Machine Learning (2022) 111:1959–1985
https://doi.org/10.1007/s10994-021-06094-4

One-Stage Tree: end-to-end tree builder and pruner

Zhuoer Xu[1] · **Guanghui Zhu**[1] · **Chunfeng Yuan**[1] · **Yihua Huang**[1]

Received: 31 January 2021 / Revised: 24 August 2021 / Accepted: 6 October 2021 /
Published online: 5 November 2021

Abstract
Decision trees have favorable properties, including interpretability, high computational efficiency, and the ability to learn from little training data. Learning a decision tree is known to be NP-complete. The researchers have proposed many greedy algorithms such as CART to learn approximate solutions. Inspired by the current popular neural networks, *soft* trees that support end-to-end training with back-propagation have attracted more and more attention. However, existing *soft* trees either lose the interpretability due to the continuous relaxation or employ the two-stage method of end-to-end building and then pruning. In this paper, we propose One-Stage Tree to build and prune the decision tree jointly through a bilevel optimization problem. Moreover, we leverage the reparameterization trick and proximal iterations to keep the tree discrete during end-to-end training. As a result, One-Stage Tree reduces the performance gap between training and testing and maintains the advantage of interpretability. Extensive experiments demonstrate that the proposed One-Stage Tree outperforms CART and the existing *soft* trees on classification and regression tasks.

Keywords Decision tree · Soft tree · End-to-end learning · Explainable AI

1 Introduction

Compared with currently popular neural networks, decision trees have several favorable properties, such as good interpretability and high computational efficiency. In many practical ML (Machine Learning) applications (Chen & Guestrin, 2016; Ke et al., 2017;

Editors: Annalisa Appice, Sergio Escalera,Jose A. Gamez, Heike Trautmann.

✉ Guanghui Zhu
zgh@nju.edu.cn

Zhuoer Xu
zhuoer.xu@smail.nju.edu.cn

Chunfeng Yuan
cfyuan@nju.edu.cn

Yihua Huang
yhuang@nju.edu.cn

[1] National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China

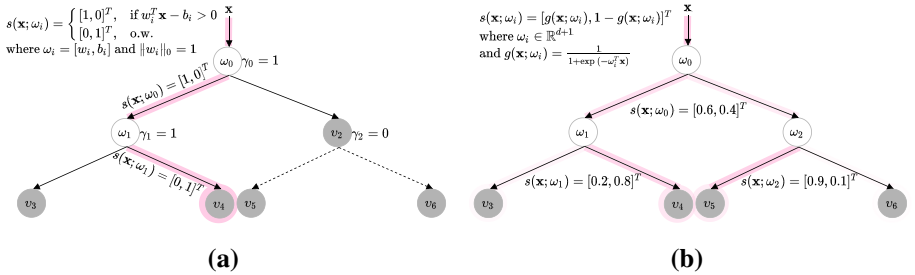


Fig. 1 Overview of a *hard* tree (**a**) and a *soft* tree (**b**). **a** The dashed lines indicate the pruned nodes (i.e., $\gamma_i = 0$). The path where the instance $\boldsymbol{x}$ is routed is shown in purple. As two parts of the internal parameters ω_i, w_i and b_i indicate the feature weight and the feature threshold respectively. **b** For the *soft* tree, the color transparency indicates the path probability

Wu et al., 2020; Li et al., 2020), the decision tree has proven its worth and achieved great success.

A decision tree is a hierarchical structure for the supervised learning task, composed of internal nodes and leaf nodes. The parameters θ of a decision tree can be divided into three parts: (1) internal parameters ω, which decide the direction of each instance $\boldsymbol{x}$ by the router function s; (2) leaf parameters υ, which are the prediction outputs of all leaves $\mathcal{T}$; (3) architecture parameters γ, which define the architecture of the tree. Then, an instance $\boldsymbol{x}$ is recursively directed to the left or right child of the internal node i by the router function $s(\boldsymbol{x};\omega_i)$. When a leaf t is reached (i.e., $\gamma_t = 0$), the leaf parameter υ_t is used as the prediction. Traditionally, splitting rules are used to learn the node (i.e., internal and leaf) parameters (ω, υ), and stopping rules are used to learn the architecture parameters γ.

As shown in Fig. 1, decision trees can be divided into *hard* trees and *soft* trees. For the *hard* tree in Fig. 1a, the response at node i has the following recursive definition:

$$f_i(\boldsymbol{x};\theta) = \begin{cases} \upsilon_i, & \text{if } i \text{ is a leaf (i.e. } \gamma_i = 0) \\ f_{L(i)}(\boldsymbol{x};\theta), & \text{if } s(\boldsymbol{x};\omega_i) = [1,0]^T \\ f_{R(i)}(\boldsymbol{x};\theta), & \text{if } s(\boldsymbol{x};\omega_i) = [0,1]^T \end{cases} \tag{1}$$

where $\gamma_i \in \{0,1\}$, $s : \boldsymbol{x} \to \{[1,0]^T, [0,1]^T\}$ is an *axis-parallel* split with a *one-hot vector* ω_i, and $L(i)/R(i)$ is the left/right child of node i. For example, CART (Breiman et al., 1984) greedily chooses the split feature and threshold by minimizing the *Gini Index* in the current node. Furthermore, top-down pre-pruning and minimal cost-complexity post-pruning (Breiman et al., 1984) are used as the stopping rules to improve the tree's generalization.

The greedy-based splitting and stopping rules in *hard* trees inevitably have the disadvantage of building sub-optimal decision trees. Moreover, despite the low empirical error, decision trees are easily overfitted (Kotsiantis, 2013). To improve learning through end-to-end training with back-propagation, *soft* trees (Norouzi et al., 2015; Irsoy et al., 2014; Hehn et al., 2019) are proposed.

As shown in Fig. 1b, the *soft* tree relaxes the parameters θ to be continuous, e.g., the discrete choices of γ (i.e., whether to prune) and s (i.e., which path to route). Thus, the response at node i can be expressed as follows:

$$f_i(\boldsymbol{x};\theta) = \left(1-\gamma_i\right)\cdot \upsilon_i + \gamma_i \cdot s\left(\boldsymbol{x};\omega_i\right)^T \begin{bmatrix} f_{L(i)}(\boldsymbol{x};\theta) \\ f_{R(i)}(\boldsymbol{x};\theta) \end{bmatrix} \tag{2}$$

where $\gamma_i \in [0,1]$, and $s : \boldsymbol{x} \to [0,1]^2$ is an *oblique* split with $\omega_i \in \mathbb{R}^d$. Due to the continuous characteristics, various techniques (e.g., Gradient Descent (Bottou, 2012; Mukkamala & Hein, 2017) and Regularization (Prechelt, 1998; Bousquet et al., 2004)) in deep learning can be used for end-to-end tree training.

However, designing effective *soft* trees is a challenging task. Soft Decision Tree (Irsoy et al., 2012; Norouzi et al., 2015) only considers the global optimization of node parameters (ω, υ), omitting the architecture parameters γ. Breiman et al. (1984) pointed out that tree quality depends more on good stopping rules than on splitting rules. That is, γ is more crucial in some way. Budding Tree (Irsoy et al., 2014) considers the architecture parameters, but its randomly pruned nodes fail to bud. All of the above methods directly utilize probabilistic trees when testing, which looses the interpretability of decision trees. End2End Tree (Hehn et al., 2019) maintains a probabilistic tree for training and discretizes it to a deterministic one for testing. However, there exists a performance gap after the discretization. Moreover, End2End Tree is a two-stage method that first learns the node parameters end-to-end and then searches γ by the greedy algorithm.

In this work, we propose One-Stage Tree, which is, to our best knowledge, the first *soft* tree that maintains discretization during training. In contrast to the two-stage methods (e.g., CART and End2End Tree) of building and then pruning, we first formalize the joint search for node and architecture parameters as a bilevel optimization problem. Then, we keep the discretization of the path and architecture during training. Specifically, we directly sample leaves by the Gumbel Softmax to predict instances according to the path probability and propose an optimization strategy for discrete γ via proximal iterations. Benefiting from the discretization, we directly find the closed-form optimal solution of υ. Moreover, we reduce the performance gap and maintain interpretability.

Extensive experimental results on both classification and regression tasks reveal the effectiveness of One-Stage Tree. One-Stage Tree has a significant improvement over the most typical CART. Compared with the existing *soft* trees, One-Stage Tree achieves better performance on most datasets. Moreover, One-Stage Tree is competitive with other standard ML methods. The implementation of One-Stage Tree is publicly available on GitHub.[1]

To summarize, our main contributions can be highlighted as follows:

- We introduce One-Stage Tree to search the node and architecture parameters jointly through a bilevel optimization problem.
- The reparameterization trick and proximal iterations are leveraged to keep the tree discrete during training. In this way, we can reduce the performance gap between training and testing and maintain interpretability.
- Extensive experimental results on both classification and regression tasks demonstrate that One-Stage Tree outperforms CART and the existing *soft* trees.

The rest of the paper is structured as follows: after introducing related work in *soft* trees (Sect. 2), we describe our approach in detail (Sect. 3). We then report experimental results on classification and regression datasets (Sect. 4) before concluding the paper (Sect. 5).

[1] https://github.com/Unkrible/One-Stage-Tree.

2 Related work

2.1 Soft tree

The decision tree is among the most popular machine learning algorithms, given its interpretability and simplicity. First, due to the axis-parallel split of each internal node, the decision tree can learn from little training data and is easy to interpret. Then, benefiting from the hierarchical architecture, the decision tree is computationally efficient, with only $\mathcal{O}(\log|\mathcal{I}|)$ nodes needing to be visited out of all $|\mathcal{I}|$ internal nodes for a binary complete tree.

The decision tree structure depends on internal parameters ω and router function $s(\boldsymbol{x};\omega_i)$. The most typical one is the *univariate discrete* tree (Quinlan, 1986, 1996; Breiman et al., 1984), also called *hard* tree, where $\|\omega_i\|_0 = 1$ and $\|s(\boldsymbol{x};\omega_i)\|_0 = 1$. *Hard* Tree selects a subpath for instances according to a specific feature and threshold. In the *multivariate* tree (Irsoy et al., 2012; Norouzi et al., 2015; Irsoy et al., 2014; Hehn et al., 2019), which is also called *soft* tree, ω_i is a continuous variable and $s(\boldsymbol{x};\omega_i)$ defines an oblique split.

Soft Decision Tree (Irsoy et al., 2012) takes the *sigmoid function* as the router function and builds a *multivariate dense* tree whose prediction is contributed by leaves with different probabilities. For Soft Decision Tree, $s(\boldsymbol{x};\omega_i) = [g_i(\boldsymbol{x};\omega_i), 1 - g_i(\boldsymbol{x};\omega_i)]^T$, where $g_i(\boldsymbol{x};\omega_i) = \frac{1}{1+\exp(-\omega_i^T\boldsymbol{x})}$, routes instances to all its children with probabilities. Although it has a smoother fit and lower bias around the split boundaries, all the leaves' paths are traversed. The computational overhead increases from $\mathcal{O}(\log(|I|))$ to $\mathcal{O}(|I|)$, where I denotes the set of internal nodes.

