Supplementary to “The Tree Ensemble Layer: Differentiability meets Conditional Computation”

Hussein Hazimeh [1] **Natalia Ponomareva** [2] **Petros Mol** [2] **Zhenyu Tan** [3] **Rahul Mazumder** [1]

A. Notation

Table A.1 lists the notation used throughout the paper.

B. Appendix for Section 2

Figure 2 was generated by training a single tree with depth 10 using the smooth-step activation function. We optimized the cross-entropy loss using Adam (Kingma & Ba, 2014) with base learning rate $= 0.1$ and batch size $= 256$. The y-axis of the plot corresponds to the average number of reachable leaves per sample (each point in the graph corresponds to a batch, and averaging is done over the samples in the batch). For details on the diabetes dataset used in this experiment and the computing setup, please refer to Section D of the appendix.

C. Appendix for Section 3

C.1. Example of Conditional Forward and Backward Passes

Figure C.1 shows an example of the tree traversed by the conditional forward pass, along with the corresponding $T_{fractional}$ used during the backward pass, for a simple regression tree of depth $d = 4$. Note that only 3 leaves are reachable (out of the 16 leaves of a perfect, depth-4 tree). The values inside the boxes at the bottom correspond to the regression values (scalars) stored at each leaf. The output of the tree for the forward pass shown on the left of Figure C.1, is given by

$$T(x) = 0.8 \cdot 0.3 \cdot 1.5 + 0.8 \cdot 0.7 \cdot (-2.0) + 0.2 \cdot 2.1 = -0.34.$$

Note also that the tree used to compute the backward pass is substantially smaller than the forward pass tree since all the hard-routing internal nodes of the latter have been removed.

C.2. Memory Complexity of the Conditional Forward Pass

The memory requirements depend on whether the forward pass is being used for inference or training. For inference, a node can be discarded as soon as it is traversed. Since we traverse the tree in a depth-first manner, the worst-case memory complexity is $\mathcal{O}(d)$. When used in the context of training, additional quantities need to be stored in order to perform the backward pass efficiently. In particular, we need to store $l.prob$ for every reachable leaf l, and $\langle w_i, x \rangle$ for every internal node $i \in \mathcal{F}$, where $\mathcal{F}$ is set of ancestors of the reachable leaves whose activation is fractional—see Section 3.2 for a formal definition of $\mathcal{F}$. Note that $|F| = U - 1$ (as discussed after Definition 1). Thus, the worst-case memory complexity when used in the context of training is $\mathcal{O}(d+U)$.

C.3. Time Complexity of the Conditional Backward Pass

Lines 8 and 9 perform $\mathcal{O}(k)$ operations so each leaf requires $\mathcal{O}(k)$ operations. Lines 11 and 12 are $\mathcal{O}(1)$ since for every $i \in \mathcal{F}$, $\langle w_i, x \rangle$ is available from the conditional forward pass. Lines 13 and 14 are $\mathcal{O}(p)$, while line 15 is $\mathcal{O}(1)$. The total number of internal nodes traversed is $|\mathcal{F}|$. Moreover, we always have $|\mathcal{F}| = U - 1$ (see the discussion following Definition 1). Therefore, the worst-case complexity is $\mathcal{O}(Up + Uk)$. By Theorem 1, the maximum number of non-zero entries in the three gradients is $p + p|\mathcal{F}| + Uk = \mathcal{O}(Up + Uk)$ (and it is easy to see that this rate is achievable). The best-case complexity of Algorithm 2 is $\mathcal{O}(k)$—this corresponds to the case where there is only one reachable leaf ($U = 1$), so the fractional tree is composed of a single node.

[1]Massachusetts Institute of Technology [2]Google Research [3]Google Brain. Correspondence to: Hussein Hazimeh <hazimeh@mit.edu>.

Proceedings of the 37th International Conference on Machine Learning, Online, PMLR 119, 2020. Copyright 2020 by the author(s).

Table A.1: List of notation used.

