Flexible Modeling and Multitask Learning using Differentiable Tree Ensembles

Citation: Ibrahim, Shibal, Hazimeh, Hussein and Mazumder, Rahul. 2022. "Flexible Modeling and Multitask Learning using Differentiable Tree Ensembles."

As Published: https://doi.org/10.1145/3534678.3539412

Publisher: ACM|Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining USB

Persistent URL: https://hdl.handle.net/1721.1/146341

Version: Final published version: final published article, as it appeared in a journal, conference proceedings, or other formally published context

Flexible Modeling and Multitask Learning using Differentiable Tree Ensembles

Shibal Ibrahim
shibal@mit.edu
Massachusetts Institute of Technology
Cambridge, MA, USA

Hussein Hazimeh
hazimeh@mit.edu
Massachusetts Institute of Technology
Cambridge, MA, USA

Rahul Mazumder
rahulmaz@mit.edu
Massachusetts Institute of Technology
Cambridge, MA, USA

ABSTRACT

Decision tree ensembles are widely used and competitive learning models. Despite their success, popular toolkits for learning tree ensembles have limited modeling capabilities. For instance, these toolkits support a limited number of loss functions and are restricted to single task learning. We propose a flexible framework for learning tree ensembles, which goes beyond existing toolkits to support arbitrary loss functions, missing responses, and multi-task learning. Our framework builds on differentiable (a.k.a. soft) tree ensembles, which can be trained using first-order methods. However, unlike classical trees, differentiable trees are difficult to scale. We therefore propose a novel tensor-based formulation of differentiable trees that allows for efficient vectorization on GPUs. We introduce FASTEL: a new toolkit (based on Tensorflow 2) for learning differentiable tree ensembles. We perform experiments on a collection of 28 real open-source and proprietary datasets, which demonstrate that our framework can lead to 100x more compact and 23% more expressive tree ensembles than those obtained by popular toolkits.

CCS CONCEPTS

• **Computing methodologies → Classification and regression trees**; **Multi-task learning**; **Ensemble methods**.

KEYWORDS

tree ensemble learning, differentiable trees, soft-trees, zero-inflated models, negative binomial regression, multi-task learning

ACM Reference Format:
Shibal Ibrahim, Hussein Hazimeh, and Rahul Mazumder. 2022. Flexible Modeling and Multitask Learning using Differentiable Tree Ensembles. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), August 14–18, 2022, Washington, DC, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3534678.3539412

1 INTRODUCTION

Decision tree ensembles are popular models that have proven successful in various machine learning applications and competitions [16, 20]. Besides their competitive performance, decision trees are appealing in practice because of their interpretability, robustness to outliers, and ease of tuning [24]. Training a decision tree naturally requires solving a combinatorial optimization problem, which can be challenging to scale to large instances. In practice, greedy heuristics are commonly used to get feasible solutions to the combinatorial problem; for example CART [10], C5.0 [42], and OC1 [39]. By building on these heuristics, highly scalable toolkits for learning tree ensembles have been developed, e.g., XGBoost [16] and LightGBM [31]. These toolkits are considered a defacto standard for training tree ensembles and have demonstrated success in various domains.

Despite their success, popular toolkits for learning tree ensembles lack modeling flexibility. For example, these toolkits support a limited set of loss functions, which may not be suitable for the application at hand. Moreover, these toolkits are limited to single task learning. In many modern applications, it is desirable to solve multiple, related machine learning tasks. In such applications, multi-task learning, i.e., learning tasks simultaneously, may be a more appropriate choice than single task learning [15, 17, 23, 33]. If the tasks are sufficiently related, multi-task learning can boost predictive performance by leveraging task relationships during training.

In this paper, we propose a flexible modeling framework for training tree ensembles that addresses the aforementioned limitations. Specifically, our framework allows for training tree ensembles with any differentiable loss function, enabling the user to seamlessly experiment with different loss functions and select what is suitable for the application. Moreover, our framework equips tree ensembles with the ability to perform multi-task learning. To achieve this flexibility, we build up on differentiable trees [25, 32], which can be trained with first-order (stochastic) gradient methods.

Previously, soft tree ensembles have been predominantly explored for classification tasks with cross-entropy loss. In such tasks, they were found to be more expressive and compact than traditional tree ensembles [25]. However, state-of-the-art toolkits, e.g., TEL [25], are slow as they only support CPU training and are difficult to customize. Our proposed framework goes beyond the latter work on soft trees by supporting a diverse collection of loss functions for classification, regression, Poisson regression, zero-inflated models, overdispersed distributions, and multi-task learning. The framework also offers seamless support for arbitrary loss functions: the user can modify the loss function with a single line. We empirically observe that the ability to customize the loss can lead to a significant reduction in ensemble sizes (up to 20x). We also propose a custom tensor-based formulation of differentiable tree ensembles, leading to more efficient training on CPUs (10×) and GPUs (20×).

Contributions. Our contributions can be summarized as follows. **(i)** We propose a flexible framework for training differentiable tree ensembles with seamless support for new loss functions. **(ii)** We introduce a novel, tensor-based formulation for differentiable tree ensembles that allows for efficient training on GPUs. Existing toolkits e.g., TEL [25], only support CPU training. **(iii)** We extend differentiable tree ensembles to multi-task learning settings by introducing

KDD '22, August 14–18, 2022, Washington, DC, USA
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9385-0/22/08.
https://doi.org/10.1145/3534678.3539412

a new regularizer that allows for soft parameter sharing across tasks — current popular multi-task tree ensemble toolkits e.g., RF [9], GRF [3] do not allow for soft information sharing. **(iv)** We introduce FASTEL[1] — a new toolkit (based on Tensorflow 2.0) for learning differentiable tree ensembles — and perform experiments on a collection of 28 open-source and real-world datasets, demonstrating that our toolkit can lead to 100x more compact ensembles and up to 23% improvement in out-of-sample performance, compared to tree ensembles learnt by popular toolkits such as XGBoost [16].

Organization. We summarize related work in Section 2. We then briefly review differentiable trees in Section 3.1. In Section 3.2, we present a careful tensor-based formulation of the tree ensemble, which allows for efficient training on both CPUs and GPUs. In Section 4, we discuss important examples of loss functions supported by our framework. In Section 5 , we propose a multitask learning formulation based on differentiable tree ensembles. In Section 6, we present an empirical study on a collection of real-world datasets.

2 RELATED WORK

Learning binary trees has been traditionally done in three broad ways. The first approach relies on greedy construction and/or optimization via methods such as CART [10], C5.0 [42], OC1 [39], TAO [13]. These methods optimize a criterion at the split nodes based on the samples routed to each of the nodes. The second approach considers probabilistic relaxations/decisions at the split nodes and performs end-to-end learning with first order methods [21, 28, 35]. The third approach considers optimal trees with mixed integer formulations and jointly optimizes over all discrete/continuous parameters with MIP solvers [5–7, 52]. Each of the three approaches have their pros and cons. The first approach is highly scalable because of greedy heuristics. In many cases, the tree construction uses a splitting criterion different from the optimization objective [10] (e.g., gini criterion when performing classification) possibly resulting in sub-optimal performance. The second approach is also scalable but principled pruning in probabilistic trees remains an open research problem. The third approach scales to small datasets: some of the largest instances reported in prior work include number of samples $N \sim 10^4$, features $p \sim 10$ and tree depths $d \sim 4$.

Jointly optimizing over an ensemble of classical decision trees is a hard combinatorial optimization problem [27]. Historically, tree ensembles have been trained with two methods. One approach uses greedy heuristics for individual trees with ensembling done via bagging/boosting. For example, individual trees are trained with CART on bootstrapped samples of the data e.g., random forests (RF) [9] and its variants [3, 22]; or sequentially trained with gradient boosting: Gradient Boosting Decision Trees [24] and efficient variants [16, 31, 41, 44]. Despite the success of ensemble methods, interesting challenges remain: (i) RF tend to under-perform gradient boosting methods such as XGBoost [16]. (ii) The tree ensembles are typically very large, making them complex and difficult to interpret. Recent work by [13, 51] improve RF with local optimization methods such as alternating minimization. However, their implementation is not open-source. (iii) Open-source toolkits for gradient boosting are limited in terms of flexibility. They lack support for multi-task learning, missing responses or customized loss functions. Modifying these toolkits for custom applications often require significant effort, technical expertise and research investment.

