GradTree: Learning Axis-Aligned Decision Trees with Gradient Descent

Sascha Marton[1], **Stefan Lüdtke**[2], **Christian Bartelt**[1] **Heiner Stuckenschmidt**[1]

[1]University of Mannheim, Germany
[2]University of Rostock, Germany
{sascha.marton, christian.bartelt, heiner.stuckenschmidt}@uni-mannheim.de, stefan.luedtke@uni-rostock.de

Abstract

Decision Trees (DTs) are commonly used for many machine learning tasks due to their high degree of interpretability. However, learning a DT from data is a difficult optimization problem, as it is non-convex and non-differentiable. Therefore, common approaches learn DTs using a greedy growth algorithm that minimizes the impurity locally at each internal node. Unfortunately, this greedy procedure can lead to inaccurate trees. In this paper, we present a novel approach for learning hard, axis-aligned DTs with gradient descent. The proposed method uses backpropagation with a straight-through operator on a dense DT representation, to jointly optimize all tree parameters. Our approach outperforms existing methods on binary classification benchmarks and achieves competitive results for multi-class tasks. The implementation is available under: https://github.com/s-marton/GradTree

1 Introduction

Decision trees (DTs) are some of the most popular machine learning models and are still frequently used today. In particular, with the growing interest in explainable artificial intelligence, DTs regained popularity due to their interpretability. However, learning a DT is a difficult optimization problem, as it is non-convex and non-differentiable. Therefore, the prevailing method to learn a DT is a greedy procedure that minimizes the impurity at each internal node. The algorithms still in use today, such as CART (Breiman et al. 1984) and C4.5 (Quinlan 1993), were developed in the 1980s and remained largely unchanged since then. Unfortunately, a greedy algorithm optimizes the objective locally at each internal node which constrains the search space, potentially leading to inaccurate trees. We illustrate this issue below:

Example 1 The Echocardiogram dataset (Dua and Graff 2017) deals with predicting one-year survival of patients after a heart attack based on tabular data from an echocardiogram. Figure 1 shows two DTs. The tree on the left is learned by a greedy algorithm, while the one on the right is learned with our gradient-based approach. We can observe that the greedy procedure leads to a tree with a significantly lower performance. Splitting on *wall-motion-score* is the locally optimal split (see Figure 1a), but globally, it is beneficial to split on *wall-motion-score* with different values conditioned on *pericardial-effusion* in the second level (Figure 1b).

In this paper, we propose a novel approach for learning hard, axis-aligned DTs based on a joint optimization of all tree parameters using gradient descent, which we call **Grad**ient-Based Decision **Tree** (GradTree). Similar to optimization in neural networks, GradTree yields a desirable local optimum of parameters that generalizes well to test data. Using a gradient-based optimization, GradTree can overcome the limitations of greedy approaches, which are constrained by sequentially selecting optimal splits, as illustrated in Figure 1. At the same time, GradTree can converge to a local optimum that offers good generalization, and thus provides an advantage over alternative non-greedy methods like optimal DTs, which often suffer from severe overfitting (Zantedeschi, Kusner, and Niculae 2021).

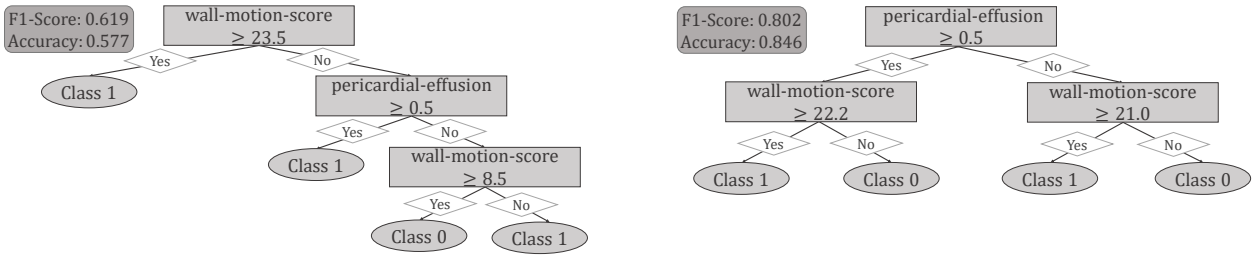
Specifically, our contributions are:

- We introduce a dense DT representation that enables a joint, gradient-based optimization of all tree parameters (Section 3.2).
- We present a procedure to deal with the non-differentiable nature of DTs using backpropagation with a straight-through (ST) operator (Section 3.3).
- We propose a novel tree routing that allows an efficient, parallel optimization of all tree parameters with gradient descent (Section 3.4).

We empirically evaluate GradTree on a large number of real-world datasets (Section 4). GradTree outperforms existing methods for binary classification tasks and achieves competitive results on multi-class datasets. On several benchmark datasets, the performance difference between GradTree and other methods is substantial. The gradient-based optimization of GradTree also provides more flexibility by allowing split adjustments during training and easy integration of custom loss functions.

2 Related Work

Greedy DT Algorithms The most prominent DT learning algorithms still frequently used, namely CART (Breiman et al. 1984) and C4.5 (Quinlan 1993), date back to the 1980s. Both follow a greedy procedure to learn a DT. Since then, many variations to those algorithms have been proposed, for instance C5.0 (Kuhn, Johnson et al. 2013) and GUIDE (Loh



(a) Greedy DT (CART)

(b) Gradient-Based DT (GradTree)

Figure 1: Greedy vs. Gradient-Based DT. Two DTs trained on the Echocardiogram dataset. The CART DT (left) makes only locally optimal splits, while GradTree (right) jointly optimizes all parameters, leading to significantly better performance.

2002, 2009). However, until today, none of these algorithms was able to consistently outperform CART and C4.5 as shown for instance by Zharmagambetov et al. (2021).