Unlike Soft Decision Tree that only searches the splitting rule, Budding Tree (Irsoy et al., 2014) relaxes γ and fits the tree architecture. The bud node i can be an internal node and a leaf at the same time according to γ_i. By gradient descent, Budding Tree splits and prunes the tree in the learning phase. However, γ_i will never be updated once it equals 0, which is called *dying* γ problem.

Being aware of the benefits of discretization in terms of interpretability, End2End Tree (Hehn et al., 2019) proposes a *multivariate discrete* tree. End2End Tree is fully probabilistic at train time but becomes deterministic at test time after an annealing process. The performance gap between training and testing still exists. Moreover, the tree architecture is still searched greedily, with the risk of a sub-optimal architecture.

In this paper, we propose One-Stage Tree to build and prune the tree jointly. One-Stage Tree directly samples the leaf node as prediction and keeps the discretization of the architecture during training. We leverage the reparameterization trick and proximal iterations to optimize the *multivariate discrete* tree.

2.2 Proximal algorithm

PA (Proximal Algorithm) (Parikh & Boyd, 2014) is a popular optimization technique for handling the following problem:

$$\min_{x\in\mathcal{C}} f(x) + g(x) \tag{3}$$

where f and g are closed proper convex, f is differentiable, and $\mathcal{C}$ is the feasible space. The crux of PA is the standard proximal step:

$$x^{(k+1)} = \text{prox}_{\mathcal{C},\epsilon g}\left(x^{(k)} - \epsilon \nabla f\left(x^{(k)}\right)\right)$$
$$\text{where } \epsilon > 0 \text{ and } \text{prox}_{\mathcal{C},\epsilon g}(x) = \arg\min_{z \in \mathcal{C}} \frac{1}{2}\|z - x\|_2^2 + \epsilon g(z) \tag{4}$$

$\text{prox}_{\mathcal{C},\epsilon g}(\cdot)$ represents the standard proximal operator of g with scale parameter ϵ constrained by $\mathcal{C}$ (Parikh & Boyd, 2014). In this form, we split the objective into two terms, one of which is differentiable. Since g can be extended-valued, it can be used to encode constraints on x.

In machine learning, PA is widely used to solve the continuously differentiable optimization problem with a constraint $\mathcal{C}$ as $\min_x f(x)$ where $x \in \mathcal{C}$. Since $g(x) = 0$, the proximal step is simplified to $x^{(k+1)} = \text{prox}_{\mathcal{C}}(x^{(k)} - \epsilon \nabla f(x^{(k)}))$. Due to its excellent theoretical guarantee and good empirical performance, it has been applied to many deep learning problems (e.g., network binarization (Bai et al., 2018) and recommendation system (Yao et al., 2020)). Another variant of PA with lazy proximal step (Xiao, 2010) maintains two copies of x, i.e.,

$$x^{(k+1)} = x^{(k)} - \epsilon \nabla f(\bar{x}^{(k)}) \text{ where } \bar{x}^{(k)} = \text{prox}_{\mathcal{C}}(x^{(k)}) \tag{5}$$

Although it has no convergence guarantee in the non-convex case, it performs well empirically on deep learning tasks (Courbariaux et al., 2015; Hou et al., 2017).

3 Methodology

In this section, we introduce One-Stage Tree, which is trained with the reparameterization trick and proximal iterations in an end-to-end manner.

3.1 Problem formulation

Consider a supervised task with input space $\mathcal{X} \subset \mathbb{R}^d$ and output space $\mathcal{Y} \subset \mathbb{R}^c$. Let $\mathcal{D}$ be the data distribution. We denote the training set sampled from $\mathcal{D}$ as $\mathcal{D}_{train}$. The training set is also defined as $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\} \subset \mathcal{X}$ with corresponding $\{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N\} \subset \mathcal{Y}$. Let l be a differentiable convex function that measures the difference between the prediction and the target. We denote the overall loss on a given dataset as $\mathcal{L}(\theta)$.

The *soft* tree relaxes the parameters θ to be continuous, recursively calculates the probability $s(\boldsymbol{x};\omega_i)$ from an internal node i to its children, and finally gets the path probability $\mu_t(\boldsymbol{x};\omega,\gamma)$ from the root to each leaf t. Let Path(t) be the node set from the root to leaf t. The path probability $\mu_t(\boldsymbol{x};\omega,\gamma)$ is calculated as follows:

$$\mu_t(\boldsymbol{x};\omega,\gamma) = \prod_{i \in \text{Path}(t)} \gamma_i \cdot s(\boldsymbol{x};\omega_i)_{(1-\mathbb{I}_{L(i) \in \text{Path}(t)})} \tag{6}$$

where $\mathbb{I}$ represents the 0-1 indicator function and $L(i)$ denotes the left child of node i. If $L(i) \in \text{Path}(t)$ is satisfied, the indicator function equals 1. For example, $s(\boldsymbol{x};\omega_i)_0$ indicates the probability of routing the left child. The response of the *soft* tree is the probability-weighted sum of the leaf values.

$$f(\boldsymbol{x};\theta) = \textstyle\sum_{t \in T} v_t \cdot \mu_t(\boldsymbol{x};\omega,\gamma) \tag{7}$$

After the continuous relaxation, the goal is to jointly learn the tree parameters and find the global optima, which can be optimized by gradient descent:

$$\theta^* = \arg\min_{\theta} \mathcal{L}(\theta) = \arg\min_{\theta} \sum_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}} l(f(\boldsymbol{x};\theta),\boldsymbol{y}) \tag{8}$$

3.2 One-Stage Tree

Although the continuous relaxation allows the whole tree to be differentiable, *soft* trees have significant limitations:

1. *Interpretability* Although the oblique split ω_i indicates feature importance, path probabilities at large depths are difficult to interpret.
2. *Performance* The continuous architecture γ needs to be discretized to $\{0, 1\}$ at test time, resulting in inconsistent performance between training and testing.

Recall that in *hard* trees, the trees are all discrete when updating node parameters. Such discretization naturally alleviates the above limitations. Thus, we aim to search the differentiable *soft* tree but keep discrete architectures and paths when updating the parameters. Like decision trees that use the validation set for pruning, we divide the training set into two parts (i.e., the training set and validation set) and minimize the following objective:

$$\begin{aligned} \min_{\gamma} \quad & \mathcal{L}_{val}(\omega^*, \upsilon^*, \gamma) \\ \text{s.t.} \quad & \omega^*, \upsilon^* = \arg\min_{\omega,\upsilon} \mathcal{L}_{train}(\omega, \upsilon, \gamma) \\ & \forall \text{node } i,\ \gamma_i \in \{0, 1\} \text{ and } s : \boldsymbol{x} \to \{[0, 1]^T, [1, 0]^T\} \end{aligned} \tag{9}$$

where $\mathcal{L}_{train}$ and $\mathcal{L}_{val}$ are the losses on the training and validation sets, respectively.

We call it One-Stage Tree because it keeps the discretization while simultaneously completing the two stages of building and pruning. The one-stage optimization is achieved by solving the bilevel optimization problem in Eq. (9). However, the problem of discretization remains. Specifically, the discretization in One-Stage Tree can be divided into two parts:

- *Discretization of Probabilistic Path* A discrete path routes an instance from root to a leaf. In *soft* trees, the paths are probabilistic and summed as prediction $\sum_{t\in\mathcal{T}} \mu_t(\boldsymbol{x};\omega,\gamma) \cdot \upsilon_t$. To discretize the path, a straightforward idea is to sample a path as the prediction based on the probability. The Monte Carlo method (Metropolis and Ulam, 1949) can be used to estimate the expectation of the loss. We use the reparameterization trick (Blum et al., 2015) to make it differentiable w.r.t ω. To approximate the sampling estimator to the true expectation, we use the Gumbel Softmax (Maddison et al., 2014) for reparameterization.
- *Discretization of Continuous Architecture* The continuous relaxation of γ unifies the forms of node and leaf. However, it makes the tree architecture difficult to interpret and leads to low computational efficiency caused by that all nodes need to be visited. To keep γ discrete but differentiable, we propose an architecture optimization strategy via proximal iterations (Parikh & Boyd, 2014; Xiao, 2010), which is inspired by NAS (Neural Architecture Search (Liu et al., 2018; Yao et al., 2020)).

To summarize, as shown in Fig. 2, One-Stage Tree retains the advantages of *hard* trees as discrete inference models and improves learning through end-to-end training with back-propagation.

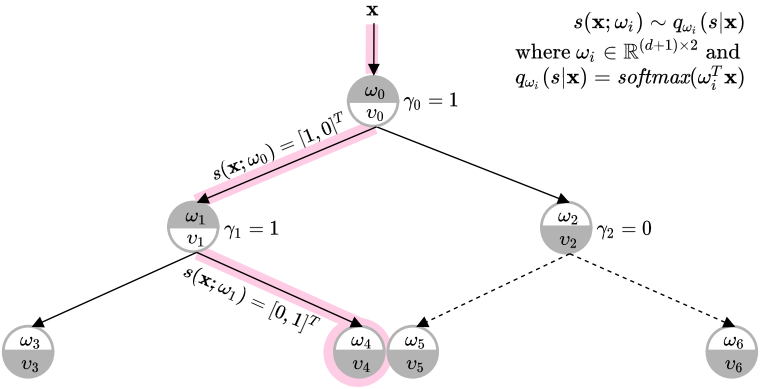


Fig. 2 Overview of One-Stage Tree. Each node i can be either an internal node (i.e., ω_i is emphasized) or a leaf (i.e., v_i is emphasized) according to γ_i. The path, where the instance $\boldsymbol{x}$ (including a bias term) is routed to, is shown in purple. Due to the discretization, $\boldsymbol{x}$ is routed to the left or right child at each node by $s(\boldsymbol{x};\omega)$ and finally reaches the leaf as the prediction

3.3 Gumbel-softmax path

The probabilistic path from the root to the leaf t lacks interpretability and leads to iterative optimization of the leaf parameters v. To discretize the probabilistic path, we express the router at node i as a random variable $s(\boldsymbol{x};\omega_i) \sim q_{\omega_i}(s|\boldsymbol{x})$. Thus, a discrete path is sampled from a continuous distribution parameterized by ω. In this way, we can explore the diversity, where each instance can belong to different leaves, and exploit the best path with the highest probability. Traditionally, each instance belongs to a fixed leaf by the splitting rule, and each leaf calculates v by its instances (e.g., the average of labels in CART). Here, by sampling from the random variable when training, we explore the case that each instance can be held by different leaves in the optimization process. Moreover, we directly choose the path with the highest probability as exploitation when testing.

$$s(\boldsymbol{x};\omega_i) = g_{\omega_i}(\epsilon,\boldsymbol{x}) \quad \text{with } \epsilon \sim p(\epsilon) \tag{10}$$

However, as a result of sampling, the loss cannot be propagated backward to ω. To train ω, we reparameterize the random variable s using a differentiable transformation $g_{\omega_i}(\epsilon,\boldsymbol{x})$, where g is parameterized by ω_i and ϵ is an (auxiliary) noise variable with independent marginal $p(\epsilon)$.