The alternative approach for tree ensemble learning extends probabilistic/differentiable trees and performs end-to-end learning [25, 32]. These works build upon the idea of hierarchical mixture of experts introduced by [30] and further developed by [21, 28, 45] for greedy construction of trees. Some of these works [25, 32] propose using differentiable trees as an output layer in a cascaded neural network for combining feature representation learning along with tree ensemble learning for classification. In this paper, we focus on learning tree (ensembles) with hyperplane splits and constant leaf nodes–this allows us to expand the scope of trees to flexible loss functions, and develop specialized implementations that can be more efficient. One might argue that probabilistic trees are harder to interpret and suffer from slower inference as a sample must follow each root-leaf path, lacking *conditional computation* present in classical decision trees. However, [25] proposed a principled way to get conditional inference in probabilistic trees by introducing a new activation function: this allows for routing samples through small parts of the tree similar to classical decision trees. We refer the reader to Section 6.1 for a study on a single tree and highlight that a soft tree with hyperplane splits and conditional inference has similar interpretability as that of a classical tree with hyperplane splits — see Figure 2. Additionally, a soft tree can lead to smaller optimal depths—see Supplemental Section S1.1.

End-to-end learning with differentiable tree ensembles appears to have several advantages. (i) Training is easy to set up in public deep learning frameworks, e.g., Tensorflow [1] and PyTorch [40]. Differentiable tree ensembles allow for flexibility in loss functions without the need for specialized algorithms. For example, mixture likelihoods can be easily implemented in Tensorflow Probability [18], which allows for handling zero-inflated data. Similarly, multi-task loss objectives can also be handled. (ii) With a careful implementation, the tree ensemble can be trained efficiently on GPUs — this is not possible with earlier toolkits such as TEL [25]. (iii) Differentiable trees can lead to more expressive and compact ensembles [25]. This can have important implications for interpretability, latency and storage requirements during inference.

3 OPTIMIZING TREE ENSEMBLES

In this section, we first introduce background on soft trees. Later, in Section 3.2, we discuss the tensor-based formulation to perform efficient training on both CPUs and GPUs. Finally, in Section 3.3, we briefly discuss our FASTEL toolkit.

We assume a supervised multi-task learning setting, with input space $\mathcal{X} \subseteq \mathbb{R}^p$ and output space $\mathcal{Y} \subseteq \mathbb{R}^k$. We learn a mapping $f : \mathbb{R}^p \to \mathbb{R}^k$, from input space $\mathcal{X}$ to output space $\mathcal{Y}$, where we parameterize function f with a differentiable tree ensemble. We consider a general optimization framework where the learning objective is to minimize any differentiable loss function $g : \mathbb{R}^p \times \mathbb{R}^k \to \mathbb{R}$. The framework can accommodate different loss functions arising in different applications and perform end-to-end learning with tree ensembles.

[1] https://github.com/ShibalIbrahim/FASTEL

Notation. For an integer $n \geq 1$, let $[n] := \{1, 2, ..., n\}$. We let 1_m denote the vector in $\mathbb{R}^m$ with all coordinates being 1. For a matrix $\boldsymbol{B} = ((B_{ij})) \in \mathbb{R}^{m \times n}$, let the j-th column be denoted by $\boldsymbol{B}_j := [B_{1j}, B_{2j}, ..., B_{mj}]^T \in \mathbb{R}^m$ for $j \in [n]$. A dot product between two vectors $\boldsymbol{u}, \boldsymbol{v} \in R^m$ is denoted as $\boldsymbol{u} \cdot \boldsymbol{v}$. A dot product between a matrix $\boldsymbol{U} \in R^{m,n}$ and a vector $\boldsymbol{v} \in \mathbb{R}^m$ is denoted as $\boldsymbol{U} \cdot \boldsymbol{v} = \boldsymbol{U}^T v \in \mathbb{R}^n$. A dot product between a tensor $\boldsymbol{\mathcal{U}} \in \mathbb{R}^{p,m,n}$ and a vector $\boldsymbol{v} \in \mathbb{R}^m$ is denoted as $\boldsymbol{\mathcal{U}} \cdot \boldsymbol{v} = \boldsymbol{\mathcal{U}}^T \boldsymbol{v} \in \mathbb{R}^{p,n}$ where the transpose operation of a tensor $\boldsymbol{\mathcal{U}}^T \in \mathbb{R}^{p,n,m}$ permutes the last two dimensions of the tensor.

3.1 Preliminaries and Setup

We learn an ensemble of m differentiable trees. Let f^j be the jth tree in the ensemble. For easier exposition, we consider a single-task regression or classification setting—see Section 5 for an extension to the multi-task setting. In a regression setting $k = 1$, while in multi-class classification setting $k = C$, where C is the number of classes. For an input feature-vector $\boldsymbol{x} \in \mathbb{R}^p$, we learn an additive model with the output being sum over outputs of all the trees:

$$\boldsymbol{f}(\boldsymbol{x}) = \sum_{j=1}^{m} \boldsymbol{f}^j(\boldsymbol{x}). \tag{1}$$

The output, $\boldsymbol{f}(\boldsymbol{x})$, is a vector in $\mathbb{R}^k$ containing raw predictions. For multiclass classification, mapping from raw predictions to $\mathcal{Y}$ is done by applying a softmax function on the vector $\boldsymbol{f}(\boldsymbol{x})$ and returning the class with the highest probability. Next, we introduce the key building block of the approach: differentiable decision tree.

Differentiable decision trees for modelling $\boldsymbol{f}^j$. Classical decision trees perform hard sample routing, i.e., a sample is routed to exactly one child at every splitting node. Hard sample routing introduces discontinuities in the loss function, making trees unamenable to continuous optimization. Therefore, trees are usually built in a greedy fashion. In this section, we first introduce a single soft tree proposed by [30], which is utilized in [8, 21, 28] and extended to soft tree ensembles in [25, 26, 32]. A soft tree is a variant of a decision tree that performs soft routing, where every internal node can route the sample to the left and right simultaneously, with different proportions. This routing mechanism makes soft trees differentiable, so learning can be done using gradient-based methods. Notably, [25] introduced a new activation function for soft trees that allowed for conditional computation while preserving differentiability.

Let us fix some $j \in [m]$ and consider a single tree $\boldsymbol{f}^j$ in the additive model (1). Recall that $\boldsymbol{f}^j$ takes an input sample and returns an output vector (logit), i.e., $\boldsymbol{f}^j : X \in \mathbb{R}^p \rightarrow \mathbb{R}^k$. Moreover, we assume that $\boldsymbol{f}^j$ is a perfect binary tree with depth d. We use the sets $\mathcal{I}^j$ and $\mathcal{L}^j$ to denote the internal (split) nodes and the leaves of the tree, respectively. For any node $i \in \mathcal{I}^j \cup \mathcal{L}^j$, we define $A^j(i)$ as its set of ancestors and use the notation $\boldsymbol{x} \rightarrow i$ for the event that a sample $\boldsymbol{x} \in \mathbb{R}^p$ reaches i.

Routing. Internal (split) nodes in a differentiable tree perform soft routing, where a sample is routed left and right with different proportions. This soft routing can be viewed as a probabilistic model. Although the sample routing is formulated with a probabilistic model, the final prediction of the tree $\boldsymbol{f}$ is a deterministic function as it assumes an expectation over the leaf predictions. Classical decision trees are modeled with either axis-aligned splits [10, 42] or hyperplane (a.k.a. oblique) splits [39]. Soft trees are based on hyperplane splits, where the routing decisions rely on a linear combination of the features. Particularly, each internal node $i \in \mathcal{I}^j$ is associated with a trainable weight vector $\boldsymbol{w}_i^j \in \mathbb{R}^p$ that defines the node's hyperplane split. Given a sample $\boldsymbol{x} \in \mathbb{R}^p$, the probability that internal node i routes $\boldsymbol{x}$ to the left is defined by $S(\boldsymbol{w}_i^j \cdot \boldsymbol{x})$, where $S : R \rightarrow [0, 1]$ is an activation function. Now we discuss how to model the probability that $\boldsymbol{x}$ reaches a certain leaf l. Let $[l \swarrow i]$ (resp. $[i \searrow l]$) denote the event that leaf l belongs to the left (resp. right) subtree of node $i \in \mathcal{I}^j$. Assuming that the routing decision made at each internal node in the tree is independent of the other nodes, the probability that $\boldsymbol{x}$ reaches l is given by:

$$P^j(\{x \rightarrow l\}) = \prod_{i \in A^j(l)} r_{i,l}^j(\boldsymbol{x}), \tag{2}$$

where $r_{i,l}^j(\boldsymbol{x})$ is the probability of node i routing $\boldsymbol{x}$ towards the subtree containing leaf l, i.e., $r_{i,l}^j(x) := S(\boldsymbol{w}_i^j \cdot \boldsymbol{x}) 1[l \swarrow i] \odot (1 - S(\boldsymbol{w}_i^j \cdot \boldsymbol{x})) 1[i \searrow l]$. Popular choices for S include logistic function [21, 26, 30, 32, 45] and smooth-step function (for conditional computation as in classical trees with oblique splits) [25]. Next, we define how the root-to-leaf probabilities in (2) can be used to make the final prediction of the tree.

