.

Latent parse trees. Our work resembles but should not be confused with the latent parse tree literature in natural language processing (Yogatama et al., 2017; Liu & Lapata, 2018; Choi et al., 2018; Williams et al., 2018; Niculae et al., 2018b; Corro & Titov, 2019b;a; Maillard et al., 2019; Kim et al., 2019a;b). This line of work has a different goal than ours: to induce a tree for each individual data point (e.g., sentence). In contrast, our work aims to learn a single tree, for all instances to pass through.

Isotonic regression. Also called monotonic regression, isotonic regression (Barlow et al., 1972) constrains the regression function to be non-decreasing/non-increasing. This is useful if one has prior knowledge of such monotonicity (*e.g.*, the mean temperature of the climate is non-decreasing). A classic algorithm is pooling-adjacent-violators (PAV), which optimizes the pooling of adjacent points that violate the monotonicity constraint (Barlow et al., 1972). This initial algorithm has been generalized and incorporated into convex programming frameworks (see Mair et al. (2009) for an excellent summary of the history of isotonic regression and its extensions). Our work builds off of the generalized PAV (GPAV) algorithm of Yu & Xing (2016).

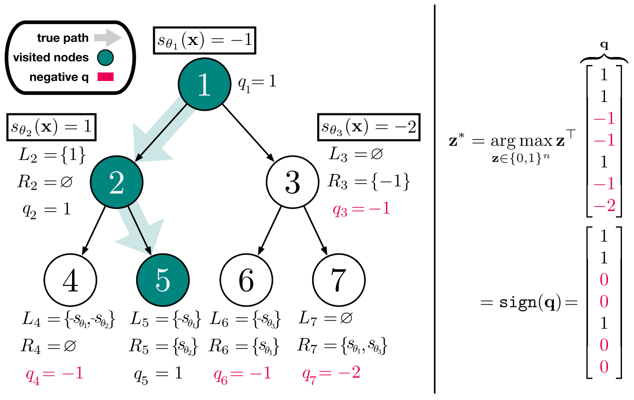

true path
visited nodes ●
negative q 🟥
$s_{\theta_1}(\mathbf{x}) = -1$
$q_1 = 1$
$s_{\theta_2}(\mathbf{x}) = 1$
$L_2 = \{1\}$
$R_2 = \varnothing$
$q_2 = 1$
$s_{\theta_3}(\mathbf{x}) = -2$
$L_3 = \varnothing$
$R_3 = \{-1\}$
$q_3 = -1$
1
2
3
4
5
6
7
$L_4 = \{-s_{\theta_1}, -s_{\theta_2}\}$
$R_4 = \varnothing$
$q_4 = -1$
$L_5 = \{-s_{\theta_1}\}$
$R_5 = \{s_{\theta_2}\}$
$q_5 = 1$
$L_6 = \{-s_{\theta_3}\}$
$R_6 = \{s_{\theta_1}\}$
$q_6 = -1$
$L_7 = \varnothing$
$R_7 = \{s_{\theta_1}, s_{\theta_3}\}$
$q_7 = -2$
$\mathbf{z}^* = \arg\max_{\mathbf{z}\in\{0,1\}^n} \mathbf{z}^\top \overbrace{\begin{bmatrix}1\\1\\-1\\-1\\1\\-1\\-2\end{bmatrix}}^{\mathbf{q}} = \mathtt{sign}(\mathbf{q}) = \begin{bmatrix}1\\1\\0\\0\\1\\0\\0\end{bmatrix}$

Figure 2: A depiction of equation (1) for optimizing tree traversals given the tree parameters $\boldsymbol{\theta}$.

3. Method

Given inputs $\{\mathbf{x}_i \in \mathcal{X}\}_{i=1}^n$, our goal is to learn a latent binary decision tree $\mathcal{T}$ with maximum depth D. This tree sends each input $\mathbf{x}$ through branching nodes to a specific leaf node in the tree. Specifically, all branching nodes $\mathcal{T}_B \subset \mathcal{T}$ split an input $\mathbf{x}$ by forcing it to go to its left child if $s_\theta(\mathbf{x}) < 0$, and right otherwise. There are three key parts of the tree that need to be identified: 1. The *continuous* parameters $\theta_t \in \mathbb{R}^d$ that describe how $s_{\theta_t}(.)$ splits inputs at every node t; 2. The *discrete* paths $\mathbf{z}$ made by each input $\mathbf{x}$ through the tree; 3. The *discrete* choice a_t of whether a node t should be active or pruned, i.e. inputs should reach/traverse it or not. We next describe how to represent each of these.

3.1. Tree Traversal & Pruning Programs

Let the splitting functions of a complete tree $\{s_{\theta_t} : \mathcal{X} \to \mathbb{R}\}_{t=1}^{|\mathcal{T}_B|}$ be fixed. The path through which a point $\mathbf{x}$ traverses the tree can be encoded in a vector $\mathbf{z} \in \{0,1\}^{|\mathcal{T}|}$, where each component z_t indicates whether $\mathbf{x}$ reaches node $t \in \mathcal{T}$. The following integer linear program (ILP) is maximized by a vector $\mathbf{z}$ that describes *tree traversal*:

$$\max_{\mathbf{z}} \ \mathbf{z}^\top \mathbf{q} \tag{1}$$

$$\begin{aligned} \text{s.t. } \forall t \in \mathcal{T} \setminus \{\text{root}\}, \quad & q_t = \min\{R_t \cup L_t\} \\ & R_t = \{s_{\theta_{t'}}(\mathbf{x}) \quad \mid \ \ \forall t' \in A_R(t)\} \\ & L_t = \{-s_{\theta_{t'}}(\mathbf{x}) \mid \ \ \forall t' \in A_L(t)\} \\ & z_t \in \{0,1\}, \end{aligned}$$

where we fix $q_1 = z_1 = 1$ (i.e., the root). Here $A_L(t)$ is the set of ancestors of node t whose left child must be followed to get to t, and similarly for $A_R(t)$. The quantities q_t (where $\mathbf{q} \in \mathbb{R}^{|\mathcal{T}|}$ is the tree-vectorized version of q_t) describe the 'reward' of sending $\mathbf{x}$ to node t. This is based on how well the splitting functions leading to t are satisfied (q_t is positive if all splitting functions are satisfied and negative otherwise). This is visualized in Figure 2 and formalized below.

Lemma 1. *For an input* $\mathbf{x}$ *and binary tree* $\mathcal{T}$ *with splits* $\{s_{\theta_t}, \forall t \in \mathcal{T}_B\}$*, the ILP in eq.* (1) *describes a valid tree traversal* $\mathbf{z}$ *for* $\mathbf{x}$ *(i.e.,* $z_t = 1$ *for any* $t \in \mathcal{T}$ *if* $\mathbf{x}$ *reaches* t*).*

Proof. Assume without loss of generality that $s_{\theta_t}(\mathbf{x}) \neq 0$ for all $t \in \mathcal{T}$ (if this does not hold for some node $\tilde{t}$ simply add a small constant ϵ to $s_{\theta_{\tilde{t}}}(\mathbf{x})$). Recall that for any non-leaf node t' in the fixed binary tree $\mathcal{T}$, a point $\mathbf{x}$ is sent to the left child if $s_{\theta_{t'}}(\mathbf{x}) < 0$ and to the right child otherwise. Following these splits from the root, every point $\mathbf{x}$ reaches a unique leaf node $f_{\mathbf{x}} \in \mathcal{F}$, where $\mathcal{F}$ is the set of all leaf nodes. Notice first that both $\min L_{f_{\mathbf{x}}} > 0$ and $\min R_{f_{\mathbf{x}}} > 0$.[2] This is because $\forall t' \in A_L(f_{\mathbf{x}})$ it is the case that $s_{\theta_{t'}}(\mathbf{x}) < 0$, and $\forall t' \in A_R(f_{\mathbf{x}})$ we have that $s_{\theta_{t'}}(\mathbf{x}) > 0$. This is due to the definition of $f_{\mathbf{x}}$ as the leaf node that $\mathbf{x}$ reaches (i.e., it traverses to $f_{\mathbf{x}}$ by following the aforementioned splitting rules). Further, for any other leaf $f \in \mathcal{F} \setminus \{f_{\mathbf{x}}\}$ we have that either $\min L_f < 0$ or $\min R_f < 0$. This is because there exists a node t' on the path to f (and so t' is either in $A_L(f)$ or $A_R(f)$) that does not agree with the splitting rules. For this node it is either the case that (i) $s_{\theta_{t'}}(\mathbf{x}) > 0$ and $t' \in A_L(f)$, or that (ii) $s_{\theta_{t'}}(\mathbf{x}) < 0$ and $t' \in A_R(f)$. In the first case $\min L_f < 0$, in the second $\min R_f < 0$. Therefore we have that $q_f < 0$ for all $f \in \mathcal{F} \setminus \{f_{\mathbf{x}}\}$, and $q_{f_{\mathbf{x}}} > 0$. In order to maximize the objective $\mathbf{z}^\top \mathbf{q}$ we will have that $z_{f_{\mathbf{x}}} = 1$ and $z_f = 0$. Finally, let $\mathcal{N}_{f_{\mathbf{x}}}$ be the set of non-leaf nodes visited by $\mathbf{x}$ on its path to $f_{\mathbf{x}}$. Now notice that the above argument applied to leaf nodes can be applied to nodes at any level of the tree: $q_t > 0$ for $t \in \mathcal{N}_{f_{\mathbf{x}}}$ while $q_{t'} < 0$ for $t' \in \mathcal{T} \setminus \mathcal{N}_{f_{\mathbf{x}}} \cup f_{\mathbf{x}}$. Therefore $\mathbf{z}_{\mathcal{N}_{f_{\mathbf{x}}} \cup f_{\mathbf{x}}} = 1$ and $\mathbf{z}_{\mathcal{T} \setminus \mathcal{N}_{f_{\mathbf{x}}} \cup f_{\mathbf{x}}} = 0$, completing the proof. □

This solution is unique so long as $s_{\theta_t}(\mathbf{x}_i) \neq 0$ for all $t \in \mathcal{T}, i \in \{1, \ldots, n\}$ (*i.e.*, $s_{\theta_t}(\mathbf{x}_i) = 0$ means equal preference to split $\mathbf{x}_i$ left or right). Further the integer constraint on z_{it} can be relaxed to an interval constraint $z_{it} \in [0, 1]$ w.l.o.g. This is because if $s_{\theta_t}(\mathbf{x}_i) \neq 0$ then $z_t = 0$ if and only if $q_t < 0$ and $z_t = 1$ when $q_t > 0$ (and $q_t \neq 0$).

