Backpropagation applied to random tessellation trees

Frank Zhang

30 June 2019

1 Abstract

Deep neural networks and decision trees operate under two distinct paradigms; the former performs representation learning with pre-specified architecture while the latter induces partitions over pre-specified features using a data-driven architecture. Inspired by the use of decision trees to construct visual codebooks, this work seeks to learn trees using a process that is both robust to large datasets and allows them to be optimized end-to-end in a unified deep learning architecture for image recognition applications. The method of tree growth is inspired by recent work on a nonparametric space partitioning methods called the Randomized Tessellation process and the stochastic routing formulation of soft decision trees is used to enable gradient descent optimization. An initial assessment of the proposed learning algorithm is conducted using the MNIST dataset that also shows how the modular compatible neural tree can be incorporated into a traditional neural network stack.

2 Introduction

Decision tree learning is a popular data modeling technique due in part to the simplicity of trees. For real-world applications, ensembles of decision trees, such as random forests and gradient boosted trees, are widely used for their competitive predictive performance and better generalization compared to single tree models. Though deep learning approaches have largely dominated current computer vision tasks such as image classification and semantic segmentation, decision trees are still of interest for the proximity measures they can induce. Specifically, once a decision tree has been trained, the proximity between samples can be calculated by determining the number of times they land in the same terminal node or the number of tree splits that separate them. Conducting such similarity assessments can aid in clinical reasoning for example by enabling image-content based retrieval of previous cases, which may also ease model interpretability. By contrast, current neural network descriptive measures lack an analogous method to perform such auxiliary comparisons in addition to the main predictive task.

Nevertheless, one of the major draws of deep learning approaches is the joint learning of feature representations together with their classifiers, which has typically outperformed conventional handcrafted feature pipelines. Current decision tree techniques fall short in this regard in that the input or feature space on which they operate is typically predefined and unchanged, unlike the multilayered transformations afforded by current neural network architectures. One way to surmount this shortcoming is to reformulate the decision tree as a neural network with differentiable split functions to guide inputs through a tree while delivering gradients through backpropagation to the lower layers of a servicing network. This approach has been taken in similar works, each using different mappings between decision trees and neural networks [1, 2, 3]. However, only a few seriously explore reformulating decision trees in a modular compatible manner.

The main contribution of this work is to provide a method for reformulating a decision tree learner as a neural network layer that is more faithful to the growing process of traditional trees [4]. A logical way forward currently exists using Bayesian nonparametric space partitioning methods such as the Mondrian process and its recent extension, the Random Tessellation process [5, 6]. At this point, no theoretical study has yet been reported on optimizing the splits produced by these methods, which are chosen independent of the labels, using a method like backpropagation. Thus, the current approach is twofold: augmenting decision trees with representation learning and optimizing the trees produced by the aforementioned stochastic processes so the splits better reflect the labels.

3 Methods

3.1 Decision Tree Framework

The general task of prediction involves N labeled examples $(\boldsymbol{x}_1, y_1)$,...,$(\boldsymbol{x}_n, y_n)$ from $\mathbb{R}_d \times \mathcal{Y}$ wherein the task is to predict labels $y \in \mathcal{Y}$ for unlabeled test points $\boldsymbol{x} \in \mathbb{R}_d$.

A decision tree on $\mathbb{R}_d$ is a hierarchical partitioning of $\mathbb{R}_d$ and a rule for predicting y given the partition containing a data item. Most approaches consider a strictly binary tree T, which contains a finite set of nodes such that every node j has exactly one parent node, except for a root node ϵ which has no parent, and every node j is the parent of exactly zero or two children nodes, called the left child $\mathrm{left}(j)$ and the right child $\mathrm{right}(j)$. Each node of the tree j in T is associated with a block B_j in $\mathbb{R}_d$ of the input space such that, at the root, $B_\epsilon = \mathbb{R}_d$ or the entire space, while each internal node with two children is a split of its parent's block into two smaller blocks. Nodes with two children are decision nodes or splits of T indexed by $\mathcal{N}$ because they are assigned a decision function parameterized by Θ, which is responsible for routing samples along the tree and determining the size of each child block. Those nodes without children are referred to as prediction nodes or leaves of T indexed by $\mathcal{L}$ and together form the partition of $\mathbb{R}_d$. During prediction, the input is first passed into the tree root node and, when it reaches a decision node, sent to the left or right subtree based on the output of the decision function. This is repeated until a leaf node is reached, at which point the rule is applied to obtain an output.