Using the reparameterization trick, we can now form MC (Monte Carlo) estimates (Metropolis & Ulam, 1949) of the expectation of the loss, which is differentiable w.r.t. ω, as follows:

$$\begin{aligned}\mathbb{E}_{q_\omega(s|\boldsymbol{x})}[\mathcal{L}(\theta)] &= \mathbb{E}_{q_\omega(s|\boldsymbol{x})}\left[\sum_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}} l(f(\boldsymbol{x};\theta),\boldsymbol{y})\right] \\ &= \mathbb{E}_{p(\epsilon)}\left[\sum_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}} l\left(\sum_{t\in\mathcal{T}} v_t \prod_{i\in\text{Path}(t)} \gamma_i \cdot g_{\omega_i}(\epsilon,\boldsymbol{x})_{(1-\mathbb{I}_{L(i)\in\text{Path}(t)})},\boldsymbol{y}\right)\right] \\ &\approx \frac{1}{M}\sum_{m=1}^{M}\sum_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}} l\left(\sum_{t\in\mathcal{T}} v_t \prod_{i\in\text{Path}(t)} \gamma_i \cdot g_{\omega_i}(\epsilon^{(m)},\boldsymbol{x})_{(1-\mathbb{I}_{L(i)\in\text{Path}(t)})},\boldsymbol{y}\right)\end{aligned} \tag{11}$$

where M is the number of samples and $\epsilon^{(m)} \sim p(\epsilon)$. If $L(i) \in \text{Path}(t)$ is satisfied, the indicator function $\mathbb{I}$ equals 1 and the probability of routing the left child is $g_{\omega_i}(\epsilon,\boldsymbol{x})_0$.

Specifically, we choose to sample noise from Gumbel Distribution (Gumbel, 1954), which can smoothly approximate the expectation (Maddison et al., 2014; Jang et al., 2016). Correspondingly, the Gumbel Softmax of $g_{\omega_i}(\epsilon,\boldsymbol{x})$ is expressed as follows:

$$\begin{aligned} g_{\omega_i}(\epsilon,\boldsymbol{x})_k &= \frac{e^{\hat{s}(\boldsymbol{x};\omega_i)_k/\tau}}{\sum_{j=0}^{1} e^{\hat{s}(\boldsymbol{x};\omega_i)_j}/\tau} \\ \text{where } \epsilon &\sim \mathit{Gumbel}(0),\ k\in\{0,1\},\ \text{and } \hat{s}(\boldsymbol{x};\omega_i)_k = s(\boldsymbol{x};\omega_i)_k + \epsilon \end{aligned} \tag{12}$$

τ denotes the temperature of the Gumbel Softmax. In practice, the value of τ can be empirically set to 1.

3.4 Architecture search via proximal iterations

Equation (9) implies a bilevel optimization problem with γ as the upper-level variable and ω as the lower-level variable. We also relax the discrete choice of whether to prune or not (i.e. $\gamma_i \in [0,1]$). As a result, γ can be optimized w.r.t. its validation set performance by gradient descent.

Budding Tree (Irsoy et al., 2014) propagates the error backwards from the root towards the leaves. f_i denotes the response at node i (Eq. (2)). Define pa(i) as the parent of node i and $\delta_i = \partial\mathcal{L}/\partial f_i$ as the *responsibility* of node i. Deriving $\mathcal{L}$ w.r.t. γ_i, we have:

$$\begin{aligned} \frac{\partial\mathcal{L}}{\partial\gamma_i} &= \delta_i\left(-v_i + \cdots\right) \\ \text{with}\quad \delta_i &= \begin{cases} \frac{\partial\mathcal{L}}{\partial f_i}, & \text{if i is the root} \\ \delta_{\text{pa}(i)} \times \gamma_i \times \cdots, & \text{if } i \text{ is a child} \end{cases} \end{aligned} \tag{13}$$

The detailed derivation of Eq. (13) can be found in "Appendix 1". From Eq. (13), we can see that once the internal node is pruned (i.e., $\gamma_i = 0$), it will never be able to bud again because its gradient is 0, which we call *dying* γ. Moreover, it is prohibitive to evaluate the gradient due to the expensive inner optimization $\arg\min_{\omega,v} \mathcal{L}_{train}(\omega, v, \gamma)$. Following the commonly used method (e.g., meta learning (Finn et al., 2017) and NAS (Liu et al., 2018)), we use a one-step gradient approximation to the optimal internal parameter ω^* to improve efficiency. Thus, the gradient of the architecture parameter γ is as follows:

$$\begin{aligned} &\nabla_\gamma \mathcal{L}_{val}(\omega^*, \upsilon^*, \gamma) \\ \approx\ &\nabla_\gamma \mathcal{L}_{val}\big(\omega - \xi \nabla_\omega \mathcal{L}_{train}(\omega, \upsilon^*, \gamma), \upsilon^*, \gamma\big) \\ =\ &\nabla_\gamma \mathcal{L}_{val}\big(\omega', \upsilon^*, \gamma\big) - \xi \nabla^2_{\gamma,\omega} \mathcal{L}_{train}(\omega, \upsilon^*, \gamma) \nabla_{\omega'} \mathcal{L}_{val}\big(\omega', \upsilon^*, \gamma\big) \end{aligned} \tag{14}$$

More specifically, the approximate procedure alternatively optimizes the node parameters (ω, υ) and the architecture parameters γ. At step k, given the current architecture $\gamma^{(k)}$, we first calculate $\upsilon^{(k+1)}$ in closed-form (Sect. 3.5). Then, we obtain $\omega^{(k+1)}$ by descending $\nabla_{\omega^{(k)}} \mathcal{L}_{\text{train}}(\omega^{(k)}, \upsilon^{(k+1)}, \gamma^{(k)})$ with the step size ξ as a one-step optimization for w^* under $\gamma^{(k)}$. Then, we update the architecture parameters $\gamma^{(k)}$ so as to minimize the validation loss. The architecture gradient is given in Eq. (14). We omit the step-index k for brevity. υ^* denotes the optimal leaf values and ω^* denotes the internal parameters with a one-step gradient decent.

However, there exist two problems in solving Eq. (14). First, the evaluation of the second-order derivative ∇^2 is expensive due to a large number of parameters. Second, the continuous trick further leads to the performance gap caused by discretizing $\gamma_i \in [0, 1]$ at training to $\{0, 1\}$ at testing.

To address the two issues, we employ the variant of Proximal Algorithm (Yao et al., 2020) for optimizing γ efficiently. Equivalently, we transform γ_i to a 2-d one-hot vector that indicates whether to prune or not, i.e., $\gamma_i \in \{[0, 1], [1, 0]\}$. Let the feasible space of γ be $\mathcal{C} = \{\gamma |\ \forall i,\ \|\gamma_i\|_0 = 1\ \wedge\ 0 \le \gamma_{i,j} \le 1\}$. We denote it as the intersection of two feasible spaces (i.e., $\mathcal{C} = \mathcal{C}_1 \cap \mathcal{C}_2$), where $\mathcal{C}_1 = \{\gamma |\ \forall i,\ \|\gamma_i\|_0 = 1\}$ and $\mathcal{C}_2 = \{\gamma |\ \forall i,\ 0 \le \gamma_{i,j} \le 1\}$. With such a constrained form, we can apply the composition of lazy and standard proximal steps proposed in Yao et al. (2020).

Specifically, as shown in Eq. (15), in each proximal iteration, we first get a discrete architecture $\bar{\gamma}$ constrained by $\mathcal{C}_1$. Then, we derive gradients w.r.t $\bar{\gamma}$ and keep γ to be optimized as continuous variable but constrained by $\mathcal{C}_2$:

$$\gamma^{(k+1)} = \text{prox}_{\mathcal{C}_2}\big(\gamma^{(k)} - \epsilon \nabla_{\bar{\gamma}^{(k)}} \mathcal{L}_{\text{val}}\big(\bar{\gamma}^{(k)}\big)\big), \text{ where } \bar{\gamma}^{(k)} = \text{prox}_{\mathcal{C}_1}(\gamma^{(k)}) \tag{15}$$

Algorithm 1 One-Stage Tree: End-to-End Tree Builder and Pruner

1: Initialize Tree Parameters $\theta = (\omega, \upsilon, \gamma)$ according to the constraints;
2: **while** not converged **do**
3: Get *discrete* architecture: $\bar{\gamma}^{(k)} = \text{prox}_{\mathcal{C}_1}(\gamma^{(k)})$, where $\mathcal{C}_1 = \{\gamma |\ \forall i,\ \|\gamma_i\|_0 \le 1\}$;
4: Update *optimal* leaves $\upsilon^{(k)}$ in closed-form by Equation (17) with the training set;
5: Update $\omega^{(k)}$ by the MC approximation of $\nabla_{\omega^{(k)}} \mathbb{E}_{q_{\omega^{(k)}}(s|\boldsymbol{x})}[\mathcal{L}_{\text{train}}(\omega^{(k)}, \upsilon^{(k)}, \bar{\gamma}^{(k)})]$ with Equation (11);
6: Update $\gamma^{(k+1)} = \text{prox}_{\mathcal{C}_2}(\gamma^{(k)} - \epsilon \nabla_{\bar{\gamma}^{(k)}} \mathcal{L}_{\text{val}}(\omega^{(k+1)}, \upsilon^{(k+1)}, \bar{\gamma}^{(k)}))$, where $\mathcal{C}_2 = \{\gamma |\ \forall i,\ 0 \le \gamma_i \le 1\}$;
7: **end while**
8: **return** Tree Parameters θ.

Algorithm 1 shows the overall workflow of One-Stage Tree that searches the architecture parameters γ via proximal iterations. In the k-th iteration, the architecture and node parameters are updated alternatively. As the lazy proximal step first projects γ into the discrete feasible space $\mathcal{C}_1$, we can obtain the discrete architecture $\bar{\gamma}^{(k)} = \text{prox}_{\mathcal{C}_1}(\gamma^{(k)})$ (Line 3). Then, we calculate the optimal leaf value υ in closed-form (Sect. 3.5) and update the

internal parameters ω on the training dataset (Sect. 3.3) based on $\bar{\gamma}^{(k)}$ (Lines 4-5). After forwarding ω one-step as in Eq. (14), we optimize $\gamma^{(k)}$ with the gradient derived from $\bar{\gamma}^{(k)}$ as continuous variable and then project it into $\mathcal{C}_2$ (Line 6).

