Improved Boosted Regression Forests Through Non-Greedy Tree Optimization

Arman Zharmagambetov, Magzhan Gabidolla, Miguel Á. Carreira-Perpiñán
Dept. of Computer Science and Engineering, University of California, Merced, CA, USA
Email: {azharmagambetov,mgabidolla,mcarreira-perpinan}@ucmerced.edu

Abstract—**Regression forests (ensembles of trees) are considered as the leading off-the-shelf method for regression. One of the main approaches of constructing such forests is based on boosting. However, majority of the current boosting implementations employ an axis-aligned tree as a base learner, where each decision node tests for a single feature. Moreover, such trees are usually trained by greedy top-down algorithms such as CART which is shown to be suboptimal. We instead use oblique trees, where each decision node tests for a linear combination of features and train them with the recently proposed non-greedy tree learning method–*Tree Alternating Optimization (TAO)*. We embed the TAO algorithm into the boosting framework and show its effectiveness in the regression setting. We show that it produces much better forests than other types of tree ensembling methods in terms of error, model size and inference time. The result has an immense practical impact on various applications such as in signal processing, data mining, computer vision, etc.**

I. Introduction

Regression analysis is an important problem studied in statistical modeling and machine learning that has been widely used in many signal processing, data mining [1] and computer vision [2] applications. A typical task in regression is to predict a continuous scalar or vector output from an input data. One of the most successful methods in solving such problems attributes to ensembles of decision trees. Their success is due to their ability to achieve low variance and low bias by combining weakly correlated trees. The main approaches are based on bagging, where individual trees are trained independently on bootstrap samples of the data and on a subset of features [3]; or on boosting [4], where individual trees are trained sequentially on the whole data but with adaptively weighted instances.

Here, we focus on boosting algorithms which are well-studied in machine learning and statistical literature. A backbone of all boosting algorithms is building a prediction model by sequentially combining a collection of weak learners which are sometimes referred as base learners. According to the literature [5]–[7], a decision tree based model is a common choice for a base learner due to its fast training time and ability to achieve a low bias. Despite the fact that such models are sensitive to the input (i.e. have high variance), the variance can be sufficiently reduced by leveraging boosting idea which produces a powerful model by ensembling weak learners. However, conventional trees that are employed by boosting algorithms have two major drawbacks. First, they are trained using greedy tree induction algorithms (in top-down fashion) such as CART [8], C4.5 [9] and variations thereof. Such algorithms are known to be suboptimal [7] since splitting a node at each recursive step is achieved by approximately optimizing a purity (or other variance-based) criterion that is indirectly related to the desired objective function. Second, such trees make axis-aligned split, i.e. each internal (or decision) node tests for a single feature which limits input feature utilization (e.g. correlation between features is ignored).

We address both of these issues by: using trees with more complex nodes (oblique, i.e., having hyperplane split) and using a better optimization algorithm to learn an individual tree. For this matter, we build on top of a recently proposed algorithm for learning decision trees, Tree Alternating Optimization (TAO) [10], [11]. TAO, in general, finds a good approximate optima of an objective function over a tree with the given structure and it applies to trees beyond axis-aligned splits. We propose a novel boosting framework for regression which utilizes an oblique TAO tree as a base learner. To the best of our knowledge, this is the first work to show the successful merge of boosting and oblique decision trees for regression problems. Obviously, oblique (or linear multivariate) splits are more powerful models compared to axis-aligned (or univariate) since it brings more flexibility and better feature utilization. We additionally apply sparsity penalty to make TAO trees more efficient. We perform extensive evaluations of our proposed method where we demonstrate that boosted TAO trees significantly improve over SoA boosting methods in terms of accuracy, model size and inference time. Additionally, we analyze different features of our algorithm that are unique to the boosted TAO trees (e.g. hyperparameters choice).

In the rest of the paper, we review related work (section II) and describe a boosting framework we use including our modifications (section III). We then evaluate our approach experimentally and analyze it from different perspectives (section IV).

II. Related Work

A. *Boosting algorithms for regression*

The first practical boosting implementation attributes to AdaBoost by [4]. Since then a large number of boosting methods has been developed such as the gradient boosting machine [12] and a variaty of literature is written which provides extensive reviews on them [5]–[7]. Initially designed for binary classification, AdaBoost has many multiclass extensions and adaptations for regression problems.

Boosting for regression has not been studied as extensively as it is for classification, nevertheless quite a few boosting algorithms have been developed for regression. The original AdaBoost paper [4] presents an adaptation of boosting for regression by reducing the problem into binary classification. It does so by partitioning the target value into equal sized intervals, and so performing binary classification on each interval renders the problem similar to AdaBoost for multi-class classification. This algorithm, AdaBoost.R, has several problems [13], and has hardly ever been used in practice. By performing some ad-hoc modification to it, Drucker [14] presents AdaBoost.R2, which is considered as the first practical boosting algorithm for regression. We focus on it in this paper, and present it in detail in section III-A. Avnimelech and Intrator [15] introduce a threshold based boosting for regression and analyzes it from a PAC perspective. Another threshold based algorithm is AdaBoost.RT [16], which is designed to improve upon [15]. However, these threshold based algorithms introduce an additional hyperparameter for the threshold, and they may have numerical problems when the target values are zero or close to it. Finally, gradient boosting framework [12] fits base learners in a greedy stagewise additive manner to perform approximate functional gradient descent, and it is applicable to both classification and regression problems. Recent highly optimized gradient boosting implementations [17]–[19] support regression tasks.