Notation	**Space or Type**	**Explanation**
$\mathcal{X}$	$\mathbb{R}^p$	Input feature space.
$\mathcal{Y}$	$\mathbb{R}^k$	Output (label) space.
m	$\mathbb{Z}_{>0}$	Number of trees in the TEL.
$\mathcal{T}(x)$	Function	The output of TEL, a function that takes an input sample and returns a logit which corresponds to the sum of all the trees in the ensemble. Formally, $\mathcal{T} : \mathcal{X} \to \mathbb{R}^k$.
$T(x)$	Function	A single perfect binary tree which takes an input sample and returns a logit, i.e., $T : \mathcal{X} \to \mathbb{R}^k$.
d	$\mathbb{Z}_{>0}$	The depth of tree T.
$\mathcal{I}$	Set	The set of internal (split) nodes in T.
$\mathcal{L}$	Set	The set of leaf nodes in T.
$\mathcal{A}(i)$	Set	The set of ancestors of node i.
$\{x \to i\}$	Event	The event that sample $x \in \mathbb{R}^p$ reaches node i.
w_i	$\mathbb{R}^p$	Weight vector of internal node i (trainable). Defines the hyperplane split used in sample routing.
W	$\mathbb{R}^{\lvert\mathcal{I}\rvert \times p}$	Matrix of all the internal nodes weights.
$\mathcal{S}$	Function	Activation function $\mathbb{R} \to [0,1]$
$\mathcal{S}(\langle w_i, x\rangle)$	$[0,1]$	Probability (proportion) that internal node i routes x to the left.
$[l \swarrow i]$	Event	The event that leaf l belongs to the left subtree of node $i \in \mathcal{I}$.
$[l \searrow i]$	Event	The event that leaf l belongs to the right subtree of node $i \in \mathcal{I}$.
o_l	$\mathbb{R}^k$	Leaf l's weight vector (trainable).
O	$\mathbb{R}^{\lvert\mathcal{L}\rvert \times k}$	Matrix of leaf weights.
γ	$\mathbb{R}_{\geq 0}$	Non-negative scaling parameter for the smooth-step activation function.
L	Function	Loss function for training (e.g., cross-entropy).
U, N	$\mathbb{Z}_{>0}$	Number of leaves and internal nodes, respectively, that a sample x reaches.
$\mathcal{R}$	Set	The set of reachable leaves.
$\mathcal{F}$	Set	The set of ancestors of the reachable leaves, whose activation is fractional, i.e., $\mathcal{F} = \{i \in \mathcal{I} \mid i \in \mathcal{A}(l),\ l \in \mathcal{R},\ 0 < \mathcal{S}(\langle x, w_i\rangle) < 1\}$.

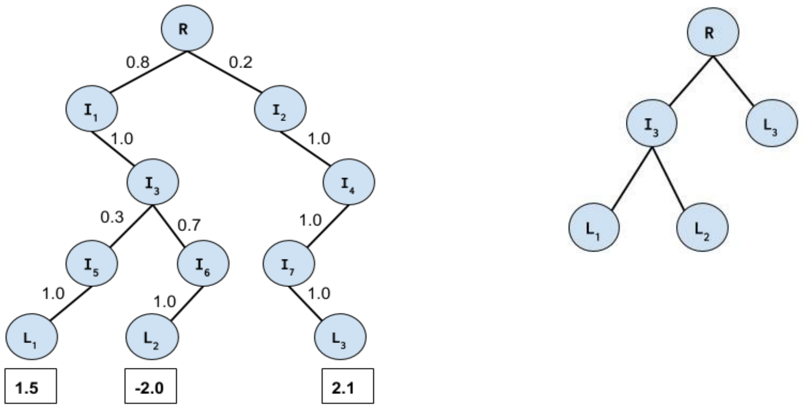

Figure C.1: **Left**: Reachable sub-tree for the conditional forward pass. **Right**: Corresponding (fractional) tree for the conditional backward pass where most of the internal (splitting) nodes of the forward pass sub-tree have been eliminated.