In each proximal iteration, we keep the architecture γ discrete when training, which contributes to reducing the performance gap caused by discretizing architecture from a continuous one. Moreover, we can ignore the second-order derivative of small magnitude $\epsilon \cdot \xi$ because γ will be projected into the discrete feasible space $\mathcal{C}_1$ in the next iteration, i.e., $\text{prox}_{\mathcal{C}_1}^{(k+1)}(\text{prox}_{\mathcal{C}_2}^{(k)}(\gamma^{(k)} - \epsilon(\nabla_{\bar{\gamma}^{(k)}} - \xi\nabla^2_{\bar{\gamma}^{(k)},\omega^{(k)}}\nabla_{\omega^{(k+1)}}))) \approx \text{prox}_{\mathcal{C}_1}^{(k+1)}(\text{prox}_{\mathcal{C}_2}^{(k)}(\gamma^{(k)} - \epsilon\nabla_{\bar{\gamma}^{(k)}}))$. Thus, the computational efficiency of updating γ can be significantly improved.

3.5 Optimal leaves in closed-form

Unlike *multivariate dense* trees (Irsoy et al., 2012, 2014) where leaf values are iteratively optimized by gradient descent, we can solve for υ in closed-form due to the discretization of path and architecture. The prediction $f(\boldsymbol{x})$ is υ_t when $\mu_t(\boldsymbol{x}) = 1$. Define $(\boldsymbol{x}, \boldsymbol{y})$ as an instance and $I_t = \{(\boldsymbol{x}, \boldsymbol{y}) | \forall (\boldsymbol{x}, \boldsymbol{y}) \sim \mathcal{D}, \mu_t(\boldsymbol{x}; \omega, \gamma) = 1\}$ as the instance set of leaf t, the derivative of the loss function can be expressed as:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \upsilon_t} &= \sum_{\boldsymbol{x},\boldsymbol{y}} \frac{\partial l}{\partial f}\frac{\partial f}{\partial \upsilon_t} = \sum_{\boldsymbol{x},\boldsymbol{y}} \frac{\partial l}{\partial f}\frac{\partial \sum_{t\in\mathcal{T}} \upsilon_t \cdot \mu_t(\boldsymbol{x};\omega,\gamma)}{\partial \upsilon_t} \\ &= \sum_{\boldsymbol{x},\boldsymbol{y}} \frac{\partial l}{\partial f}\mu_t(\boldsymbol{x};\omega,\gamma) = \sum_{(\boldsymbol{x},\boldsymbol{y})\in I_t} \frac{\partial l}{\partial f} \end{aligned} \tag{16}$$

From Eq. (16), the optimal leaves are solved in closed-form by simply deriving l w.r.t. the tree prediction. Let $\partial\mathcal{L}/\partial\upsilon_t = 0$, we show the optimal solution for the leaf values under the common *MSE* and *CrossEntropy* losses:

- *MSE*

$$\begin{aligned} &\text{Let } l(\boldsymbol{y}, f) = \frac{1}{2}(\boldsymbol{y} - f)^2 \\ &\therefore \frac{\partial \mathcal{L}}{\partial \upsilon_t} = \sum_{(\boldsymbol{x},\boldsymbol{y})\in I_t} \frac{\partial l}{\partial f} = \sum_{(\boldsymbol{x},\boldsymbol{y})\in I_t} (f - \boldsymbol{y}) = \sum_{(\boldsymbol{x},\boldsymbol{y})\in I_t} \left(\upsilon_t - \boldsymbol{y}\right) \\ &\text{Let } \frac{\partial \mathcal{L}}{\partial \upsilon_t} = 0 \quad \therefore \upsilon_t^* = \frac{\sum_{(\boldsymbol{x},\boldsymbol{y})\in I_t} \boldsymbol{y}}{|I_t|} \end{aligned}$$

- *CrossEntropy* with constraint $\sum_{i=0}^{c} f_i = 1$ (i.e., c is the number of classes, and the probability sum is 1):

Table 1 Characteristic comparison between One-Stage Tree and other tree models including *hard* tree and existing *soft* trees

Characteristic	Training				Inference	
	Joint θ optimization	End-to-End training ω	Arch. γ search	Optimal υ in closed-form	Discrete path	Performance consistency
Hard Tree	×	×	✓	✓	✓	✓
Soft Decision Tree	×	✓	×	×	×	✓
Budding Tree	✓	✓	Dying γ	×	×	✓
End2End Tree	×	✓	✓	✓	✓	×
One-Stage Tree	✓	✓	✓	✓	✓	✓

$$\text{Let } l(\boldsymbol{y}, f, \lambda) = -\sum_{i=0}^{c} \boldsymbol{y}_i \log f_i + \lambda\left(1 - \sum_{i=0}^{c} f_i\right)$$

$$\therefore \frac{\partial \mathcal{L}}{\partial \upsilon_{t,i}} = \sum_{(\boldsymbol{x},\boldsymbol{y})\in I_t} \frac{\partial l}{\partial f_i} = \sum_{(\boldsymbol{x},\boldsymbol{y})\in I_t} \left(-\frac{\boldsymbol{y}_i}{f_i} - \lambda\right) = \sum_{(\boldsymbol{x},\boldsymbol{y})\in I_t} \left(-\frac{\boldsymbol{y}_i}{\upsilon_{t,i}} - \lambda\right)$$

$$\text{Let } \frac{\partial \mathcal{L}}{\partial \upsilon_{t,i}} = 0 \quad \therefore \upsilon^*_{t,i} = \frac{-\sum_{(\boldsymbol{x},\boldsymbol{y})\in I_t} \boldsymbol{y}_i}{\lambda |I_t|}$$

$$\text{Let } \frac{\partial l}{\partial \lambda} = 1 - \sum_{i=0}^{c} f_i = 1 - \sum_{i=0}^{c} \upsilon_{t,i} = \sum_{i=0}^{c} \frac{-\sum_{(\boldsymbol{x},\boldsymbol{y})\in I_t} \boldsymbol{y}_i}{\lambda |I_t|} = 0$$

$$\therefore \lambda^* = \frac{-\sum_{(\boldsymbol{x},\boldsymbol{y})\in I_t} \sum_{i=0}^{c} \boldsymbol{y}_i}{|I_t|} == -1, \ \upsilon^*_t = \frac{\sum_{(\boldsymbol{x},\boldsymbol{y})\in I_t} \boldsymbol{y}}{|I_t|}$$

In summary, the optimal solution for υ under both *MSE* and *CrossEntropy* is:

$$\upsilon^*_t = \frac{\sum_{(\boldsymbol{x},\boldsymbol{y})\in I_t} \boldsymbol{y}}{|I_t|} \tag{17}$$

3.6 In-depth discussion

Table 1 shows the characteristic comparison between One-Stage Tree and other tree models from both training and inference perspectives.

In the training phase, One-Stage Tree improves learning in an end-to-end manner. Unlike the existing *soft* trees, One-Stage Tree can achieve joint optimization for node and architecture parameters. Soft Decision Tree does not support the optimization of the architecture parameters. Due to the lack of any pruning strategy, Soft Decision Tree is easy to fall into overfitting. Budding Tree considers the search of the architecture parameters, but the dying γ problem may occur. Compared to One-Stage Tree, End2End Tree is a two-stage method that first learns the node parameters end-to-end and then searches the architecture

parameters greedily. Moreover, due to the discretization of path and architecture, One-Stage Tree can efficiently solve v in closed form.

In the inference phase, One-Stage Tree can keep the same advantage of interpretability as *hard* trees due to maintaining discretization. Unlike End2End Tree, which transforms from the probabilistic tree to the deterministic one during inference, One-Stage Tree does not require the additional transformation and thus can reduce the performance gap between training and testing.

4 Experiments

In this section, we conduct extensive experiments on public datasets to answer the following research questions:

- *RQ1* How effective is the proposed One-Stage Tree?
- *RQ2* Is One-Stage Tree robust to hyperparameters?
- *RQ3* How do different components of One-Stage Tree (e.g., Proximal Algorithm) contribute to the performance?
- *RQ4* How to reflect the interpretability of One-Stage Tree?

4.1 Experimental setting

We use a total of 22 public datasets from OpenML,[2] UCI repository,[3] and Kaggle.[4] There are 17 classification (C) datasets and 5 regression (R) datasets that have various numbers of features (5 to 57) and instances (100 to 30000).

Benefiting from *soft* trees, One-Stage Tree can be trained using tools from deep learning. We choose the Adam optimizer (Kingma & Ba, 2014) to train One-Stage Tree. The number of epochs is up to 200, the batch size is 32, and the learning rate is 0.01. The other hyperparameters of the Adam optimizer are all the same as default settings. EarlyStopping (Prechelt, 1998), which monitors the validation loss, is used to prevent overfitting with patience of 15. Except for Sect. 4.3.1, the depths of all tree models are set to 6 for comparison.

We use *MSE* loss for regression tasks and *CrossEntropy* loss for classification tasks in all experiments. Moreover, to evaluate the trees, we use *r2-score* and *accuracy* for regression tasks and classification tasks respectively. For clarity, we multiply all metrics by 100 in all tables.

Due to the bilevel optimization, One-Stage Tree splits the raw data as 6:2:2 (train:validation:test) and uses the validation set for optimizing the architecture parameters (i.e., tree pruning). For the other methods (i.e., CART, Soft Decision Tree, and End2End Tree), the raw data is divided using a ratio of 8:2 (train:test). These methods do not require a validation set during training.

[2] https://www.openml.org.
[3] https://archive.ics.uci.edu/ml/index.php.
[4] https://www.kaggle.com.

Table 2 Comparison between One-Stage Tree, CART, and the existing *soft* trees on classification datasets from UCIrvine

Dataset	Inst.\Feat.	CART[a]	Soft Decision Tree[a]	End2End Tree[a]	One-Stage Tree
PimaIndian	768\7	75.97 ± 0.58	30.52 ± 0.00	68.54 ± 2.10	**81.82 ± 0.92**
SpectF	267\44	58.52 ± 1.01	**83.33 ± 0.00**	82.32 ± 4.00	79.26 ± 1.55
German Credit	1001\24	70.50 ± 0.22	29.00 ± 0.00	71.00 ± 0.00	**75.70 ± 1.52**
Ionosphere	351\34	92.96 ± 1.00	66.58 ± 8.17	87.45 ± 3.86	**93.24 ± 0.63**
Credit Default	30000\25	82.53 ± 0.01	21.62 ± 0.00	78.37 ± 0.03	**82.74 ± 0.08**
Messidor_features	1150\19	59.74 ± 0.31	66.23 ± 3.86	59.07 ± 2.94	**66.93 ± 1.36**
Wine Quality Red	999\12	58.31 ± 0.17	42.50 ± 14.76	51.90 ± 3.72	**61.38 ± 0.68**
Wine Quality White	4900\12	50.41 ± 0.00	*Err.*	41.48 ± 0.78	**53.82 ± 0.44**
SpamBase	4601\57	91.21 ± 0.05	75.53 ± 1.85	75.31 ± 6.70	**93.25 ± 0.43**
Credit-a	690\6	79.71 ± 0.32	65.74 ± 6.14	67.26 ± 2.51	**85.07 ± 0.83**
Fertility	100\9	80.00 ± 2.24	87.27 ± 4.10	**90.00 ± 3.16**	**90.00 ± 0.00**
Heaptitis	155\6	65.81 ± 2.89	70.09 ± 2.54	64.81 ± 4.67	**83.87 ± 3.23**
Megawatt1	253\37	81.96 ± 1.64	13.73 ± 0.00	86.27 ± 0.00	**88.24 ± 2.77**

Bold indicates the best

[a]The results obtained with the open-source code, *Err.* unknown bug when running the open-source code, Inst. is short for Instance, Feat. is short for Feature

Table 3 Comparison between One-Stage Tree and CART on regression datasets

Dataset	Source	Instances\Features	CART[a]	One-Stage Tree
Housing Boston	UCIrvine	506\13	61.55 ± 0.66	**63.73 ± 2.49**
Airfoil	UCIrvine	1503\5	**74.00 ± 0.00**	71.08 ± 2.18
Openml_589	OpenML	1000\25	77.12 ± 0.32	**79.44 ± 1.94**
Openml_620	OpenML	1000\25	**74.83 ± 0.15**	73.64 ± 1.87

Bold indicates the best

[a]The results obtained with the open-source code

4.2 Effectiveness of One-Stage Tree (RQ1)

In this subsection, we demonstrate the effectiveness of One-Stage Tree.