B. *Decision trees as base learners*

A cornerstone of any boosting algorithm is to sequentially train a collection of base learners and then combine them to build a prediction model. Many machine learning models have been used as base learners for boosting. These include decision trees, support vector machines (SVMs) [20], multilayer perceptrons (MLPs) [21], [22], and many others. According to the literature in ensemble learning [5], models with high variance (e.g. decision trees) and with sensitivity to initialization and training run (e.g. neural networks) are more preferable than others in ensemble learning. Due to its high variance and fast training time, decision trees are the most widely used choice for boosting. For instance, the most successful and commonly used boosting frameworks [17]–[19] are based on decision trees. However, all these trees are induced greedily using top-down recursive partition algorithms such as CART [8] or C4.5 [9], which are suboptimal and they usually apply a thresholding function based on only one feature at a time (i.e. axis-aligned or univariate split). The latter one clearly leads to the underutilization of the features and correlations between them. Decision trees having linear splits (i.e. oblique) address this issue and we employ them in our modified boosting framework. It is worth mentioning that there were previous attempts to train boosted oblique trees [23], [24], however, they mostly focus on classification rather than regression and their performance has been found similar or marginally better to that of axis-aligned trees. Whereas, we demonstrate that our boosting framework with oblique trees can significantly outperform traditional tree ensembles in number of regression benchmarks. Moreover, as will be discussed in a later section, oblique TAO trees possess the property of random parameter initialization, which is considered to be beneficial in ensemble learning.

III. Boosted TAO Trees for Regression Problems

A. *Boosting framework*

Out of many boosting algorithms which exist today, we choose a simple one: AdaBoost [4]. Initially developed for the classification task, this meta-algorithm is considered as one of the first boosting implementations. We use a direct extension of this algorithm for the regression setting known as AdaBoost.R2 [14] which is considered as the first practical boosting algorithm for regression. The reason of such a choice is that we really wanted to make our algorithm as simple as possible so that it can be directly used in practical applications. Moreover, the main goal of this paper is to verify the concept and one can straightforwardly use more recent and efficient boosting framework if so desired (e.g. XGBoost [17]). That said, we demonstrate that our approach works extremely well even with such relatively simple boosting technique.

Algorithm 1 shows the pseudocode of our boosting framework. We directly extend AdaBoost.R2 and employ TAO regression tree as a base learner. Denote a tree output as $\mathbf{T}_t(\cdot)$ parametrized by $\mathbf{\Theta}$, then each boosting step involves training a base learner (TAO tree) to minimize the following weighted regression loss:

$$E = \sum_{n=1}^{N} w_n L(y_n, \mathbf{T}_t(\mathbf{x}_n; \mathbf{\Theta})) \tag{1}$$

where L can be any regression loss such as squared error (used throughout this paper), absolute error, robust losses, etc. Additionally, one can use various regularizations added to the loss (e.g. ℓ_2 penalty) to obtain more compact models and to avoid overfitting. Here, w_n is the weight per training instance. We emphasize that we *directly handle these weights in the base learner training phase* whereas most of the practical boosting implementations (including the original AdaBoost.R2) apply resampling techniques (with probabilities proportional to the weights). Once the base learner is trained, the algorithm updates its weights according to the formula shown in the algorithm and these weights are used for the next base learner. The algorithm increases the weights of the instances which have greater error and decreases the weights of the instances with small error. The final prediction of the model is given as a weighted median of all base learners' outputs.

B. *Learning a single tree*

Consider the optimization problem at each boosting step shown in eq. (1) which involves training a single tree with the weighted regression objective function. We consider a binary oblique tree where the decision function at each node has a linear form: $f_i(\mathbf{x}; \boldsymbol{\theta}_i) = \boldsymbol{\theta}_i^T \mathbf{x} + \boldsymbol{\theta}_{i0}$ and it sends $\mathbf{x}$ to the right child if $f_i(\mathbf{x}; \boldsymbol{\theta}_i) \geq 0$ and to the left, otherwise. Leaves (terminal nodes) are either constant (denoted by TAO-c) or

Algorithm 1: AdaBoost.R2 with TAO modification

```
Result: Forest of boosted TAO trees $\mathbf{F}$
input training set $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$;
  number of trees $T$;
  learning rate $\eta$;
  initial weights $\{w_n = 1/N\}_{n=1}^N$;
for $t = 1$ to $T$ do
    $\mathbf{T} \leftarrow$ fit TAO using Algorithm 2 with the current
      weights per instance $\{w_n\}_{n=1}^N$;
    obtain predictions $\{\hat{y}_n\}_{n=1}^N \leftarrow \mathbf{T}(\{\mathbf{x}_n\}_{n=1}^N)$;
    calculate a loss per instance $l_n \leftarrow \frac{|\hat{y}_n - y_n|}{D}$;
      where $D = \max_n |\hat{y}_n - y_n|$;
    calculate an average loss $\bar{L} = \Sigma_{n=1}^N w_n l_n$
    if $\bar{L} > 0.5$ then
        set $T = t - 1$;
        exit the loop;
    end
    form $\beta_t = \frac{\bar{L}}{1-\bar{L}}$;
    weight of the current tree $\alpha_t = \log(1/\beta_t)$;
    update the instance weights $w_n \leftarrow w_n \beta_t^{1-l_n}$
end
$F(\mathbf{x})$ = weighted median of $\{\mathbf{T}(\mathbf{x})\}_{t=1}^T$ ;
```