Optimal DTs To overcome the issues of a greedy DT induction, many researchers focused on finding an efficient alternative. Optimal DTs aim to optimize an objective (e.g., the purity) through an approximate brute force search to find a globally optimal tree with a certain specification (Zharmagambetov et al. 2021). Therefore, they most commonly use mixed integer optimization (Bertsimas and Dunn 2017) or a branch-and-bound algorithm to remove irrelevant parts from the search space (Aglin, Nijssen, and Schaus 2020; Lin et al. 2020). MurTree (Demirović et al. 2022) further uses dynamic programming, which reduces the runtime significantly. However, most state-of-the-art approaches still require binary data and therefore a discretization of continuous features (Bertsimas and Dunn 2017; Aglin, Nijssen, and Schaus 2020; Demirović et al. 2022), which can lead to information loss. An exception is the approach by Mazumder, Meng, and Wang (2022), which can handle continuous features out-of-the-box. However, their method is optimized for very sparse trees and limited to a maximum depth of 3.

While optimal DTs search for a global optimum, GradTree does not necessarily pursue this. Instead, like optimization in neural networks, it aims for a local optimum that offers good generalization to test data. We further want to emphasize that the local optima that can be reached by GradTree have a significant advantage over the local optimum of a greedy approach: While the local optimum of greedy approaches is constrained by sequentially selecting the optimal split at each node, GradTree overcomes this limitation by optimizing all parameters jointly.

Genetic DTs Another way to learn DTs in a non-greedy fashion is by using evolutionary algorithms. Evolutionary algorithms perform a robust global search in the space of candidate solutions based on the concept of survival of the fittest (Barros et al. 2011). This usually results in smaller trees and a better identification of feature interactions compared to a greedy, local search (Freitas 2002).

Oblique DTs In contrast to vanilla DTs that make a hard decision at each internal node, many hierarchical mixture of expert models (Jordan and Jacobs 1994) have been proposed. They usually make soft splits, where each branch is associated with a probability (Irsoy, Yıldız, and Alpaydın 2012; Frosst and Hinton 2017). Further, the models do not comprise univariate, axis-aligned splits, but are oblique with respect to the axes. These adjustments to the tree architecture allow for the application of further optimization algorithms, including gradient descent. Blanquero et al. (2020) aim to increase the interpretability of oblique trees by optimizing for sparsity, using fewer variables at each split and simultaneously fewer splits in the whole tree. Tanno et al. (2019) combine the benefits of neural networks and DTs, using so-called adaptive neural trees (ANTs). They employ a stochastic routing based on a Bernoulli distribution and utilize non-linear transformer modules at the edges, making the resulting trees soft and oblique. Xu et al. (2022) propose One-Stage Tree as a novel method for learning soft DTs, including the tree structure, while maintaining discretization during training, which results in a higher interpretability compared to existing soft DTs. However, in contrast to GradTree, the routing is instance-wise, which significantly hampers a global interpretation of the model. Norouzi et al. (2015) proposed an approach to overcome the need for soft decisions to apply gradient-based algorithms by minimizing a convex-concave upper bound on the tree's empirical loss. While this allows the use of hard splits, the approach is still limited to oblique trees. Zantedeschi, Kusner, and Niculae (2021) use argmin differentiation to simultaneously learn all tree parameters by relaxing a mixed-integer program for discrete parameters to allow for gradient-based optimization. This allows hard splits, but in contrast to GradTree, they still require a differentiable split function (e.g., a linear function which results in oblique trees). Similarly, Karthikeyan et al. (2022) developed a gradient-based approach to learn hard DTs. Like to GradTree, they use an ST operator to handle the hard step functions. While their formulation is limited to oblique trees, our approach permits axis-aligned DTs.

In summary, unlike oblique DTs, GradTree allows hard, axis-aligned splits that consider only a single feature at each split, providing significantly higher interpretability, especially at the split-level. This is supported by Molnar (2020) where the authors argue that humans cannot comprehend explanations involving more than three dimensions at once.

Oblivious DT Ensembles Popov, Morozov, and Babenko (2019) proposed an oblivious tree ensemble for deep learning. Oblivious DTs use the same splitting feature and thresh-

old in all internal nodes of the same depth, making them only suitable as weak learners in an ensemble. They use an entmax transformation of the choice function and a two-class entmax as split function. This results in oblique and soft trees, while GradTree is axis-aligned and hard. Chang, Caruana, and Goldenberg (2021) proposed a temperature annealing procedure to gradually turn the input to an entmax function one-hot, which can enforce axis-aligned trees. In contrast, our approach employs an ST operator immediately following an entmax transformation to yield a one-hot encoded vector. Our experiments substantiate that our method achieves superior results in the context of individual DTs.

Deep Neural Decision Trees (DNDTs) Yang, Morillo, and Hospedales (2018) propose DNDTs that realize tree models as neural networks, utilizing a soft binning function for splitting. Therefore, the resulting trees are soft, but axis-aligned, which makes this work closely related to our approach. Since DNDTs are generated via the Kronecker product of the binning layers, the structure depends on the number of features and classes (and the number of bins). As discussed by the authors, this results in poor scalability w.r.t. the number of features, which currently can only be solved by using random forests for high-dimensional datasets (> 12 features). Our approach, in contrast, scales linearly with the number of features, making it efficient for high-dimensional datasets. Furthermore, using the Kronecker product to build the tree prevents splitting on the same feature with different thresholds in the same path, which can be crucial to achieve a good performance. For GradTree, both the split threshold and the split index are learned parameters, inherently allowing the model to split on the same feature multiple times.

3 GradTree: Gradient-Based Decision Trees

In this section, we introduce GradTree. We present a new DT representation and a novel algorithm that allows learning hard, axis-aligned DTs with gradient descent. More specifically, we will use backpropagation with a straight-through (ST) operator (Section 3.3) on a dense DT representation (Section 3.2) to adjust the model parameters during the training. Furthermore, our novel tree routing (Section 3.4) allows an efficient optimization of all parameters over an entire batch with a single set of matrix operations.

3.1 Arithmetic Decision Tree Formulation

Here, we introduce a notation for DTs with respect to their parameters. We formulate DTs as an arithmetic function based on addition and multiplication, rather than as a nested concatenation of rules, which is necessary for a gradient-based learning. Note that our notation and training procedure assume fully-grown (i.e. complete, full) DTs. After training, we apply a basic post-hoc pruning to reduce the tree size for application. Our formulation aligns with Kontschieder et al. (2015). However, they only consider stochastic routing and oblique trees, whereas our formulation emphasizes deterministic routing and axis-aligned trees.