While the above program works for any fixed tree, we would like to be able to also learn the structure of the tree itself. We do so by learning a pruning optimization variable $a_t \in \{0, 1\}$, indicating if node $t \in \mathcal{T}$ is active (if 1) or pruned (if 0). We adapt eq. (1) into the following pruning-aware mixed integer program (MIP) considering all inputs $\{\mathbf{x}_i\}_{i=1}^n$:

$$\begin{aligned} \max_{\mathbf{z}_1, \ldots, \mathbf{z}_n, \mathbf{a}} \quad & \sum_{i=1}^n \mathbf{z}_i^\top \mathbf{q}_i - \frac{\lambda}{2} \|\mathbf{a}\|^2 \qquad (2) \\ \text{s.t. } \forall i \in [n], \quad & a_t \leq a_{p(t)}, \quad \forall t \in \mathcal{T} \setminus \{\text{root}\} \\ & z_{it} \leq a_t \\ & z_{it} \in [0, 1], a_t \in \{0, 1\} \end{aligned}$$

[2]We use the convention that $\min \varnothing = \infty$ (i.e., for paths that go down the rightmost or leftmost parts of the tree).

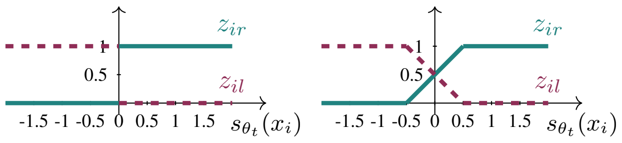


Figure 3: Routing of point $\mathbf{x}_i$ at node t, without pruning, without (left) or with (right) quadratic relaxation. Depending on the decision split $s_{\theta_t}(\cdot)$, $\mathbf{x}_i$ reaches node t's left child l (right child r respectively) if $z_{il} > 0$ ($z_{ir} > 0$). The relaxation makes $\mathbf{z}_i$ continuous and encourages points to have a margin ($|s_{\theta_t}| > 0.5$).

with $\|\cdot\|$ denoting the l_2 norm. We remove the first three constraints in eq. (1) as they are a deterministic computation independent of $\mathbf{z}_1, \ldots, \mathbf{z}_n, \mathbf{a}$. We denote by $p(t)$ the parent of node t. The new constraints $a_t \leq a_{p(t)}$ ensure that child nodes t are pruned if the parent node $p(t)$ is pruned, hence enforcing the tree hierarchy, while the other new constraint $z_{it} \leq a_t$ ensures that no point $\mathbf{x}_i$ can reach node t if node t is pruned, resulting in losing the associated reward q_{it}. Overall, the problem consists in a trade-off, controlled by hyper-parameter $\lambda \in \mathbb{R}$, between minimizing the number of active nodes through the pruning regularization term (since $a_t \in \{0, 1\}$, $\|\mathbf{a}\|^2 = \sum_t a_t^2 = \sum_t \mathbb{I}(a_t = 1)$) while maximizing the reward from points traversing the nodes.

3.2. Learning Tree Parameters

We now consider the problem of learning the splitting parameters θ_t. A natural approach would be to do so in the MIP itself, as in the optimal tree literature. However, this would severely restrict allowed splitting functions as even linear splitting functions can only practically run on at most thousands of training inputs (Bertsimas & Dunn, 2017). Instead, we propose to learn s_{θ_t} via gradient descent.

To do so, we must be able to compute the partial derivatives $\partial \mathbf{a} / \partial \mathbf{q}$ and $\partial \mathbf{z} / \partial \mathbf{q}$. However, the solutions of eq. (2) are piecewise-constant, leading to null gradients. To avoid this, we relax the integer constraint on $\mathbf{a}$ to the interval $[0, 1]$ and add quadratic regularization $-{}^1\!/\!_4 \sum_i (\|\mathbf{z}_i\|^2 + \|1 - \mathbf{z}_i\|^2)$. The regularization term for $\mathbf{z}$ is symmetric so that it shrinks solutions toward even left-right splits (see Figure 3 for a visualization, and the supplementary for a more detailed justification with an empirical evaluation of the approximation gap). Rearranging and negating the objective yields

$$\begin{aligned} & \mathrm{T}_\lambda(\mathbf{q}_1, \ldots, \mathbf{q}_n) = \qquad (3) \\ & \quad \operatorname*{arg\,min}_{\mathbf{z}_1, \ldots, \mathbf{z}_n, \mathbf{a}} \ {}^\lambda\!/\!_2 \|\mathbf{a}\|^2 + {}^1\!/\!_2 \sum_{i=1}^n \|\mathbf{z}_i - \mathbf{q}_i - {}^1\!/\!_2\|^2 \\ & \text{s.t. } \forall i \in [n], \quad a_t \leq a_{p(t)}, \quad \forall t \in \mathcal{T} \setminus \{\text{root}\} \\ & \qquad z_{it} \leq a_t \\ & \qquad z_{it} \in [0, 1], a_t \in [0, 1]. \end{aligned}$$

Algorithm 1 Pruning via isotonic optimization

```
initial partition $\mathcal{G} \leftarrow \{\{1\}, \{2\}, \cdots\} \subset 2^{\mathcal{T}}$
for $G \in \mathcal{G}$ do
    $a_G \leftarrow \arg\min_a \sum_{t \in G} g_t(a)$        {Eq. (7), Prop. 2}
end for
repeat
    $t_{\max} \leftarrow \arg\max_t \{a_t : a_t > a_{p(t)}\}$
    merge $G \leftarrow G \cup G'$ where $G' \ni t_{\max}$ and $G \ni p(t_{\max})$.
    update $a_G \leftarrow \arg\min_a \sum_{t \in G} g_t(a)$        {Eq. (7), Prop. 2}
until no more violations $a_t > a_{p(t)}$ exist
```

The regularization makes the objective strongly convex. It follows from convex duality that T_λ is Lipschitz continuous (Zalinescu, 2002, Corollary 3.5.11). By Rademacher's theorem (Borwein & Lewis, 2010, Theorem 9.1.2), T_λ is thus differentiable almost everywhere. Generic methods such as OptNet (Amos & Kolter, 2017) could be used to compute the solution and gradients. However, by using the tree structure of the constraints, we next derive an efficient specialized algorithm. The main insight, shown below, reframes pruned binary tree learning as isotonic optimization.

Proposition 1. *Let* $\mathcal{C} = \left\{\mathbf{a} \in \mathbb{R}^{|\mathcal{T}|} : a_t \leq a_{p(t)} \text{ for all } t \in \mathcal{T} \setminus \{root\}\right\}$. *Consider* $\mathbf{a}^\star$ *the optimal value of*

$$\operatorname*{arg\,min}_{\mathbf{a} \in \mathcal{C} \cap [0,1]^{|\mathcal{T}|}} \sum_{t \in \mathcal{T}} \left(\frac{\lambda}{2} a_t^2 + \sum_{i : a_t < q_{it} + 1/2} (a_t - q_{it} - 1/2)^2\right). \quad (4)$$

Define[3] $\mathbf{z}_i^\star$ *such that* $z_{it}^\star = \mathrm{Proj}_{[0, a_t^\star]}(q_{it} + 1/2)$. *Then,* $\mathrm{T}_\lambda(\mathbf{q}_1, \ldots, \mathbf{q}_n) = \mathbf{z}_1^\star, \ldots, \mathbf{z}_n^\star, \mathbf{a}^\star$.

Proof. The constraints and objective of eq. (3) are separable, so we may push the minimization *w.r.t.* $\mathbf{z}$ inside the objective:

$$\operatorname*{arg\,min}_{\mathbf{a} \in \mathcal{C} \cap [0,1]^{|\mathcal{T}|}} \frac{\lambda}{2} \|\mathbf{a}\|^2 + \sum_{t \in \mathcal{T}} \sum_{i=1}^{n} \min_{0 \leq z_{it} \leq a_t} 1/2 (z_{it} - q_{it} - 1/2)^2 . \quad (5)$$

Each inner minimization $\min_{0 \leq z_{it} \leq a_t} 1/2 (z_{it} - q_{it} - 1/2)^2$ is a one-dimensional projection onto box constraints, with solution $z_{it}^\star = \mathrm{Proj}_{[0, a_t]}(q_{it} + 1/2)$. We use this to eliminate $\mathbf{z}$, noting that each term $1/2 (z_{it}^\star - q_{it} - 1/2)^2$ yields

$$\begin{cases} 1/2 (q_{it} + 1/2)^2, & q_{it} < -1/2 \\ 0, & -1/2 \leq q_{it} < a_t - 1/2 \\ 1/2 (a_t - q_{it} - 1/2)^2, & q_{it} \geq a_t - 1/2. \end{cases} \quad (6)$$

The first two branches are constants *w.r.t.* a_t The problems in eq. (3) and eq. (4) differ by a constant and have the same constraints, therefore they have the same solutions. $\square$

[3]Here $\mathrm{Proj}_{\mathcal{S}}(x)$ is the projection of x onto set $\mathcal{S}$. If $\mathcal{S}$ are box constraints, projection amounts to clipping.