Algorithm 2: Learning a single TAO tree with boosting weights

```
Result: trained tree $\mathbf{T}$
input training set $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$;
  initial random tree $\mathbf{T}(\cdot; \mathbf{\Theta})$ of depth $\Delta$;
  AdaBoost weights $\{w_n\}_{n=1}^N$;
repeat
    for depth $d = 0$ to $\Delta$ do
        for $i \in$ nodes at depth $d$ do
            if $i$ is a leaf then
                $y_i \leftarrow$ fit a weighted regressor (constant
                  or linear) on a reduced set $\mathcal{R}_i$ with the
                  current weights $\{w_n\}$;
            else
                $\boldsymbol{\theta}_i \leftarrow$ fit a weighted binary classifier to
                  minimize eq. (2) with the current
                  weights $\{w_n\}$;
            end
        end
    end
until convergence occurs or max iteration;
postprocessing: remove dead or pure subtrees;
```

linear predictors (denoted by TAO-l). We noticed that the performance improvement is drastic with TAO-l and this is the motivation of having trees with linear leaves. We use TAO algorithm to train such tree and modify the algorithm to handle the weighted loss.

The Tree Alternating Optimization (TAO) proposed in [10], [11] is an iterative algorithm which non-greedily optimizes a desired objective function over a tree and each iteration is guaranteed to decrease that objective. It has been successfully applied to train a single tree [25], [26] and ensemble of bagged trees [27], [28]. Furthermore, decision trees trained via TAO algorithm have nodes with hard splits (input follows exactly one child of a node) rather than a soft split [29] where an input instance is routed to each leaf with a certain probability. Although such trees still have hierarchical structure, it is generally accepted to consider them as separate models since they sufficiently differ from conventional trees (lack of conditional computation, interpretability, etc.).

The basis of TAO is given by the following two theorems: a *separability condition* and a *reduced problem* formulation. We refer a reader to [11] for in-depth analysis of the method and all the proofs which carry over with little modification. Separability condition states that by picking any set of nodes that are non-descendants and fixing the remaining nodes (i.e. parameters) of a tree, the loss function in eq. (1) *separates* over the parameters of the nodes in that set. This leads to the second theorem (reduced problem) which states that we can train them (i.e. non-descendant set of nodes) independently. Algorithm 2 shows how to train each node of a tree depending on its type. If node is a leaf then the solution is given either by taking an average response (if leaves are constant) or fitting a linear regressor (if leaves are linear) on a subset of points that reach the leaf. In the case of decision nodes, one can prove that the original problem in eq. (1) reduces to the following weighted binary classification problem on the training instances that reach the node i :

$$\min_{\boldsymbol{\theta}_i} \sum_{n \in \mathcal{R}_i} w_n \overline{L}(\overline{y}_n, f_i(\mathbf{x}_n; \boldsymbol{\theta}_i)) + \lambda \phi_i(\boldsymbol{\theta}_i) \tag{2}$$

where $\overline{L}$ is a 0/1 misclassification loss and $\overline{y}_n \in \{\text{right,left}\}$ is a "pseudolabel" indicating the child which gives a lower value of E for instance $\mathbf{x}_n$ under the current tree. Additionally, we use a sparsity penalty $\phi(\boldsymbol{\theta}_i) = \|\boldsymbol{\theta}_i\|_1$ to obtain more compact models. Note that the original TAO formulation solves *unweighted* binary classification problem instead. We further approximate the loss in eq. (2) with a convex surrogate (e.g. logistic) loss and solve it efficiently using LIBLINEAR [30]. We denote decision trees trained with the TAO algorithms as *TAO trees* to distinguish between conventional trees.

C. Training a tree with the boosting weights

Generic boosting framework involves fitting a base learner (decision trees in our case) to minimize the *weighted* loss function in eq. 1 at each boosting step, where the weights $\{w_n\}_{n=1}^N$ come from a particular boosting algorithm. Depending on how tree learning is done, some algorithms may not directly handle these weights in the loss. The problem comes from the difficulty of learning decision trees, which are non-differentiable functions. Therefore, various approximation techniques are used to simulate the weights and one such technique is "resampling" [31]–[33] which is utilized by majority of the practical AdaBoost implementations. The idea is to sample N instances with replacement and with probabilities

TABLE I
ROTATION ANGLE AND IMAGE PATCH LOCATION (GIVEN AS X,Y COORDINATES OF THE LOWER LEFT AND UPPER RIGHT CORNERS, RESPECTIVELY) FOR EACH CLASS OF THE MNIST REGRESSION PROBLEM.

class	0	1	2	3	4	5	6	7	8	9
angle	$-86°$	$-10°$	$-29°$	$16°$	$21°$	$-23°$	$-2°$	$-52°$	$-90°$	$-5°$
patch	[6 6 14 14]	[10 11 18 19]	[5 19 13 27]	[15 14 23 22]	[6 9 14 17]	[3 17 11 25]	[13 10 21 18]	[12 6 20 14]	[9 8 17 16]	[5 10 13 18]

TABLE II
SPECIFICATIONS OF THE DATASETS USED IN OUR EXPERIMENTS. N IS SAMPLE SIZE, D IS INPUT DIMENSION (NUMBER OF FEATURES) AND K IS OUTPUT DIMENSION.

Dataset	N_{train}	N_{test}	D	K
abalone	2 506	1 671	8	1
cpuact	4 915	3 277	21	1
CT slice	42 800	10 700	384	1
MNIST-rotated	60 000	10 000	784	64
YearPredictionMSD	463 715	51 630	90	1

$\{w_n\}_{n=1}^N$ for each data point. Indeed, their sum is always equal to one due to the normalization applied after each boosting step and each weight is equivalent to the "importance" of the corresponding point.