Prediction. As with classical decision trees, we assume that each leaf stores a weight vector $\boldsymbol{o}_l^j \in R^k$ (learned during training). Note that, during the forward pass, $\boldsymbol{o}_l^j$ is a constant vector, meaning that it is not a function of the input sample(s). For a sample $\boldsymbol{x} \in \mathbb{R}^p$, we define the prediction of the tree as the expected value of the leaf outputs, i.e.,

$$\boldsymbol{f}^j(\boldsymbol{x}) = \sum_{l \in L} P^j(\{\boldsymbol{x} \rightarrow l\}) \boldsymbol{o}_l^j. \tag{3}$$

3.2 Efficient Tensor Formulation

Current differentiable tree ensemble proposals and toolkits, for example deep neural decision forests[2] [32] and TEL [25] model trees individually. This leads to slow CPU-training times and makes these implementations hard to vectorize for fast GPU training. In fact, TEL [25] does not support GPU training. We propose a tensor-based formulation of a tree ensemble that parallelizes routing decisions in nodes across the trees in the ensemble. This can lead to 10x faster CPU training times if the ensemble sizes are large e.g., 100. Additionally, the tensor-based formulation is GPU-friendly, which provides an additional 40% faster training times. See Figure 1 for a timing comparison on CPU training without/with tensor formulation. Next, we outline the tensor-based formulation.

We propose to model the internal nodes in the trees across the ensemble jointly as a "supernodes". In particular, an internal node $i \in \mathcal{I}^j$ at depth d in all trees can be condensed together into a supernode $i \in \mathcal{I}$. We define a learnable weight matrix $\boldsymbol{W}_i \in \mathbb{R}^{p,m}$, where each j-th column of the weight matrix contains the learnable weight vector $\boldsymbol{w}_i^j$ of the original j-th tree in the ensemble. Similarly, the leaf nodes are defined to store a learnable weight matrix $\boldsymbol{O}_l \in \mathbb{R}^{m,k}$, where each j-th row contains the learnable weight vector $\boldsymbol{o}_l^j$

[2] https://keras.io/examples/structured_data/deep_neural_decision_forests/

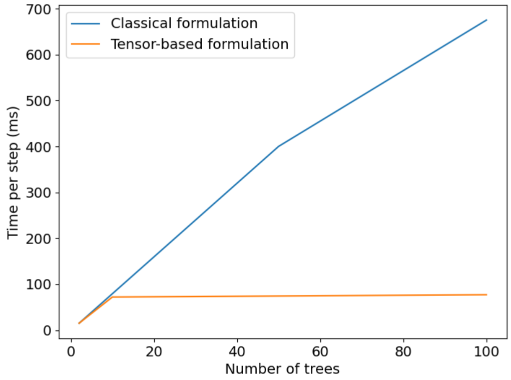

Figure 1: Timing comparison of classical formulation against our tensor-based formulation of a tree ensemble. Classical formulation models trees in the ensemble individually as pointed out in Section 3.2. Tensor-based formulation with CPU training is up to 10× faster than classical formulation. Tensor-based formulation with GPU training leads to an additional 40% improvement, leading to an effective 20× gain over classical formulation.

in the original j-th tree in the ensemble. The prediction of the tree with supernodes can be written as

$$f(\boldsymbol{x}) = \left(\sum_{l \in L} \boldsymbol{O}_l \odot \prod_{i \in A(l)} \boldsymbol{R}_{i,l}\right) \cdot \mathbf{1}_m \tag{4}$$

where $\odot$ denotes the element-wise product, $\boldsymbol{R}_{i,l} = S(\boldsymbol{W}_i \cdot \boldsymbol{x})1[l \swarrow i] \odot (1 - S(\boldsymbol{W}_i \cdot \boldsymbol{x}))1[i \searrow l] \in \mathbb{R}^{m,1}$ and the activation function S is applied element-wise. This formulation of tree ensembles via supernodes allows for sharing of information across tasks via tensor formulation in multi-task learning — see Section 5 for more details.

3.3 Toolkit

Our FASTEL toolkit is built in Tensorflow (TF) 2.0 and integrates with Tensorflow-Probability. The toolkit allows the user to write a custom loss function, and TF provides automatic differentiation. Popular packages, such as XGBoost, require users to provide first/second order derivatives. In addition to writing a custom loss, the user can select from a wide range of predefined loss and likelihood functions from Tensorflow-Probability. By relying on TF in the backend, our toolkit can easily exploit distributed computing. It can also run on multiple CPUs or GPUs, and on different platforms, including mobile platforms.

4 FLEXIBLE LOSS FUNCTIONS

Our framework can handle any differentiable loss function. Such flexibility is important as various applications require flexibility in loss functions beyond what is provided by current tree ensemble learning toolkits. Our framework is built on Tensorflow, which allows for scalable gradient-based optimization. This coupled with our efficient differentiable tree ensemble formulation gives a powerful toolkit to seamlessly experiment with different loss functions and select what is suitable for the intended application. A few examples of flexible distributions that our toolkit supports — due to compatibility with Tensorflow-Probability — are normal, Poisson, gamma, exponential, mixture distributions e.g., zero-inflation models [36], and compound distributions e.g., negative binomial [49]. Other loss functions such as those robust to outliers [4] can also be handled. To demonstrate the flexibility of our framework, we deeply investigate two specific examples: zero-inflated Poisson and negative binomial regression. These cannot be handled by the popular gradient boosting toolkits such as XGBoost [16] and LightGBM [31].

Zero-inflated Poisson Regression. Zero-inflation occurs in many applications, e.g., understanding alcohol and drug abuse in young adults [29], characterizing undercoverage and overcoverage to gauge the on-going quality of the census frames [50], studying popularity of news items on different social media platforms [38], financial services applications [36] etc. Despite the prevalence of these applications, there has been limited work on building decision tree-based approaches for zero-inflated data perhaps due to a lack of support public toolkits. Therefore, practitioners either resort to Poisson regression with trees or simpler linear models to handle zero-inflated responses. A Poisson model can lead to sub-optimal performance due to the limiting equidispersion constraint (mean equals the variance). Others take a two-stage approach [12], where a classification model distinguishes the zero and non-zero and a second model is used to model the non-zero responses. This can be sup-optimal as errors in the first model can deteriorate the performance of the second model. We employ a more well-grounded approach by formulating the joint mixture model, where one part of the model tries to learn the mixture proportions (zero vs non-zero) and the other part models the actual non-zero responses. Such a mixture model permits a differentiable loss function when both components of the model are parameterized with differentiable tree ensembles and can be optimized with gradient descent method in an end-to-end fashion without the need for a custom solver. We provide an extensive study with our framework on small to large-scale real world zero-inflated datasets and demonstrate that such flexibility in distribution modeling can lead to significantly more compact and expressive tree ensembles. This has large implications for faster inference, storage requirements and interpretability.

We briefly review Poisson regression and then dive into zero-inflated Poisson models. Poisson regression stems from the generalized linear model (GLM) framework for modeling a response variable in the exponential family of distributions. In general, GLM uses a link function to provide the relationship between the linear predictors, $\boldsymbol{x}$ and the conditional mean of the density function: $g[\mathbb{E}(y|\boldsymbol{x})] = \boldsymbol{\beta} \cdot \boldsymbol{x}$, where $\boldsymbol{\beta}$ are parameters and $g(\cdot)$ is the link function. When responses y_n (for $n \in [N]$), are independent and identically distributed (i.i.d.) and follow the Poisson distribution conditional on $\boldsymbol{x}_n$'s, we use $log(\cdot)$ as the link function and call the model a Poisson regression model: $log(\mu_n|\boldsymbol{x}_n) = \boldsymbol{\beta} \cdot \boldsymbol{x}_n$. We consider more general parameterizations with tree ensembles as given by

$$log(\mu_n|\boldsymbol{x}_n) = f(\boldsymbol{x}_n; \mathcal{W}, \boldsymbol{O}). \tag{5}$$