4.2.1 Comparison with trees

We compare One-Stage Tree on 22 datasets with the state-of-the-art and baseline tree methods, including:

1. CART (Breiman et al., 1984): the most typical *univariate discrete* tree, which uses *MSE* for regression and *Gini Index* for classification as the splitting rules. We choose the widely-used *sklearn.tree* package[5] to run CART.

[5] https://scikit-learn.org/stable/.

Table 4 Comparison between One-Stage Tree, CART, and Budding Tree with the same experimental setting in Irsoy et al. (2014)

Tree Dataset	Housing Boston	German Credit	Magic	Pima Indian	Spam Base	Ecoli	Glass	Yeast
Budding Tree*	78.20	68.70	**86.30**	67.10	91.40	**83.56**	53.78	59.31
CART[a]	73.51	70.00	83.38	72.66	90.74	76.79	**70.83**	**59.80**
One-Stage Tree	**80.80**	**70.06**	85.61	**78.52**	**93.09**	79.46	62.60	58.59

Bold indicates the best

*The results reported in the paper

[a]The results obtained with the open-source code

2. Soft Decision Tree (Irsoy et al., 2012): a *multivariate dense* tree, of which all the paths to all the leaves contribute to the final prediction with different probabilities. It only supports classification tasks. We use the open-source code[6] with most stars on GitHub to obtain the experimental results.
3. Budding Tree (Irsoy et al., 2014): a *multivariate dense* tree, which searches the tree architecture in the learning phase. It supports both classification and regression tasks.
4. End2End Tree (Hehn et al., 2019): the state-of-the-art *multivariate discrete* tree, which is fully probabilistic at the training phase but becomes deterministic after an annealing process at the testing phase. It is open-source[7] and only supports classification tasks.

For the open-source methods including CART, Soft Decision Tree, and End2End Tree, we directly use the default hyperparameters in the open-source codes. To investigate the stability of the training process, we randomly select 5 random seeds and obtain the mean and standard deviation of the trees' performance. Table 2 shows the comparison results between One-Stage Tree and the open-source methods including CART, Soft Decision Tree, and End2End Tree on classification datasets. Since Soft Decision Tree and End2End Tree do not support regression tasks, we only show the comparison results with CART in Table 3.

Moreover, since Budding Tree is not completely open-source, we directly use the available datasets and the experimental results reported in the original paper (Irsoy et al., 2014). To set up the same experimental setting, we separate 1/3 of the dataset as a test set to evaluate the final performance. The comparison results between One-Stage Tree and Budding Tree are shown in Table 4. According to the comparison results, we can observe that:

- The comparison results shown in Table 2 indicate that One-Stage Tree outperforms the existing tree methods and achieves the best performance in all but 1 case on classification datasets.

[6] https://github.com/kimhc6028/soft-decision-tree.

[7] https://github.com/tomsal/endtoenddecisiontrees.

Table 5 *p*-values for each pairwise comparison using the Nemenyi post-hoc test for the *soft* tree models (Confidence level $\alpha = 0.05$)

	CART	Soft Decision Tree	End2End Tree	One-Stage Tree
CART	1.0000	0.5177	0.9000	**0.0255**
Soft Decision Tree	0.5177	1.0000	0.8162	**0.0001**
End2End Tree	0.9000	0.8162	1.0000	**0.0046**
One-Stage Tree	**0.0255**	**0.0001**	**0.0046**	1.0000

Bold indicates the statistically significant difference (i.e., $p < 0.05$)

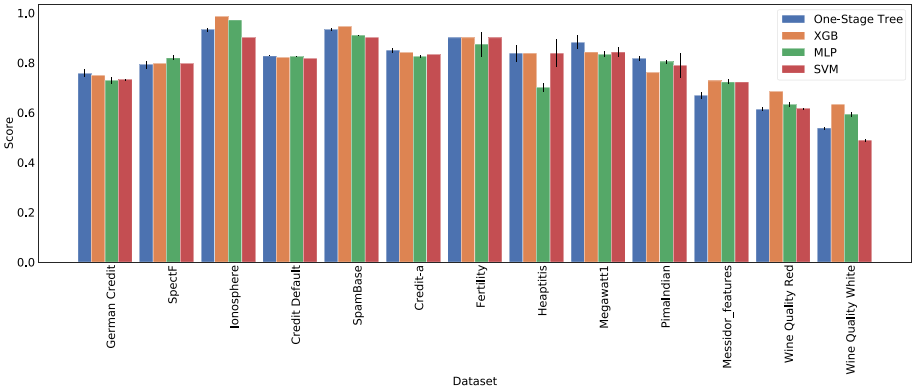


Fig. 3 Comparison between One-Stage Tree and other ML Methods on classification datasets

- Besides the classification tasks, One-Stage Tree can also support regression tasks. As shown in Table 3, for regression datasets from different sources, One-Stage Tree outperforms CART on 3/5 datasets.
- The comparison results shown in Table 4 demonstrate that One-Stage Tree outperforms Budding Tree and CART on 5/8 and 6/8 datasets respectively. Although CART and Budding Tree also search the node and architecture parameters in the training phase, One-Stage Tree shows better performance in learning the tree parameters θ.
- For *soft* trees, end-to-end training based on gradient descent inevitably has a degree of randomness. Table 2 shows that One-Stage Tree has a smaller standard deviation on most datasets, achieving more stable performance compared to the existing *soft* trees.

4.2.2 Statistical comparison

To further statistically evaluate the difference between the *soft* trees, we perform the Friedman test (Demšar, 2006), which is a non-parametric equivalent of the repeated-measures ANOVA. It is used to determine whether or not there is a statistically significant difference between the *soft* tree models.

For the comparison results in Table 2, we first calculate the Friedman statistic. Let r_i^j be the rank of the j-th of k *soft* tree models ($k = 4$, i.e., CART, Soft Decision Tree, End2End Tree, and One-Stage Tree) on the i-th of N classification datasets. The Friedman test compares the average ranks of models, $R_j = \frac{1}{N} \sum_i r_i^j$. The null-hypothesis states that all the tree

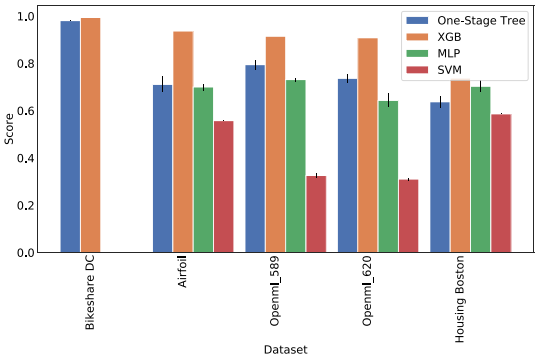


Fig. 4 Comparison between One-Stage Tree and other ML Methods on regression datasets

Table 6 Average Rank of One-Stage Tree and other ML Methods

Method	Avg. Rank
XGBoost	1.63
One-Stage Tree	2.47
MLP	2.63
LinearSVM	3.26

models are equivalent and so their ranks R_j should be equal. We employ the scipy tool[8] to calculate the Friedman statistic. The Friedman statistic is 19.837209 and the corresponding p-value is 0.00018. Since the p-value is less than 0.05, we can reject the null hypothesis that the performance is the same for all four types of *soft* trees. In other words, we have sufficient evidence to conclude that the trees lead to statistically significant differences in terms of performance. Since the p-value of the Friedman test is statistically significant, we perform the Nemenyi post-hoc test (Nemenyi, 1963) to further determine exactly which trees have different means. Table 5 shows the p-values for each pairwise comparison. We can conclude that One-Stage Tree is significantly different from other trees for a confidence level of $\alpha = 0.05$. Additionally, according to the Friedman test, there is no significant difference for the three trees compared in Table 4.

For the regression tasks shown in Table 3, we perform Wilcoxon signed-rank test (Demšar, 2006) to compare the two tree models statistically. The statistic is 5.0 and the corresponding p-value is 0.24886. Thus, there is no statistically significant difference between CART and One-Stage Tree in the regression tasks.

4.2.3 Comparison with other standard ML methods

Moreover, we compare One-Stage Tree with other standard ML methods (i.e., XGBoost (Chen & Guestrin, 2016),[9] Support Vector Machine,[10] and Multi-Layer Perception[11]) for reference. In particular, a linear kernel for SVM is used. We use the open-source

[8] https://github.com/scipy/scipy.
[9] https://github.com/dmlc/xgboost.
[10] https://github.com/scikit-learn/scikit-learn/tree/main/sklearn/svm.
[11] https://github.com/scikit-learn/scikit-learn/tree/main/sklearn/neural_network.

Table 7 Mean number of internal nodes per tree depth

Depth	1	2	3	4	5	6	7	8	9
One-Stage Tree	1.0	2.6	7.0	14.9	21.9	44.0	59.6	129.3	187.4
Perfect Binary Tree	1	3	7	15	31	63	127	255	511

implementation of the above methods and perform grid search to select the best hyperparameters for the learners in each dataset. The hyperparameter search space can be seen in "Appendix 2". Similarly, 5 randomly selected seeds are used to obtain the performance mean and standard deviation. Figures 3 and 4 show the performance comparison on the classification and regression datasets respectively. The mean performance is present in a bar chart with an error line (i.e., the standard deviation). MLP and SVM perform very poor on the dataset BikeShare DC due to the large range of regression values. Thus, we truncate their performance to 0 in Fig. 4. This also reflects the advantage of tree models that each leaf takes the mean value of its samples as the prediction output.