On the other hand, the reduced optimization problem in eq. (2) directly minimizes the weighted loss function which involves solving a weighted binary classification problem (for internal nodes). Although the given problem is NP-hard, it can be solved efficiently by approximating it with a convex surrogate (e.g. logistic) loss. Therefore, our base learners presented in the previous section do not resample the training set but rather directly handle the weighted losses which is methodologically more justified approach according to the problem formulation.

IV. EXPERIMENTS

We perform extensive evaluations of our proposed learning method across a diverse group of datasets (see Table II) from various domains: computer vision, signal processing, etc. We compare against state-of-the-art and established regression forests (e.g. XGBoost [17], Random Forest denoted as RF) that have stood the test of time and other recent baselines: refined RF [34], cRF [35], ARF [36], GIF [37]. Moreover, we empirically analyze boosted TAO trees from various perspectives such as hyperparameters choice.

Most of the datasets that we use are taken from the UCI Machine Learning Repository [38], except MNIST which is obtained from [39]. MNIST is originally designed for the classification task and we modify it to obtain a regression dataset as follows. We define a mapping that is locally linear but globally severely nonlinear and where both the input and output are high-dimensional. For each MNIST image, we generate its ground-truth output $\mathbf{y}_n \in \mathbb{R}^K$ by picking 8×8 patch of an image (patch location is defined separately for each class) and apply a class-specific image rotation to that patch. So, the output dimension is $K = 64$ resulting to the multi-output regression task. We assign an angle in $[-90°\ 90°]$ and a patch location at random to each class according to the Table I. Dataset specifications can be found in Table II.

A. Setup

We implemented TAO in C++ (with support of a Python interface). We initialize each TAO tree from a complete binary tree of depth Δ and having random weights at each node. At each boosting step, we train a TAO tree using $I = 20$ iterations throughout the experiments and tune a sparsity penalty (λ in eq. (2)) separately for each dataset. We parallelize the training of nodes at a given depth. We implemented 2 versions of the boosted TAO trees: one with constant leaves (R2 TAO-c) and another one with linear leaves (R2 TAO-l). For solving the reduced problem in a leaf, we either simply take average response values (for R2 TAO-c) or solve ℓ_1-regularized linear regression problem (a.k.a LASSO [40]) implemented in scikit-learn [41]. For the latter one, we set the same regularization penalty $\alpha = \lambda$. For solving the reduced problem in a decision (internal) node, we use an ℓ_1-regularized logistic regression solver in LIBLINEAR (v2.30 with support for instance weights) [30]. Here, the value of the C in LIBLINEAR is inversely proportional to the λ, i.e. $C = 1/\lambda$. Note that ℓ_1-regularized logistic regression in LIBLINEAR has randomized behavior, but for the fixed random seed it is deterministic. However, in TAO, we cannot control the order in which the *rand()* function in LIBLINEAR is called during parallel training, and, therefore, our TAO implementation is not deterministic. That said, this random behavior is negligible since the difference that one can obtain on the test error is not significant.

As for the baseline methods, we either use their available implementations or report their published results (rRF [34], cRF [35], ARF [36], GIF [37]). The most important baselines are Random Forests (RF) and conventional AdaBoost with CART-style trees (denoted as R2 CART). We use the Python implementation in scikit-learn [41] for both of them. Gradient boosting [12] is another important baseline. It fits approximate functional descent in forward stagewise additive modeling. In this paper, we use the XGBoost implementation (v0.81, Python package). It is highly optimized and supports parallel training. For all three methods above, we tune (as best as we could) the most important hyperparameters: maximum depth Δ of the base learner, learning rate η (when applicable) and the number of boosting iterations T (which is equivalent to the number of trees or `n_estimators`). We report the mean error (test) and standard deviation over 5 independent runs, where the error is the rooted mean squared error (RMSE) $E = \sqrt{\frac{1}{NK}\sum_{n=1}^N \|\mathbf{y}_n - \hat{\mathbf{y}}_n\|_2^2}$. Experiments were performed

TABLE III

COMPARISON OF DIFFERENT REGRESSION FORESTS, SORTED BY TEST ERROR E_{TEST}. FOR EACH DATASET, WE GIVE (N, D, K) = DATASET SIZE, INPUT AND OUTPUT DIMENSIONS. WE REPORT THE TEST RMSE (AVG$\pm$STDEV OVER 5 REPEATS), NO. OF PARAMETERS AND INFERENCE FLOPS (NUMBERS IN PARENTHESES ARE ESTIMATES), NUMBER OF TREES T AND MAXIMUM DEPTH OF THE FOREST Δ. BOOSTED TAO TREES ARE IN BOLDFACE.