where f is parameterized with a tree ensemble as in (1) and $\mathcal{W}, \boldsymbol{O}$ are the learnable parameters in the supernodes and the leaves of the tree ensemble. When a count data has excess zeros, the equidispersion assumption of the Poisson is violated. The Poisson model is not an appropriate model for this situation anymore. [34] proposed zero-inflated-Poisson (ZIP) models that address the mixture of excess zeros and Poisson count process. The mixture is indicated by the latent binary variable d_n using a logit model and the density for the Poisson count given by the log-linear model. Thus, $y_n = y_n^* \odot 1[d_n \neq 0]$, where the latent indicator $d_n \sim Bernoulli(\pi_n)$ with $\pi_n = P(d_n = 1)$ and $y_n^* \sim Poisson(\mu_n)$. The mixture yields the marginal probability mass function of the observed y_n given as:

$$ZIP(y_n|\mu_n, \pi_n) = \begin{cases} (1-\pi_n) + \pi_n e^{-\mu_n}, & \text{if } y_n = 0 \\ \pi_n e^{-\mu_n} \mu_n^{y_n} / y_n!, & \text{if } y_n = 1, 2, \cdots \end{cases} \quad (6)$$

where μ_n and π_n are modeled by

$$log\left(\frac{\pi_n}{1-\pi_n}|\boldsymbol{x}_n\right) = f(\boldsymbol{x}_n; \mathcal{Z}, \boldsymbol{\mathcal{U}}) \quad (7)$$

$$log(\mu_n|\boldsymbol{x}_n) = f(\boldsymbol{x}_n; \boldsymbol{\mathcal{W}}, \boldsymbol{O}). \quad (8)$$

where $\mathcal{Z}, \boldsymbol{\mathcal{U}}$ are the learnable parameters in the splitting internal supernodes and the leaves of the tree ensemble for the logit model for π_n and $\boldsymbol{\mathcal{W}}, \boldsymbol{O}$ are the learnable parameters in the supernodes and the leaves of the tree ensemble for the log-tree model for μ_n respectively. The likelihood function for this ZIP model is given by

$$L(y_n, f(\boldsymbol{x}_n)) = \prod_{y_n=0} (1-\pi_n) + \pi_n e^{-\mu_n} \prod_{y_n>0} \pi_n e^{-\mu_n} \mu_n^{y_n} / y_n! \quad (9)$$

where $\mu_n = e^{f(\boldsymbol{x}_n; \boldsymbol{\mathcal{W}}, \boldsymbol{O})}$ and $\pi_n = e^{f(\boldsymbol{x}_n, \mathcal{Z}; \boldsymbol{\mathcal{U}})} / (1 + e^{f(\boldsymbol{x}_n; \mathcal{Z}, \boldsymbol{\mathcal{U}})})$. Such a model can be overparameterized and we observed that sharing the learnable parameters $\mathcal{Z} = \boldsymbol{\mathcal{W}}$ in the splitting internal supernodes across the log-mean and logit models can lead to better test performance — see Section 6 for a thorough evaluation on real-world datasets.

Negative Binomial Regression. An alternative distribution to zero-inflation modeling that can cater to over-dispersion in the responses is Negative Binomial (NB) distribution. A negative binomial distribution for a random variable y with a non-negative mean $\mu \in \mathbb{R}_+$ and dispersion parameters $\phi \in \mathbb{R}_+$ is given by:

$$NB(\mathrm{y}|\mu, \phi) = \binom{\mathrm{y} + \phi - 1}{\mathrm{y}} \left(\frac{\mu}{\mu + \phi}\right)^{\mathrm{y}} \left(\frac{\phi}{\mu + \phi}\right)^{\phi} \quad (10)$$

The mean and variance of a random variable $\mathrm{y} \sim NB(\mathrm{y}|\mu, \phi)$ are $\mathbb{E}[\mathrm{y}] = \mu$ and $\mathrm{Var}[\mathrm{y}] = \mu + \mu^2/\phi$. Recall that Poisson($\mu$) has variance μ, so $\mu^2/\phi > 0$ is the additional variance of the negative binomial above that of the Poisson with mean μ. So the inverse of parameter ϕ controls the overdispersion, scaled by the square of the mean, μ^2.

When the responses y_n (for $n \in [N]$) are i.i.d, and follow NB distribution conditioned on $\boldsymbol{x}_n$'s, we can use the $log(.)$ as a link function to parameterize the log-mean and log-dispersion as linear functions of the covariates $\boldsymbol{x}_n$. In our parameterization with Tree Ensembles, we model them as given by:

$$log(\mu_n|\boldsymbol{x}_n) = f(\boldsymbol{x}_n; \boldsymbol{\mathcal{W}}, \boldsymbol{O}) \quad (11)$$

$$log\,(\phi_n|\boldsymbol{x}_n) = f(\boldsymbol{x}_n; \mathcal{Z}, \boldsymbol{\mathcal{U}}). \quad (12)$$

where $\mathcal{Z}, \boldsymbol{\mathcal{U}}$ are the learnable parameters in the supernodes and the leaves of the tree ensemble for the log-mean and $\boldsymbol{\mathcal{W}}, \boldsymbol{O}$ are the learnable parameters in the supernodes and the leaves of the tree ensemble for the log-dispersion model for ϕ_n respectively. Such a model can be overparameterized and we observed that sharing the learnable parameters $\mathcal{Z} = \boldsymbol{\mathcal{W}}$ in the splitting internal supernodes across the log-mean and log-dispersion models can lead to better out-of-sample performance. See Section 6.4 for empirical validation on a large-scale dataset.

5 MULTI-TASK LEARNING WITH TREE ENSEMBLES

Multi-task Learning (MTL) aims to learn multiple tasks simultaneously by using a shared model. Unlike single task learning, MTL can achieve better generalization performance through exploiting task relationships [14, 15]. One key problem in MTL is how to share model parameters between tasks [43]. For instance, sharing parameters between unrelated tasks can potentially degrade performance. MTL approaches for classical decision trees approaches e.g., RF [37], GRF [3] have shared weights at the splitting nodes across the tasks. Only the leaf weights are task specific. However this can be limiting in terms of performance, despite easier interpretability associated with the same split nodes across tasks.

To perform flexible multi-task learning, we extend our formulation in Section 3.2 by using task-specific nodes in the tree ensemble. We consider T tasks. For easier exposition, we consider tasks of the same kind: multilabel classification or multi-task regression. For multilabel classification, each task is assumed to have same number of classes (with $k = C$) for easier exposition — our framework can handle multilabel settings with different number of classes per task. Similarly, for regression settings, $k = 1$. For multi-task zero-inflated Poisson or negative binomial regression, when two model components need to be estimated, we set $k = 2$ to predict log-mean and logit components for zero-inflated Poisson and log-mean and log-dispersion components for negative binomial.

We define a trainable weight tensor $\boldsymbol{\mathcal{W}}_i \in \mathbb{R}^{T,p,m}$ for supernode $i \in \mathcal{I}$, where each t-th slice of the tensor $\boldsymbol{\mathcal{W}}_i[t, :, :]$ denotes the trainable weight matrix associated with task t. The prediction in this case is given by

$$\boldsymbol{f}(\boldsymbol{x}) = \left(\sum_{l \in L} \boldsymbol{O}_l \odot \prod_{i \in A(l)} \boldsymbol{\mathcal{R}}_{i,l}\right) \cdot \mathbf{1}_m \quad (13)$$

where $\boldsymbol{O}_l \in \mathbb{R}^{T,m,k}$ denotes the trainable leaf tensor in leaf l, $\boldsymbol{\mathcal{R}}_{i,l} = S(\boldsymbol{\mathcal{W}}_i \cdot \boldsymbol{x})1[l \swarrow i] \odot (1 - S(\boldsymbol{\mathcal{W}}_i \cdot \boldsymbol{x}))1[i \searrow l] \in \mathbb{R}^{T,m,1}$.

In order to share information across the tasks, our framework imposes a closeness penalty on the hyperplanes $\boldsymbol{\mathcal{W}}_i$ in the supernodes across the tasks. This results in the optimization formulation:

$$\min_{\boldsymbol{\mathcal{W}}, \boldsymbol{O}} \sum_{t \in T} \sum_{\boldsymbol{x}, y_t} g_t(y_t, f_t(\boldsymbol{x})) + \lambda \sum_{s<t, t \in T} \|\boldsymbol{\mathcal{W}}_{:,s,:,:} - \boldsymbol{\mathcal{W}}_{:,t,:,:}\|^2, \quad (14)$$

where $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{\mathcal{I},T,m,p}$ denotes all the weights in all the supernodes, $\boldsymbol{O} \in \mathbb{R}^{\mathcal{L},m,k}$ denotes all the weights in the leaves, and $\lambda \in [0, \infty)$ is a non-negative regularization penalty that controls how close the weights across the tasks are. For $\lambda = 0$, the model behaves similar to a single-task learning setting. When $\lambda \to \infty$, the model shares complete information in the splitting nodes and the weights across the tasks in each of the internal supernodes become the same — this is similar to hard parameter sharing. The latter case can be separately handled more efficiently by using the function definition in (4) for $f(\boldsymbol{x})$ without any closeness regularization in (14). Our model can control the level of sharing across the tasks by controlling λ. In practice, we tune over $\lambda \in [1e-5, 10]$ and select the optimal value based on a validation set. This penalty assumes that the hyperplanes across the tasks should be equally close as we go down the depth of the trees. However this assumption maybe less accurate as we go down the tree. Empirically, we found that decaying λ exponentially as $\lambda/2^d$ with depth d of the supernodes in the ensemble can achieve better test performance.