To present the comparison results clearly, we further calculate the average rank of each method in Table 6. Compared to other methods, One-Stage Tree outperforms traditional machine learning methods MLP and SVM. Although XGBoost ensembles 100 trees with a max depth of 6 via GBDT (Friedman, 2001), One-Stage Tree is still competitive on several datasets. We also consider combining One-Stage Tree with ensemble learning methods such as bagging and boosting to further improve the performance in future work.

In summary, our proposed One-Stage Tree is effective for both classification and regression tasks and outperforms CART and the existing *soft* trees on most datasets. One-Stage Tree also shows good performance in comparison with other standard machine learning methods.

4.3 Robustness of One-Stage Tree (RQ2)

In this subsection, we evaluate whether One-Stage Tree is sensitive to the key hyperparameters, i.e., the tree depth and the validation size ρ. We perform experiments on all classification datasets with the same experimental setting as in RQ1.

4.3.1 Tree depth

The size of tree depth ranges from 1 to 9. We run One-Stage Tree 5 times with different random seeds and report the mean number of internal nodes per tree depth. As shown in Table 7, the optimization of the architecture parameters in One-Stage Tree is effective for tree pruning. At a depth of 9, One-Stage Tree can even prune 60% of the internal nodes. Despite the larger depth, trees may be pruned as shallow ones to avoid the risk of overfitting.

With the increase of the tree depth, the search space of the tree structure is growing. Meanwhile, the ability to find effective trees becomes crucial. Figure 5 shows the performance curves of all tree models with respect to the tree depth. From Fig. 5, we can observe that:

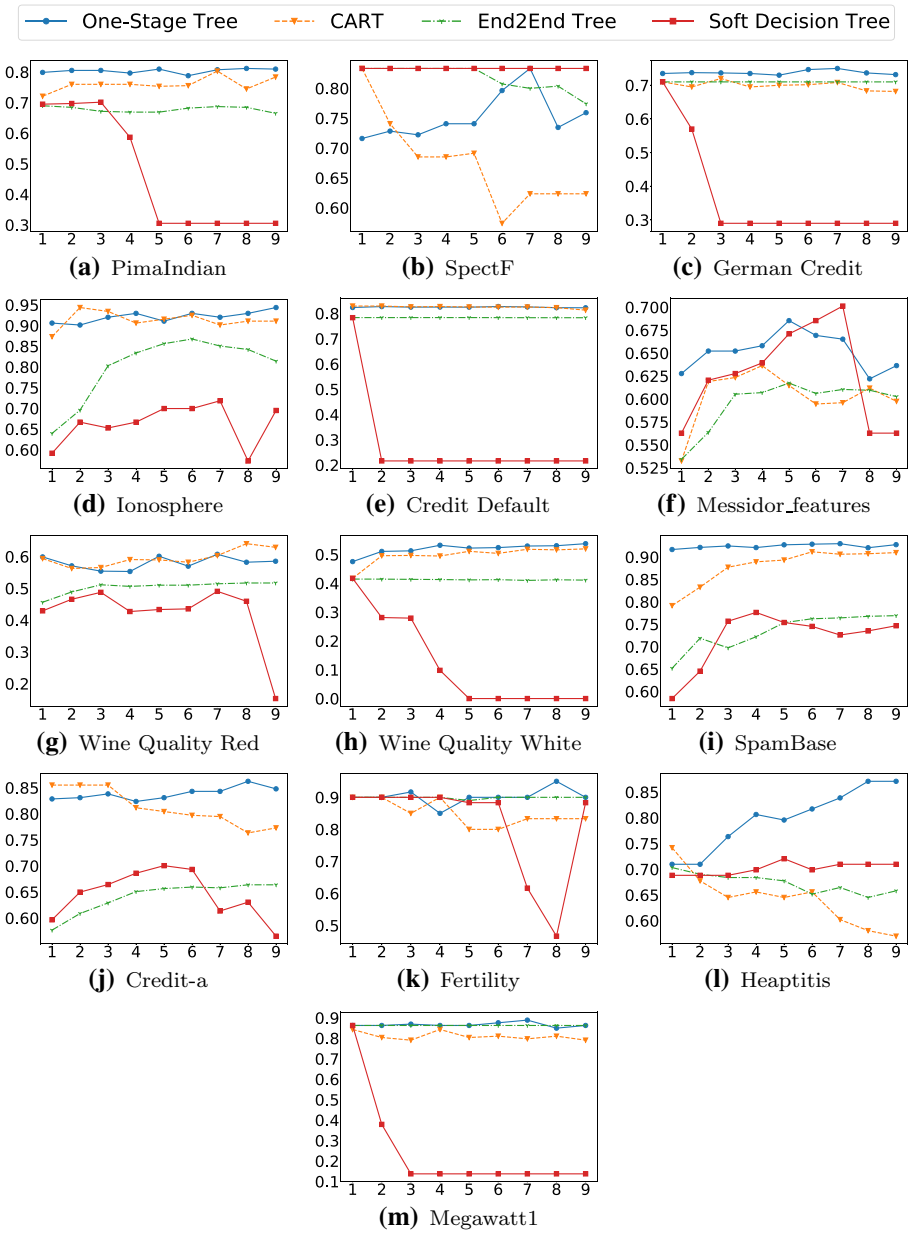


Fig. 5 Effect of tree depth

- Due to the joint optimization of the node and architecture parameters, One-Stage Tree can achieve better performance than other tree models at different depths. Moreover, from a global perspective, the performance of One-Stage Tree increases with the tree

Table 8 Comparison results of One-Stage Tree with different validation sizes ρ

Dataset	Features	$\rho = 0.05$	$\rho = 0.10$	$\rho = 0.15$	$\rho = 0.20$	$\rho = 0.25$	$\rho = 0.30$
Credit-a	6	84.78	84.06	84.78	85.07	84.06	**86.23**
Heaptitis	6	77.42	80.65	77.42	**83.87**	80.65	77.42
PimaIndian	7	80.36	81.17	81.82	**81.82**	80.52	81.17
Fertility	9	90.00	88.75	86.25	**90.00**	88.75	86.25
Wine Quality Red	12	61.56	60.94	**61.88**	61.38	57.19	58.13
Wine Quality White	12	53.47	53.88	51.53	53.82	**53.98**	52.14
Messidor_features	19	**71.00**	70.13	70.13	66.93	67.97	68.83
German Credit	24	74.50	74.00	74.50	**75.70**	75.50	74.00
Credit Default	25	**82.88**	82.58	82.85	82.74	82.60	82.72
Ionosphere	34	91.20	89.08	**94.01**	93.24	92.86	91.55
Megawatt1	37	86.27	**92.16**	90.20	88.24	88.24	90.20
SpectF	44	74.07	**79.63**	77.31	79.26	75.93	72.69
SpamBase	57	92.73	**93.70**	93.49	93.25	93.49	93.05

Bold indicates the best

depth. When the tree depth is 9, One-Stage Tree can still achieve performance improvement on several datasets (e.g., Ionosphere and Wine Equality White).

- As the tree pruning is not supported, Soft Decision Tree can easily fall into overfitting. Especially at the larger depth, the performance of Soft Decision Tree may decrease dramatically.
- End2End Tree can achieve stable performance at different depths. However, as a two-stage method of building and then pruning, it does not perform as well as One-Stage Tree.
- For the *hard* tree CART that greedily achieves building and pruning, the performance at a large tree depth may fall into local optimal. For example, at a depth of 9, the performance of CART is even much worse than that at a depth of 1 on datasets such as SpectF and Hepatitis.

In summary, One-Stage Tree not only achieves stable performance at different tree depths but also achieves performance improvement as the tree depth increases. Moreover, considering that the best performance may be achieved at depth $\{5, 6, 7\}$, the uncertainty in deep learning is worth noting. More regularization techniques need to be added to alleviate such problems in future work.

4.3.2 Validation size

To evaluate the impact of the validation size, we use the same experimental setting as in RQ1 (e.g., depth of 6 and patience of 15). As shown in Table 8, the performance of One-Stage Tree remains stable with respect to the validation size. Moreover, the comparison results in Table 8 demonstrate that the optimal ρ increases as the number of features decreases. The validation size can be seen as a trade-off for the training of the internal and architecture parameters. As ρ grows, more instances are used to train the architecture

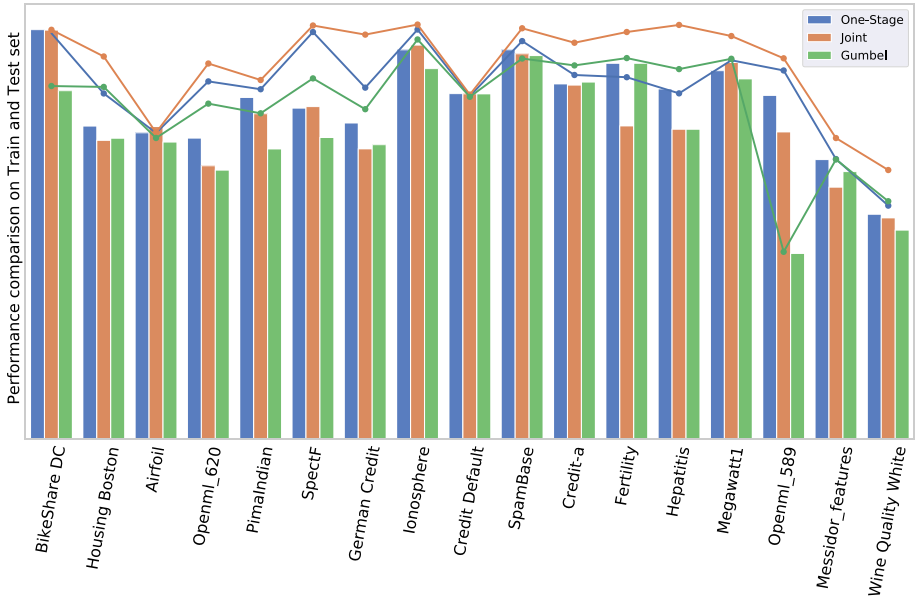


Fig. 6 Performance comparison between One-Stage Tree and Joint Tree (Bar chart shows the comparison in the test set, and line chart shows the comparison in the training set)

Table 9 Statistics (i.e., the number of top-ranked datasets for each tree) on the performance of One-Stage Tree, Joint Tree and Gumbel Tree (0.5 means that two tree models tie on the dataset)

	One-Stage Tree	Joint Tree	Gumbel Tree
Classification Tasks			
Train	0	13	0
Test	7.5	4	1.5
Regression Tasks			
Train	0	5	0
Test	4	1	0

parameters γ. At the same time, the internal parameters ω is much less optimized. The number of the internal parameters is d (i.e., the number of features) times greater than the number of the architecture parameters. Thus, for the datasets with more features, training ω requires more instances (i.e., the smaller ρ).

4.4 Ablation study (RQ3)

In this subsection, we conduct experiments to check whether the discretization of One-Stage Tree influences the performance gap. To validate the effectiveness of proximal iterations, we propose two variants:

- *Joint Tree* which is a variant of One-Stage Tree without proximal iterations. Joint Tree optimizes the node and architecture parameters according to Eq. (8).