	Forest	E_{test}	#pars.	FLOPS	T	Δ
abalone	CART	3.01±0.01	2 891	20	1	20
	XGBoost	2.22±0.00	31k	(1 089)	100	10
	XGBoost	2.20±0.00	220k	(9 349)	1k	10
	GIF [37]	2.18	(50k)	–	10	–
	TAO-c	2.18±0.05	287	41	1	6
	R2 CART	2.16±0.01	53k	(1k)	100	10
	R2 CART	2.15±0.00	0.5M	(10k)	1k	10
	RF	2.12±0.01	230k	(2 473)	100	29
	rRF [34]	2.10±0.01	(100k)	(1 000)	100	10
	ARF [36]	2.10±0.03	(100k)	(1 000)	100	10
	RF	2.10±0.00	2M	(25k)	1k	34
	R2 TAO-c	2.09±0.01	4.3k	1040	30	10
	R2 TAO-c	2.08±0.01	15k	3470	100	10
cpuact	CART	3.63±0.32	9 691	25	1	25
	TAO-c	2.71±0.04	498	51	1	6
	RF	2.62±0.04	0.6M	(2 842)	100	36
	ARF [36]	2.62±0.01	(98k)	750	50	15
	R2 CART	2.61±0.16	72k	(1k)	100	10
	RF	2.60±0.01	6M	(28k)	1k	37
	XGBoost	2.60±0.00	40k	(1 000)	100	10
	XGBoost	2.57±0.00	294k	(8 780)	1k	10
	R2 CART	2.56±0.11	0.7M	(10k)	1k	10
	R2 TAO-c	2.40±0.01	22k	1 772	30	8
	R2 TAO-c	2.32±0.00	92k	6 733	100	8
CT slice	CART	2.71±0.06	85k	51	1	51
	TAO-c	1.54±0.05	7k	1 123	1	7
	R2 CART	1.48±0.03	122k	(1 000)	100	10
	XGBoost	1.45±0.00	71k	(1 000)	100	10
	R2 CART	1.31±0.01	1M	(10k)	1k	10
	XGBoost	1.18±0.00	465k	(10k)	1k	10
	RF	1.03±0.01	5M	(5 818)	100	71
	cRF [35]	1.00	(17M)	–	1k	–
	RF	0.97±0.01	54M	(57k)	1k	78
	R2 TAO-c	0.59±0.00	441k	22k	30	8
	R2 TAO-c	0.51±0.00	1.5M	77k	100	8
	R2 TAO-c	0.31±0.00	4.2M	74k	100	12
YearPredictionMSD	CART	13.41±0.11	621k	49	1	49
	RF	9.31±0.00	40M	(5 237)	100	68
	R2 CART	9.25±0.01	2.5M	(1 500)	100	15
	RF	9.23±0.00	401M	(52k)	1k	73
	R2 CART	9.21±0.03	24M	(15k)	1k	15
	TAO-c	9.11±0.05	7k	448	1	8
	XGBoost	9.04±0.00	103k	(1 000)	100	10
	XGBoost	9.01±0.00	1.1M	(10k)	1k	10
	cRF [35]	8.90	(184M)	–	1000	–
	R2 TAO-c	8.85±0.00	1.2M	13k	30	10
	R2 TAO-c	8.83±0.00	3.9M	45k	100	10
	R2 TAO-l	8.82±0.00	874k	22k	50	7
MNIST-rotated	CART	28.51±0.11	119k	49	1	49
	RF	17.87±0.04	7.6M	(4 669)	100	59
	RF	17.18±0.03	71M	(42k)	1k	61
	XGBoost	16.79±0.00	44M	(84k)	3.2k	25
	R2 CART	16.65±0.09	109M	(80k)	3.2k	25
	R2 TAO-c	15.93±0.06	7M	32k	30	14
	R2 TAO-c	15.60±0.05	12M	55k	50	14
	TAO-l	15.22±0.54	20k	1 137	1	7
	R2 TAO-l	8.87±0.05	1.4M	68k	30	7

on Intel(R) Xeon(R) CPU E5-2699 v3 @ 2.30GHz. For implementations that support parallel processing, we set the number of threads to 8.

B. Results

Table III reports the results (sorted by decreasing test error) on single and multi-output (see MNIST-rotated) datasets. We differentiate two versions of the AdaBoost as follows: "R2 CART" is the conventional AdaBoost (R2 version) which employs CART trees, whereas "R2 TAO" is our modification. First of all, results indicate that even a single TAO tree ($T = 1$) already performs well by outperforming some forests (e.g. in "CT slice" and "YearPredictionMSD") which advocates for the power of optimization in tree learning. Next, consider ensembles of boosted TAO trees ($T > 1$). Our experimental results show that they achieve the *lowest test error in all datasets*, often by quite a considerable margin (e.g. MNIST-rotated, YearPredictionMSD). Moreover, our resulting forests have a fewer number of trees and they are shallower. For example, in most of these datasets, TAO achieves the lowest error with just 30 trees. However, it is also true that our oblique trees use more parameters at each decision node (since they are linear). Therefore, we also report the actual number of parameters and our estimates of the inference FLOPS (see below on how we estimate them) and they show that TAO forests have comparable model sizes. This is partly due to the sparsity penalty that we apply.

Next, we perform a more challenging task to validate the performance on multi-output regression (i.e. output is a continuous vector) task. To do so, we create a synthetic problem where the input is an MNIST digit (of dimension $D = 28 \times 28$ grayscale pixels) and the output is an image patch created using a nonlinear transformation (as described above). The last section of the Table III shows the results. Boosted TAO trees achieve significantly better accuracy on this multioutput regression problem. This is especially true with the forest of TAO-l, where its error is twice as low as the second best model. Please note that, for both TAO-c and TAO-l, *we are not creating a separate tree for each output dimension*, but rather handle it at each leaf, i.e. each leaf outputs a vector in $\mathbb{R}^K$ whereas previous boosting implementations (e.g. XGBoost) usually create K separate trees at each boosting step.

