The tree ensemble layer

Studying the paper by Hazimeh *et al*

Elsa

https://ididio.com

April 30, 2020

Outline

1: Introduction

1: Introduction

Conditional computation is the ability of a model to use only a small part of its architecture based on what the inputs require

Representation learning is the ability to choose the features that matter the most for learning

Trees can do the former, and neural networks can do the latter.

2: The Tree Ensemble Layer (TEL)

Setting

TEL adds up a bunch of differentiable decision trees

The input and output

You have a p-dimensional input space $\mathcal{X}$ and a k dimensional output Y. For a regression problem $k = 1$ and for classification k is the number of classes. You will use m trees and your final output is

$$\mathcal{T}(x) = T^{(1)}(x) + T^{(2)}(x) + ... + T^{(m)}(x) \tag{1}$$

and we map

$$\mathcal{T} \subseteq \mathbb{R}^k \longrightarrow Y$$

using a softmax to find the class with the highest probability. Each tree $T^{(j)}$ is actually a **deterministic** function: it is the expectation over all its leaves.

Why do we need soft trees?

Classically, trees use *hard routing* in which input values determine a definite direction at each branch, and this leads to a discontinuous loss function that doesn't allow continuous optimization. So we want *soft trees* that perform *soft routing* and basically probabilities determine tree decisions. This allows gradient-based learning.

Notation: one binary tree T of depth d

- I: internal nodes
- $\mathcal{L}$: leaves
- $\mathcal{A}(i)$: the ancestors of any leaf or node i
- $\{x \rightsquigarrow i\}$ means the p-dim input x can reach the node i

Soft routing: weights determine prob of going left

Each *internal* node i has a weight vector w_i in the input space $\mathbb{R}^p$. For an activation function $\mathcal{S} : \mathbb{R} \to [0, 1]$ and an input x, the dot product of the weight and the input vector are sent to this activation function:

$$\mathcal{S}(\langle w_i, x \rangle) \text{ is the probability that } i \text{ routes } x \text{ to the left}$$

Soft routing: Probability that you reach a leaf

Notation:

- $[l \swarrow i]$ means the leaf l belongs to the left subtree of the node i
- $[l \searrow i]$: l belongs to the right subtree i

The probability that node i sends input x to the subtree that contains l is

$$\begin{aligned} r_l(x; w_i) &:= \text{prob route left} \cdot (1 \text{ or } 0) + \text{prob route right} \cdot (0 \text{ or } 1) \\ &= \text{prob route left}^{1 \text{ or } 0} \cdot \text{prob route right}^{0 \text{ or } 1} \\ &= \mathcal{S}(\langle x, w_i \rangle)^{\mathbb{1}[l \swarrow i]} (1 - \mathcal{S}(\langle x, w_i \rangle))^{\mathbb{1}[l \searrow i]} \end{aligned}$$

Because node routing choices are independent

$$P(x \rightsquigarrow l) = \prod_{i \in \mathcal{A}(l)} r_l(x; w_i) \tag{2}$$

Prediction and backwards-pass weights

Each leaf has a weight vector $o_l \in \mathbb{R}^k$ that is constant during the forward pass – it is not a function of the inputs.