6 EXPERIMENTS

We study the performance of differentiable tree ensembles in various settings and compare against the relevant state-of-the-art baselines for each setting. The different settings can be summarized as follows: (i) Comparison of a single soft tree with state-of-the-art classical tree method in terms of test performance and depth. We include both axis-aligned and oblique classical tree in our comparisons. (ii) Flexible zero-inflation models with tree ensembles. We compare against Poisson regression with tree ensembles and gradient boosting decision trees (GBDT). We consider test Poisson deviance and tree ensemble compactness for model evaluation (iii) We evaluate our proposed multi-task tree ensembles and compare them against multioutput RF, multioutput GRF and single-task GBDT. We consider both fully observed and partially observed responses across tasks. (iv) We also validate our tree ensemble methods with flexible loss functions (zero-inflated Poisson and negative binomial regression) on a large-scale multi-task proprietary dataset.

Model Implementation. Differentiable tree ensembles in our toolkit are implemented in TensorFlow 2.0 using Keras interface.

Datasets. We use 27 open-source regression datasets from various domains (e.g., social media platforms, human behavior, finance). 9 are from Mulan [48], 2 are from UCI data repository [19], 12 are from Delve database [2] and the 5 remaining are SARCOS [47], Youth Risk Behavior Survey [29], Block-Group level Census data [46], and a financial services loss dataset from Kaggle[3]. We also validate our framework on a proprietary multi-task data with millions of samples from a multi-national financial services company.

6.1 Studying a Single Tree

In this section, we compare performance and model compactness of a single tree on 12 regression datasets from Delve database: abalone, pumadyn-family, comp-activ (cpu, cpuSmall) and concrete.

Competing Methods and Implementation. We focus on two baselines from classical tree literature: CART [10] and Tree Alternating Optimization (TAO) method proposed by [13]. The authors in [51] performed an extensive comparison of various single tree learners and demonstrated TAO to be the best performer. Hence, we include both axis-aligned and oblique decision tree versions of TAO in our comparisons. Given that the authors in [13, 51] do not provide an open-source implementation for TAO, we implemented our own version of TAO. For a fair comparison, we use binary decision trees for both axis-aligned and oblique versions of TAO. For more details about our implementation, see Supplemental Section S1.1.

Table 1: Test mean squared error of *single* axis-aligned and oblique decision trees on various regression datasets.

	Axis-Aligned		Oblique	
Data	**CART**	**TAO**	**TAO**	**Soft Tree**
abalone	7.901E-03	8.014E-03	7.205E-03	**6.092E-03**
pumadyn-32nh	9.776E-03	9.510E-03	1.146E-02	**8.645E-03**
pumadyn-32nm	2.932E-03	2.741E-03	4.942E-03	**1.280E-03**
pumadyn-32fh	1.324E-02	1.307E-02	1.370E-02	**1.166E-02**
pumadyn-32fm	2.246E-03	2.177E-03	3.566E-03	**1.602E-03**
pumadyn-8nh	1.995E-02	1.983E-02	2.027E-02	**1.670E-02**
pumadyn-8nm	4.878E-03	4.584E-03	4.206E-03	**2.352E-03**
pumadyn-8fh	2.182E-02	2.200E-02	2.159E-02	**2.048E-02**
pumadyn-8fm	4.398E-03	4.347E-03	4.074E-03	**3.543E-03**
cpu	9.655E-04	1.475E-03	1.312E-03	**9.159E-04**
cpuSmall	1.450E-03	2.319E-03	1.171E-03	**9.159E-04**
concrete	7.834E-03	6.619E-03	1.166E-02	**4.139E-03**

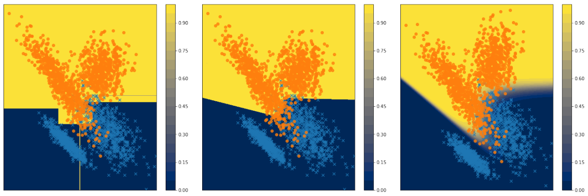

Figure 2: Classifier boundaries for CART [Left], TAO (oblique) [Middle] and Soft tree [Right] on a synthetic dataset with $N_{train} = N_{val} = N_{test} = 2500$ from sklearn [11]. We tune for 50 trials over depths in the range $[2-4]$ for TAO (oblique) and soft trees and $[2-10]$ for CART. Optimal depths for CART, TAO, Soft tree are 5, 4 and 2 respectively. Test AUCs are 0.950, 0.957, and 0.994 respectively.

Results. We present the out-of-sample mean-squared-error performance and optimal depths in Tables 1 and S1 (in Supplemental Section S1.1) respectively. Notably, in all 12 cases, soft tree outperforms all 3 baseline methods in terms of test performance. The soft tree finds a smaller optimal depth in majority cases in comparison with its classical counterpart i.e., oblique TAO tree — See Table S1 in Supplemental Section S1.1. This may be due to the end-to-end learning in a soft tree, unlike TAO that performs local search.

6.2 Zero-inflation

We consider a collection of real-world applications with zero-inflated data. The datasets include (i) yrbs: nationwide drug use behaviors of high school students as a function of demographics, e-cigarettes/mariyuana use etc.; (ii) news: popularity of news items on social media platforms [38] e.g., Facebook, Google+ as a function of topic and sentiments; (iii) census: number of people with zero, one, or two health insurances across all Census blocks in the ACS population as a function of housing and socio-economic demographics; (iv) fin-services-loss: financial services losses as a function of geodemographics, information on crime rate, weather.

[3] https://bit.ly/3swGnTo

Table 2: Test poisson deviance across models on various datasets. Flexible modeling via zero-inflated Poisson for Soft Tree Ensembles leads to better poisson deviance.

			GBDT	**Soft Trees**	
Data	**N**	**p**	**Poisson**		**ZIP**
yrbs-cocaine	12172	55	3.14E-02	3.00E-02	**2.82E-02**
yrbs-heroine	12711	55	1.81E-02	1.60E-02	**1.54E-02**
yrbs-meth	12690	55	2.38E-02	2.21E-02	**2.09E-02**
yrbs-lsd	9564	55	3.51E-02	3.59E-02	**3.43E-02**
news-facebook	81637	3	4.70E-03	**4.68E-03**	**4.68E-03**
news-google+	87495	3	5.97E-03	**5.93E-03**	**5.93E-03**
census-health0	220333	64	1.51E-04	**1.28E-04**	1.32E-04
census-health1	220333	64	**4.63E-04**	5.11E-04	4.66E-04
census-health2+	220333	64	2.73E-03	3.06E-03	**2.72E-03**
fin-services-losses	452061	300	**2.20E-03**	2.28E-03	**2.20E-03**
#wins	-	-	2	3	8

Table 3: Tree ensemble compactness (# trees and depth) for GBDT and Soft Tree Ensembles for different datasets. Flexible modeling via zero-inflated Poisson for Soft Tree Ensembles can lead to more compact tree ensembles, which can improve interpretability

	#Trees			**Depth**		
	GBDT	**Soft Trees**		**GBDT**	**Soft Trees**	
Data	**Poisson**		**ZIP**	**Poisson**		**ZIP**
yrbs-cocaine	575	77	**13**	4	**2**	3
yrbs-heroine	1425	83	**4**	4	4	**2**
yrbs-meth	1475	16	**4**	4	**2**	3
yrbs-lsd	1225	**17**	45	4	3	**2**
news-facebook	200	**65**	85	4	4	4
news-google+	750	81	**74**	4	4	4
census-health0	1275	**10**	**10**	4	3	**2**
census-health1	1275	**17**	**17**	4	2	**2**
census-health2+	1375	73	**56**	8	4	**3**
fin-services-losses	1225	32	**4**	4	3	**2**
#wins	0	4	**7**	-	5	**8**

Competing methods. We consider Poisson regression with GBDT and differentiable tree ensembles. We also consider zero-inflation modeling with differentiable tree ensembles. We use GBDT from sklearn [11]. For additional details about the tuning experiments, please see Supplemental Section S1.2.