Estimating model sizes. We use the following methodology to calculate model sizes reported in Table III. For tree ensembles, it is calculated by summing up individual model (tree) sizes. The number of parameters of a single tree is equal to the sum of the number of parameters of all its nodes. For axis-aligned trees, it is equal to 2 (feature index and a threshold), and for oblique trees, this equals to the number of nonzero weights and can be at most $D + 1$ (input feature dimension and a bias term). Similarly, the number of parameters in a leaf is equal to K (output dimension) for constant leaves and at most $(D + 1)K$ for linear leaves. We estimate inference FLOPS for a single tree as the number of parameters an input instance encounters in the root-to-leaf path. We calculate the

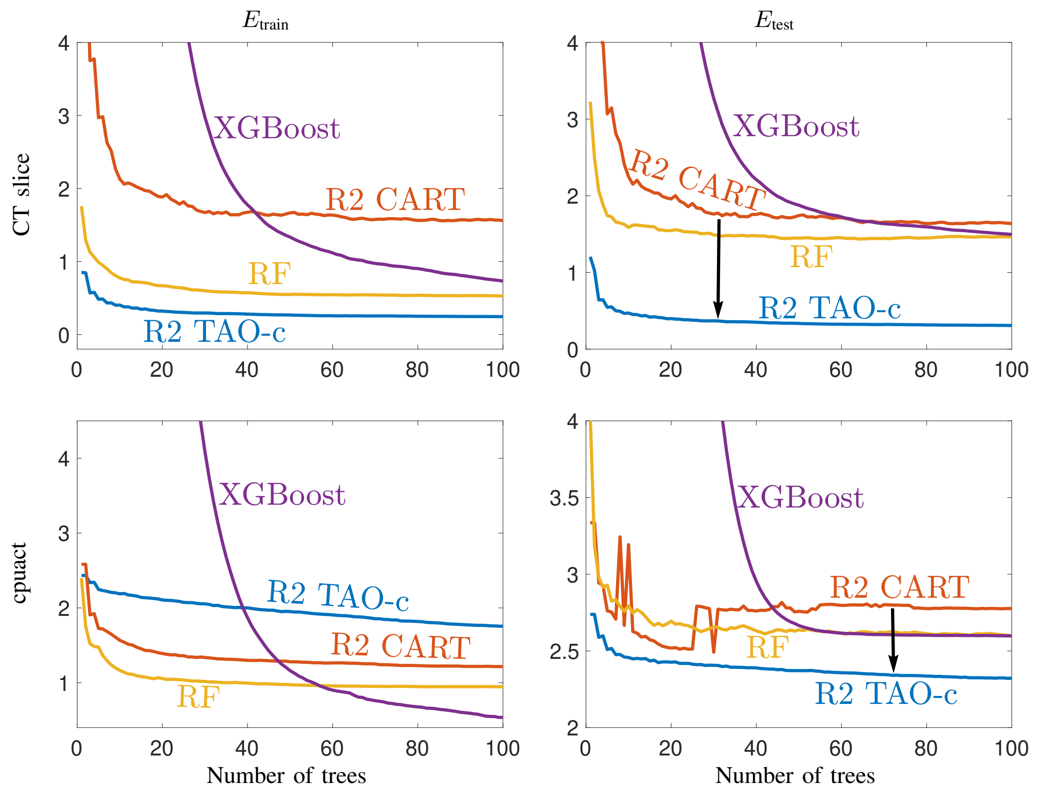

Fig. 1. Comparison of different regression forests on the CT slice and cpuact datasets as a function of the number of trees. "R2" refers to AdaBoost.R2, "RF" refers to Random Forest. All errors are RMSE.

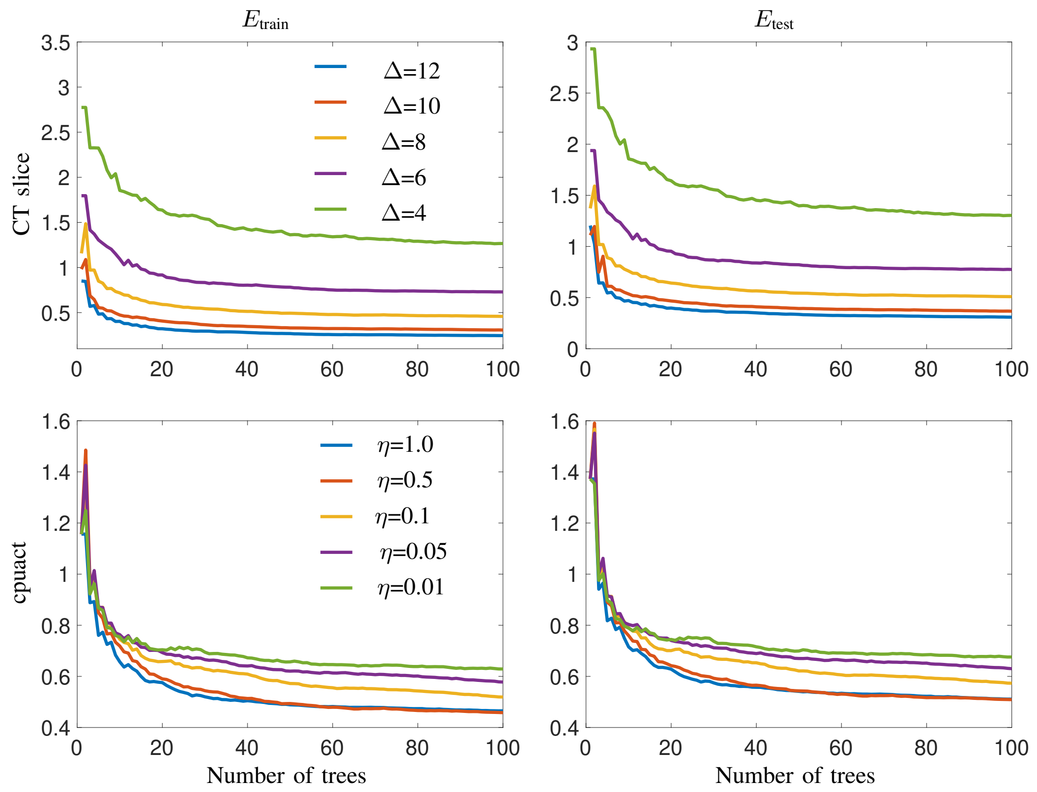

Fig. 2. The effect of the depth Δ (top row) and the learning rate η (bottom row) of the boosted TAO trees on CT slice and cpuact. All errors are RMSE.

inference FLOPS for each instance in the training set, and report the average result. For some methods where we do not have access to the trees (this includes cited results), we provide an upper bound on the number of parameters and inference FLOPS. We use the following assumptions in the calculation of upper bounds: number of parameters—we upper bound the number of decision (internal) nodes and the number of leaf nodes in a single tree by $\min(N, 2^{\Delta} - 1)$ and $\min(N, 2^{\Delta})$, respectively (N is the number of training points); inference FLOPS—we assume each root-leaf path to have depth Δ.