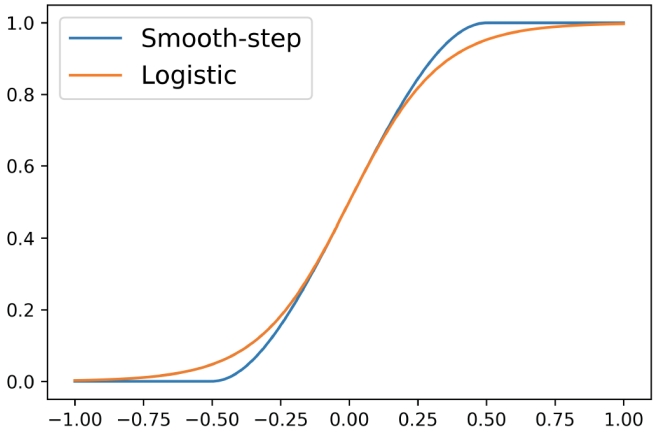
The prediction for each tree is

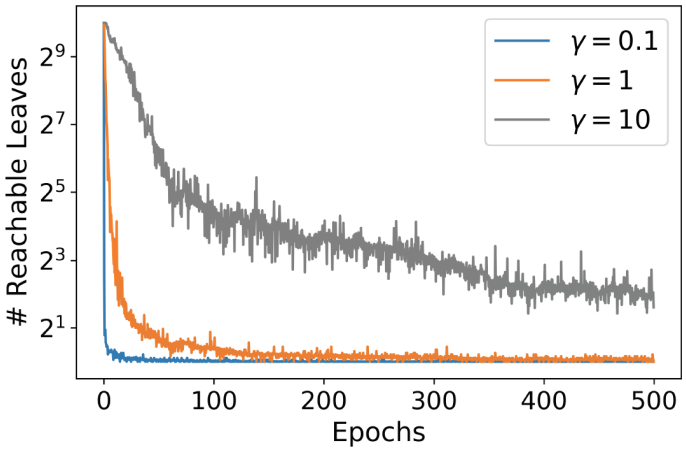
$$\begin{aligned} T^{(j)}(x) &= \sum_{leaves} \text{prob reach leaf} \times \text{a } k\text{-dim weight assigned later} \\ &= \sum_{l \in \mathcal{L}} P(x \rightsquigarrow l) o_l \end{aligned} \tag{3}$$

Choice of activation function S

Clever choice to avoid calc at each node: the **smooth step activation function** can output 0s and 1s leaving the tree itself as differentiable.

$$\mathcal{S}(t) = \begin{cases} 0 & \text{if } t \leq -\frac{\gamma}{2} \\ -\frac{2}{\gamma^3}t^3 + \frac{3}{2\gamma}t + \frac{1}{2} & \text{if } -\frac{\gamma}{2} \leq t \leq \frac{\gamma}{2} \\ 1 & \text{if } t \geq \frac{\gamma}{2} \end{cases} \tag{4}$$





Making the smooth-step S work

You need to avoid calculating at a lot of nodes and you need to avoid zero gradients at the internal nodes

- Performing batch normalization before the tree layer keeps $\mathcal{S}$ smooth and bounded and helps prevent zero gradients
- Tuning γ allows balance between performance and too many zero gradients
- For small γ the figure shows that the number of reachable leaves quickly converges to 1

3A: Conditional Forward Pass

How do you optimize TEL?

- Use a first-order method such as SGD (stochastic gradient descent)
- Avoid bottleneck in gradient computation that makes large ensembles inaccessible by exploiting sparcity

Setup

- TEL is a hidden layer that outputs $T(x)$, and for now 1 sample, 1 tree
- The net has layers after TEL, and we have the backpropagation algorithm for those layers that gives us $\frac{\partial L}{\partial T}$
- We have weight matrices

$$O = \begin{vmatrix} \text{leaf 1's } k\text{-dim weight vector} \\ \dots \\ \text{leaf } |\mathcal{L}|\text{'s weight vector} \end{vmatrix}$$

and

$$W = \begin{vmatrix} \text{internal node 1's } p\text{-dim weight vector} \\ \dots \\ \text{internal node } |\mathcal{I}|\text{'s weight vector} \end{vmatrix}$$

- For a cross-entropy or other loss function L, we need $\frac{\partial L}{\partial O}$ and $\frac{\partial L}{\partial W}$ to update the weight vectors and $\frac{\partial L}{\partial x}$ to continue the back-propagation

Conditional Forward Pass

This takes place before the gradients are calculated.

Algorithm summary

Traverse the tree from top to bottom only going down paths the activation function didn't turn off, and then calculate the tree expectation as given in equation (3)

More notation:

- Let $left(i)$ and $right(i)$ be the left and right children of any node
- Let $i.prob := P(x \rightsquigarrow i\ )$

Algorithm: Conditional Forward Pass

$$x \in \mathbb{R}^p, W \in \mathbb{R}^{|\mathcal{I}|} \times \mathbb{R}^p, O \in \mathbb{R}^{|\mathcal{L}|} \times \mathbb{R}^k \to T(x)$$