For a DT of depth d, the parameters include one split threshold and one feature index for each internal node, represented as vectors $\boldsymbol{\tau} \in \mathbb{R}^{2^d-1}$ and $\boldsymbol{\iota} \in \mathbb{N}^{2^d-1}$ respectively, where $2^d - 1$ equals the number of internal nodes in a fully-grown DT. Additionally, each leaf node comprises a class membership, in the case of a classification task, which we denote as the vector $\boldsymbol{\lambda} \in \mathcal{C}^{2^d}$, where $\mathcal{C}$ is the set of classes and 2^d equals the number of leaf nodes in a fully-grown DT.

Formally, a DT can be expressed as a function $DT(\cdot|\boldsymbol{\tau}, \boldsymbol{\iota}, \boldsymbol{\lambda}) : \mathbb{R}^n \to \mathcal{C}$ with respect to its parameters:

$$DT(\boldsymbol{x}|\boldsymbol{\tau}, \boldsymbol{\iota}, \boldsymbol{\lambda}) = \sum_{l=0}^{2^d-1} \lambda_l \, \mathbb{L}(\boldsymbol{x}|l, \boldsymbol{\tau}, \boldsymbol{\iota}) \tag{1}$$

The function $\mathbb{L}$ indicates whether a sample $\boldsymbol{x} \in \mathbb{R}^n$ belongs to a leaf l, and can be defined as a multiplication of the split functions of the preceding internal nodes. We define the split function $\mathbb{S}$ as a Heaviside step function

$$\mathbb{S}_{\text{Heaviside}}(\boldsymbol{x}|\iota, \tau) = \begin{cases} 1, \text{if } x_\iota \geq \tau \\ 0, \text{otherwise} \end{cases} \tag{2}$$

where ι is the index of the feature considered at a certain split and τ is the corresponding threshold.

By enumerating the internal nodes of a fully-grown tree with depth d in a breadth-first order, we can now define the indicator function $\mathbb{L}$ for a leaf l as

$$\begin{aligned} \mathbb{L}(\boldsymbol{x}|l, \boldsymbol{\tau}, \boldsymbol{\iota}) = \prod_{j=1}^{d} & \left(1 - \mathfrak{p}(l,j)\right) \, \mathbb{S}(\boldsymbol{x}|\tau_{\mathfrak{i}(l,j)}, \iota_{\mathfrak{i}(l,j)}) \\ & + \mathfrak{p}(l,j) \, \left(1 - \mathbb{S}(\boldsymbol{x}|\tau_{\mathfrak{i}(l,j)}, \iota_{\mathfrak{i}(l,j)})\right) \end{aligned} \tag{3}$$

Here, $\mathfrak{i}$ is the index of the internal node preceding a leaf node l at a certain depth j and can be calculated as

$$\mathfrak{i}(l,j) = 2^{j-1} + \left\lfloor \frac{l}{2^{d-(j-1)}} \right\rfloor - 1 \tag{4}$$

Additionally, $\mathfrak{p}$ indicates whether the left ($\mathfrak{p} = 0$) or the right branch ($\mathfrak{p} = 1$) was taken at the internal node preceding a leaf node l at a certain depth j. We can calculate $\mathfrak{p}$ as

$$\mathfrak{p}(l,j) = \left\lfloor \frac{l}{2^{d-j}} \right\rfloor \bmod 2 \tag{5}$$

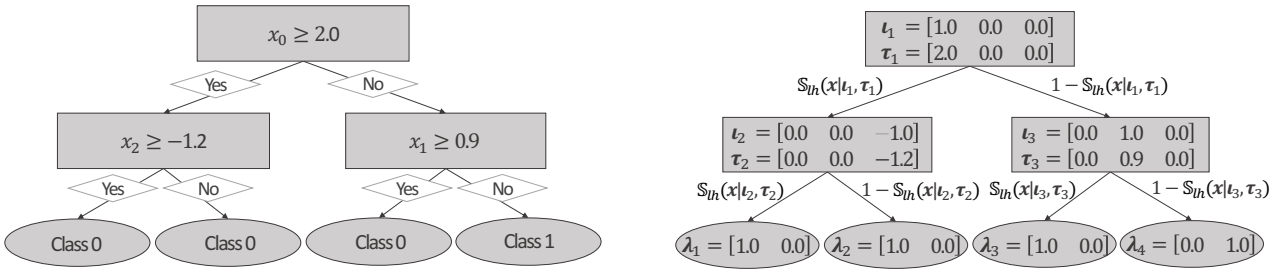
As becomes evident, DTs involve non-differentiable operations in terms of the split function, including the split feature selection (Equation 2). This precludes the application of backpropagation for learning the parameters. Specifically, to efficiently learn a DT using backpropagation, we must address three challenges:

C1 The index ι for the split feature selection is defined as $\iota \in \mathbb{N}$. However, the index ι is a parameter of the DT and a gradient-based optimization requires $\iota \in \mathbb{R}$.

C2 The split function $\mathbb{S}(\boldsymbol{x}|\iota, \tau)$ is a Heaviside step function with an undefined gradient for $x_\iota = \tau$ and 0 gradient elsewhere, which precludes an efficient optimization.

C2 Leafs in a vanilla DT comprise a class membership $\lambda \in \mathcal{C}$. To calculate an informative loss and optimize the leaf parameters with gradient descent, we need $\lambda \in \mathbb{R}$.

Additionally, the computation of the internal node index $\mathfrak{i}$ and path position $\mathfrak{p}$ involves non-differentiable operations. However, given our focus on fully-grown trees, these values remain constant, allowing for their computation prior to the optimization process.



(a) Vanilla DT Representation

(b) Dense DT Representation

Figure 2: Standard vs. Dense DT Representation. Comparison of a standard and the equivalent dense representation for an exemplary DT with depth 2 and a dataset with 3 variables and 2 classes. Here, $\mathbb{S}_{\text{lh}}$ stands for $\mathbb{S}_{\text{logistic\_hard}}$ (Equation 7).