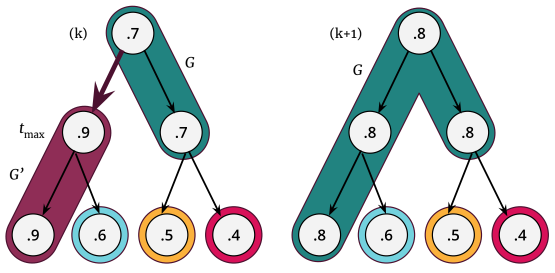


Figure 4: One step in the tree isotonic optimization process from Algorithm 1. Shaded tubes depict the partition groups $\mathcal{G}$, and the emphasized arc is the largest among the violated inequalities. The group of $t_{\max}$ merges with the group of its parent, and the new value for the new, larger group a_G is computed.

Efficiently inducing trees as isotonic optimization. From Proposition 1, notice that eq. (4) is an instance of tree-structured isotonic optimization: the objective decomposes over *nodes*, and the inequality constraints correspond to edges in a rooted tree:

$$\operatorname*{arg\,min}_{\mathbf{a} \in \mathcal{C}} \sum_{t \in \mathcal{T}} g_t(a_t), \quad \text{where} \quad (7)$$

$$g_t(a_t) = \frac{\lambda}{2} a_t^2 + \sum_{i : a_t \leq q_{it} + 1/2} 1/2 (a_t - q_{it} - 1/2)^2 + \iota_{[0,1]}(a_t).$$

where $\iota_{[0,1]}(a_t) = \infty$ if $a_t \notin [0,1]$ and 0 otherwise. This problem can be solved by a generalized *pool adjacent violators* (PAV) algorithm: Obtain a tentative solution by ignoring the constraints, then iteratively remove violating edges $a_t > a_{p(t)}$ by *pooling together* the nodes at the end points. Figure 4 provides an illustrative example of the procedure. At the optimum, the nodes are organized into a partition $\mathcal{G} \subset 2^{\mathcal{T}}$, such that if two nodes t, t' are in the same group $G \in \mathcal{G}$, then $a_t = a_{t'} := a_G$.

When the inequality constraints are the edges of a rooted tree, as is the case here, the PAV algorithm finds the optimal solution in at most $|\mathcal{T}|$ steps, where each involves updating the a_G value for a newly-pooled group by solving a one-dimensional subproblem of the form (Yu & Xing, 2016)[4]

$$a_G = \operatorname*{arg\,min}_{a \in \mathbb{R}} \sum_{t \in G} g_t(a), \quad (8)$$

resulting in Algorithm 1. It remains to show how to solve eq. (8). The next result, proved in the supplementary, gives an exact and efficient solution, with an algorithm that requires finding the nodes with highest q_{it} (*i.e.*, the nodes which $\mathbf{x}_i$ is most highly encouraged to traverse).

[4]Compared to Yu & Xing (2016), our tree inequalities are in the opposite direction. This is equivalent to a sign flip of parameter $\mathbf{a}$, *i.e.*, to selecting the *maximum* violator rather than the minimum one at each iteration.

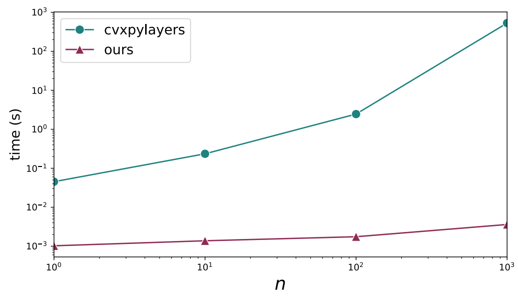


Figure 5: Comparison of Algorithm 1's running time (ours) with the running time of cvxpylayers (Agrawal et al., 2019a) for tree depth $D = 3$, varying n.

Proposition 2. *The solution to the one-dimensional problem in eq.* (8) *for any* G *is given by*

$$\operatorname*{arg\,min}_{a \in \mathbb{R}} \sum_{t \in G} g_t(a) = \mathrm{Proj}_{[0,1]}\left(a(k^\star)\right) \tag{9}$$

$$\textit{where} \quad a(k^\star) := \frac{\sum_{(i,t) \in S(k^\star)} (q_{it} + 1/2)}{\lambda |G| + k^\star},$$

$S(k) = \{j^{(1)}, \ldots, j^{(k)}\}$ *is the set of indices* $j = (i,t) \in \{1, \ldots, n\} \times G$ *into the* k *highest values of* $\mathbf{q}$, *i.e.,* $q_{j^{(1)}} \geq q_{j^{(2)}} \geq \ldots \geq q_{j^{(m)}}$, *and* $k^\star$ *is the smallest* k *satisfying* $a(k) > q_{j^{(k+1)}} + 1/2$.

Figure 5 compares our algorithm to a generic differentiable solver (details and additional comparisons in supplementary material).

Backward pass and efficient implementation details. Algorithm 1 is a sequence of differentiable operations that can be implemented *as is* in automatic differentiation frameworks. However, because of the prominent loops and indexing operations, we opt for a low-level implementation as a C++ extension. Since the $\mathbf{q}$ values are constant *w.r.t.* $\mathbf{a}$, we only need to sort them once as preprocessing, resulting in a overall time complexity of $\mathcal{O}(n|\mathcal{T}| \log(n|\mathcal{T}|))$ and space complexity of $\mathcal{O}(n|\mathcal{T}|)$. For the backward pass, rather than relying on automatic differentiation, we make two remarks about the form of $\mathbf{a}$. Firstly, its elements are organized in groups, *i.e.*, $a_t = a'_t = a_G$ for $\{t, t'\} \subset G$. Secondly, the value a_G inside each group depends only on the optimal support set $S^\star_G := S(k^\star)$ as defined for each subproblem by Proposition 2. Therefore, in the forward pass, we must store only the node-to-group mappings and the sets $S^\star_G$. Then, if G contains t,

$$\frac{\partial a^\star_t}{\partial q_{it'}} = \begin{cases} \frac{1}{\lambda |G| + k^\star}, & 0 < a^\star_t < 1, \text{ and } (i, t') \in S^\star_G, \\ 0, & \text{otherwise.} \end{cases}$$

As T_λ is differentiable almost everywhere, these expressions yield the unique Jacobian at all but a measure-zero set of points, where they yield one of the Clarke generalized Jacobians (Clarke, 1990). We then rely on automatic differentiation to propagate gradients from $\mathbf{z}$ to $\mathbf{q}$, and from $\mathbf{q}$ to the split parameters $\boldsymbol{\theta}$: since $\mathbf{q}$ is defined element-wise via min functions, the gradient propagates through the minimizing path, by Danskin's theorem (Proposition B.25, Bertsekas, 1999; Danskin, 1966).

Algorithm 2 Learning with decision tree representations.

```
initialize neural network parameters $\boldsymbol{\phi}, \boldsymbol{\theta}$
repeat
    sample batch $\{\mathbf{x}_i\}_{i \in B}$
    induce traversals:
    $\{\mathbf{z}_i\}_{i \in B}, \mathbf{a} = \mathrm{T}_\lambda(\{q_{\boldsymbol{\theta}}(\mathbf{x}_i)\})$ {Algorithm 1; differentiable}
    update parameters using $\nabla_{\{\boldsymbol{\theta},\boldsymbol{\phi}\}} \ell(f_{\boldsymbol{\phi}}(\mathbf{x}_i, \mathbf{z}_i))$    {autograd}
until convergence or stopping criterion is met
```

3.3. The Overall Objective

We are now able to describe the overall optimization procedure that simultaneously learns tree parameters: (a) input traversals $\mathbf{z}_1, \ldots, \mathbf{z}_n$; (b) tree pruning $\mathbf{a}$; and (c) split parameters $\boldsymbol{\theta}$. Instead of making predictions using heuristic scores over the training points assigned to a leaf (e.g., majority class), we learn a prediction function $f_\phi(\mathbf{z}, \mathbf{x})$ that minimizes an arbitrary loss $\ell(\cdot)$ as follows:

$$\begin{aligned} &\min_{\boldsymbol{\theta},\boldsymbol{\phi}} \sum_{i=1}^{n} \ell\big(f_{\boldsymbol{\phi}}(\mathbf{x}_i, \mathbf{z}_i)\big) \\ &\text{where } \mathbf{z}_1, \ldots, \mathbf{z}_n, \mathbf{a} := \mathrm{T}_\lambda\big(q_{\boldsymbol{\theta}}(\mathbf{x}_1), \ldots, q_{\boldsymbol{\theta}}(\mathbf{x}_n)\big) . \end{aligned} \tag{10}$$

This corresponds to embedding a decision tree as a layer of a deep neural network (using $\mathbf{z}$ as intermediate representation) and optimizing the parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ of the whole model by back-propagation. In practice, we perform mini-batch updates for efficient training; the procedure is sketched in Algorithm 2. Here we define $q_\theta(\mathbf{x}_i) := \mathbf{q}_i$ to make explicit the dependence of $\mathbf{q}_i$ on $\boldsymbol{\theta}$.

4. Experiments

In this section we showcase our method on both: (a) *Classification/Regression* for tabular data, where tree-based models have been demonstrated to have superior performance over MLPs (Popov et al., 2020); and (b) *Hierarchical clustering* on unsupervised data. Our experiments demonstrate that our method leads to predictors that are competitive with state-of-the-art tree-based approaches, scaling better with the size of datasets and generalizing to many tasks. Further we visualize the trees learned by our method and how sparsity is easily adjusted by tuning the hyper-parameter λ.

Architecture details. We use a linear function or a multi-layer perceptron (L fully-connected layers with ELU activation (Clevert et al., 2016) and dropout) for $f_\phi(\cdot)$ and choose between linear or linear followed by ELU splitting

Table 1: Results on tabular regression datasets. We report average and standard deviations over 4 runs of MSE and **bold** best results, and those within a standard deviation from it, for each family of algorithms (single tree or ensemble). For the single tree methods we additionally report the average training times (s).