Table 10 Factors for the interpretability of decision trees

Interpretability	*Hard* Tree	Soft Decision Tree	One-Stage Tree
Routing Rule	✓	×	✓
Feature Importance	✓	×	✓
Node Instance Distribution	✓	×	✓
Node Impurity	✓	×	✓
Deterministic Predicted Value	✓	×	✓

- *Gumbel Tree* which is another variant of One-Stage Tree that discretizes the architecture by the Gumbel Softmax in the outer minimization of Eq. (9).
  The details of Gumbel Tree can be found in "Appendix 3".

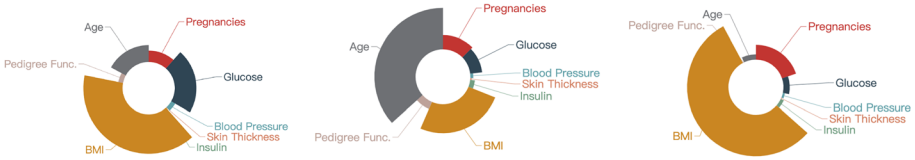
We perform the experiments on all classification and regression datasets used in RQ1 with the same experimental settings.

The performance comparison is presented in Fig. 6 and Table 9. In Fig. 6, the bar chart shows the performance on the test set to indicate the generalization of the trees, and the line chart shows the performance on the training set to represent the fit of the trees. The height difference between points and bars of the same color represents the performance gap of the tree between training and testing. For greater clarity, we count the number of top-ranked datasets for each tree in Table 9.

Without dividing the validation set to optimize γ, Joint Tree obtains a better fit on the training set in all datasets. However, One-Stage Tree achieves a significant performance improvement over Joint Tree on the test set. Compared with Joint Tree, One-Stage Tree reduces the fit to the training set and greatly improves the generalization ability on the test set. The performance gap between training and testing is indeed reduced by proximal iterations.

Additionally, from Fig. 6 and Table 9, we can see that One-Stage Tree performs much better than Gumbel Tree. Figure 6 shows that Gumbel Tree has the worst performance on the training set on most datasets, which indicates that Gumbel Tree is not fully trained. The main reason why the underfitting problem occurs in Gumbel Tree is that the Gumbel Softmax prefers to sample different architectures in the early stage of training. Since the internal parameters ω and the architecture parameters γ are optimized alternatively in Eq. (9), sampling a significantly different architecture each time plays a negative impact on the one-step approximation of the optimal internal parameters ω^*. In contrast, One-Stage Tree gradually optimizes the current architecture with a small difference to ensure the effectiveness.

Furthermore, as discussed in Sect. 4.2.2, we perform the Friedman test (Demšar, 2006) to compare One-Stage Tree with the two variants statistically. The Friedman statistic is 14.2121 and the corresponding p-value is 0.00082. Since the p-value is less than 0.05, we conclude that the trees lead to statistically significant differences. Next, according to the Nemenyi post-hoc test, we further conclude that One-Stage Tree is significantly different from other two variants for a confidence level of $\alpha = 0.05$.



(a) Instance$_0$ that is routed to right child mainly by *BMI*

(b) Instance$_1$ that is routed to right child mainly by *Age*

(c) Instance$_2$ that is routed to left child mainly by *BMI*

Fig. 7 Feature contribution of 3 different instances in Node$_0$

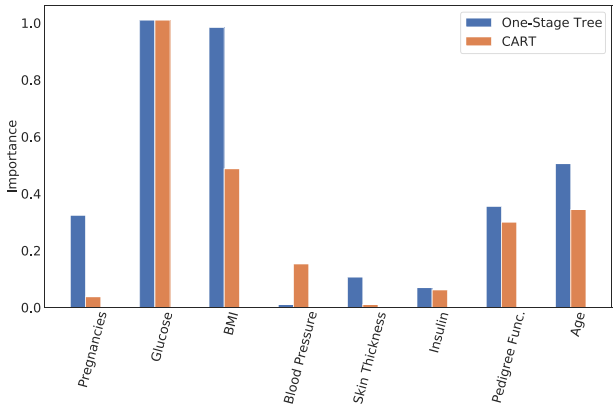


Fig. 8 Feature importance of CART and One-Stage Tree in the dataset PimaIndian

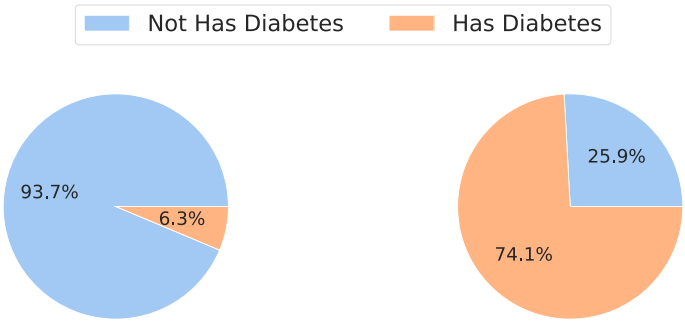
4.5 Discussion of interpretability (RQ4)

The interpretability of *hard* trees helps to understand the mechanism of the tree model and explore the patterns of the dataset. In contrast to *hard* trees, One-Stage Tree cannot be transformed into several 'if-else' rules based on features and thresholds. In One-Stage Tree, different instances may be routed based on different features at the same node. Thus, unlike *hard* trees, it is difficult to visualize One-Stage Tree. In Table 10, we summarize the factors for the interpretability of decision trees. Taking a tree of depth 6 trained on the dataset PimaIndian as an example, we discuss the interpretability of One-Stage Tree.

4.5.1 Routing rule

Hard trees use a greedy algorithm to select a split feature at each node with a threshold. It can be seen that a *hard* tree learns rules from the dataset to route instances to different leaves for prediction. Such 'if-else'-based routing rules are dataset-wise, i.e., each node routes different instances based on the same feature in the inference phase. Thus, they are easy to be understood.

In One-Stage Tree, the routing rules are instance-wise. At the same node, the features contribute differently to the routing of different instances. In the inference phase, the path with the highest probability is chosen, which is equivalent to:



(a) Leaf_1 **(b)** Leaf_2

Fig. 9 Instance distribution in leaves of One-Stage Tree trained on the dataset PimaIndian

$$s(\boldsymbol{x};\omega_i) = \begin{cases} [1,0]^T, & \text{if } (\omega_{i,0} - \omega_{i,1})^T \boldsymbol{x} > 0 \\ [0,1]^T, & \text{o.w.} \end{cases} \tag{18}$$

where $\omega_i \in \mathbb{R}^{(d+1)\times 2}$ and $\omega_{i,0}^T \boldsymbol{x}$ means the logit that $\boldsymbol{x}$ is routed to the left node. Thus, $|(\omega_{i,0,j} - \omega_{i,1,j}) \cdot x_j|$ can be viewed as the contribution of feature j at node i for the instance $\boldsymbol{x}$. We visualize the contribution of each feature to the node router through a pie chart to obtain the instance-wise routing rules. As shown in Fig. 7a, b, different features (i.e., *BMI* and *Age*) make the major contribution to the two instances that are both routed to the right node. Meanwhile, in Fig. 7a, c, two instances are routed to different children mainly by the same feature *BMI*.

Since the traditional *soft* trees (e.g., Soft Decision Tree and Budding Tree) are probabilistic, the instances are not directly routed to child nodes, which is difficult to interpret. Compared to probabilistic trees, One-Stage Tree directly routes an instance to a child node rather than weighting it by probabilities. Thus, One-Stage Tree is more interpretable.

4.5.2 Feature importance

Feature importance is an important way to explore data patterns. For *hard* trees, feature importance is calculated as the decrease in node impurity weighted by the probability of reaching that node. One-Stage Tree also provides the feature importance to explore the dataset. Due to the same discrete architecture of *hard* trees, One-Stage Tree calculates feature importance in the same way but weights the decreased impurity by $|\omega_{i,0,j} - \omega_{i,1,j}|$ for each feature j at node i.

In Fig. 8, we show the feature importance of One-Stage Tree and CART in the dataset PimaIndian. We observe that two features, i.e., Glucose and BMI, play an important role in both trees.

4.5.3 Node instance distribution, node impurity, and predicted value

Benefiting from the discrete architecture, One-Stage Tree deterministically routes each instance to a leaf for prediction. As a result, the instance distribution and node impurity can be computed in the same way as for the *hard* tree. In Fig. 9, we show the instance

distribution in two leaves of One-Stage Tree trained on the dataset PimaIndian. Furthermore, we can calculate the node impurity and other criteria according to the instance distribution.

4.6 Discussion

Tabular data is generally dominated by tree models. One-Stage Tree can be seen as an attempt at deep learning on tabular data. Although One-Stage Tree inherits the advantages of the decision tree, it has the disadvantages of deep learning. For example, due to the use of gradient descent for optimization, the efficiency of constructing One-Stage Tree needs to be improved. Moreover, One-Stage Tree is a single decision tree. To achieve better performance, we need to further construct a tree ensemble model based on One-Stage Tree. Due to the continuous nature, we can perform joint tuning on all trees.

5 Conclusion and future work

In this work, we proposed One-Stage Tree, which retains the advantages of traditional decision trees as the inference model and improving learning through end-to-end training with back-propagation. Based on the continuous relaxation of *soft* trees, One-Stage Tree optimizes the node and architecture parameters jointly through a bilevel optimization problem. Moreover, One-Stage Tree leverages the reparameterization trick and proximal iterations to keep the tree discrete when training the continuous parameters. As a benefit, the performance gap between training and testing is reduced and the interpretability is maintained. The experimental results show that One-Stage Tree is effective on both classification and regression tasks and can outperform CART and the existing *soft* trees.

In the future, we plan to improve the efficiency of *soft* trees on GPU by parallelizing sequential decisions. Additionally, using ensemble methods such as bagging and boosting to build a forest of One-Stage Tree is also an important future work.

Appendix 1: Derivation of Equation (13) in budding tree

Similar to One-Stage Tree, Budding Tree (Irsoy et al., 2014) jointly optimizes the node and architecture parameters in the learning phase. Formally, for regression, given the training set $\mathcal{D}_{train} = \{\boldsymbol{x}_m, \boldsymbol{y}_m\}_{i=m}^{N}$, the optimization objective of Budding Tree is as follows. For simplicity, we omit the regularization term on γ_i.

$$\min_{\theta} \mathcal{L} = \sum_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}_{train}} \frac{1}{2}(f_r(\boldsymbol{x};\theta) - \boldsymbol{y})^2 \tag{19}$$

$f_r(\boldsymbol{x};\theta)$ denotes the response at the root node calculated using Eq. (2) recursively. Specifically, the response at node i is calculated as follows:

$$f_i(\boldsymbol{x};\theta) = \left(1-\gamma_i\right)\cdot \upsilon_i + \gamma_i \cdot \left[s\left(\boldsymbol{x};\omega_i\right)_0 \cdot f_{L(i)}(\boldsymbol{x};\theta) + s\left(\boldsymbol{x};\omega_i\right)_1 \cdot f_{R(i)}(\boldsymbol{x};\theta)\right]$$

where $\gamma_i \in [0,1]$ denotes the architecture parameter. $\gamma_i = 0$ indicates that node i is a leaf.