3.2 Dense Decision Tree Representation

In this subsection, we present a differentiable representation of the feature indices $\boldsymbol{\iota}$ to facilitate gradient-based optimization, which is illustrated in Figure 2.

To this end, we expand the vector $\boldsymbol{\iota} \in \mathbb{R}^{2^d-1}$ to a matrix $I \in \mathbb{R}^{2^d-1} \times \mathbb{R}^n$. This is achieved by one-hot encoding the feature index as $\boldsymbol{\iota} \in \mathbb{R}^n$ for each internal node. This adjustment is necessary for the optimization process to account for the fact that feature indices are categorical instead of ordinal. Although our matrix representation for feature selection has parallels with that proposed by Popov, Morozov, and Babenko (2019), we introduce a novel aspect: A matrix representation for split thresholds. We denote this representation as $T \in \mathbb{R}^{2^d-1} \times \mathbb{R}^n$. Instead of representing a single value for all features, we store individual values for each feature, denoted as $\boldsymbol{\tau} \in \mathbb{R}^n$. This modification is tailored to support the optimization process, particularly in recognizing that split thresholds are feature-specific and non-interchangeable. In essence, a viable split threshold for one feature may not be suitable for another. This adjustment acts as a memory mechanism, ensuring that a given split threshold is exclusively associated with the corresponding feature. Consequently, this refinement enhances the exploration of feature selection at every split during the training.

Besides the previously mentioned advantages, using a dense DT representation allows the use of matrix multiplications for an efficient computation. Accordingly, we can reformulate the Heaviside split function (Equation 2) as

$$\mathbb{S}_{\text{logistic}}(\boldsymbol{x}|\boldsymbol{\iota}, \boldsymbol{\tau}) = S\left(\sum_{i=0}^{n} \iota_i x_i - \sum_{i=0}^{n} \iota_i \tau_i\right) \tag{6}$$

$$\mathbb{S}_{\text{logistic\_hard}}(\boldsymbol{x}|\boldsymbol{\iota}, \boldsymbol{\tau}) = \lfloor \mathbb{S}_{\text{logistic}}(\boldsymbol{x}|\boldsymbol{\iota}, \boldsymbol{\tau}) \rceil \tag{7}$$

where $S(x) = \frac{1}{1+e^{-x}}$ denotes the logistic function and $\lfloor \cdot \rceil$ represents for rounding to the nearest integer. In our context, with $\boldsymbol{\iota}$ being one-hot encoded, $\mathbb{S}_{\text{logistic\_hard}}(\boldsymbol{x}|\boldsymbol{\iota}, \boldsymbol{\tau}) = \mathbb{S}_{\text{Heaviside}}(\boldsymbol{x}|\iota, \tau)$ holds.

3.3 Backpropagation of Decision Tree Loss

While the dense representation introduced previously emphasizes an efficient learning of axis-aligned DTs, it does not solve **C1**-**C3**. In this subsection, we will address those challenges by using the ST operator for backpropagation.

For the function value calculation in the forward pass, we need to assure that $\boldsymbol{\iota}$ is a one-hot encoded vector. This can be achieved by applying a hardmax function on the feature index vector for each internal node. However, applying a hardmax is a non-differentiable operation, which precludes gradient computation. To overcome this issue, we use the ST operator (Bengio, Léonard, and Courville 2013): For the forward pass, we apply the hardmax as is. For the backward pass, we exclude this operation and directly propagate back the gradients of $\boldsymbol{\iota}$. Accordingly, we can optimize the parameters of $\boldsymbol{\iota}$ where $\boldsymbol{\iota} \in \mathbb{R}$ while still using axis-aligned splits during training (**C1**). However, this procedure introduces a mismatch between the forward and backward pass. To reduce this mismatch, we additionally perform an entmax transformation (Peters, Niculae, and Martins 2019) to generate a sparse distribution over $\boldsymbol{\iota}$ before applying the hardmax.

Similarly, we employ the ST operator to ensure hard splits (Equation 7) by excluding $\lfloor \cdot \rceil$ for the backward pass (**C2**). Using the sigmoid logistic function before applying the ST operator (see Equation 6) utilizes the distance to the split threshold as additional information for the gradient calculation. If the feature considered at an internal node is close to the split threshold for a specific sample, this will result in smaller gradients compared to a sample that is more distant.

Furthermore, we need to adjust the leaf nodes of the DT to allow an efficient loss calculation (**C3**). Vanilla DTs contain the predicted class for each leaf node and are functions $DT : \mathbb{R}^n \to \mathcal{C}$. We use a probability distribution at each leaf node and therefore define DTs as a function $DT : \mathbb{R}^n \to \mathbb{R}^c$ where c is the number of classes. Consequently, the parameters of the leaf nodes are defined as $L \in \mathbb{R}^{2^n} \times \mathbb{R}^c$ for the whole tree and $\boldsymbol{\lambda} \in \mathbb{R}^c$ for a specific leaf node. This adjustment allows the application of standard loss functions.

3.4 Deterministic Tree Routing and Training

In the previous subsections, we introduced the adjustments that are necessary to apply gradient descent to DTs. During the optimization, we calculate the gradients with backpropagation based on the computation graph of the tree pass function. The tree pass function to calculate the function values is summarized in Algorithm 1 and utilizes the adjust-

Algorithm 1: Tree Pass Function

function PASS($I, T, L, \boldsymbol{x}$)
  $I \leftarrow \text{entmax}(I)$
  $I \leftarrow I - c_1^*$ where $c_1^* = I - \text{hardmax}(I)$ ▷ ST operator
  $\hat{\boldsymbol{y}} \leftarrow [0]^c$
  for $l = 0, \ldots, 2^d - 1$ **do**
    $p \leftarrow 1$
    for $j = 1, \ldots, d$ **do**
      $\mathfrak{i} \leftarrow 2^{j-1} + \left\lfloor \frac{l}{2^{d-(j-1)}} \right\rfloor - 1$ ▷ Equation 4
      $\mathfrak{p} \leftarrow \left\lfloor \frac{l}{2^{d-j}} \right\rfloor \bmod 2$ ▷ Equation 5
      $s \leftarrow S\left(\sum_{i=0}^{n} T_{\mathfrak{i},i}\, I_{\mathfrak{i},i} - \sum_{i=0}^{n} x_i\, I_{\mathfrak{i},i}\right)$ ▷ Equation 6
      $s \leftarrow s - c_2^*$ where $c_2^* = s - \lfloor s \rceil$ ▷ ST operator
      $p \leftarrow p\,((1-\mathfrak{p})\,s + \mathfrak{p}\,(1-s))$ ▷ Equation 3
    end for
    $\hat{\boldsymbol{y}} \leftarrow \hat{\boldsymbol{y}} + L_l\, p$ ▷ Equation 1
  end for
  return $\sigma(\hat{\boldsymbol{y}})$ ▷ Softmax σ to get probability distribution
end function