	METHOD	YEAR	MICROSOFT	YAHOO
Single	CART	96.0 ± 0.4	$0.599 \pm$ 1e-3	$0.678 \pm$ 17e-3
	ANT	$\mathbf{77.8 \pm 0.6}$	$\mathbf{0.572 \pm}$ **2e-3**	$\mathbf{0.589 \pm}$ **2e-3**
	Ours	$\mathbf{77.9 \pm 0.4}$	$\mathbf{0.573 \pm}$ **2e-3**	$\mathbf{0.591 \pm}$ **1e-3**
Ens.	NODE	$\mathbf{76.2 \pm 0.1}$	$0.557 \pm$ 1e-3	$0.569 \pm$ 1e-3
	XGBoost	78.5 ± 0.1	$\mathbf{0.554 \pm}$ **1e-3**	$\mathbf{0.542 \pm}$ **1e-3**
Time	CART	23s	20s	26s
	ANT	4674s	1457s	4155s
	ours	1354s	1117s	825s

functions $s_\theta(\cdot)$ (we limit the search for simplicity, there are no restrictions except differentiability).

4.1. Supervised Learning on Tabular Datasets

Our first set of experiments is on supervised learning with heterogeneous tabular datasets, where we consider both regression and binary classification tasks. We minimize the Mean Square Error (MSE) on regression datasets and the Binary Cross-Entropy (BCE) on classification datasets. We compare our results with tree-based architectures, which either train a single or an ensemble of decision trees. Namely, we compare against the greedy CART algorithm (Breiman et al., 1984) and two optimal decision tree learners: OPTREE with local search (Optree-LS, Dunn, 2018) and a state-of-the-art optimal tree method (GOSDT, Lin et al., 2020). We also consider three baselines with probabilistic routing: deep neural decision trees (DNDT, Yang et al., 2018), deep neural decision forests (Kontschieder et al., 2015) configured to jointly optimize the routing and the splits and to use an ensemble size of 1 (NDF-1), and adaptive neural networks (ANT, Tanno et al., 2019). As for the ensemble baselines, we compare to NODE (Popov et al., 2020), the state-of-the-art method for training a forest of differentiable oblivious decision trees on tabular data, and to XGBoost (Chen & Guestrin, 2016), a scalable tree boosting method. We carry out the experiments on the following datasets. **Regression:** *Year* (Bertin-Mahieux et al., 2011), Temporal regression task constructed from the Million Song Dataset; *Microsoft* (Qin & Liu, 2013), Regression approach to the MSLR-Web10k Query–URL relevance prediction for learning to rank; *Yahoo* (Chapelle & Chang, 2011), Regression approach to the C14 learning-to-rank challenge. **Binary classification:** *Click*, Link click prediction based on the KDD Cup 2012 dataset, encoded and subsampled following Popov et al. (2020); *Higgs* (Baldi et al., 2014), prediction of Higgs boson–producing events.

For all tasks, we follow the preprocessing and task setup from (Popov et al., 2020). All datasets come with training/test splits. We make use of 20% of the training set as validation set for selecting the best model over training and for tuning the hyperparameters. We tune the hyperparameters for all methods. and optimize eq. (10) and all neural network methods (DNDT, NDF, ANT and NODE) using the Quasi-Hyperbolic Adam (Ma & Yarats, 2019) stochastic gradient descent method. Further details are provided in the supplementary. Tables 1 and 2 report the obtained results on the regression and classification datasets respectively.[5] Unsurprisingly, ensemble methods outperfom single-tree ones on all datasets, although at the cost of being harder to visualize/interpret. Our method has the advantage of (a) generalizing to any task; (b) outperforming or matching all single-tree methods; (c) approaching the performance of ensemble-based methods; (d) scaling well with the size of datasets. These experiments show that our model is also significantly faster to train, compared to its differentiable tree counterparts NDF-1 and ANT, while matching or beating the performance of these baselines, and it generally provides the best trade-off between time complexity and accuracy over all datasets (visualizations of this trade-off are reported in the supplementary material). Further results on smaller datasets are available in the supplementary material to provide a comparison with optimal tree baselines.

Table 2: Results on tabular classification datasets. We report average and standard deviations over 4 runs of error rate. Best result for each family of algorithms (single tree or ensemble) are in **bold**. Experiments are run on a machine with 16 CPUs and 64GB of RAM, with a training time limit of 3 days. We denote methods that exceed this memory and training time as OOM and OOT, respectively. For the single tree methods we additionally report the average training times (s) when available.

	METHOD	CLICK	HIGGS
Single Tree	GOSDT	OOM	OOM
	OPTREE-LS	OOT	OOT
	DNDT	$0.4866 \pm$ 1e-2	OOM
	NDF-1	$0.3344 \pm$ 5e-4	$0.2644 \pm$ 8e-4
	CART	$0.3426 \pm$ 11e-3	$0.3430 \pm$ 8e-3
	ANT	0.4035 ± 0.1150	$0.2430 \pm$ 6e-3
	Ours	$\mathbf{0.3340 \pm}$ **3e-4**	$\mathbf{0.2201 \pm}$ **3e-4**
Ens.	NODE	$\mathbf{0.3312 \pm}$ **2e-3**	$\mathbf{0.210 \pm}$ **5e-4**
	XGBoost	$\mathbf{0.3310 \pm}$ **2e-3**	$0.2334 \pm$ 1e-3
Time	DNDT	681s	-
	NDF-1	3768s	43593s
	CART	3s	113s
	ANT	75600s	62335s
	ours	524s	18642s

4.2. Self-Supervised Hierarchical Clustering

To show the versatility of our method, we carry out a second set of experiments on hierarchical clustering tasks. Inspired by the recent success of self-supervised learning approaches

[5] GOSDT/DNDT/Optree-LS/NDF are for classification only.

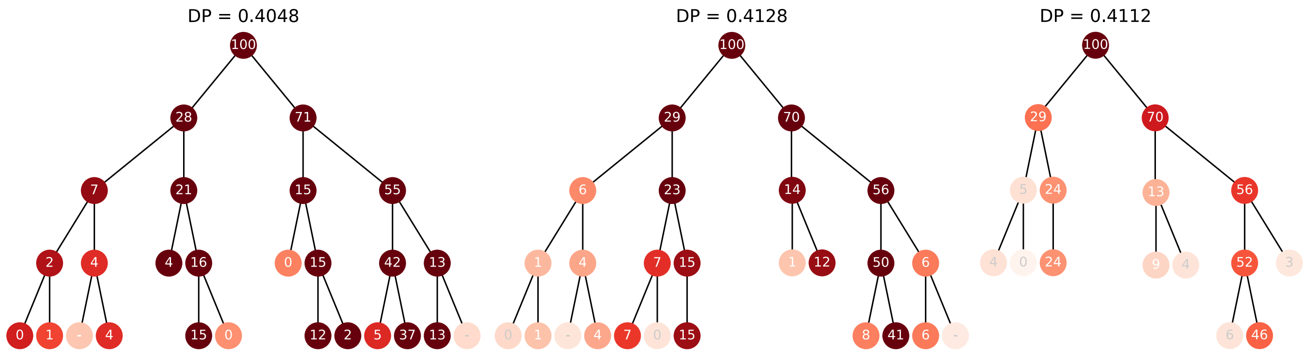


Figure 6: *Glass* tree routing distribution, in rounded percent of dataset, for λ left-to-right in $\{1, 10, 100\}$. The larger λ, the more nodes are pruned. We report dendrogram purity (DP) and represent the nodes by the percentage of points traversing them (normalized at each depth level) and with a color intensity depending on their $\mathbf{a}_t$ (the darker the higher $\mathbf{a}_t$). The empty nodes are labeled by a dash $-$ and the inactive nodes ($a_t = 0$) have been removed.

Table 3: Results for hierarchical clustering. We report average and standard deviations of *dendrogram purity* over four runs.

METHOD	GLASS	COVTYPE
Ours	0.468 ± 0.028	$\mathbf{0.459 \pm 0.008}$
gHHC	0.463 ± 0.002	0.444 ± 0.005
HKMeans	$\mathbf{0.508 \pm 0.008}$	0.440 ± 0.001
BIRCH	0.429 ± 0.013	0.440 ± 0.002

(Lan et al., 2019; He et al., 2020), we learn a tree for hierarchical clustering in a self-supervised way. Specifically, we regress a subset of input features from the remaining features, minimizing the MSE. This allows us to use eq. (10) to learn a hierarchy (tree). To evaluate the quality of the learned trees, we compute their dendrogram purity (DP; Monath et al., 2019). DP measures the ability of the learned tree to separate points from different classes, and corresponds to the expected purity of the least common ancestors of points of the same class.

We experiment on the following datasets: *Glass* (Dua & Graff, 2017): glass identification for forensics, and *Covtype* (Blackard & Dean, 1999; Dua & Graff, 2017): cartographic variables for forest cover type identification. For *Glass*, we regress features 'Refractive Index' and 'Sodium,' and for *Covtype* the horizontal and vertical 'Distance To Hydrology.' We split the datasets into training/validation/test sets, with sizes 60%/20%/20%. Here we only consider linear f_ϕ. As before, we optimize Problem 10 using the Quasi-Hyperbolic Adam algorithm and tune the hyper-parameters using the validation reconstruction error.

As baselines, we consider: BIRCH (Zhang et al., 1997) and Hierarchical KMeans (HKMeans), the standard methods for performing clustering on large datasets; and the recently proposed gradient-based Hyperbolic Hierarchical Clustering (gHHC, Monath et al., 2019) designed to construct trees in hyperbolic space. Table 3 reports the dendrogram purity scores for all methods. Our method yields results

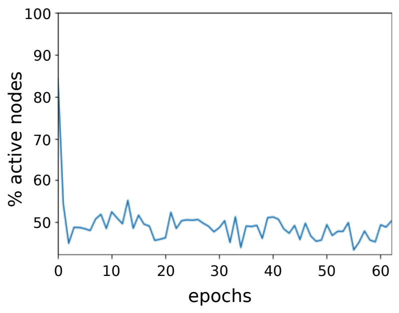


Figure 7: Percentage of active nodes during training as a function of the number of epochs on *Glass*, $D = 6$, $\lambda = 10$.

comparable to all baselines, even though not specifically tailored to hierarchical clustering.

