Pros and cons of soft vs hard decision trees

Kuat Gazizov
Dept. CSE, UC Merced
kgazizov@ucmerced.edu

Arman Zharmagambetov
Meta AI (FAIR)
armanz@meta.com

Miguel Á. Carreira-Perpiñán
Dept. CSE, UC Merced
mcarreira-perpinan@ucmerced.edu

Decision trees come in two basic types. In *soft decision trees (SDT)* (also called *hierarchical mixtures of experts* [4]), an input instance follows each root-leaf path, each having a positive probability, and the tree output is the probability-weighted average of its leaf predictions. This happens because each decision node produces a positive probability for each of its children. In *(hard) decision trees (HDT)* [1, 5], an input instance follows exactly one root-leaf path because each decision node picks only one child. Both types have been around for decades, but we find no proper comparison of them (across factors such as accuracy, interpretability, inference time, etc.), which is surprising given the importance of decision trees. We have carried out a theoretical and empirical comparison and provide a brief summary of our conclusions here. We focus on a single tree (not a forest) with oblique (not axis-aligned) decision nodes and constant leaves for classification.

Use in applications Thousands of research papers have been written about both SDTs and HDTs, and they appear in any machine learning textbook. HDTs are also widely used in actual practice, particularly with tabular data, whether in isolation or ensembled into a forest. Many commercial statistical software packages implement them, as well as open-source ones such as scikit-learn. However, we find no significant uses of SDTs in practical applications, and the few software implementations we find are limited projects. This already suggests that SDTs have not found a place in real-world applications, unlike HDTs. We give more concrete evidence for this next.

Training algorithm SDTs define a smooth model, so (maximum likelihood) training is relatively straighforward [4]. In our empirical evaluation we use a GPU-enabled, gradient-based implementation. In contrast, HDTs define a very difficult, nondifferentiable optimization problem. For decades, greedy recursive partitioning algorithms such as CART [1] and C4.5 [5] have been used, even though they are quite suboptimal. More recently, the Tree Alternating Optimization (TAO) algorithm [3] has made it possible to train HDTs much like any other ML model: by monotonically decreasing at each iteration an essentially arbitrary choice of loss function, regularization and type of decision nodes and leaves, over a fixed-structure tree. This results in trees both smaller and more accurate.

Predictive accuracy Given the same structure, a SDT will always be at least as accurate as a HDT, because the latter is constrained to predict using a single leaf rather than a weighted average of all leaves. However, the far faster inference time on a HDT can often compensate for this by making the HDT larger, as shown in fig. 1. (Also, using a small forest of HDTs will easily beat the SDT.)

Interpretability: do the experts specialize in a SDT? Can we "harden" a SDT? HDTs are recognized among the most interpretable models because the instance follows a single root-leaf path, whose logic can be inspected. In a SDT, the instance follows all paths, which defeats this purpose. Can we "round" or "harden" a SDT so it becomes a HDT at inference time, as sometimes proposed in the literature (e.g. [6])? Empirically, we find this works badly, as follows. We can harden a SDT in two different ways. 1) We can make each decision node hard once and for all (replacing the logistic function with the step function), i.e., we greedily choose the most likely child at each decision node and thus follow a single root-leaf path. 2) We can use the highest-probability leaf for the given input instance (which is very slow since we have to compute all path probabilities anyway). Either way, we consistently observe (see fig. 1) that the resulting HDT's accuracy drops drastically, and is much worse than that of a properly trained HDT (with TAO). Another way to see this is by looking at the entropy of the leaf distribution (on average over all training instances), or

Preprint. Under review.

the probability of a given decision node to choose the right child, or the effective overlap in number of training instances between pairs of leaves (fig. 2). We find that a typical instance reaches many leaves, a typical decision node is often uncertain, and pairs of leaves ("experts") are far from local and overlap considerably.

That is, either we follow all the paths and the SDT is really no different than a regular mixture of experts, or we follow a single path greedily and then we do obtain a HDT, but a very inaccurate one. A SDT really is a (flat) mixture of experts, but with instance-dependent weights for the experts. The hierarchical structure does not buy us much; a neural network with some other architecture will likely be more accurate. A decision tree is really useful when we follow a single path (both for interpretability and fast inference), i.e., a HDT. Empirically, we find we learn much better (smaller, more accurate) HDTs using TAO than training a SDT and hardening it a posteriori.

Runtime At inference time given a test instance, a SDT follows 2^{Δ} paths if the depth is Δ. For training, computing the gradient of a SDT is also exponential on the depth for every training instance (since it visits each node). This makes deep SDTs exceedingly slow for both training and inference.