$output \leftarrow 0$, $toTraverse \leftarrow \{root\}$, $root.prob \leftarrow 1$
while $toTraverse \neq \emptyset$ **do**
  let $i \in toTraverse$ and $toTraverse \leftarrow toTraverse - \{i\}$
 if $i \in \mathcal{I}$ **then**
   $left(i).prob = i.prob \cdot S(\langle w_i, x \rangle)$
   $right(i).prob = i.prob \cdot (1 - S(\langle w_i, x \rangle))$
   $S(\langle w_i, x \rangle) > 0 \implies toTraverse \leftarrow toTraverse \cup left(i)$
   $S(\langle w_i, x \rangle) < 1 \implies toTraverse \leftarrow toTraverse \cup right(i)$
  else
   $output \leftarrow output + i.prob \cdot o_i$
  end if
end while

Forward Pass Complexity

- Calculating $\mathcal{S}(\langle w_i, x \rangle)$ for each reachable node is $\mathcal{O}(p)$
- For each reachable leaf the output update is $\mathcal{O}(k)$

So overall complexity is

$$\mathcal{O}\left(|\text{reachable int nodes}| \cdot p + |\text{reachable leaves}| \cdot k\right)$$

whereas a dense pass would be

$$\mathcal{O}\left(2^d p + 2^d k\right)$$

When γ is sufficiently small this algorithm has better complexity.

3B: Conditional Backward Pass

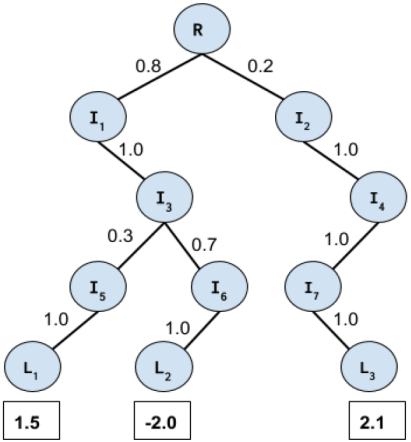
Conditional Backwards Pass – The bottom line

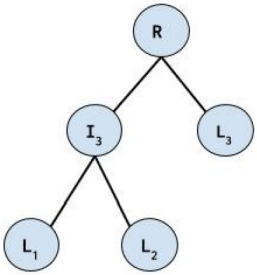
Here's what you need in order to calculate the gradients in the backwards pass:

- Travel all nodes that end in reachable leaves up to the leaves themselves
- Differentiate your nice cubic activation function S with respect to its scalar argument
- You'll need $P(\{x \rightsquigarrow l\})$ for all of those leaves but those were calculated in the forward pass

In order to do this efficiently, you need a fractional tree in which you remove nodes aren't needed for decision-making.

The fractional tree





You can create this tree during the forward pass for no real cost. There will always be one fewer internal node than leaves. The worse-case complexity is $\mathcal{O}(Np + Uk)$ where N and U are the number of reachable internal nodes leaves.

Conditional Backward Pass - Definitions

$$\begin{aligned}\mathcal{R} &:= \text{the reachable leaves} \\ &= \{l \in \mathcal{L} : l \text{ is reached}\} \\ \mathcal{F} &:= \text{the set of ancestors of the reachable leaves with frac. act.} \\ &= \{i \in \mathcal{I} : i \in \mathcal{A}(l), l \in \mathcal{R}, 0 < \mathcal{S}(\langle w_i, x\rangle) < 1\}\end{aligned}$$

Only $\mathcal{R}$ and $\mathcal{F}$ are needed to calculate the gradients – they are zero on the other nodes.

$$i.sum_g := \sum_{l \in R : i \in \mathcal{A}(l)} g(l) = \sum_{l \in R : i \in \mathcal{A}(l)} P(\{x \rightsquigarrow l\}) \langle \frac{\partial L}{\partial T}, o_l \rangle$$

Algorithm: Conditional Backward Pass

$$x \in \mathbb{R}^p, \text{tree parameters}, \frac{\partial L}{\partial T} \longrightarrow \frac{\partial L}{\partial x}, \frac{\partial L}{\partial W}, \frac{\partial L}{\partial O}$$