Tree Pruning The hyper-parameter λ in Problem 3 controls how aggressively the tree is pruned, hence the amount of tree splits that are actually used to make decisions. This is a fundamental feature of our framework as it allows to smoothly trim the portions of the tree that are not necessary for the downstream task, resulting in lower computing and memory demands at inference. In Figure 6, we study the effects of pruning on the tree learned on *Glass* with a depth fixed to $D = 4$. We report how inputs are distributed over the learned tree for different values of λ. We notice that increasing λ effectively prune nodes and entire portions of the tree, without significantly impact performance (as measured by dendrogram purity).

To look into the evolution of pruning during training, we further plot the % of active (unpruned) nodes within a training epoch in Figure 7. We observe that (a) it appears possible to increase and decrease this fraction through training (b) the fraction seems to stabilize in the range 45%-55% after a few epochs.

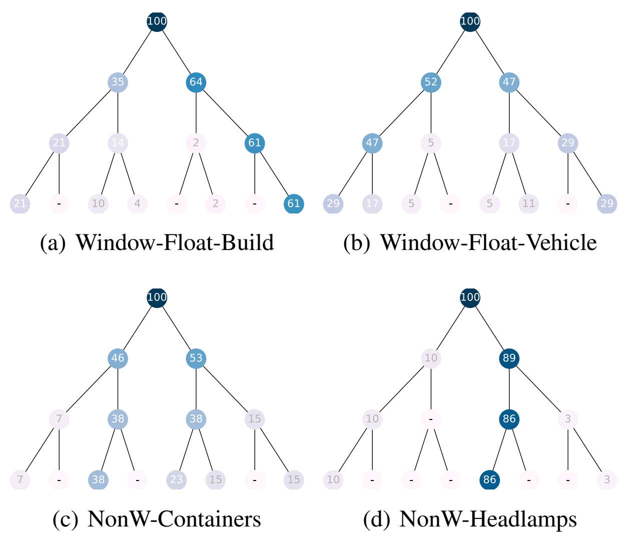


Figure 8: Class routing distributions on *Glass*, with distributions normalized over each depth level. Trees were trained with optimal hyper-parameters (depth $D = 5$), but we plot nodes up to $D = 3$ for visualization ease. Empty nodes are labeled by a dash $-$.

Class Routing To gain insights on the latent structure learned by our method, we study how points are routed through the tree, depending on their class. The *Glass* dataset is particularly interesting to analyze as its classes come with an intrinsic hierarchy, *e.g.*, with superclasses *Window* and *NonWindow*. Figure 8 reports the class routes for four classes. As the trees are constructed without supervision, we do not expect the structure to exactly reflect the class partition and hierarchy. Still, we observe that points from the same class or super-class traverse the tree in a similar way. Indeed, trees for class *Build* 8(a) and class *Vehicle* 8(b) that both belong to the *Window* super-class, share similar paths, unlike the classes Containers 8(c) and Headlamps 8(d).

5. Discussion

In this work we have presented a new optimization approach to learn trees for a variety of machine learning tasks. Our method works by sparsely relaxing a ILP for tree traversal and pruning, to enable simultaneous optimization of these parameters, alongside splitting parameters and downstream functions via argmin differentiation. Our approach nears or improves upon recent work in both supervised learning and hierarchical clustering. We believe there are many exciting avenues for future work. One particularly interesting direction would be to unify recent advances in tight relaxations of nearest neighbor classifiers with this approach to learn efficient neighbor querying structures such as ball trees. Another idea is to adapt this method to learn instance-specific trees such as parse trees.

Acknowledgements

The authors are thankful to Mathieu Blondel and Caio Corro for useful suggestions. They acknowledge the support of Microsoft AI for Earth grant in the form of Microsoft Azure computing credits. VN acknowledges support from the European Research Council (ERC StG DeepSPIN 758969) and the Fundação para a Ciência e Tecnologia through contract UIDB/50008/2020 while at the Instituto de Telecomunicações.

References

Sina Aghaei, Mohammad Javad Azizi, and Phebe Vayanos. Learning optimal and fair decision trees for non-discriminative decision-making. In *Proc. of AAAI*, 2019.

Sina Aghaei, Andres Gomez, and Phebe Vayanos. Learning optimal classification trees: Strong max-flow formulations. *preprint arXiv:2002.09142*, 2020.

A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and Z. Kolter. Differentiable convex optimization layers. In *Proc. of NeurIPS*, 2019a.

Akshay Agrawal, Robin Verschueren, Steven Diamond, and Stephen Boyd. A rewriting system for convex optimization problems. *Journal of Control and Decision*, 2018.

Akshay Agrawal, Shane Barratt, Stephen Boyd, Enzo Busseti, and Walaa M Moursi. Differentiating through a cone program. *Journal of Applied and Numerical Optimization*, 2019b. ISSN 25625527. doi: 10.23952/jano.1.2019.2.02.

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proc. of ACM SIGKDD*, 2019.

Brandon Amos. *Differentiable Optimization-Based Modeling for Machine Learning*. PhD thesis, Carnegie Mellon University, May 2019.

Brandon Amos and J Zico Kolter. OptNet: Differentiable optimization as a layer in neural networks. In *Proc. of ICML*, 2017.

Elaine Angelino, Nicholas Larus-Stone, Daniel Alabi, Margo Seltzer, and Cynthia Rudin. Learning certifiably optimal rule lists for categorical data. *Journal of Machine Learning Research*, 2017.

Pierre Baldi, Peter Sadowski, and Daniel Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature Communications*, 2014.

RE Barlow, DJ Bartholomev, JM Brenner, and HD Brunk. Statistical inference under order restrictions: The theory and application of isotonic regression, 1972.

Kristin P Bennett. Decision tree construction via linear programming. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 1992.

Kristin P Bennett and Jennifer A Blue. Optimal decision trees. *Rensselaer Polytechnic Institute Math Report*, 1996.

Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. The million song dataset. In *Proc. of ISMIR*, 2011.

Dimitri P Bertsekas. *Nonlinear Programming*. Athena Scientific Belmont, 1999.

Dimitris Bertsimas and Jack Dunn. Optimal classification trees. *Machine Learning*, 2017.

Jock A Blackard and Denis J Dean. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and Electronics in Agriculture*, 1999.

Jonathan Borwein and Adrian S Lewis. *Convex analysis and nonlinear optimization: theory and examples*. Springer Science & Business Media, 2010.

Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and Regression Trees*. CRC press, 1984.

Miguel A Carreira-Perpinán and Pooya Tavallali. Alternating optimization of decision trees, with application to learning sparse oblique trees. In *Proc. of NeurIPS*, 2018.

Olivier Chapelle and Yi Chang. Yahoo! learning to rank challenge overview. In *Proc. of the learning to rank challenge*, 2011.

Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proc. of KDD*, 2016.

Jihun Choi, Kang Min Yoo, and Sang-goo Lee. Learning to compose task-specific tree structures. In *Proc. of AAAI*, 2018.

Frank H Clarke. *Optimization and nonsmooth analysis*. SIAM, 1990.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (ELUs). *Proc. of ICLR*, 2016.

Benoît Colson, Patrice Marcotte, and Gilles Savard. An overview of bilevel optimization. *Annals of operations research*, 2007.

Caio Corro and Ivan Titov. Learning latent trees with stochastic perturbations and differentiable dynamic programming. In *Proc. of ACL*, 2019a.

Caio Corro and Ivan Titov. Differentiable Perturb-and-Parse: Semi-Supervised Parsing with a Structured Variational Autoencoder. In *Proc. of ICLR*, 2019b.

John M Danskin. The theory of max-min, with applications. *SIAM Journal on Applied Mathematics*, 1966.

Sanjoy Dasgupta. A cost function for similarity-based hierarchical clustering. In *Proc. of STOC*, 2016.

Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 2016.

Josip Djolonga and Andreas Krause. Differentiable learning of submodular models. In *Proc. of NeurIPS*, 2017.

Justin Domke. Learning graphical model parameters with approximate marginal inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2013.

Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

Jack William Dunn. *Optimal trees for prediction and prescription*. PhD thesis, Massachusetts Institute of Technology, 2018.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proc. of ICML*, 2017.

Stephen Gould, Basura Fernando, Anoop Cherian, Peter Anderson, Rodrigo Santa Cruz, and Edison Guo. On differentiating parameterized argmin and argmax problems with application to bi-level optimization. *CoRR*, abs/1607.05447, 2016.

Oktay Günlük, Jayant Kalagnanam, Matt Menickelly, and Katya Scheinberg. Optimal decision trees for categorical data via integer programming. *Journal of Global Optimization*, 2021.

Moritz Hardt, Ben Recht, and Yoram Singer. Train faster, generalize better: Stability of stochastic gradient descent. In *Proc. of ICML*, 2016.

Hussein Hazimeh, Natalia Ponomareva, Petros Mol, Zhenyu Tan, and Rahul Mazumder. The tree ensemble layer: Differentiability meets conditional computation. In *Proc. of ICML*, 2020.

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proc. of CVPR*, 2020.

Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *Proc. of NeurIPS*, 2017.

Xiyang Hu, Cynthia Rudin, and Margo Seltzer. Optimal sparse decision trees. In *Proc. of NeurIPS*, 2019.

Ozan İrsoy, Olcay Taner Yıldız, and Ethem Alpaydın. Soft decision trees. In *Proc. of ICPR*, 2012.

Yoon Kim, Chris Dyer, and Alexander Rush. Compound probabilistic context-free grammars for grammar induction. In *Proc. of ACL*, 2019a.

Yoon Kim, Alexander Rush, Lei Yu, Adhiguna Kuncoro, Chris Dyer, and Gábor Melis. Unsupervised recurrent neural network grammars. In *Proc. of NAACL-HLT*, 2019b.