ments introduced in the previous sections. Our tree routing facilitates the computation of the tree pass function over a complete batch as a single set of matrix operations, which allows an efficient computation. We also want to note that our dense representation can be converted into an equivalent vanilla DT representation at each point in time. Similarly, the fully-grown nature of GradTree is only required during the gradient-based optimization and standard pruning techniques to reduce the tree size are applied post-hoc.

Furthermore, our implementation optimizes the gradient descent algorithm by leveraging common stochastic gradient descent techniques, including mini-batch calculation and momentum using the Adam optimizer (Kingma and Ba 2014) with weight averaging (Izmailov et al. 2018). Moreover, we implement early stopping and random restarts to avoid bad initial parametrizations, where the best parameters are selected based on the validation loss. Further details can be found in the supplementary material.

4 Experimental Evaluation

The following experiments aim to evaluate the predictive capability of GradTree against existing methods.

4.1 Experimental Setup

Datasets and Preprocessing The experiments were conducted on several benchmark datasets, mainly from the UCI repository (Dua and Graff 2017). For all datasets, we performed a standard preprocessing: Similar to Popov, Morozov, and Babenko (2019), we applied leave-one-out encoding to all categorical features and further performed a quantile transform, making each feature follow a normal distribution. We used a $80\%/20\%$ train-test split for all datasets. To account for class imbalance, we rebalanced datasets using SMOTE (Chawla et al. 2002) if the minority class accounts for less than $\frac{25}{c-1}\%$ of the data. For GradTree and DNDT, we used 20% of the training data as validation data for early stopping. As DL8.5 requires binary features, we discretized numeric features using quantile binning with 5 bins and one-hot encoded categorical features. Details and sources of the datasets are available in the supplementary material.

Methods We compared GradTree[1] to the most prominent approach from each category (see Section 2) to ensure a concise, yet holistic evaluation focusing on hard, axis-aligned DTs. Specifically, we selected the following methods:

- **CART**: We use the sklearn (Pedregosa et al. 2011) implementation, which uses an optimized version of the CART algorithm. CART typically employs the Gini impurity measure, but we additionally allowed entropy.
- **Evolutionary DTs**: We use GeneticTree (Pysiak 2021) for an efficient learning of DTs with a genetic algorithm.
- **DNDT**: We use the official DNDT implementation (Yang, Morillo, and Hospedales 2022). For a fair comparison, we enforce binary trees by setting the number of cut points to 1 and ensure hard splits during inference. As suggested by Yang, Morillo, and Hospedales (2018), we limited DNDTs to datasets with no more than 12 features, due to scalability issues.
- **DL8.5 (Optimal DTs)**: We use the official DL8.5 implementation (Aglin, Nijssen, and Schaus 2022) including improvements from MurTree (Demirović et al. 2022) which reduces the runtime significantly.

To ensure a fair comparison, we further applied a simple post-hoc pruning for GradTree to remove all branches with zero samples based on one pass of the training data. Similar to DNDT, we used a cross-entropy loss.

Hyperparameters We conducted a random search with cross-validation to determine the optimal hyperparameters. The complete list of relevant hyperparameters for each approach along with additional details on the selection are in the supplementary material.

4.2 Results

GradTree outperforms existing DT learners for binary classification First, we evaluated the performance of GradTree against existing approaches on the benchmark datasets in terms of the macro F1-Score, which inherently considers class imbalance. We report the relative difference to the best model (MRD) and mean reciprocal rank (MRR), following the approach of Yang, Morillo, and Hospedales (2018). Overall, GradTree outperformed existing approaches for binary classification tasks (best MRR of 0.758 and MRD of 0.008) and achieved competitive results for multi-class tasks (second-best MRR of 0.619 and MRD of 0.069). More specifically, GradTree significantly outperformed state-of-the-art non-greedy DT methods, including DNDTs as our gradient-based benchmark.

For binary classification (Table 1), GradTree demonstrated superior performance over CART, achieving the best performance on 13 datasets as compared to only 6 datasets for CART. Notably, the performance difference between GradTree and existing methods was substantial for several datasets, such as *Echocardiogram*,