$\frac{\partial L}{\partial x} \leftarrow 0$
traverse $T_{\text{fractional}}$ in post order ($L1$, $L2$, $L3$, $I3$, R): call current i
 if $i \in \mathcal{L}$ **then**
  $\frac{\partial L}{\partial o_i} = \frac{\partial L}{\partial T} P(\{x \rightsquigarrow i\})$
  $i.sum_g = g(i)$
 else
  $a = \frac{\mathcal{S}'}{S} left(i).sum_g$
  $b = \frac{\mathcal{S}'}{1-S} right(i).sum_g$
  $\frac{\partial L}{\partial x} += w_i^T (a - b)$
  $\frac{\partial L}{\partial w_i} += x^T (a - b)$
  $i.sum_g = left(i).sum_g + right(i).sum_g$
 end if

Appendix

Conditional Backward Pass - $\frac{\partial L}{\partial x}$

Chain rule:

$$\begin{aligned} \frac{\partial L}{\partial x} &= \frac{\partial L}{\partial T} \cdot \frac{\partial T}{\partial x} \\ 1 \times p &= (1 \times k) \cdot (k \times p) \\ &\qquad \uparrow \text{available from backprop} \end{aligned} \tag{5}$$

Using equation (3) and additive property of derivatives followed by definition of P, and finally the product rule that would get very messy if you wrote it out:

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

Simplifying $\frac{\partial L}{\partial x}$ Slide 1

$$\frac{\partial T}{\partial x} = \sum_{l \in \mathcal{L}} o_l \frac{\partial}{\partial x} \prod_{j \in \mathcal{A}(l)} r_l(x; w_j)$$
$$= \sum_{l \in \mathcal{L}} o_l \sum_{i \in \mathcal{A}(l)} \frac{\partial}{\partial x} r_l(x; w_i) \prod_{j \in \mathcal{A}(l), j \neq i} r_l(x; w_j)$$

- $r_l(x; w_i)$ is the probability that l can be reached through node i, so the summation only needs to be over reachable leaves $\mathcal{R}$
- If $r_l(x; w_i) \in \{0, 1\}$, $\frac{\partial}{\partial x} r_l(x; w_i) = 0$, limiting inner sum to $\mathcal{A}(l) \cap \mathcal{F}$
- By the definition of $P(x \rightsquigarrow l)$ in equation (2), the product is $\frac{P(x \rightsquigarrow l)}{r_l(x; w_i)}$

$$\frac{\partial T}{\partial x} = \sum_{l \in \mathcal{R}} o_l \sum_{i \in \mathcal{A}(l) \cap \mathcal{F}} \frac{\partial}{\partial x} r_l(x; w_i) \frac{P(x \rightsquigarrow l)}{r_l(x; w_i)} \tag{7}$$

Simplifying $\frac{\partial L}{\partial x}$ Slide 2

$$\frac{\partial T}{\partial x} = \sum_{l \in \mathcal{R}} o_l \sum_{i \in \mathcal{A}(l) \cap \mathcal{F}} \frac{\boldsymbol{\partial}}{\boldsymbol{\partial x}} \boldsymbol{r_l(x; w_i)} \frac{P(x \rightsquigarrow l)}{r_l(x; w_i)} \tag{7}$$

$$\begin{aligned} r_l(x; w_i) &= \text{prob route left}^{1 \text{ or } 0} \cdot \text{prob route right}^{0 \text{ or } 1} \\ &= \mathcal{S}(\langle x, w_i \rangle)^{\mathbb{1}[l \swarrow i]} (1 - \mathcal{S}(\langle x, w_i \rangle))^{\mathbb{1}[l \searrow i]} \end{aligned}$$

So either the first or second term is 1, so we need $\frac{d}{dx}\mathcal{S}(\langle x, w_i \rangle)$ possibly times -1. Finally, apply the chain rule with respect to the inner product that feeds $\mathcal{S}$

$$\frac{\partial}{\partial x} r_l(x; w_i) = \boldsymbol{\mathcal{S}'(\langle x, w_i \rangle) w_i^T (-1)^{\mathbb{1}[l \searrow i]}} \tag{8}$$

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial T} \cdot \frac{\partial T}{\partial x} \tag{5}$$

$$\frac{\partial T}{\partial x} = \sum_{l \in \mathcal{R}} o_l \sum_{i \in \mathcal{A}(l) \cap \mathcal{F}} \frac{\boldsymbol{\partial}}{\boldsymbol{\partial x}} \boldsymbol{r_l(x; w_i)} \frac{P(x \rightsquigarrow l)}{r_l(x; w_i)} \tag{7}$$

$$\frac{\partial}{\partial x} r_l(x; w_i) = \boldsymbol{\mathcal{S}'(\langle x, w_i \rangle) w_i^T (-1)^{\mathbb{1}[l \searrow i]}} \tag{8}$$