C. *Direct comparison of the base learners: TAO trees vs CART trees*

In fig. 1, we plot the test and train errors of the different tree ensembles as a function of the number of trees for the CT slice and cpuact datasets. For any fixed number of trees, R2-TAO-c achieves considerably better test error than all the other regression forests. The most important baseline is R2 CART since it uses the same boosting framework (AdaBoost.R2) but different base learners. Arrows on the right plots illustrate the improvement in performance achieved by replacing CART trees with TAO trees as base learners in AdaBoost.R2. We can clearly see a significant difference on the test error for the chosen two datasets and we have observed similar behavior in other datasets, too. It is worth mentioning that we have tuned hyperparameters for all algorithms as best as we could. This result clearly shows the superiority of oblique trees and the power of optimization performed by TAO.

D. *Model hyperparameters*

The most important hyperparameters in AdaBoost.R2 are the learning rate η (or shrinkage factor) and depth Δ of the individual learner. In fig. 2, we explore how these two hyperparameters affect the performance of our boosted TAO trees. According to the literature, the shrinkage factor η has a significant impact on model generalization [7]. Indeed, these figures show that the test error varies as we try different values for η. It seems that the larger values of η achieve the lowest train error, however it may result to overfitting (left bottom plot). Nevertheless, defining a good value for η depends on problem and one might need to tune this hyperparameter (e.g. using cross-validation) to find the best option. Comparison of different Δ suggests that it should be as large as possible but avoiding overfitting.

V. Conclusion

Our idea is to boost more powerful oblique trees which are better optimized thanks to the TAO algorithm. In this paper, we showed that a simple boosting technique such as AdaBoost.R2 together with TAO trees demonstrates excellent results for regression problems both in terms of accuracy and model size (we have also verified similar gains using AdaBoost.M1 and SAMME, two other different versions of AdaBoost [42]). Additionally, the proposed method is relatively simple to implement and use. As for the training time, AdaBoost trains each base model sequentially and TAO non-greedily optimizes over the parameters of a tree which adds additional cost to the runtime. Despite the parallelism inside TAO, experiments show quite slower runtime compared to Random Forests and XGboost, but similar runtime w.r.t. traditional AdaBoost. However, such long runtime is reasonable and justified by the consistently low test error they achieve. This makes our boosted trees a powerful machine learning framework which can be widely-used in many applications including: signal processing, computer vision, data mining, and many others.

References

[1] A. Verikas, A. Gelzinis, and M. Bacauskiene, "Mining data with random forests: A survey and results of new tests," *Pattern Recognition*, vol. 44, no. 2, pp. 330–349, 2011.

[2] A. Criminisi, J. Shotton, and E. Konukoglu, "Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning," *Foundations and Trends in Computer Graphics and Vision*, vol. 7, no. 2–3, pp. 81–227, 2012.

[3] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct. 2001.

[4] Y. Freund and R. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *J. Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[5] L. I. Kuncheva, *Combining Pattern Classifiers: Methods and Algorithms*, 2nd ed. John Wiley & Sons, 2014.

[6] Z.-H. Zhou, *Ensemble Methods: Foundations and Algorithms*, ser. Chapman & Hall/CRC Machine Learning and Pattern Recognition Series. CRC Publishers, 2012.

[7] T. J. Hastie, R. J. Tibshirani, and J. H. Friedman, *The Elements of Statistical Learning—Data Mining, Inference and Prediction*, 2nd ed., ser. Springer Series in Statistics. Springer-Verlag, 2009.

[8] L. J. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Belmont, Calif.: Wadsworth, 1984.

[9] J. R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[10] M. Á. Carreira-Perpiñán and P. Tavallali, "Alternating optimization of decision trees, with application to learning sparse oblique trees," in *Advances in Neural Information Processing Systems (NEURIPS)*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. MIT Press, Cambridge, MA, 2018, pp. 1211–1221.

[11] M. Á. Carreira-Perpiñán, "The Tree Alternating Optimization (TAO) algorithm: A new way to learn decision trees and tree-based models," 2021, arXiv.

[12] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001.

[13] D. Nigel and D. Helmbold, "Boosting methods for regression," *Machine Learning*, vol. 47, pp. 153–200, May 2002.

[14] H. Drucker, "Improving regressors using boosting techniques," in *Proc. of the 14th Int. Conf. Machine Learning (ICML'97)*, D. H. Fisher, Ed., Nashville, TN, Jul. 6–12 1997, pp. 107–115.

[15] R. Avnimelech and N. Intrator, "Boosting regression estimators," *Neural Computation*, vol. 11, pp. 499–520, Feb. 1999.

[16] D. Shrestha and D. Solomatine, "Experiments with adaboost.rt, an improved boosting scheme for regression," *Neural Computation*, vol. 18, no. 7, pp. 1678–1710, Jul. 2006. [Online]. Available: https://doi.org/10.1162/neco.2006.18.7.1678

[17] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. of the 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (SIGKDD 2016)*, San Francisco, CA, Aug. 13–17 2016, pp. 785–794.