[1] Available under: https://github.com/s-marton/GradTree

	Gradient-Based		Non-Greedy		Greedy
	GradTree (ours)	DNDT	GeneticTree	DL8.5 (Optimal)	CART
Blood Transfusion	**0.628 ± .036 (1)**	0.543 ± .051 (5)	0.575 ± .094 (4)	0.590 ± .034 (3)	0.613 ± .044 (2)
Banknote Authentication	**0.987 ± .007 (1)**	0.888 ± .013 (5)	0.922 ± .021 (4)	0.962 ± .011 (3)	0.982 ± .007 (2)
Titanic	**0.776 ± .025 (1)**	0.726 ± .049 (5)	0.730 ± .074 (4)	0.754 ± .031 (2)	0.738 ± .057 (3)
Raisins	0.840 ± .022 (4)	0.821 ± .033 (5)	**0.857 ± .021 (1)**	0.849 ± .027 (3)	0.852 ± .017 (2)
Rice	0.926 ± .007 (3)	0.919 ± .012 (5)	0.927 ± .005 (2)	0.925 ± .008 (4)	**0.927 ± .006 (1)**
Echocardiogram	**0.658 ± .113 (1)**	0.622 ± .114 (3)	0.628 ± .105 (2)	0.609 ± .112 (4)	0.555 ± .111 (5)
Wisconcin Breast Cancer	0.904 ± .022 (2)	**0.913 ± .032 (1)**	0.892 ± .028 (4)	0.896 ± .021 (3)	0.886 ± .025 (5)
Loan House	**0.714 ± .041 (1)**	0.694 ± .036 (2)	0.451 ± .086 (5)	0.607 ± .045 (4)	0.662 ± .034 (3)
Heart Failure	0.750 ± .070 (3)	0.754 ± .062 (2)	0.748 ± .068 (4)	0.692 ± .062 (5)	**0.775 ± .054 (1)**
Heart Disease	**0.779 ± .047 (1)**	$n > 12$	0.704 ± .059 (4)	0.722 ± .065 (2)	0.715 ± .062 (3)
Adult	0.743 ± .034 (2)	$n > 12$	0.464 ± .055 (4)	0.723 ± .011 (3)	**0.771 ± .011 (1)**
Bank Marketing	**0.640 ± .027 (1)**	$n > 12$	0.473 ± .002 (4)	0.502 ± .011 (3)	0.608 ± .018 (2)
Congressional Voting	**0.950 ± .021 (1)**	$n > 12$	0.942 ± .021 (2)	0.924 ± .043 (4)	0.933 ± .032 (3)
Absenteeism	**0.626 ± .047 (1)**	$n > 12$	0.432 ± .073 (4)	0.587 ± .047 (2)	0.564 ± .042 (3)
Hepatitis	0.608 ± .078 (2)	$n > 12$	0.446 ± .024 (4)	0.586 ± .083 (3)	**0.622 ± .078 (1)**
German	**0.592 ± .068 (1)**	$n > 12$	0.412 ± .006 (4)	0.556 ± .035 (3)	0.589 ± .065 (2)
Mushroom	**1.000 ± .001 (1)**	$n > 12$	0.984 ± .003 (4)	0.999 ± .001 (2)	0.999 ± .001 (3)
Credit Card	0.674 ± .014 (4)	$n > 12$	**0.685 ± .004 (1)**	0.679 ± .007 (3)	0.683 ± .010 (2)
Horse Colic	**0.842 ± .039 (1)**	$n > 12$	0.496 ± .169 (4)	0.708 ± .038 (3)	0.786 ± .062 (2)
Thyroid	0.905 ± .010 (2)	$n > 12$	0.605 ± .116 (4)	0.682 ± .018 (3)	**0.922 ± .011 (1)**
Cervical Cancer	**0.521 ± .043 (1)**	$n > 12$	0.514 ± .034 (2)	0.488 ± .027 (4)	0.506 ± .034 (3)
Spambase	0.903 ± .025 (2)	$n > 12$	0.863 ± .019 (3)	0.863 ± .011 (4)	**0.917 ± .011 (1)**
Mean Reciprocal Rank (MRR) ↑	**0.758 ± .306 (1)**	0.370 ± .268 (3)	0.365 ± .228 (4)	0.335 ± .090 (5)	0.556 ± .293 (2)
Mean Relative Diff. (MRD) ↓	**0.008 ± .012 (1)**	0.056 ± .051 (3)	0.211 ± .246 (5)	0.084 ± .090 (4)	0.035 ± .048 (2)

Table 1: Binary Classification Performance. We report macro F1-scores (mean ± stdev over 10 trials) on test data with optimized hyperparameters. The rank of each method is presented in brackets. The datasets are sorted by the number of features.

Heart Disease and *Absenteeism*. For multi-class datasets, GradTree achieved the second-best overall performance. While GradTree still achieved a superior performance for low-dimensional datasets (top part of Table 2), CART achieved the best results for high-dimensional datasets with a high number of classes. We can explain this by the dense representation used for the gradient-based optimization. Using our representation, the difficulty of the optimization task increases with the number of features (more parameters at each internal node) and the number of classes (more parameters at each leaf node). In future work, we aim to optimize our dense representation, e.g., by using parameter sharing.

GradTree has a small effective tree size The effective tree size (= size after pruning) of GradTree is smaller than CART for binary and marginally higher for multi-class tasks (Table 3). Only the tree size for GeneticTree is significantly smaller, which is caused by the complexity penalty of the genetic algorithm. DL8.5 also has a smaller average tree size than CART and GradTree. We can attribute this to DL8.5 being only feasible up to a depth of 4 due to the high computational complexity. The tree size for DNDTs scales with the number of features (and classes) which quickly results in large trees. Furthermore, pruning DNDTs is non-trivial due to the use of the Kronecker product (it is not sufficient to prune subtrees bottom-up).

GradTree is robust to overfitting We can observe that gradient-based approaches were more robust and less prone to overfitting compared to a greedy optimization with CART and alternative non-greedy methods. We measure overfitting by the difference between the mean train and test performance (see Table 3). For binary tasks, GradTree exhibits a train-test performance difference of 0.051, considerably smaller than that of CART (0.183), GeneticTree (0.204), and DL8.5 (0.202). DNDTs, which are also gradient-based, achieved an even smaller difference of 0.039. For multi-class tasks, the difference was significantly smaller for GradTree compared to any other approach.

GradTree does not rely on extensive hyperparameter optimization Besides their interpretability, a distinct advantage of DTs over more sophisticated models is that they typically do not rely on an extensive hyperparameter optimization. In this experiment, we show that the same is true for GradTree by evaluating the performance with default configurations (see Table 3). When using the default parameters, GradTree still outperformed the other methods on binary tasks (highest MRR and most wins) and achieved competitive results for multi-class tasks (second-highest MRR).