C.4. Proof of Theorem 1

Gradient of Loss w.r.t. x: By the chain rule, we have:

$$\underbrace{\frac{\partial L}{\partial x}}_{1\times p} = \underbrace{\frac{\partial L}{\partial T}}_{1\times k}\underbrace{\frac{\partial T}{\partial x}}_{k\times p} \tag{1}$$

The first term in the RHS above is available from backpropagation. The second term can be written more explicitly as follows:

$$\begin{aligned}\frac{\partial T}{\partial x} &= \frac{\partial \sum_{l\in\mathcal{L}} P(\{x\to l\})o_l}{\partial x}\\ &= \sum_{l\in\mathcal{L}} o_l \frac{\partial}{\partial x}\prod_{j\in\mathcal{A}(l)} r_{j,l}(x)\\ &= \sum_{l\in\mathcal{L}} o_l \sum_{i\in\mathcal{A}(l)} \frac{\partial}{\partial x} r_{i,l}(x) \prod_{j\in\mathcal{A}(l),j\neq i} r_{j,l}(x)\end{aligned}$$

We make three observations that allows us to simplify the expression above. First, if a leaf l is not reachable by x, then the inner term in the second summation above must be 0. This means that the outer summation can be restricted to the set of reachable leaves $\mathcal{R}$. Second, if an internal node i has a non-fractional $r_{i,l}(x)$, then $\frac{\partial}{\partial x} r_{i,l}(x) = 0$. This implies that we can restrict the inner summation to be only over $\mathcal{A}(l)\cap\mathcal{F}$. Third, the second term in the inner summation can be simplified by noting that the following holds for any $l\in\mathcal{R}$:

$$\prod_{j\in\mathcal{A}(l),j\neq i} r_{j,l}(x) = \frac{P(\{x\to l\})}{r_{i,l}(x)} \tag{2}$$

Note that in the above, $r_{i,l}(x)$ cannot be zero since $l\in\mathcal{R}$ (otherwise, l will be unreachable). Combining the three observations above, $\frac{\partial T}{\partial x}$ simplifies to:

$$\frac{\partial T}{\partial x} = \sum_{l\in\mathcal{R}} o_l \sum_{i\in\mathcal{A}(l)\cap\mathcal{F}} \frac{\partial}{\partial x} r_{i,l}(x)\frac{P(\{x\to l\})}{r_{i,l}(x)}. \tag{3}$$

Note that:

$$\frac{\partial}{\partial x} r_{i,l}(x) = \mathcal{S}'(\langle x, w_i\rangle) w_i^T (-1)^{\mathbb{1}[i\searrow l]}. \tag{4}$$

Plugging (4) into (3), and then using (1), we get:

$$\frac{\partial L}{\partial x} = \sum_{l\in\mathcal{R}} g(l) \sum_{i\in\mathcal{A}(l)\cap\mathcal{F}} w_i^T (-1)^{\mathbb{1}[i\searrow l]} \frac{\mathcal{S}'(\langle x, w_i\rangle)}{r_{i,l}(x)} \tag{5}$$

where

$$g(l) = P(\{x\to l\})\langle\frac{\partial L}{\partial T}, o_l\rangle. \tag{6}$$

Finally, we switch the order of the two summations in (5) to get:

$$\begin{aligned}\frac{\partial L}{\partial x} &= \sum_{i\in\mathcal{F}}\sum_{l\in\mathcal{R}|i\in\mathcal{A}(l)} g(l) w_i^T (-1)^{\mathbb{1}[i\searrow l]} \frac{\mathcal{S}'(\langle x, w_i\rangle)}{r_{i,l}(x)}\\ &= \sum_{i\in\mathcal{F}} \frac{\mathcal{S}'(\langle x, w_i\rangle)}{\mathcal{S}(\langle x, w_i\rangle)} w_i^T \sum_{l\in\mathcal{R}|[l\swarrow i]} g(l)\\ &- \sum_{i\in\mathcal{F}} \frac{\mathcal{S}'(\langle x, w_i\rangle)}{1-\mathcal{S}(\langle x, w_i\rangle)} w_i^T \sum_{l\in\mathcal{R}|[i\searrow l]} g(l)\end{aligned}$$

Gradient of Loss w.r.t. w_i: By the chain rule:

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial T}\frac{\partial T}{\partial w_i} \tag{7}$$

The first term in the summation is provided from backpropagation. The second term can be simplified as follows:

$$\begin{aligned}\frac{\partial T}{\partial w_i} &= \frac{\partial \sum_{l\in\mathcal{L}} P(\{x\to l\})o_l}{\partial w_i}\\ &= \sum_{l\in\mathcal{L}} o_l \frac{\partial}{\partial w_i}\prod_{j\in\mathcal{A}(l)} r_{j,l}(x)\\ &= \sum_{l\in\mathcal{L}|i\in\mathcal{A}(l)} o_l \frac{\partial}{\partial w_i} r_{i,l}(x) \prod_{j\in\mathcal{A}(l),j\neq i} r_{j,l}(x),\end{aligned} \tag{8}$$

If $i\in\mathcal{F}^c$ then the term inside the summation above must be zero, which leads to $\frac{\partial T}{\partial w_i} = 0$.