Budding Tree employs stochastic gradient-descent (the error is computed with respect to a single instance $\boldsymbol{x}$) to solve Eq. (19). Next, we focus on the optimization of the architecture

parameters γ. We omit θ for brevity. Define pa(i) as the parent of node i and $\delta_i = \partial\mathcal{L}/\partial f_i$ as the *responsibility* of node i, by deriving $\mathcal{L}$ w.r.t. γ_i, we have:

$$\begin{aligned}\frac{\partial\mathcal{L}}{\partial\gamma_i} &= \frac{\partial\mathcal{L}}{\partial f_i(\boldsymbol{x})}\frac{\partial f_i(\boldsymbol{x})}{\partial\gamma_i} \\ &= \delta_i(-\gamma_i + s\left(\boldsymbol{x};\omega_i\right)_0 \cdot f_{L(i)}(\boldsymbol{x}) + s\left(\boldsymbol{x};\omega_i\right)_1 \cdot f_{R(i)}(\boldsymbol{x}))\end{aligned} \tag{20}$$

If node i is the root node, we can easily get $\delta_i = f_i(\boldsymbol{x}) - \boldsymbol{y}$ according to Eq. (19). For other nodes, since $f_i(\boldsymbol{x})$ is calculated recursively, we can compute δ_i as follows:

$$\delta_i = \frac{\partial\mathcal{L}}{\partial f_{pa(i)}(\boldsymbol{x})}\frac{\partial f_{pa(i)}(\boldsymbol{x})}{\partial f_i(\boldsymbol{x})} = \begin{cases}\delta_{pa(i)} \times \gamma_i \times s\left(\boldsymbol{x};\omega_i\right)_0, & \text{if i is the left child} \\ \delta_{pa(i)} \times \gamma_i \times s\left(\boldsymbol{x};\omega_i\right)_1, & \text{if } i \text{ is the right child}\end{cases} \tag{21}$$

In summary, we conclude that Eq. (13) holds based on Eqs. (20) and (21).

Appendix 2: Hyperparemeter search space of standard ML methods

To evaluate the performance of standard ML methods including MLP and SVM more accurately, we perform hyperparameter optimization with grid search. The hyperparameter search space of MLP and SVM are as follows.

- *MLP*
  - *activation*: tanh, relu, logistic
  - *solver*: sgd, adam
  - *alpha (L2 penalty parameter)*: $10^{-4}, 10^{-3}, 10^{-2}$
  - *early stopping*: true, false
  - *hidden layer sizes*: 100, (100, 150), (50, 100, 100)
- *LinearSVC (SVM for Classification)*
  - *penalty*: L1, L2
  - *loss*: hinge, squared hinge
  - *dual*: true, false
  - *tol*: $10^{-5}, 10^{-4}, 10^{-3}$
  - *C (Regularization parameter)*: 1, 10, 100
- *LinearSVR (SVM for Regression)*
  - *dual*: true, false
  - *loss*: epsilon insensitive, squared epsilon insensitive
  - *tol*: $10^{-5}, 10^{-4}, 10^{-3}$
  - *C (Regularization parameter)*: 1, 10, 100

Appendix 3: Details of gumbel tree

Gumbel Tree is a variant of One-Stage Tree that discretizes the tree architecture by the Gumbel Softmax instead of the lazy proximal step (Yao et al., 2020). Algorithm 2 shows the overall workflow of Gumbel Tree. As shown in Algorithm 2, both Gumbel Tree and One-Stage Tree

solve the bilevel optimization problem in Eq. (9) by alternatively optimizing the node and architecture parameters. The main difference is the optimization strategy of the architecture parameters γ (Line 3 and Line 6). The optimization processes of (ω, υ) (Line 4 and Line 5) are exactly the same as One-Stage Tree.

Unlike One-Stage Tree that searches the architecture parameters γ via proximal iterations, Gumbel Tree first obtains the discrete architecture $\bar{\gamma}^{(k)}$ using the Gumbel Softmax (Line 3) (Maddison et al., 2014). Based on the discrete architecture $\bar{\gamma}^{(k)}$, we further calculate the optimal leaf values in closed-form (Line 4) and update the internal parameters (Line 5) on the training set. After forwarding ω one step, we optimize $\gamma^{(k)}$ with gradient decent on the validation set (Line 6). Additionally, both Gumbel Tree and One-Stage Tree employ the early-stopping mechanism to optimize the architecture parameters γ.

Algorithm 2 Gumbel Tree

1: Initialize Tree Parameters $\theta = (\omega, \upsilon, \gamma)$ according to the constraints;
2: **while** not converged **do**
3: Get *discrete* architecture: $\bar{\gamma}^{(k)} = gumbel\_softmax(\gamma^{(k)})$;
//The optimization processes of (ω, υ) are the same as One-Stage Tree.
4: Update *optimal* leaves $\upsilon^{(k)}$ in closed-form by Equation (17) with the training set;
5: Update $\omega^{(k)}$ by the MC approximation of $\nabla_{\omega^{(k)}} \mathbb{E}_{q_{\omega^{(k)}}(s|\boldsymbol{x})}[\mathcal{L}_{\text{train}}(\omega^{(k)}, \upsilon^{(k)}, \bar{\gamma}^{(k}$
Equation (11);
6: Update $\gamma^{(k+1)} = \gamma^{(k)} - \epsilon \nabla_{\bar{\gamma}^{(k)}} \mathcal{L}_{\text{val}}(\omega^{(k+1)}, \upsilon^{(k+1)}, \bar{\gamma}^{(k)})$;
7: **end while**
8: **return** Tree Parameters θ.

Acknowledgments This work was supported in part by the National Natural Science Foundation of China (#U1811461 and #62102177), the Natural Science Foundation of Jiangsu Province, China (#BK20210181), the National Key R&D Program of China (#2019YFC1711000), and the Collaborative Innovation Center of Novel Software Technology and Industrialization, Jiangsu, China.

References

Bai, Y., Wang, Y. X., & Liberty, E. (2018). Proxquant: Quantized neural networks via proximal operators. In *International conference on learning representations*.

Blum, A., Haghtalab, N., & Procaccia, A. D. (2015). Variational dropout and the local reparameterization trick. In *NIPS*.

Bottou, L. (2012). Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade* (pp. 421–436). Springer.

Bousquet, O., Chapelle, O., & Hein, M. (2004). Measure based regularization. In *Advances in Neural Information Processing Systems* (pp. 1221–1228).

Breiman, L., Friedman, J., Stone, C. J., & Olshen, R. A. (1984). *Classification and regression trees*. CRC Press.

Chen, T., & Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (pp. 785–794).

Courbariaux, M., Bengio, Y., & David, J. P. (2015). Binaryconnect: Training deep neural networks with binary weights during propagations. *Advances in Neural Information Processing Systems, 28,* 3123–3131.
Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research, 7*(1), 1–30.
Finn, C., Abbeel, P., & Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th international conference on machine learning* (vol. 70, pp. 1126–1135).
Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *Annals of Statistics,* 1189–1232.
Gumbel, E. J. (1954). Statistical theory of extreme values and some practical applications: A series of lectures (vol. 33). US Government Printing Office.
Hehn, T. M., Kooij, J. F., & Hamprecht, F. A. (2019). End-to-end learning of decision trees and forests. *International Journal of Computer Vision*, 1–15 .
Hou, L., Yao, Q., & Kwok, J. T. (2017). Loss-aware binarization of deep networks. In *5th international conference on learning representations, ICLR 2017, Toulon, France April 24–26, 2017, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=S1oWlN9ll
Irsoy, O., Yıldız, O. T., Alpaydın, E. (2012). Soft decision trees. In *Proceedings of the 21st international conference on pattern recognition (ICPR2012)* (pp. 1819–1822). IEEE.
Irsoy, O., Yildiz, O. T., & Alpaydin, E. (2014). Budding trees. In *2014 22nd international conference on pattern recognition* (pp. 3582–3587). IEEE.
Jang, E., Gu, S., & Poole, B. (2016). Categorical reparameterization with gumbel-softmax. Preprint arXiv: 1611.01144
Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., et al. (2017). Lightgbm: A highly efficient gradient boosting decision tree. *Advances in Neural Information Processing Systems, 30,* 3146–3154.
Kingma, D., & Ba, J. (2014). Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.
Kotsiantis, S. B. (2013). Decision trees: A recent overview. *Artificial Intelligence Review, 39*(4), 261–283.
Li, Q., Wen, Z., & He, B. (2020). Practical federated gradient boosting decision trees. In *Proceedings of the AAAI conference on artificial intelligence* (vol. 34, pp. 4642–4649).
Liu, H., Simonyan, K., & Yang, Y. (2018). Darts: Differentiable architecture search. In *International conference on learning representations*.
Maddison, C. J., Tarlow, D., & Minka, T. (2014). A* sampling. *Advances in Neural Information Processing Systems, 27,* 3086–3094.
Metropolis, N., & Ulam, S. (1949). The monte carlo method. *Journal of the American Statistical Association, 44*(247), 335–341.
Mukkamala, M. C., & Hein, M. (2017). Variants of rmsprop and adagrad with logarithmic regret bounds. In *International conference on machine learning* (pp. 2545–2553). PMLR.
Nemenyi, P. (1963). Distribution-free multiple comparisons. Princeton University.
Norouzi, M., Collins, M., Johnson, M. A., Fleet, D. J., & Kohli, P. (2015). Efficient non-greedy optimization of decision trees. In *Advances in neural information processing systems* (pp. 1729–1737).
Parikh, N., & Boyd, S. (2014). Proximal algorithms. *Foundations and Trends in Optimization, 1*(3), 127–239.
Prechelt, L. (1998). Early stopping-but when? In *Neural networks: Tricks of the trade* (pp. 55–69). Springer.
Quinlan, J. (1996). Bagging, boosting, and c4. s. In *Proceedings of the thirteenth national conference on Artificial intelligence* (vol. 1, pp. 725–730).
Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning, 1*(1), 81–106.
Wu, L., Rao, Y., Liang, H., Nazir, A., et al. (2020). Dtca: Decision tree-based co-attention networks for explainable claim verification. In *Proceedings of the 58th annual meeting of the association for computational linguistics* (pp. 1024–1035).
Xiao, L. (2010). Dual averaging methods for regularized stochastic learning and online optimization. *The Journal of Machine Learning Research, 11,* 2543–2596.
Yao, Q., Chen, X., Kwok, J. T., Li, Y., & Hsieh, C. J. (2020). Efficient neural interaction function search for collaborative filtering. In *Proceedings of The web conference 2020* (pp. 1660–1670).
Yao, Q., Xu, J., Tu, W. W., & Zhu, Z. (2020). Efficient neural architecture search via proximal iterations. In *AAAI* (pp. 6664–6671).