Results. We present the out-of-sample Poisson deviance performance in Table 2. Notably, tree ensembles with zero-inflated loss function leads the chart. We also present the optimal selection of tree ensemble sizes and depths in Table 3. We can observe that zero-inflation modeling can lead to significant benefits in terms of model compression. Both tree ensemble sizes and depths can potentially be made smaller, which have implications for faster inferences, memory footprint and interpretability.

6.3 Multi-task Regression

We compare performance and model compactness of our proposed regularized multi-task tree ensembles on 11 multi-task regression datasets from Mulan (atp1d, atp7d, sf1, sf2, jura, enb, slump, scm1d, scm20d), and UCI data repository (bike) and SARCOS dataset.

Table 4: Test MSE of RF, GRF and multi-task Differentiable Tree Ensembles on 11 multi-task regression datasets with fully observed responses across tasks.

		Multi-task		
Data	**Task**	**RF**	**GRF**	**Soft Trees**
atp1d	1	2.242E-02	1.847E-02	**5.383E-03**
	2	2.498E-02	1.894E-02	**7.217E-03**
	3	1.127E-02	**9.625E-03**	1.128E-02
	4	1.574E-02	1.504E-02	**1.403E-02**
	5	2.040E-02	1.905E-02	**1.182E-02**
	6	1.571E-02	**1.333E-02**	1.527E-02
atp7d	1	3.244E-03	**2.691E-03**	8.965E-03
	2	**3.914E-03**	3.931E-03	4.456E-03
	3	1.231E-02	1.078E-02	**1.059E-02**
	4	**5.459E-03**	6.542E-03	6.001E-03
	5	**1.842E-03**	1.922E-03	3.358E-03
	6	**4.042E-03**	5.303E-03	5.033E-03
sf1	1	4.384E-02	**3.151E-02**	3.345E-02
	2	1.077E-02	4.638E-03	**3.828E-03**
	3	4.252E-02	**2.863E-02**	2.993E-02
sf2	1	7.883E-03	8.807E-03	**7.789E-03**
	2	3.247E-03	2.583E-03	**2.206E-03**
	3	**1.262E-03**	3.744E-03	3.595E-03
jura	1	3.027E-02	3.008E-02	**2.233E-02**
	2	1.483E-02	1.405E-02	**1.015E-02**
	3	7.896E-03	7.586E-03	**6.036E-03**
enb	1	2.473E-04	**1.865E-04**	2.063E-04
	2	2.830E-03	3.305E-03	**1.054E-03**
slump	1	1.732E-01	1.396E-01	**1.001E-01**
	2	1.224E-01	9.827E-02	**7.368E-02**
	3	3.878E-02	2.944E-02	**5.149E-03**
scm1d	1	3.040E-03	2.530E-03	**1.794E-03**
	2	3.397E-03	3.003E-03	**2.226E-03**
	3	4.178E-03	3.611E-03	**2.940E-03**
	4	3.991E-03	3.376E-03	**2.150E-03**
scm20d	1	4.457E-03	3.650E-03	**2.198E-03**
	2	4.766E-03	3.632E-03	**2.410E-03**
	3	4.892E-03	3.506E-03	**2.620E-03**
	4	5.573E-03	4.072E-03	**2.632E-03**
bike	1	3.466E-03	2.558E-03	**1.730E-03**
	2	4.636E-03	4.039E-03	**3.728E-03**
	3	5.123E-03	4.303E-03	**3.822E-03**
# wins	-	5	6	**26**

Table 5: Tree ensemble sizes for soft trees, RF, and GRF.

	atp1d	**atp7d**	**sf1**	**sf2**	**jura**	**enb**	**slump**	**scm1d**	**scm20d**	**bike**
RF	100	125	500	225	150	100	825	175	100	100
GRF	1050	950	350	100	150	100	350	50	100	250
Ours	10	15	12	44	17	13	54	49	25	93

Competing Methods. We focus on 4 tree ensemble baselines from literature: single-task soft tree ensembles, sklearn GBDT, sklearn multioutput RF [11] and r-grf package for GRF [3]. We consider two multi-task settings: (i) All Fully observed responses for all tasks, (ii) Partially observed responses across tasks. In the former case, we

Table 6: Test MSE of GBDT, single-task and multi-task Soft Tree Ensembles on 11 multi-task regression datasets, with 50% missing responses per task.

		Single-Task		Multi-Task
Data	**Task**	**GBDT**	**Soft Trees**	
atp1d	1	1.469E-02	3.091E-02	**1.295E-02**
	2	**7.698E-03**	2.598E-02	1.137E-02
	3	2.172E-02	2.915E-02	**1.807E-02**
	4	**5.905E-03**	**1.417E-02**	9.434E-03
	5	2.421E-02	5.631E-02	**2.105E-02**
	6	**5.646E-03**	5.880E-02	2.724E-02
atp7d	1	5.562E-02	6.261E-02	**3.601E-02**
	2	4.033E-02	2.216E-02	**1.067E-02**
	3	2.140E-02	4.989E-02	**1.680E-02**
	4	1.107E-02	2.089E-02	**9.926E-03**
	5	4.254E-02	6.476E-02	**3.053E-02**
	6	**4.195E-03**	2.416E-02	, 2.199E-02
sf1	1	**2.674E-02**	2.763E-02	2.732E-02
	2	5.825E-03	1.680E-02	**5.435E-03**
	3	3.030E-02	3.704E-02	**2.964E-02**
sf2	1	**1.003E-02**	1.179E-02	**1.009E-02**
	2	1.123E-02	6.843E-03	**2.809E-03**
	3	9.271E-03	1.039E-02	**8.064E-03**
jura	1	1.779E-02	1.934E-02	**1.503E-02**
	2	1.117E-02	1.665E-02	**9.304E-03**
	3	1.311E-02	1.514E-02	**1.262E-02**
enb	1	**1.509E-04**	2.738E-04	4.167E-04
	2	**1.071E-03**	1.227E-03	1.160E-03
slump	1	1.622E-01	**7.611E-02**	9.485E-02
	2	8.823E-02	1.050E-01	**4.734E-02**
	3	8.423E-03	**1.737E-03**	7.744E-03
scm1d	1	**1.598E-03**	1.903E-03	2.058E-03
	2	**1.952E-03**	2.703E-03	2.490E-03
	3	3.029E-03	3.194E-03	**2.919E-03**
	4	**2.666E-03**	3.656E-03	3.272E-03
scm20d	1	2.541E-03	2.672E-03	**2.533E-03**
	2	3.640E-03	3.174E-03	**3.146E-03**
	3	3.658E-03	4.015E-03	**3.201E-03**
	4	3.756E-03	4.115E-03	**3.670E-03**
bike	1	**2.122E-03**	2.558E-03	2.300E-03
	2	**3.680E-03**	4.038E-03	3.846E-03
	3	**3.731E-03**	4.303E-03	3.910E-03
sarcos	1	2.582E-04	**1.317E-04**	1.518E-04
	2	1.643E-04	8.310E-05	**8.307E-05**
	3	3.325E-04	**1.788E-04**	1.933E-04
# wins	-	14	5	**23**

compare against RF and GRF. In the latter case, we compare against single-task soft tree ensembles and GBDT. Note the open-source implementations for RF and GRF do not support partially observed responses for multi-task settings and GBDT does not have support for multi-task setting. We refer the reader to Supplemental Section S1.3 for tuning experiments details.

Results. We present results for fully observed response settings in Table 4 and partially observed response settings in Table 6. In both cases, regularized multi-task soft trees lead the charts over the corresponding baselines in terms of out-of-sample mean squared error performance. For the fully observed response setting, we also show tree ensemble sizes in Table 5. We see a large reduction in the number of trees with out proposed multi-task tree ensembles.

6.4 Large-scale multi-task data from a multinational financial services company

We study the performance of our differentiable tree ensembles in a real-word, large-scale multi-task setting from a multinational financial services company. The system encompasses costs and fees for millions of users for different products and services. The dataset has the following characteristics: (i) It is a multi-task regression dataset with 3 tasks. (ii) Each task has high degree of over-dispersion. (iii) All tasks are not fully observed as each user signs up for a subset of products/services. The degree of missing responses on average across tasks is ∼ 50%. (iv) Number of features is also large (∼ 600).

We validate the flexibility of our end-to-end tree-ensemble learning framework with soft trees on a dataset of 1.3 million samples. We study the following flexible aspects of our framework: (i) Flexible loss handling with zero-inflation Poisson regression and negative binomial regression for single-task learning. (ii) Multi-task learning with our proposed regularized multi-task soft tree ensembles in the presence of missing responses across tasks. (iii) Flexible loss handling with zero-inflation Poisson/negative binomial regression in the context of multi-task learning.