Combining the above:

$$\frac{\partial L}{\partial x} = \sum_{l \in \mathcal{R}} g(l) \sum_{i \in \mathcal{A}(l) \cap \mathcal{F}} w_i^T (-1)^{\mathbb{1}[i \searrow l]} \frac{\mathcal{S}'(\langle x, w_i \rangle)}{r_l(x; w_i)} \tag{9}$$

where

$$g(l) = P(\{x \rightsquigarrow l\}) \langle \frac{\partial L}{\partial T}, o_l \rangle \tag{10}$$

Simplifying $\frac{\partial L}{\partial x}$ Slide 4

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

$$= \sum_{i \in \mathcal{F}} \sum_{l \in \{\ell \in \mathcal{R} : i \in \mathcal{A}(\ell)\}} g(l) w_i^T (-1)^{\mathbb{1}[i \searrow l]} \frac{\mathcal{S}'(\langle x, w_i \rangle)}{r_l(x; w_i)}$$

$$= \sum_{i \in \mathcal{F}} \frac{S'(\langle x, w_i \rangle)}{S(\langle x, w_i \rangle)} w_i^T \sum_{l \in \{\ell \in \mathcal{R} : [l \swarrow i]\}} g(l)$$

$$- \sum_{i \in \mathcal{F}} \frac{S'(\langle x, w_i \rangle)}{1 - S(\langle x, w_i \rangle)} w_i^T \sum_{l \in \{\ell \in \mathcal{R} : [l \searrow i]\}} g(l)$$

Conditional Backward Pass - $\frac{\partial L}{\partial w_i}$ - just like $\frac{\partial L}{\partial x}$

Chain rule:

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial T} \cdot \frac{\partial T}{\partial w_i} \tag{11}$$

These steps are just like the previous derivation:

$$\begin{aligned}
\frac{\partial T}{\partial w_i} &= \frac{\sum_{l \in \mathcal{L}} P(\{x \rightsquigarrow l\}) o_l}{\partial w_i} \\
&= \sum_{l \in \mathcal{L}} o_l \frac{\partial}{\partial w_i} \prod_{j \in \mathcal{A}(l)} r_l(x; w_j) \\
&= \sum_{l \in \mathcal{L}: l \in \mathcal{A}(l)} o_l \frac{\partial}{\partial w_i} r_l(x; w_i) \prod_{j \in \mathcal{A}(l), j \neq i} r_l(x; w_j)
\end{aligned}$$

Conditional Backward Pass - $\frac{\partial L}{\partial w_i}$ - slide 2

$$\frac{\partial T}{\partial w_i} = \sum_{l \in \mathcal{L}: l \in \mathcal{A}(l)} o_l \frac{\partial}{\partial w_i} r_l(x; w_i) \prod_{j \in \mathcal{A}(l), j \neq i} r_l(x; w_j)$$

If $i \in \mathcal{F}^C$ that means i never reaches the set $\mathcal{L}$ so $r_l(x; w_i)$ is constant and its derivative is 0. If $i \in \mathcal{F}$ then we're only dealing with reachable leaves l. Again using the definition of $P(x \rightsquigarrow l)$ in equation (2), we remember the product is $\frac{P(x \rightsquigarrow l)}{r_l(x; w_i)}$. So:

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

Conditional Backward Pass - $\frac{\partial L}{\partial o_l}$

From the definition of our prediction T in equation (3), and noting that the only dependence is on the weight itself we get

$$\begin{aligned} \frac{\partial T}{\partial o_l} &= \frac{\partial}{\partial o_l}\left[\sum_{v\in\mathcal{L}} P\left(x \rightsquigarrow v\right) o_v\right] \\ &= P\left(\{x \rightsquigarrow l\}\right) \end{aligned} \tag{13-14}$$

where I_k is the $k \times k$ identity matrix.
Finally,

$$\frac{\partial L}{\partial o_l} = \frac{\partial L}{\partial T} P(\{x \rightsquigarrow l\}) \tag{15}$$