3.2 Soft Decision Trees

Suppose a tree has already been provided. To allow for training based on gradient backpropagation, the rule and especially the decision function should be differentiable. The approach taken here has been described in previous works as stochastic routing [1]. As opposed to the hard decision node which redirects instances to one of its children, a soft decision node redirects instances to all its children with probabilities calculated by the decision function. The probability of either going left or right at node j, or p_j^L or p_j^R respectively, is described by the following functions:

$$p_j^L = 1 - g_j(x) \tag{1}$$

$$p_j^R = g_j(x) \tag{2}$$

$$g_j(x) = \frac{1}{1 + exp[-(w_j^T x + w_{j0})]} = \sigma(w_j^T x + w_{j0}) \tag{3}$$

where $\sigma(x)$ is the sigmoid function, and the linear gating function $w_j^T x + w_{j0}$ defines an oblique split, which is in contrast to the axis-orthogonal splits often used by other tree algorithms. A pre-

viously proposed interpretation is to consider each $g_j(x)$ as a linear output unit of a deep network that will be turned into a probabilistic routing decision by the action of a sigmoid activation [1].

To determine the output of the tree, the rules of each leaf node are averaged by the probability of reaching the leaf, which is simply the product of all decision node probabilities along the path from the root node to the leaf. Typically, for most decision tree models, the rule is a class probability distribution for classification or constant output for regression determined by computing the appropriate statistics of the data partition that reaches that leaf. In line with maintaining differentiability through the various levels of the model, one can instead view the rule as an embedding layer of a neural network that maps the leaf index to vectors of real numbers as appropriate for the task. Alternatively, with the benefit of using backpropagation to perform optimization, more complicated terminal models within the leaves can be considered, such as linear models and even additional neural networks. In essence, the soft decision tree is a mixture of experts model that uses the tree membership probability as the gating network [7].

3.3 Tree Construction

The tree structure is inferred from the data based on some procedure which forms the crux of the learning algorithm. Of interest here is the method of inferring the order of splits and how those splits are initially oriented in the input space. While randomly initialized splits have been considered in the past, the approach taken here builds on the sampling procedures outlined in the Mondrian and Random Tessellation processes for trees [5, 6].

A Randomized Tessellation process, of which the Mondrian process is a special case, is a continuous-time Markov process ($\mathcal{M}_t : t \geq 0$), in which events are cuts (specified by hyperplanes) of $\mathbb{R}_\mathrm{d}$ with an associated split time t $t_j \geq 0$ for each node j. Split times increase with depth, i.e. $t_j > t_{\mathrm{parent}(j)}$, where $\mathrm{parent}(j)$ denotes the parent of node j, and $t_{\mathrm{parent}(\epsilon)} = 0$. The expected depth of a Mondrian tree is parameterized by a non-negative lifetime parameter $\lambda > 0$, though $\lambda = \infty$ is commonly used. The depth can also be controlled by only splitting nodes that contain more than a predetermined minimum number of data points. Both stopping criterion are described here.

The algorithm builds an unpruned decision tree according to the classical top-down procedure. Each block is associated with a bound that is approximated by a closed ball whose radius r_j and position c_j are computed using the dimension-wise maximum $max_d(x)$ standard deviation and mean respectively of an isotropic Gaussian. The approximation is used for block lifetime calculations for reasons described in the Randomized Tessellation process [6]. Like the Randomized Tessellation process, the splits are sampled independent of the labels and are not necessarily axis-aligned. More precisely, the sampled hyperplane intersects the ball centered at the mean of the data. However, only splits that separate the input data into two partitions are accepted. This filters out splits that do not further separate the dataset predictors.

An additional element when considering a smoothened decision boundary is the contrast of the sigmoid activation or the sharpness of the transition from 0 to 1. To achieve this, the weights and bias of the linear gating function $w_j^T x + w_{j0}$ are both scaled by a hyperparameter that is inversely proportional to the radius r_j. The default suggested here is scaling by a factor $\frac{3}{r_j}$ which ensures that the probability of separation of data at the periphery of the sphere is 95% initially.

4 Experiments

Behavior of the proposed learning algorithm was evaluated for initial feasibility using the MNIST classification dataset [8]. All models are implemented in Mxnet [9].

For all the experiments detailed, the following training protocol was employed: (1) to constrain

the number of splits in these experiments, a random subset of the data was chosen and trees were grown until each point was totally partitioned (e.g. 9 data points would produce 8 splits); (2) parameters were optimized using stochastic gradient descent with learning rate determined by experimentation to yield stable network training that did not cause numerical instability with minibatches of size 64; (3) networks were trained for 1 epoch (i.e. 1 iteration) over the entire dataset.