We present our results in Table 7. We can see that we achieve the lowest Poisson deviance and highest AUC with multi-task regression via zero-inflated Poisson/negative binomial regression.

7 CONCLUSION

We propose a flexible and scalable framework for learning differentiable tree ensembles. Our framework supports a diverse set of loss functions and allows for easily adding new loss functions. It also has novel support for multi-task learning. For scalability, we propose a new tensor-based formulation of tree ensembles, which allows for 10x faster training on CPUs and also adds support for GPU training. We perform experiments on a collection of 28 open-source and real-world datasets, demonstrating that our new FASTEL toolkit can lead to 100x more compact ensembles and up to 23% improvement in out-of-sample performance, compared to tree ensembles learnt by popular toolkits such as XGBoost.

ACKNOWLEDGMENTS

We thank Denis Sai for his help with the experiments on large-scale multi-task data. This research was supported in part, by grants from the Office of Naval Research: ONR-N00014-21-1-2841 and award from Liberty Mutual Insurance.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, et al. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.

[2] Uchenna Akujuobi and Xiangliang Zhang. 2017. Delve: A Dataset-Driven Scholarly Search and Analysis System. *SIGKDD Explor. Newsl.* 19, 2 (nov 2017), 36–46.

[3] Susan Athey, Julie Tibshirani, and Stefan Wager. 2019. Generalized random forests. *The Annals of Statistics* (2019).

[4] Jonathan T. Barron. 2019. A General and Adaptive Robust Loss Function. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 4326–4334.

[5] K.P. Bennett. 1992. *Decision Tree Construction Via Linear Programming*. Number no. 1067 in Computer sciences technical report. University of Wisconsin-Madison, Computer Sciences Department.

Table 7: Out-of-sample performance of single-task and multi-task tree ensembles with flexible loss functions for zero-inflation/overdispersion. We evaluate performance with weighted Poisson deviance and AUC across tasks.

		GBDT	**Soft Trees**					
		Single-task	**Single-task**			**Multi-task**		
Metric	**Task**	**Poisson**	**Poisson**	**ZIP**	**NB**	**Poisson**	**ZIP**	**NB**
Poisson Deviance	1	2.643E-04	2.624E-04	2.623E-04	2.623E-04	2.607E-04	**2.605E-04**	2.608E-04
	2	8.029E-04	8.050E-04	8.029E-04	8.044E-04	8.022E-04	**8.014E-04**	**8.014E-04**
	3	1.044E-03	1.045E-03	1.043E-03	1.042E-03	1.041E-03	**1.040E-03**	1.041E-03
AUC	1	0.710	0.721	0.722	0.721	0.730	**0.734**	0.727
	2	0.690	0.689	0.690	0.688	0.691	0.691	**0.692**
	3	0.684	0.683	0.686	0.685	0.687	**0.689**	**0.689**

[6] Kristin P. Bennett and Jennifer A. Blue. 1996. *Optimal Decision Trees.* Technical Report. R.P.I. Math Report No. 214, Rensselaer Polytechnic Institute.

[7] Dimitris Bertsimas and Jack Dunn. 2017. Optimal classification trees. *Machine Learning* 106 (2017), 1039–1082.

[8] Rafael Blanquero, Emilio Carrizosa, Cristina Molero-Río, et al. 2021. Optimal randomized classification trees. *Computers & Operations Research* 132 (2021), 105281.

[9] Leo Breiman. 2001. Random Forests. *Machine Learning* 45, 1 (2001), 5–32.

[10] L. Breiman, J. Friedman, C.J. Stone, and R.A. Olshen. 1984. *Classification and Regression Trees.* Taylor & Francis.

[11] Lars Buitinck, Gilles Louppe, Mathieu Blondel, et al. 2013. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning.* 108–122.

[12] A. Colin Cameron and Pravin K. Trivedi. 2013. *Regression Analysis of Count Data* (2 ed.). Cambridge University Press. https://doi.org/10.1017/CBO9781139013567

[13] Miguel A. Carreira-Perpinan and Pooya Tavallali. 2018. Alternating optimization of decision trees, with application to learning sparse oblique trees. In *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, et al. (Eds.), Vol. 31. Curran Associates, Inc.

[14] Rich Caruana. 1997. Multitask Learning. *Machine Learning* 28, 1 (1997), 41–75.

[15] Olivier Chapelle, Pannagadatta K. Shivaswamy, Srinivas Vadrevu, et al. 2010. Boosted multi-task learning. *Machine Learning* 85 (2010), 149–173.

[16] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Francisco, California, USA) *(KDD '16)*. Association for Computing Machinery, New York, NY, USA, 785–794.

[17] Michael Crawshaw. 2020. Multi-Task Learning with Deep Neural Networks: A Survey. *ArXiv* abs/2009.09796 (2020).

[18] Joshua V. Dillon, Ian Langmore, Dustin Tran, et al. 2017. TensorFlow Distributions. *CoRR* abs/1711.10604 (2017). arXiv:1711.10604

[19] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository.

[20] Chandra Erdman and Nancy Bates. 2016. The Low Response Score (LRS). *Public Opinion Quarterly* 81, 1 (Dec. 2016), 144–156.

[21] Nicholas Frosst and Geoffrey Hinton. 2017. Distilling a Neural Network Into a Soft Decision Tree. arXiv:1711.09784

[22] Pierre Geurts, Damien Ernst, and Louis Wehenkel. 2006. Extremely randomized trees. *Machine Learning* 63, 1 (2006), 3–42.

[23] Lei Han and Yu Zhang. 2015. Learning Tree Structure in Multi-Task Learning. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Sydney, NSW, Australia) *(KDD '15)*. Association for Computing Machinery, New York, NY, USA, 397–406.

[24] T. J. Hastie, R. J. Tibshirani, and J. Friedman. 2009. *The Elements of Statistical Learning* (2 ed.). Springer.

[25] Hussein Hazimeh, Natalia Ponomareva, Petros Mol, et al. 2020. The Tree Ensemble Layer: Differentiability meets Conditional Computation. In *Proceedings of the 37th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 119)*, Hal Daumé III and Aarti Singh (Eds.). PMLR, 4138–4148.

[26] Thomas M. Hehn, Julian F. P. Kooij, and Fred A. Hamprecht. 2019. End-to-End Learning of Decision Trees and Forests. *International Journal of Computer Vision* 128 (2019), 997–1011.

[27] Laurent Hyafil and Ronald L. Rivest. 1976. Constructing optimal binary decision trees is NP-complete. *Inform. Process. Lett.* 5, 1 (1976), 15–17.

[28] Ozan Irsoy, O. T. Yildiz, and Ethem Alpaydin. 2012. Soft decision trees. *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)* (2012), 1819–1822.

[29] Wura Jacobs, Ehikowoicho Idoko, LaTrice Montgomery, et al. 2021. Concurrent E-cigarette and marijuana use and health-risk behaviors among U.S. high school students. *Preventive Medicine* 145 (2021), 106429.

[30] Michael I. Jordan and Robert A. Jacobs. 1994. Hierarchical Mixtures of Experts and the EM Algorithm. *Neural Comput.* 6, 2 (mar 1994), 181–214.

[31] Guolin Ke, Qi Meng, Thomas Finley, et al. 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Advances in Neural Information Processing Systems*, Vol. 30. Curran Associates, Inc.

[32] Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, et al. 2015. Deep Neural Decision Forests. In *2015 IEEE International Conference on Computer Vision (ICCV)*. 1467–1475.

[33] Abhishek Kumar and Hal Daumé. 2012. Learning Task Grouping and Overlap in Multi-Task Learning. In *Proceedings of the 29th International Coference on International Conference on Machine Learning* (Edinburgh, Scotland) *(ICML'12)*. Omnipress, Madison, WI, USA, 1723–1730.

[34] Diane Lambert. 1992. Zero-Inflated Poisson Regression, with an Application to Defects in Manufacturing. *Technometrics* 34, 1 (feb 1992), 1–14.

[35] Nathan Lay, Adam P. Harrison, Sharon Schreiber, et al. 2018. Random Hinge Forest for Differentiable Learning. *CoRR* abs/1802.03882 (2018). arXiv:1802.03882

[36] Simon C. K. Lee. 2020. Addressing Imbalanced Insurance Data Through Zero-Inflated Poisson Regression With Boosting. *ASTIN Bulletin* 51 (2020), 27 – 55.

[37] Henrik Linusson. 2013. Multi-Output Random Forests.

[38] Nuno Moniz and Luís Torgo. 2018. Multi-Source Social Feedback of Online News Feeds. *ArXiv* abs/1801.07055 (2018).