Ari Kobren, Nicholas Monath, Akshay Krishnamurthy, and Andrew McCallum. A hierarchical algorithm for extreme clustering. In *Proc. of KDD*, 2017.

Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, and Samuel Rota Bulo. Deep neural decision forests. In *Proc. of ICCV*, 2015.

Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *Proc. of ICLR*, 2018.

Akshay Krishnamurthy, Sivaraman Balakrishnan, Min Xu, and Aarti Singh. Efficient active algorithms for hierarchical clustering. In *Proc. of ICML*, 2012.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. In *Proc. of ICLR*, 2019.

Nathan Lay, Adam P Harrison, Sharon Schreiber, Gitesh Dawer, and Adrian Barbu. Random hinge forest for differentiable learning. *preprint arXiv:1802.03882*, 2018.

Benjamin Letham, Cynthia Rudin, Tyler H McCormick, David Madigan, et al. Interpretable classifiers using rules and bayesian analysis: Building a better stroke prediction model. *The Annals of Applied Statistics*, 2015.

Jimmy Lin, Chudi Zhong, Diane Hu, Cynthia Rudin, and Margo Seltzer. Generalized and scalable optimal sparse decision trees. In *Proc. of ICML*, 2020.

Yang Liu and Mirella Lapata. Learning structured text representations. *TACL*, 2018.

Jerry Ma and Denis Yarats. Quasi-hyperbolic momentum and adam for deep learning. *Proc. of ICLR*, 2019.

Jean Maillard, Stephen Clark, and Dani Yogatama. Jointly learning sentence embeddings and syntax with unsupervised tree-lstms. *Natural Language Engineering*, 2019.

Patrick Mair, Kurt Hornik, and Jan de Leeuw. Isotone optimization in r: pool-adjacent-violators algorithm (pava) and active set methods. *Journal of Statistical Software*, 2009.

Nicholas Monath, Manzil Zaheer, Daniel Silva, Andrew McCallum, and Amr Ahmed. Gradient-based hierarchical clustering using continuous representations of trees in hyperbolic space. In *Proc. of KDD*, 2019.

Benjamin Moseley and Joshua Wang. Approximation bounds for hierarchical clustering: Average linkage, bisecting k-means, and local search. In *Proc. of NeurIPS*, 2017.

Vlad Niculae, André FT Martins, Mathieu Blondel, and Claire Cardie. SparseMAP: Differentiable sparse structured inference. In *Proc. of ICML*, 2018a.

Vlad Niculae, André FT Martins, and Claire Cardie. Towards dynamic computation graphs via sparse latent structure. In *Proc. of EMNLP*, 2018b.

Harold J. Payne and William S. Meisel. An algorithm for constructing optimal binary decision trees. *IEEE Transactions on Computers*, 1977.

Fabian Pedregosa. Hyperparameter optimization with approximate gradient. In *Proc. of ICML*, 2016.

Sergei Popov, Stanislav Morozov, and Artem Babenko. Neural oblivious decision ensembles for deep learning on tabular data. *Proc. of ICLR*, 2020.

Tao Qin and Tie-Yan Liu. Introducing LETOR 4.0 datasets. *CoRR*, abs/1306.2597, 2013.

J Ross Quinlan. Induction of decision trees. *Machine Learning*, 1986.

J Ross Quinlan. C4. 5: Programs for machine learning. 1993.

Aravind Rajeswaran, Chelsea Finn, Sham Kakade, and Sergey Levine. Meta-learning with implicit gradients. In *Proc. of NeurIPS*, 2019.

Cynthia Rudin and Şeyda Ertekin. Learning customized and optimized lists of rules with mathematical programming. *Mathematical Programming Computation*, 2018.

Jeffrey C Schlimmer. Concept acquisition through representational adjustment. 1987.

Veselin Stoyanov, Alexander Ropson, and Jason Eisner. Empirical risk minimization of graphical model parameters given approximate inference, decoding, and model structure. In *Proc. of AISTATS*, 2011.

Ryutaro Tanno, Kai Arulkumaran, Daniel Alexander, Antonio Criminisi, and Aditya Nori. Adaptive neural trees. In *Proc. of ICML*, 2019.

Sicco Verwer and Yingqian Zhang. Learning optimal classification trees using a binary linear program formulation. In *Proc. of AAAI*, 2019.

Dwi H Widyantoro, Thomas R Ioerger, and John Yen. An incremental approach to building a cluster hierarchy. In *Proc. of ICDM*, 2002.

Adina Williams, Andrew Drozdov, and Samuel R Bowman. Do latent tree learning models identify meaningful structure in sentences? *TACL*, 2018.

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

D Yogatama, P Blunsom, C Dyer, E Grefenstette, and W Ling. Learning to compose words into sentences with reinforcement learning. In *Proc. of ICLR*, 2017.

Yao-Liang Yu and Eric P Xing. Exact algorithms for isotonic regression and related. In *Journal of Physics: Conference Series*, 2016.

Constantin Zalinescu. *Convex analysis in general vector spaces*. World Scientific, 2002.

Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: A new data clustering algorithm and its applications. *Data Mining and Knowledge Discovery*, 1997.

Haoran Zhu, Pavankumar Murali, Dzung T. Phan, Lam M. Nguyen, and Jayant Kalagnanam. A scalable mip-based method for learning optimal multivariate decision trees. In *Proc. of NeurIPS*, 2020.

A. Supplementary Material

A.1. Motivation and derivation of Equation 3.

Smoothing and symmetric regularization. Consider the Heaviside function used to make a binary decision, defined as

$$h(q) = \begin{cases} 1, & q \geq 0, \\ 0, & q < 0. \end{cases} \tag{11}$$

To relax this discontinuous function into a continuous and differentiable one, we remark that it is almost equivalent to the variational problem $\arg\max \{zq : z \in \{0,1\}\}$ with the exception of the "tied" case $q = 0$. We may relax the binary constraints into the convex interval $[0,1]$ and add any strongly-concave regularizer to ensure the relaxation has a unique maximizer everywhere and thus defines a continuous differentiable mapping:.

$$\arg\max_{0 \leq z \leq 1} zq - \omega(z). \tag{12}$$

A tempting choice would be $\omega(z) = z^2/2$. Note hovewer that as $|q|$ goes to zero, the problem converges to $\arg\max_z -z^2/2$ which is 0. This does not match our intuition for a good continuous relaxation, where a small q should suggest lack of commitment toward either option. More natural here would be a regularizer that prefers $z = 1/2$ as $|q|$ goes to zero. We may obtain such a regularizer by encoding the choice as a two-dimensional vector $[z, 1-z]$ and penalizing its squared l_2 norm, giving:

$$\omega(z) = \frac{1}{4}\big\|[z, 1-z]\big\|^2 = \frac{1}{4}\big(z^2 + (1-z)^2\big). \tag{13}$$

As $|q| \to 0$, $\arg\max_z -\omega(z)$ goes toward $z = .5$. This regularizer, applied over an entire vector $\sum_i \omega(z_i)$, is exactly the one we employ. The $1/4$ weighting factor will simplify the derivation in the next paragraph.

Rewriting as a scaled projection. For brevity consider the terms for a single node, dropping the i index:

$$\mathbf{z}^\top \mathbf{q} - \frac{1}{4}\big(\|\mathbf{z}\|^2 + \|\mathbf{1} - \mathbf{z}\|^2\big) \tag{14}$$

$$= \mathbf{z}^\top \mathbf{q} - \frac{1}{4}\|\mathbf{z}\|^2 - \frac{1}{4}\|\mathbf{1}\|^2 - \frac{1}{4}\|\mathbf{z}\|^2 + \frac{1}{2}\mathbf{z}^\top \mathbf{1} \tag{15}$$

$$= \mathbf{z}^\top \underbrace{\left(\mathbf{q} + \frac{1}{2}\mathbf{1}\right)}_{\tilde{\mathbf{q}}} - \frac{1}{2}\|\mathbf{z}\|^2 - \frac{1}{4}\|\mathbf{1}\|^2. \tag{16}$$

We highlight and ignore terms that are constant w.r.t. $\mathbf{z}$, as they have no effect on the optimization.

$$= \mathbf{z}^\top \tilde{\mathbf{q}} - \frac{1}{2}\|\mathbf{z}\|^2 + \text{const} \tag{17}$$

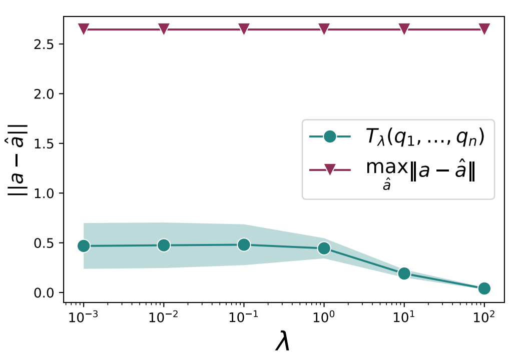
We complete the square by adding and subtracting $.5\|\tilde{\mathbf{q}}\|^2$, also constant w.r.t. $\mathbf{z}$.

$$= \mathbf{z}^\top \tilde{\mathbf{q}} - \frac{1}{2}\|\mathbf{z}\|^2 - \frac{1}{2}\|\tilde{\mathbf{q}}\|^2 + \text{const} \tag{18}$$

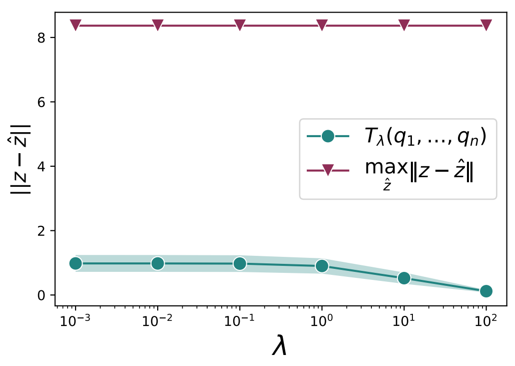
$$= -\frac{1}{2}\|\mathbf{z} - \tilde{\mathbf{q}}\|^2 + \text{const}. \tag{19}$$

We may now add up these terms for every i and flip the sign, turning the maximization into a minimization, yielding the desired result.