[18] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "LightGBM: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems (NIPS)*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. MIT Press, Cambridge, MA, 2017, pp. 3146–3154.

[19] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "CatBoost: Unbiased boosting with categorical features," in *Advances in Neural Information Processing Systems (NEURIPS)*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. MIT Press, Cambridge, MA, 2018, pp. 6638–6648.

[20] X. Li, L. Wang, and E. Sung, "Adaboost with svm-based component classifiers," *Engineering Applications of Artificial Intelligence*, vol. 21, no. 5, pp. 785–795, Aug. 2008.

[21] L. J. Breiman, "Bias, variance, and arcing classifiers," Dept. of Statistics, University of California, Tech. Rep. 460, 1996.

[22] H. Schwenk and Y. Bengio, "Boosting neural networks," *Neural Computation*, vol. 12, no. 8, pp. 1869–1887, Aug. 2000.

[23] C. Yu and D. B. Skillicorn, "Parallelizing boosting and bagging," Dept. of Computing and Information Science, Queens University, Tech. Rep. 2001–442, Feb. 2001.

[24] C. Henry, R. Nock, and F. Nielsen, "Real boosting *a la Carte* with an application to boosting oblique decision tree," in *Proc. of the 20th Int. Joint Conf. Artificial Intelligence (IJCAI'07)*, Hyderabad, India, Jan. 6–12 2007, pp. 842–847.

[25] A. Zharmagambetov, S. S. Hada, M. Á. Carreira-Perpiñán, and M. Gabidolla, "An experimental comparison of old and new decision tree algorithms," Mar. 20 2020, arXiv:1911.03054.

[26] A. Zharmagambetov, M. Gabidolla, and M. Á. Carreira-Perpiñán, "Improved boosted regression forests through non-greedy tree optimization," in *Int. J. Conf. Neural Networks (IJCNN'21)*, Virtual event, Jul. 18–22 2021.

[27] A. Zharmagambetov and M. Á. Carreira-Perpiñán, "Smaller, more accurate regression forests using tree alternating optimization," in *Proc. of the 37th Int. Conf. Machine Learning (ICML 2020)*, H. Daumé III and A. Singh, Eds., Online, Jul. 13–18 2020, pp. 11 398–11 408.

[28] M. Á. Carreira-Perpiñán and A. Zharmagambetov, "Ensembles of bagged TAO trees consistently improve over random forests, AdaBoost and gradient boosting," in *Proc. of the 2020 ACM-IMS Foundations of Data Science Conference (FODS 2020)*, Seattle, WA, Oct. 19–20 2020, pp. 35–46.

[29] O. İrsoy, O. T. Yıldız, and E. Alpaydın, "Soft decision trees," in *Proc. 21st Int. Conf. Pattern Recognition (ICPR'12)*, Tsukuba Science City, Japan, Nov. 11–15 2012, pp. 1819–1822.

[30] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "LIBLINEAR: A library for large linear classification," *J. Machine Learning Research*, vol. 9, pp. 1871–1874, Aug. 2008.

[31] J. Friedman, T. Hastie, and R. Tibshirani, "Additive logistic regression: A statistical view of boosting," *Annals of Statistics*, vol. 28, no. 2, pp. 337–407, Apr. 2000.

[32] R. E. Schapire and Y. Singer, "Improved boosting algorithms using confidence-rated predictions," *Machine Learning*, vol. 37, pp. 297–336, Dec. 1999.

[33] R. E. Schapire and Y. Freund, *Boosting. Foundations and Algorithms*, ser. Adaptive Computation and Machine Learning Series. MIT Press, 2012.

[34] S. Ren, X. Cao, Y. Wei, and J. Sun, "Global refinement of random forest," in *Proc. of the 2015 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'15)*, Boston, MA, Jun. 7–12 2015, pp. 723–730.

[35] M. Denil, D. Matheson, and N. de Freitas, "Narrowing the gap: Random forests in theory and in practice," in *Proc. of the 31st Int. Conf. Machine Learning (ICML 2014)*, E. P. Xing and T. Jebara, Eds., Beijing, China, Jun. 21–26 2014, pp. 665–673.

[36] S. Schulter, C. Leistner, P. Wohlhart, P. M. Roth, and H. Bischof, "Alternating regression forests for object detection and pose estimation," in *Proc. 14th Int. Conf. Computer Vision (ICCV'13)*, Sydney, Australia, Dec. 1–8 2013, pp. 417–424.

[37] J.-M. Begon, A. Joly, and P. Geurts, "Globally induced forest: A prepruning compression scheme," in *Proc. of the 34th Int. Conf. Machine Learning (ICML 2017)*, D. Precup and Y. W. Teh, Eds., Sydney, Australia, Aug. 6–11 2017, pp. 420–428.

[38] M. Lichman, "UCI machine learning repository," http://archive.ics.uci.edu/ml, 2013.

[39] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[40] T. Hastie, R. Tibshirani, and M. Wainwright, *Statistical Learning with Sparsity: The Lasso and Generalizations*, ser. Monographs on Statistics and Applied Probability. Chapman & Hall/CRC, 2015.

[41] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, "Scikit-learn: Machine learning in Python," *J. Machine Learning Research*, vol. 12, pp. 2825–2830, Oct. 2011, available online at https://scikit-learn.org.

[42] M. Gabidolla, A. Zharmagambetov, and M. Á. Carreira-Perpiñán, "Improved multiclass adaboost using sparse oblique decision trees," 2021, submitted.