A HDT is obviously much faster at inference time because it follows a single path, but it is also much faster at training time. In TAO, a separability condition [3] guarantees that the optimization over each node depends only on the instances that currently reach it (the "reduced set"). This means that one pass over all nodes is actually linear, not exponential, on the tree depth. Besides, non-descendant nodes (e.g. all nodes at the same depth) can be optimized in parallel.

Sparsity and pruning TAO can use an ℓ_1 penalty $\|\mathbf{w}_i\|_1$ on the decision nodes' weight vectors to make them sparse. This also results in pruning the initial tree structure: a node with $\mathbf{w}_i = \mathbf{0}$ becomes redundant (it sends all instances to the same child) and can be removed. This is not so with SDTs: a node with $\mathbf{w}_i = \mathbf{0}$ continues to send each instance to both children (although with a constant probability), so it cannot be pruned. This disappointing fact is unlike pruning in neural nets via an ℓ_1 or ℓ_0 term [2], where a neuron having a zero weight vector can be removed. (In fact, we find no work on training a SDT with sparsity constraints, or with axis-aligned decision nodes.)

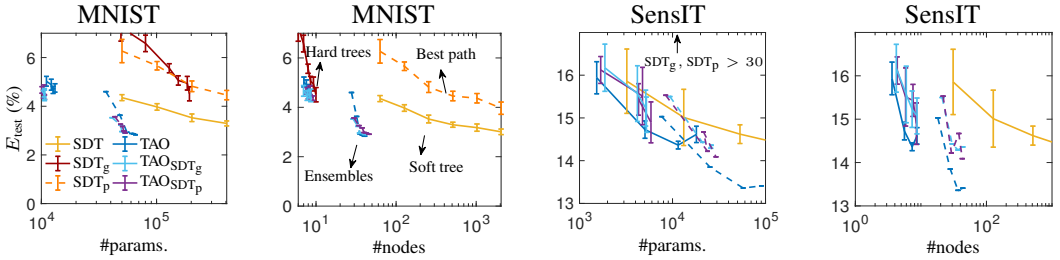

Figure 1: Test error vs total number of parameters and number of nodes traversed in a HDT or a SDT, hardened in different ways (for MNIST and SensIT datasets). SDT_p (or Best path) is a hardened SDT by using root-to-leaf path with the highest probability and SDT_g by replacing the sigmoid by the step function. The $\text{TAO}_\bullet$ tree refers to a TAO initialized from the given tree.

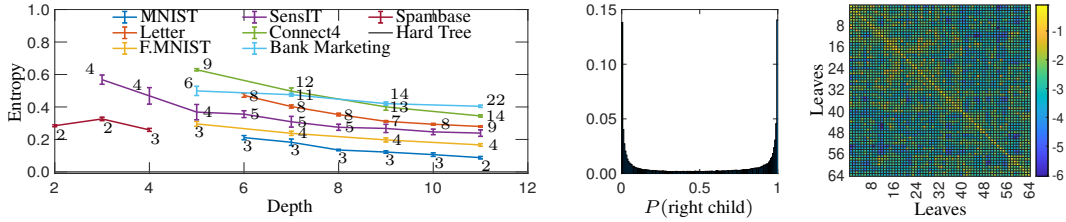

Figure 2: *Left*: leaf distribution entropy (normalised by the tree depth) for SDTs on several datasets; the numbers on the curve equal the effective number of leaves reached. *Middle*: normalized histogram of the right child probability (root of SDT of depth 8 in MNIST). *Right*: matrix of the overlap between pairs of leaves in effective number of instances for a SDT (for a HDT, the matrix is always the identity).

Acknowledgments: work funded in part by NSF award IIS–2007147.

References

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

[2] M. Á. Carreira-Perpiñán and Y. Idelbayev. "Learning-compression" algorithms for neural net pruning. In *Proc. of the 2018 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'18)*, pages 8532–8541, Salt Lake City, UT, June 18–22 2018.

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

[4] M. I. Jordan and R. A. Jacobs. Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6(2):181–214, Mar. 1994.

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

[6] R. Tanno, K. Arulkumaran, D. C. Alexander, A. Criminisi, and A. Nori. Adaptive neural trees. In K. Chaudhuri and R. Salakhutdinov, editors, *Proc. of the 36th Int. Conf. Machine Learning (ICML 2019)*, pages 6166–6175, Long Beach, CA, June 9–15 2019.