To study how $\mathbf{a}$ and $\mathbf{z}$ are impacted by the relaxation, in Figure 9 we report an empirical comparison of the solutions of the original Problem (2) and of Problem (3), after applying the quadratic relaxation. For this experiment, we generated 1000 random problems by uniformly drawing $\mathbf{q} \in [-2, 2]^{n|\mathcal{T}|}$, with $n = 10$ points and a tree of depth $D = 2$ and $|\mathcal{T}| = 7$ nodes, and upper bounding the value of any child node with the one of its parent ($q_{it} = \min(q_{it}, q_{ip(t)})$), so that the tree constraints are satisfied. We notice that the solutions of the relaxed problem are relatively close to the solutions of the original problem and that they tend to the optimal ones as the pruning hyper-parameter λ increases.



(a) Pruning variable $\mathbf{a}$.



(b) Tree traversals $\mathbf{z}$.

Figure 9: Comparison of the solutions of Problem (2), denoted by $\mathbf{a}$ and $\mathbf{z}$, with the solutions of Problem (3), denoted by $\hat{\mathbf{a}}$ and $\hat{\mathbf{z}}$. Average and one-standard-deviation intervals over 1000 random problems are represented as a function of the pruning hyper-parameter λ, as well as the maximal possible gap.

A.2. Proof of Proposition 2

Let $G \subset \mathcal{T}$ be a subset of (pooled) node indices. We seek to solve

$$\underset{a \in \mathbb{R}}{\arg\min} \sum_{t \in G} g_t(a) \;:=\; \underset{a \in [0,1]}{\arg\min} \frac{\lambda}{2} \sum_{t \in G} a^2 + \sum_{i: a \leq q_{it}} {}^1\!/\!_2 (a - q_{it} - {}^1\!/\!_2)^2 \tag{20}$$

Note that the final summation implicitly depends on the unknown a. But, regardless of the value of a, if $q_{it} \leq q_{i't'}$ and q_{it} is included in the sum, then $q_{i't'}$ must also be included by transitivity. We may therefore characterize the solution $a^\star$ via the number of active inequality constraints $k^\star = \left|\{(i,t) : a^\star \leq q_{i,t} + {}^1\!/\!_2\}\right|$. We do not know $a^\star$, but it is trivial to find by testing all possible values of k. For each k, we may find the set $S(k)$ defined in the proposition. For a given k, the candidate objective is

$$J_k(a) = \frac{\lambda}{2} \sum_{t \in G} a^2 + \sum_{(i,t) \in S(k)} {}^1\!/\!_2 (a - q_{it} - {}^1\!/\!_2)^2 \tag{21}$$

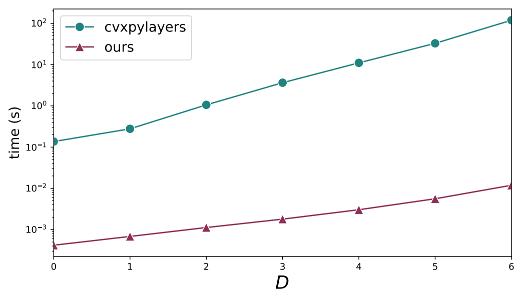
and the corresponding $a(k)$ minimizing it can be found by setting the gradient to zero:

$$J_k'(a) = \lambda \sum_{t \in G} a + \sum_{(i,t) \in S(k)} (a - q_{i,t} - {}^1\!/\!_2) := 0 \quad \iff \quad a(k) = \frac{\sum_{(i,t) \in S(k)} (q_{it} + {}^1\!/\!_2)}{\lambda |G| + k}. \tag{22}$$

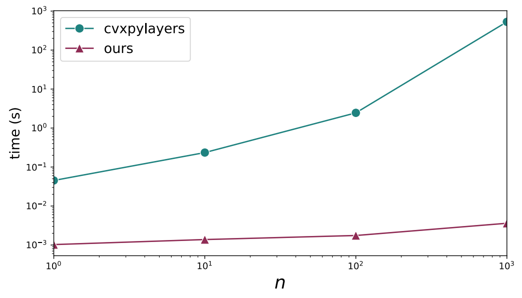
Since $|S(k)| = k$ and each increase in k adds a non-zero term to the objective, we must have $J_1\big(a(1)\big) \leq J_1\big(a(2)\big) \leq J_2\big(a(2)\big) \leq \ldots$, so we must take k to be as small as possible, while also ensuring the condition $|\{(i,t) : a(k) \leq q_{it} + {}^1\!/\!_2\}| = k$, or, equivalently, that $a(k) > q_{j([k+1])} + {}^1\!/\!_2$. The box constraints may be integrated at this point via clipping, yielding $a^\star = \operatorname{Proj}_{[0,1]}\big(a(k^\star)\big)$.

A.3. Benchmarking solvers

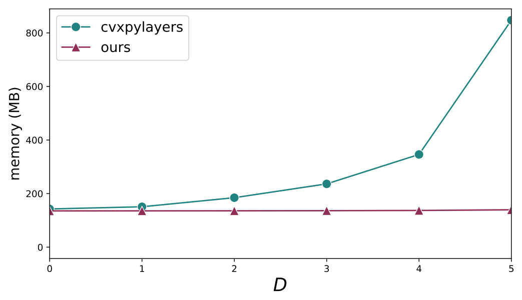
We study the running time and memory usage of Algorithm 1 depending on the number of data points n and the chosen tree depth D. We compare its solving time and memory with those needed by the popular differentiable convex optimization framework **Cvxpylayers** (Agrawal et al., 2019a) (based on Cvxpy (Diamond & Boyd, 2016; Agrawal et al., 2018)) to solve Problem 3. As this library is based on solvers implemented in Objective C and C we implement our approach in C++ for a fair comparison. We report the solving times and memory consumption in Figure 10 for a forward-and-backward pass, where we vary one parameter n or D at a time and fix the other. The algorithm that we specifically designed to solve problem (3) is indeed several magnitude faster than the considered generic solver and consumes significantly less memory, overall scaling better to the size of the tree and the number of inputs.



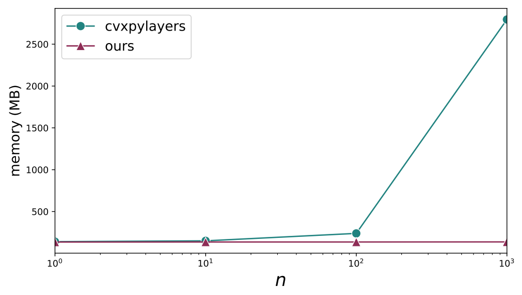
(a) Fixed number of points $n = 100$



(b) Fixed tree depth $D = 3$



(c) Fixed number of points $n = 100$



(d) Fixed tree depth $D = 3$

Figure 10: Comparison of Algorithm 1 and **Cvxpylayers** in terms of (a-b) running time and (c-d) memory usage. n takes values in a range that covers common choices of batch size. Time and n axis are represented in logarithmic scale.

A.4. Further experimental details

We tune the hyper-parameters of all methods with Optuna Bayesian optimizer (Akiba et al., 2019), fixing the number of trials to 100. For all back-propagation-based methods, we fix the learning rate to 0.001 and the batch size to 512, use a scheduler reducing this parameter of a factor of 10 every 2 epochs where the validation loss has not improved, and fix the maximal number of epochs to 100 and patience equal to 10 for early stopping. For our method, we further initialize the bias $\boldsymbol{b} \in \mathbb{R}^{|\mathcal{T}|}$ for the split function $s_\theta(x)$ (explicitly defining $s_\theta(x) = s_{\theta\setminus\boldsymbol{b}}(x) + \boldsymbol{b}$) to ensure that points are equally distributed over the leaves at initialization. We hence initialize the bias to minus the initial average value of the training points traversing each node: $b_t = -\frac{1}{|\{x_i | q_{it} > 0\}_{i=1}^n|} \sum_{i=1}^n s_{\theta_t \setminus b_t}(x_i) \mathbb{1}[q_{it} > 0]$.

Experiments on tabular datasets The other hyper-parameters for these experiments are chosen as follows:

- **Ours**: we tune D in $\{2, \dots, 6\}$, λ in $[1\text{e-}3, 1\text{e+}3]$ with log-uniform draws, the number of layers of the MLP L in $\{2, \dots, 5\}$ and its dropout probability uniformly in $[0, 0.5]$, and the choice of activation for the splitting functions as linear or Elu;
- **Optree-LS**: we fix the tree depth $D = 6$;
- **CART**: we not bound the maximal depth and tune *feature rate* uniformly in $[0.5, 1]$, *minimal impurity decrease* in $[0, 1]$, minimal Cost-Complexity Pruning α log-uniformly in $[1\text{e-}16, 1\text{e+}2]$ and *splitter* chosen between best or random;
- **Node** and **XGBoost**: all results are reported from Popov et al. (2020), where they used the same experimental set-up;
- **DNDT**: we tune the softmax temperature uniformly between $[0, 1]$ and the number of feature cuts in $\{1, 2\}$;
- **NDF**: we tune D in $\{2, \dots, 6\}$ and fix the feature rate to 1;
- **ANT**: as this method is extremely modular (like ours), to allow for a fair comparison we choose a similar configuration. We then select as transformer the identity function, as router a linear layer followed by the Relu activation and with soft (sigmoid) traversals, and as solver a MLP with L hidden layers, as defined for our method; we hence tune L in

Table 4: Number of parameters for single-tree methods on tabular datasets.