Next, we assume that $i\in\mathcal{F}$. If if a leaf l is not reachable, then the term inside the summation of (8) is zero. Using this observation along with (2), and simplifying we get the following for every $i\in\mathcal{F}$:

$$\begin{aligned}\frac{\partial L}{\partial w_i} &= \frac{\mathcal{S}'(\langle x, w_i\rangle)}{\mathcal{S}(\langle x, w_i\rangle)} x^T \sum_{l\in\mathcal{R}|[l\swarrow i]} g(l)\\ &- \frac{\mathcal{S}'(\langle x, w_i\rangle)}{1-\mathcal{S}(\langle x, w_i\rangle)} x^T \sum_{l\in\mathcal{R}|[i\searrow l]} g(l)\end{aligned}$$

Gradient of Loss w.r.t. O: Note that

$$\frac{\partial T}{\partial o_l} = \frac{\partial \sum_{v\in\mathcal{L}} P(\{x\to v\})o_v}{\partial o_l} \tag{9}$$

$$= P(\{x\to l\})I_k, \tag{10}$$

where I_k is the $k\times k$ identity matrix. Applying the chain rule, we get:

$$\frac{\partial L}{\partial o_l} = \frac{\partial L}{\partial T} P(\{x\to l\}). \tag{11}$$

D. Appendix for Section 4

Datasets: We consider 23 classification datasets from the Penn Machine Learning Benchmarks (PMLB) repository[1] (Olson et al., 2017). No additional preprocessing was done as the PMLB datasets are already preprocessed—see Olson et al. (2017) for details. We randomly split each of the PMLB datasets into 70% training and 30% testing sets. The three remaining datasets are CIFAR-10 (Krizhevsky et al., 2009), MNIST (LeCun et al., 1998), and Fashion MNIST (Xiao et al., 2017). For these, we kept the original training/testing splits (60K/10K for MNIST and Fashion MNIST, and 50K/10K for CIFAR) and normalized the pixel values to the range $[0, 1]$. A summary of the 26 datasets considered is in Table D.2.

Table D.2: Dataset Statistics

Dataset	# samples	# features	# classes
ann-thyroid	7200	21	3
breast-cancer-w.	569	30	2
car-evaluation	1728	21	4
churn	5000	20	2
CIFAR	60000	3072	10
crx	690	15	2
dermatology	366	34	6
diabetes	768	8	2
dna	3186	180	3
ecoli	327	7	5
Fashion MNIST	70000	784	10
flare	1066	10	2
heart-c	303	13	2
hypothyroid	3163	25	2
MNIST	70000	784	10
nursery	12958	8	4
optdigits	5620	64	10
pima	768	8	2
satimage	6435	36	6
sleep	105908	13	5
solar-flare_2	1066	12	6
spambase	4601	57	2
texture	5500	40	11
twonorm	7400	20	2
vehicle	846	18	4
yeast	1479	8	9

Computing Setup: We used a cluster running CentOS 7 and equipped with Intel Xeon Gold 6130 CPUs (with a 2.10GHz clock). The tuning and training was done in parallel over the competing models and datasets (i.e., each (model,dataset) pair corresponds to a separate job). For the experiments of Sections 4.1 and 4.2, each job involving TEL and XGBoost was restricted to 4 cores and 8GB of RAM, whereas LR and CART were restricted to 1 core and 2GB. The jobs in the experiment of Section 4.3 were each restricted to 8 cores and 32GB RAM. We used Python 3.6.9 to run the experiments with the following libraries: TensorFlow 2.1.0-dev20200106, XGBoost 0.90, Sklearn 0.19.0, Hyperopt 0.2.2, Numpy 1.17.4, Scipy 1.4.1, and GCC 6.2.0 (for compiling the custom forward/backward passes).