GradTree is efficient for large and high-dimensional datasets For each dataset, a greedy optimization using CART was substantially faster than other methods, taking less than a second. Nevertheless, for most datasets, training GradTree took less than 30 seconds (mean runtime of 35 seconds). DNDT had comparable runtimes to GradTree. For most datasets, DL8.5 had a low runtime of less than 10 seconds. However, scalability issues become apparent with DL8.5, especially with an increasing number of features and

	Gradient-Based		Non-Greedy		Greedy
	GradTree (ours)	DNDT	GeneticTree	DL8.5 (Optimal)	CART
Iris	**0.938 ± .057 (1)**	0.870 ± .063 (5)	0.912 ± .055 (3)	0.909 ± .046 (4)	0.937 ± .046 (2)
Balance Scale	**0.593 ± .045 (1)**	0.475 ± .104 (5)	0.529 ± .043 (3)	0.525 ± .039 (4)	0.574 ± .030 (2)
Car	0.440 ± .085 (3)	0.485 ± .064 (2)	0.306 ± .068 (4)	0.273 ± .063 (5)	**0.489 ± .094 (1)**
Glass	0.560 ± .090 (3)	0.434 ± .072 (5)	0.586 ± .090 (2)	0.501 ± .100 (4)	**0.663 ± .086 (1)**
Contraceptive	**0.496 ± .050 (1)**	0.364 ± .050 (3)	0.290 ± .048 (5)	0.292 ± .036 (4)	0.384 ± .075 (2)
Solar Flare	0.151 ± .033 (3)	**0.171 ± .051 (1)**	0.146 ± .018 (4)	0.144 ± .034 (5)	0.157 ± .022 (2)
Wine	**0.933 ± .031 (1)**	0.858 ± .041 (4)	0.888 ± .039 (3)	0.852 ± .022 (5)	0.907 ± .042 (2)
Zoo	0.874 ± .111 (3)	$n > 12$	0.782 ± .111 (4)	0.911 ± .106 (2)	**0.943 ± .076 (1)**
Lymphography	**0.610 ± .191 (1)**	$n > 12$	0.381 ± .124 (4)	0.574 ± .196 (2)	0.548 ± .154 (3)
Segment	0.941 ± .009 (2)	$n > 12$	0.715 ± .114 (4)	0.808 ± .013 (3)	**0.963 ± .010 (1)**
Dermatology	0.930 ± .030 (2)	$n > 12$	0.785 ± .126 (4)	0.885 ± .036 (3)	**0.957 ± .026 (1)**
Landsat	0.807 ± .011 (2)	$n > 12$	0.628 ± .084 (4)	0.783 ± .008 (3)	**0.835 ± .011 (1)**
Annealing	0.638 ± .126 (3)	$n > 12$	0.218 ± .053 (4)	0.787 ± .121 (2)	**0.866 ± .094 (1)**
Splice	0.873 ± .030 (2)	$n > 12$	0.486 ± .157 (3)	> 60 min	**0.881 ± .021 (1)**
Mean Reciprocal Rank (MRR) ↑	0.619 ± .303 (2)	0.383 ± .293 (3)	0.288 ± .075 (5)	0.315 ± .116 (4)	**0.774 ± .274 (1)**
Mean Relative Diff. (MRD) ↓	0.069 ± .102 (2)	0.188 ± .200 (3)	0.521 ± .749 (5)	0.215 ± .249 (4)	**0.040 ± .081 (1)**

Table 2: Multi-Class Classification Performance. We report macro F1-scores (mean ± stdev over 10 trials) on test data with optimized hyperparameters. The rank of each method is presented in brackets. The datasets are sorted by the number of features.

	Tree Size		Train-Test Difference		Default Setting (MRR ↑)	
	Binary	Multi	Binary	Multi	Binary	Multi
GradTree	54	86	0.051	0.174	**0.670**	0.470
DNDT	887	907	0.039	0.239	0.306	0.295
GeneticTree	7	20	0.204	0.258	0.427	0.315
DL8.5	28	29	0.202	0.260	0.371	0.331
CART	67	76	0.183	0.247	0.571	**0.929**

Table 3: Summarized Results. Left: Average tree size. Mid: Mean difference between train and test performance as overfitting indicator. Right: Test performance with default parameters. Detailed results are in the supplementary material.

		ST Entmax (ours)	ST Gumbel	Temp. Annealing
Default	Binary	**0.764**	0.560	0.757
	Multi	**0.638**	0.272	0.602
Optimized	Binary	**0.771**	0.569	0.759
	Multi	**0.699**	0.297	0.601

Table 4: Ablation Study. We compare our approach to deal with the non-differentiable nature of DTs with alternative methods, reporting the average macro F1-scores over 10 trials with optimized and default hyperparameters. The complete results are listed in the supplementary material.

samples. Its runtime surpassed GradTree on various datasets, taking around 300 seconds for *Credit Card* and exceeding 830 seconds for *Landsat*. For *Splice*, DL8.5 did not find a solution within 60 minutes and the optimization was terminated. Detailed runtimes are in the supplementary material.

ST entmax outperforms alternative methods In an ablation study (Table 4), we evaluated our design choice of utilizing an ST operator directly after an entmax transformation to address the non-differentiability of DTs. We contrasted this against alternative strategies found in the literature. Our approach notably surpassed ST Gumbel Softmax (Jang, Gu, and Poole 2016) and outperformed the temperature annealing technique proposed by Chang, Caruana, and Goldenberg (2021) to gradually turn the entmax one-hot.

5 Conclusion and Future Work

In this paper, we proposed GradTree, a novel method for learning hard, axis-aligned DTs based on a joint optimization of all tree parameters with gradient descent. Our empirical evaluations indicate that GradTree excels over existing methods in binary tasks and offers competitive performance in multi-class tasks. The substantial performance increase achieved by GradTree across multiple datasets highlights its importance as a noteworthy contribution to the existing repertoire of DT learning methods.

Moreover, gradient-based optimization provides greater flexibility, allowing for easy integration of custom loss functions tailored to specific application scenarios. Another advantage is the ability to relearn the threshold value as well as the split index. Therefore, GradTree is suitable for dynamic environments, such as online learning tasks.

Currently, GradTree employs conventional post-hoc pruning. In future work, we want to consider pruning already during the training, for instance through a learnable choice parameter to decide if a node is pruned, similar to Zantedeschi, Kusner, and Niculae (2021). Although our focus was on stand-alone DTs aiming for intrinsic interpretability, GradTree holds potential as a foundational method for learning hard, axis-aligned tree ensembles end-to-end via gradient descent. Exploring this performance-interpretability trade-off is an interesting direction for future research.

Acknowledgments

This research was supported in part by the German Federal Ministry for Economic Affairs and Climate Action of Germany (BMWK), and in part by the German Federal Ministry of Education and Research (BMBF).