	METHOD	YEAR	MICROSOFT	YAHOO	CLICK	HIGGS
Single Tree	CART	164	58	883	12	80
	DNDT	-	-	-	4096	-
	NDF-1	-	-	-	78016	47168
	ANT	179265	17217	53249	81	7874
	Ours	186211	52309	55806	43434	701665

$\{2, \ldots, 5\}$ and its dropout probability uniformly in $[0, 0.5]$, and fix the maximal tree depth D to 6; we finally fix the number of epochs for growing the tree and the number of epochs for fine-tuning it both to 50.

Experiments on hierarchical clustering For this set of experiments, we make us of a linear predictor and of linear splitting functions without activation. We fix the learning rate to 0.001 and the batch size 512 for *Covtype* and 8 for *Glass*. The other hyper-parameters of our method are chosen as follows: we tune D in $\{2, \ldots, 6\}$, λ in $[1\text{e-}3, 1\text{e+}3]$ with log-uniform draws. The results of the baselines are reported from Monath et al. (2019).

A.5. Additional experiments

In Figure 11 we represent the average test Error Rates or Mean Square Error as a function of the training time for each single-tree method on the tabular datasets of Section 4.1. Notice that our method provides the best trade-off between time complexity and accuracy over all datasets. In particular, it achieves Error Rates comparable on *Click* and significantly better on *Higgs* w.r.t. **NDF-1** while being several times faster. Table 4 shows that this speed-up is not necessarily due to a smaller number of model parameters, but it comes from the deterministic routing of points through the tree. Despite having model sizes bigger than ANT's ones, our method is significantly faster than this baseline as it offers an efficient way for optimizing the tree structure (via the joint optimization of pruning vector $\boldsymbol{a}$). In comparison, ANT needs to incrementally grow trees in a first phase, to then fine-tune them in a second phase, resulting in a computational overhead. Finally, *DNDT* implements a soft binning with trainable binning thresholds (or cut points) for each feature, hence scaling badly w.r.t this parameter and resulting in a memory overflow on *HIGGS*.

Further comparison with optimal tree baselines We run a set of experiments on small binary classification datasets to compare our method with optimal tree methods, which do not scale well with the dataset size. Specifically we compare against two versions of Optree: one that solves the MIP exactly (Optree-MIP) (Bertsimas & Dunn, 2017), and another that solves it with local search Optree-LS (Dunn, 2018). We also compare with the state-of-the-art optimal tree method GOSDT and to CART. We consider the binary classification datasets

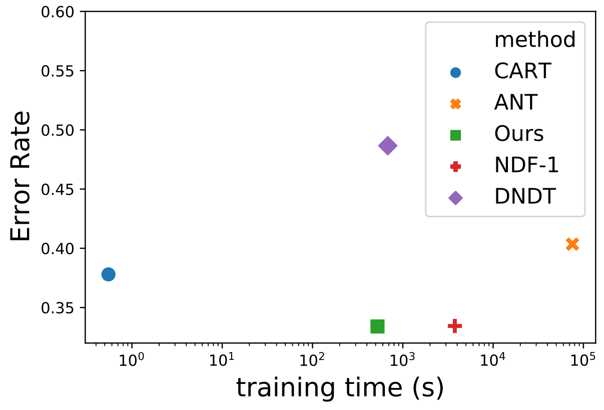
- *Mushrooms* (Schlimmer, 1987): prediction between edible and poisonous mushrooms, with 8124 instances and 22 features;
- *FICO*[6]: loan approval prediction, with 10459 instances and 24 features;
- *Tictactoe*[7]: endgame winner prediction, with 958 instances and 9 features.

We apply a stratified split on all datasets to obtain 60%-20%-20% training-validation-test sets, convert categorical features to ordinal, and z-score them. For our method, we apply the Quasi-Hyperbolic Adam optimization algorithm, with batch size equal to 512 for *Mushrooms* and *FICO*, and 64 for *Tictactoe*.

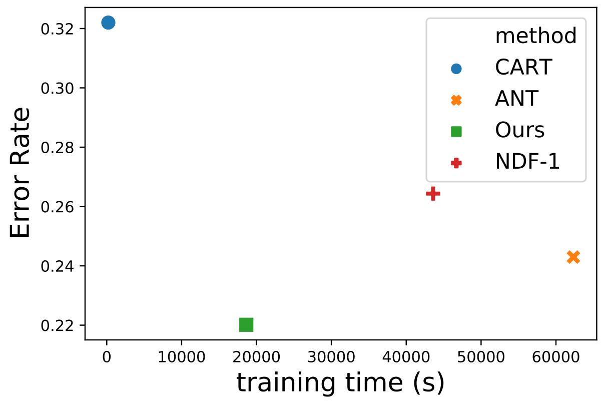
We chose the hyper-parameters as follows:

- **Ours**: we tune D in $\{2, \ldots, 6\}$, λ in $[1\text{e-}3, 1\text{e+}3]$ with log-uniform draws, and make use of a linear predictor and of linear splitting functions without activation;
- **Optree-MIP**: we fix the tree depth to $D = 6$;

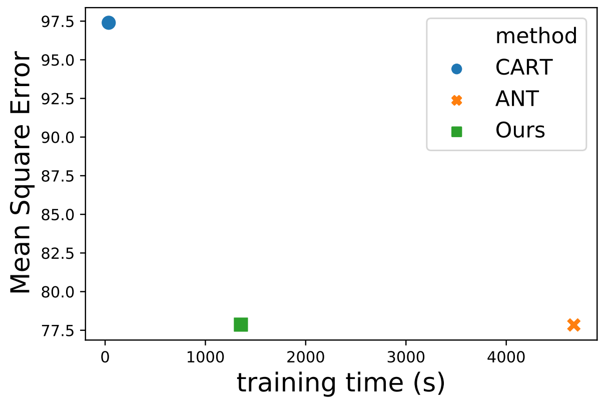
[6]https://community.fico.com/s/explainable-machine-learning-challenge
[7]https://archive.ics.uci.edu/ml/datasets/Tic-Tac-Toe+Endgame



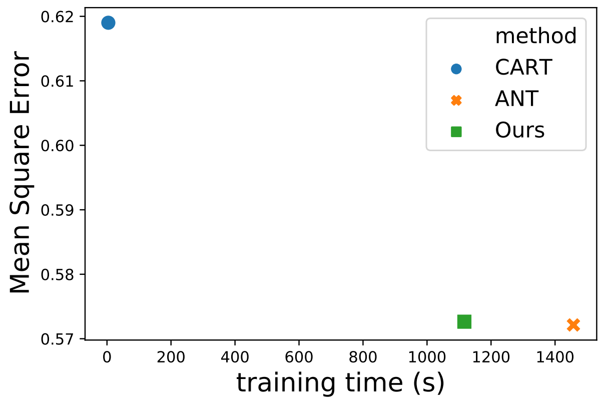
(a) **Click**



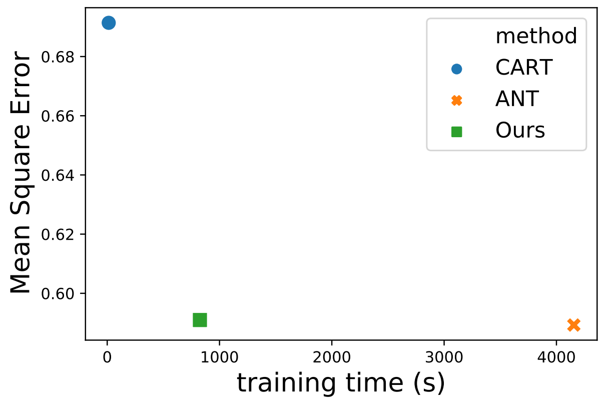
(b) **Higgs**



(c) **Year**



(d) **Microsoft**



(e) **Yahoo**

Figure 11: Average (a,b) Error Rate (c-e) Mean Square Error vs average training time required by each method.

Table 5: Results on the Mushrooms tabular dataset. We report average training time (s), and average and standard deviations of test error rate over 4 runs for binary classification datasets. We **bold** the best result (and those within a standard deviation from it). Experiments are run on a machine with 16 CPUs and 63GB RAM. We limit the training time to 3 days.

METHOD	MUSHROOMS		FICO		TICTACTOE	
	training time	error rate	training time	error rate	training time	error rate
CART	0.004s	$\mathbf{0.0003 \pm 0.0003}$	0.01s	0.3111 ± 0.0042	0.001s	$\mathbf{0.1576 \pm 0.0203}$
OPTREE-MIP	OOT	-	OOT	-	OOT	-
OPTREE-LS	OOT	-	OOT	-	751s	0.449 ± 0.0184
GOSDT	214s	0.0125 ± 0.0027	634s	0.3660 ± 0.0090	40s	0.3490 ± 0.0010
Ours	20s	$\mathbf{0.0005 \pm 0.0008}$	10s	$\mathbf{0.2884 \pm 0.0020}$	13s	0.2669 ± 0.0259

- **Optree-LS**: we tune D in $\{2, \dots, 10\}$;
- **GOSDT**: we tune the regularization parameter λ in $[5\text{e-}3, 1\text{e+}3]$ with log-uniform draws, and set accuracy as the objective function.
- **CART**: we tune D in $\{2, \dots, 20\}$, *feature rate* uniformly in $[0.5, 1]$, *minimal impurity decrease* in $[0, 1]$, α log-uniformly in $[1\text{e-}16, 1\text{e+}2]$ and *splitter* chosen between best or random.

Table 5 reports the performance of all methods across 4 runs. Both OPTREE variants do not finish in under 3 days on *Mushrooms* and *FICO*, which have more than 1000 instances. CART is the fastest method and ranks second in terms of error rate on all datasets but *Tictactoe*, where it surprisingly beats all non-greedy baselines. Our method outperforms optimal tree baselines in terms of error rate and training time on all datasets. We believe the slow scaling of GOSDT is due to the fact that it binarizes features, working with 118, 1817, 27 final binary attributes respectively on *Mushrooms*, *FICO* and *Tictactoe*. Apart from achieving superior performance on supervised classification tasks, we stress the fact that our method generalizes to more tasks, such as supervised regression and clustering as shown in the main text.