[1] https://github.com/EpistasisLab/penn-ml-benchmarks

D.1. Tuning Parameters and Architectures:

A list of all the tuning parameters and their distributions is given for every experiment below. For experiment 4.3, we also describe the architectures used in detail.

Experiment of Section 4.1 For the predictive performance experiment, we use the following:

- Learning rate: Uniform over $\{10^{-1}, 10^{-2}, \dots, 10^{-5}\}$.
- Batch size: Uniform over $\{32, 64, 128, 256, 512\}$.
- Number of Epochs: Discrete uniform with range $[5, 100]$.
- α: Log uniform over the range $[10^{-4}, 10^{4}]$.
- γ: Log uniform over the range $[10^{-4}, 1]$.

Experiment of Section 4.2:

TEL:

- Learning rate: Uniform over $\{10^{-1}, 10^{-2}, \dots, 10^{-5}\}$.
- Batch size: Uniform over $\{32, 64, 128, 256, 512\}$.
- Number of Epochs: Discrete uniform over $[5, 100]$.
- γ: Log uniform over $[10^{-4}, 1]$.
- Tree Depth: Discrete uniform over $[2, 8]$.
- Number of Trees: Discrete uniform over $[1, 100]$.
- L2 Regularization for W: Mixture model of 0 and the log uniform distribution over $[10^{-8}, 10^{2}]$. Mixture weights are 0.5 for each.

XGBoost:

- Learning rate: Uniform over $\{10^{-1}, 10^{-2}, \dots, 10^{-5}\}$.
- Tree Depth: Discrete uniform over $[2, 20]$.
- Number of Trees: Discrete uniform over $[1, 500]$.

- L2 Regularization (λ): Mixture model of 0 and the log uniform distribution over $[10^{-8}, 10^{2}]$. Mixture weights are 0.5 for each.
- min_child_weight = 0.

Logistic Regression: We used Sklearn's default optimizer and increased the maximum number of iterations to 1000. We tuned over the L2 regularization parameter (C): Log uniform over $[10^{-8}, 10^{4}]$.

CART: We used Sklearn's "DecisionTreeClassifier" and tuned over the depth: discrete uniform over $[2, 20]$.

Experiment of Section 4.3: CNN-Dense has the following architecture:

- Convolutional layer 1: has f filters and a 3×3 kernel (where f is a tuning parameter).
- Convolutional layer 2: has $2f$ filters and a 3×3 kernel.
- 2×2 max pooling
- Flattening
- Dropout: the dropout rate is a tuning parameter.
- Batch Normalization
- Dense layers: a stack of ReLU-activated dense layers, where the number of layers and the units in each is a tuning parameter.
- Output layer: dense layer with softmax activation.

CNN-TEL has the following architecture:

- Convolutional layer 1: has f filters and a 3×3 kernel (where f is a tuning parameter).
- Convolutional layer 2: has $2f$ filters and a 3×3 kernel.
- 2×2 max pooling
- Flattening
- Dropout: the dropout rate is a tuning parameter.
- Batch Normalization
- Dense layer: the number of units is a tuning parameter.
- TEL
- Output layer: softmax.

We used the following hyperparameter distributions:

- Learning rate: Uniform over $\{10^{-1}, 10^{-2}, \dots, 10^{-5}\}$.
- Batch size: Uniform over $\{32, 64, 128, 256, 512\}$.
- Number of Epochs: Discrete uniform over $[1, 100]$.
- γ: Log uniform over $[10^{-4}, 1]$.
- Tree Depth: Discrete uniform over $[2, 6]$.
- Number of Trees: Discrete uniform over $[1, 50]$.
- f: Uniform over $\{4, 8, 16, 32\}$.
- Number of dense layers in CNN-Dense: Discrete Uniform over $[1, 5]$.
- Number of units in dense layers of CNN-Dense: Uniform over $\{16, 32, 64, 128, 256, 512\}$.
- Number of units in the dense layer of CNN-TEL: Uniform over $\{16, 32, 64\}$.
- Dropout rate: Uniform over $[0.1, 0.5]$.