References

Aglin, G.; Nijssen, S.; and Schaus, P. 2020. Learning optimal decision trees using caching branch-and-bound search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 3146–3153.

Aglin, G.; Nijssen, S.; and Schaus, P. 2022. PyDL8.5. https://github.com/aia-uclouvain/pydl8.5. Accessed 13.11.2022.

Barros, R. C.; Basgalupp, M. P.; De Carvalho, A. C.; and Freitas, A. A. 2011. A survey of evolutionary algorithms for decision-tree induction. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(3): 291–312.

Bengio, Y.; Léonard, N.; and Courville, A. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*.

Bertsimas, D.; and Dunn, J. 2017. Optimal classification trees. *Machine Learning*, 106(7): 1039–1082.

Blanquero, R.; Carrizosa, E.; Molero-Río, C.; and Morales, D. R. 2020. Sparsity in optimal randomized classification trees. *European Journal of Operational Research*, 284(1): 255–272.

Breiman, L.; Friedman, J. H.; Olshen, R. A.; and Stone, C. J. 1984. *Classification and Regression Trees*. Wadsworth. ISBN 0-534-98053-8.

Chang, C.-H.; Caruana, R.; and Goldenberg, A. 2021. Nodegam: Neural generalized additive model for interpretable deep learning. *arXiv preprint arXiv:2106.01613*.

Chawla, N. V.; Bowyer, K. W.; Hall, L. O.; and Kegelmeyer, W. P. 2002. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16: 321–357.

Demirović, E.; Lukina, A.; Hebrard, E.; Chan, J.; Bailey, J.; Leckie, C.; Ramamohanarao, K.; and Stuckey, P. J. 2022. MurTree: Optimal Decision Trees via Dynamic Programming and Search. *Journal of Machine Learning Research*, 23(26): 1–47.

Dua, D.; and Graff, C. 2017. UCI Machine Learning Repository.

Freitas, A. A. 2002. *Data mining and knowledge discovery with evolutionary algorithms*. Springer Science & Business Media.

Frosst, N.; and Hinton, G. 2017. Distilling a neural network into a soft decision tree. *arXiv preprint arXiv:1711.09784*.

Irsoy, O.; Yıldız, O. T.; and Alpaydın, E. 2012. Soft decision trees. In *Proceedings of the 21st international conference on pattern recognition (ICPR2012)*, 1819–1822. IEEE.

Izmailov, P.; Podoprikhin, D.; Garipov, T.; Vetrov, D.; and Wilson, A. G. 2018. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*.

Jang, E.; Gu, S.; and Poole, B. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

Jordan, M. I.; and Jacobs, R. A. 1994. Hierarchical mixtures of experts and the EM algorithm. *Neural computation*, 6(2): 181–214.

Karthikeyan, A.; Jain, N.; Natarajan, N.; and Jain, P. 2022. Learning Accurate Decision Trees with Bandit Feedback via Quantized Gradient Descent. *Transactions of Machine Learning Research*.

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kontschieder, P.; Fiterau, M.; Criminisi, A.; and Bulo, S. R. 2015. Deep neural decision forests. In *Proceedings of the IEEE international conference on computer vision*, 1467–1475.

Kuhn, M.; Johnson, K.; et al. 2013. *Applied predictive modeling*, volume 26. Springer.

Lin, J.; Zhong, C.; Hu, D.; Rudin, C.; and Seltzer, M. 2020. Generalized and scalable optimal sparse decision trees. In *International Conference on Machine Learning*, 6150–6160. PMLR.

Loh, W.-Y. 2002. Regression tress with unbiased variable selection and interaction detection. *Statistica sinica*, 361–386.

Loh, W.-Y. 2009. Improving the precision of classification trees. *The Annals of Applied Statistics*, 1710–1737.

Mazumder, R.; Meng, X.; and Wang, H. 2022. Quant-BnB: A Scalable Branch-and-Bound Method for Optimal Decision Trees with Continuous Features. In *International Conference on Machine Learning*, 15255–15277. PMLR.

Molnar, C. 2020. *Interpretable machine learning*. Lulu. com.

Norouzi, M.; Collins, M.; Johnson, M. A.; Fleet, D. J.; and Kohli, P. 2015. Efficient non-greedy optimization of decision trees. *Advances in neural information processing systems*, 28.

Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; and Duchesnay, E. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12: 2825–2830.

Peters, B.; Niculae, V.; and Martins, A. F. 2019. Sparse sequence-to-sequence models. *arXiv preprint arXiv:1905.05702*.

Popov, S.; Morozov, S.; and Babenko, A. 2019. Neural oblivious decision ensembles for deep learning on tabular data. *arXiv preprint arXiv:1909.06312*.

Pysiak, K. 2021. GeneticTree. https://github.com/pysiakk/GeneticTree. Accessed 17.08.2022.

Quinlan, J. R. 1993. *C4.5: programs for machine learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. ISBN 1-55860-238-0.

Tanno, R.; Arulkumaran, K.; Alexander, D.; Criminisi, A.; and Nori, A. 2019. Adaptive neural trees. In *International Conference on Machine Learning*, 6166–6175. PMLR.

Xu, Z.; Zhu, G.; Yuan, C.; and Huang, Y. 2022. One-Stage Tree: end-to-end tree builder and pruner. *Machine Learning*, 111(5): 1959–1985.

Yang, Y.; Morillo, I. G.; and Hospedales, T. M. 2018. Deep neural decision trees. *arXiv preprint arXiv:1806.06988*.

Yang, Y.; Morillo, I. G.; and Hospedales, T. M. 2022. Deep Neural Decision Trees. https://github.com/wOOL/DNDT. Accessed 13.11.2022.

Zantedeschi, V.; Kusner, M.; and Niculae, V. 2021. Learning binary decision trees by argmin differentiation. In *International Conference on Machine Learning*, 12298–12309. PMLR.

Zharmagambetov, A.; Hada, S. S.; Gabidolla, M.; and Carreira-Perpinán, M. A. 2021. Non-greedy algorithms for decision tree optimization: An experimental comparison. In *2021 International Joint Conference on Neural Networks (IJCNN)*, 1–8. IEEE.