[39] Sreerama K. Murthy, Simon Kasif, and Steven Salzberg. 1994. A System for Induction of Oblique Decision Trees. *J. Artif. Int. Res.* 2, 1 (aug 1994), 1–32.

[40] Adam Paszke, Sam Gross, Francisco Massa, et al. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, et al. (Eds.). Curran Associates, Inc., 8024–8035.

[41] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, et al. 2018. CatBoost: Unbiased Boosting with Categorical Features. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (Montréal, Canada) *(NIPS'18)*. Curran Associates Inc., Red Hook, NY, USA, 6639–6649.

[42] J. Ross Quinlan. 1993. *C4.5: Programs for Machine Learning.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[43] Sebastian Ruder. 2017. An Overview of Multi-Task Learning in Deep Neural Networks. *ArXiv* abs/1706.05098 (2017).

[44] Robert E. Schapire and Yoav Freund. 2012. *Boosting: Foundations and Algorithms.* The MIT Press.

[45] Ryutaro Tanno, Kai Arulkumaran, Daniel C. Alexander, et al. 2019. Adaptive Neural Trees. *ArXiv* abs/1807.06699 (2019).

[46] US Census Bureau. 2021. Planning Database.

[47] Sethu Vijayakumar and Stefan Schaal. 2000. Locally Weighted Projection Regression : An O(n) Algorithm for Incremental Real Time Learning in High Dimensional Space.

[48] Eleftherios Spyromitros Xioufis, Grigorios Tsoumakas, William Groves, et al. 2016. Multi-target regression via input space expansion: treating targets as inputs. *Machine Learning* 104 (2016), 55–98.

[49] Ashenafi A. Yirga, Sileshi F. Melesse, Henry G. Mwambi, et al. 2020. Negative binomial mixed models for analyzing longitudinal CD4 count data. *Scientific Reports* 10, 1 (2020), 16742.

[50] Derek S. Young, Andrew M. Raim, and Nancy R. Johnson. 2017. Zero-inflated modelling for characterizing coverage errors of extracts from the US Census Bureau's Master Address File. *Journal of The Royal Statistical Society Series A-statistics in Society* 180 (2017), 73–97.

[51] Arman S. Zharmagambetov and Miguel Á. Carreira-Perpiñán. 2020. Smaller, more accurate regression forests using tree alternating optimization. In *ICML*.

[52] Haoran Zhu, Pavankumar Murali, Dzung Phan, et al. 2020. A Scalable MIP-based Method for Learning Optimal Multivariate Decision Trees. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, et al. (Eds.), Vol. 33. Curran Associates, Inc., 1771–1781.

SUPPLEMENTARY MATERIAL

S1 ADDITIONAL DETAILS FOR EXPERIMENTAL SECTION 6

Datasets. We use a collection of 27 open-source regression datasets from various domains (e.g., social media platforms, human behavior, financial risk data). 9 of these are from Mulan: A Java library for multi-label learning (Mulan) [48], 2 of them are from University of California Irvine data repository (UCI) [19], 12 of them are from Delve database [2] and the 5 remaining are SARCOS [1] [47], Youth Risk Behavior Survey[2] [29], Block-Group level data from US Census Planning Database[3][46], and financial services loss data from Kaggle[4]. For scm1d and scm20d from Mulan[48], we consider the first 4 tasks (out of the 16 tasks in the original dataset). For SARCOS, we consider 3 torques for prediction (torque-3, torque-4 and torque-7; we ignore the other torques as those seem to have poor correlations with these.)

For all datasets, we split the datasets into 64%/16%/20% training/validation/test splits. We train the models on the training set, perform hyperparameter tuning on the validation set and report out-of-sample performance on the test set.

S1.1 Tuning parameters and optimal depths comparison for a single tree in Section 6.1

TAO Implementation. We wrote our own implementation of the TAO algorithm proposed in [13]. We considered binary trees with TAO for both axis-aligned and oblique trees. In the case of axis-aligned splits, we initialize the tree with CART solution and run TAO iterations until there is no improvement in training objective. In the case of oblique trees, we initialize with a complete binary tree with random parameters for the hyperplanes in the split nodes and use logistic regression to solve the decision node optimization. We run the algorithm until either a maximum number of iterations are reached or the training objective fails to improve.

Tuning parameters. For CART and axis-aligned TAO, we tune the depth in the range $[2-20]$. We also optimize over the maximum number of iterations in the interval $[20 - 100]$. For oblique TAO and soft tree, we tune the depth between $2 - 10$ and the number of iterations between $20 - 100$. Additionally, for soft tree, we also tune over the learning rates $[1e-5, 1e-2]$ with Adam optimizer and batch sizes $\{64, 128, 256, 512\}$. For a fair comparison, we run all 4 methods for 100 trials.

Optimal depths. We make a comparison of optimal depths between CART, TAO (both axis-aligned and oblique) and soft tree. The soft tree finds a smaller optimal depth in majority cases in comparison with its classical counterpart i.e., oblique TAO tree — See Table S1. This is hypothesized to be due to the end-to-end optimization done by soft tree as opposed to a local search performed by the TAO algorithm.

Table S1: Optimal depth of a *single* axis aligned and oblique decision tree on various regression datasets.

	Axis-Aligned		Oblique	
Data	**CART**	**TAO**	**TAO**	**Soft Tree**
abalone	5	5	4	**2**
pumadyn-32nh	6	6	4	**3**
pumadyn-32nm	8	8	5	**4**
pumadyn-32fh	**3**	**3**	**3**	5
pumadyn-32fm	6	6	5	**4**
pumadyn-8nh	6	6	5	**3**
pumadyn-8nm	9	8	7	**5**
pumadyn-8fh	5	5	4	**2**
pumadyn-8fm	7	7	6	**3**
cpu	10	8	**5**	**5**
cpuSmall	8	8	**5**	**5**
concrete	15	8	**5**	9

S1.2 Tuning parameters for Sections 6.2

We use HistGBDT from sklearn [11] (GBDT in sklearn does not support Poisson regression). We tune over depths in the range $[2 - 20]$, number of trees between $50 - 1500$ and learning rates on the log-uniform scale in the interval $[1e-5, 1e-1]$. For differentiable tree ensembles, we tune number of trees in the range $[2, 100]$, depths in the set $[2 - 4]$, batch sizes $\{64, 128, 256, 512\}$, learning rates $[1e - 5, 1e - 1]$ with Adam optimizer and perform early stopping with a patience of 25 based on the validation set. For all models, we perform a random search with 1000 hyperparameter tuning trials.

S1.3 Tuning parameters for Sections 6.3

We tune number of trees in the interval $[50 - 1500]$ for RF, GRF and GBDT. For RF and GBDT, we also tune over depths between $2 - 20$. For GBDT, we tune learning rates between $[1e - 5, 1e - 1]$. For GRF, we also tune over min_node_size in the set $[2 - 20]$ and $\alpha \in [1e - 3, 1e - 1]$. For single-task and multi-task trees, we tune over depths $[2 - 4]$, number of trees $[5 - 100]$, batch sizes $\{64, 128, 256, 512\}$, epochs $[20 - 500]$, Adam learning rates $[1e - 5, 1e - 2]$. We also optimize over the regularization penalty for multi-task soft decision trees $[1e-5, 1e1]$. All single-task models (soft tree ensembles, GBDT) are tuned for 1000 trials per task. All multi-task models (RF, GRF, multi-task soft trees) are tuned for 1000 trials in total.

S1.4 Tuning parameters for Sections 6.4

We use GBDT from XGBoost [16], where we tune number of trees in the interval $[50 - 1500]$, depths between $2 - 20$ and learning rates between $[1e - 4, 1e - 0]$. For single-task and multi-task trees, we tune over depths $[2 - 4]$, number of trees $[5 - 100]$, batch sizes $\{64, 128, 256, 512\}$, epochs $[20 - 200]$, Adam learning rates $[1e - 5, 1e - 2]$. We also optimize over the regularization penalty for multi-task soft decision trees $[1e - 5, 1e1]$. All single-task models (soft tree ensembles, GBDT) with Poisson, Zero-Inflated-Poisson, Negative Binomial are tuned for 1000 trials per task. All multi-task soft-trees are tuned for 1000 trials in total.

[1] The original test set has significant data leakage as noted by https://www.datarobot.com/blog/running-code-and-failing-models/. Following their guidance we discard the original test set and use the original train set to generate train/validation/test splits.
[2] https://www.cdc.gov/healthyyouth/data/yrbs/data.htm
[3] https://www.census.gov/topics/research/guidance/planning-databases.html
[4] https://bit.ly/3swGnTo