Algorithm 1: Splitting algorithm (for numerical attributes)

1 **Procedure** `splitNode` $(j, \mathcal{D}_{N(j)})$
  Input : node j, data points at node j
2 Add j to T
3 Set $c_j = mean(X_{N(j)})$, $r_j = 2\sqrt{max_d(variance(X_{N(j)}))}$
4 Sample E from exponential distribution with rate r_j
5 **if** `stopSplit()` *is FALSE* **then**
6 Set $\tau_j = \tau_{parent(j)} + E$
7 Sample split (w_j^T, w_{j0}) according to `sampleRandomSplit()`
8 Set $N(left(j)) = \{n \in N(j) : 0 > w_j^T X_{N(j)} + w_{j0}\}$ and
  $N(right(j)) = \{n \in N(k) : 0 < w_j^T X_{N(j)} + w_{j0}\}$
9 `splitNode` $(left(j), \mathcal{D}_{N(left(j))})$
10 `splitNode` $(right(j), \mathcal{D}_{N(right(j))})$
11 **else**
12 Add j to $leaves(T)$
13 **end**
14 **Function** `sampleRandomSplit` $(\mathcal{D}_{N(j)}, c_j, r_j)$
  Input : data points at node j, center of j, radius of j
  Output: weight and bias of oblique split
15 Sample n and r_0 from the surface of unit sphere with the same dimension as $X_{N(j)}$
16 Sample $u \sim Uniform[0, r_j]$
17 Set $r_0 = u(r_0 + c_j)$
18 **if** *plane* $n \cdot (r - r_0)$ *intersects* $\mathcal{D}_{N(j)}$ **then**
19 Set $b = \frac{3}{r_j}$
20 Set $w_j^T = bn$
21 Set $w_{j0} = -b(r_0 \cdot n)$
22 $RETURN\ (w_j^T, w_{j0})$
23 **else**
24 $GOTO$ line 15
25 **end**
26 **Function** `stopSplit` $(j, \mathcal{D}_{N(j)}, E)$
  Input : node j, data points at node j, additional lifetime
  Output: boolean
27 **if** $\tau_{parent(j)+E>\lambda}$ **then**
28 $RETURN\ TRUE$
29 **else if** *number of samples in* $\mathcal{D}_{N(j)} < min\_samples\_split$ **then**
30 $RETURN\ TRUE$
31 **else if** *number of samples in* $\mathcal{D}_{N(j)} < 2$ **then**
32 $RETURN\ TRUE$
33 **else**
34 $RETURN\ FALSE$

Several different model architectures were investigated. The first variant is a single decision tree with varying number of splits. The second variant (referred to as forest) is an ensemble akin to

Method	Accuracy	Learning Rate
linear classifier	91.1%	0.3
tree (1 split)	91.1%	0.1
tree (1 split) + conv5-32	92.9%	0.01
tree (4 split)	91.3%	0.1
tree (4 split) + conv5-32	90.2%	0.01
tree (8 split)	91.5%	0.1
tree (16 split)	90.8%	0.1
forest (1 split, 4 trees)	88.8%	0.1
forest (1 split, 8 trees)	88.6%	0.1
forest (1 split, 16 trees)	87.8%	0.1
forest (8 split, 8 trees)	87.9%	0.1
ml forest (1 split, 4 trees)	85.0%	0.1
ml forest (1 split, 8 trees)	91.1%	0.1
ml forest (1 split, 16 trees)	92.7%	0.1
ml forest (8 split, 8 trees)	89.3%	0.1
boosted trees (1 split, 4 trees)	94.2%	0.1
boosted trees (1 split, 4 trees) + conv5-32	94.8%	0.01
boosted trees (1 split, 8 trees)	92.7%	0.01
boosted trees (1 split, 8 trees) + conv5-32	96.0%	0.01
boosted trees (1 split, 16 trees)	88.8%	0.01
boosted trees (8 split, 8 trees)	92.5%	0.01
boosted trees (8 split, 8 trees) + conv5-32	96.8%	0.01

Table 1: Comparison of performance of different models on MNIST. The model columns indicate how many splits were used for each tree and how many trees were used for each ensemble and whether or not there was a convolutional feature map.

random forests where the predictions of several trees are averaged. The third variant (referred to a as multilayer forest) is an ensemble similar to a forest; however, the output is instead used as input to a subsequent prediction layer in a multilayer fashion. The fourth variant (referred to a as boosted trees) is an ensemble akin to boosted trees where the predictions of individual trees are the model residuals of previous estimates and the final estimate is obtained by adding all of the tree estimates together. Lastly, to show that the trees can guide learning of feature representations, several of the above models are trained either on the raw data features or fed a feature map using a jointly learned convolutional layer (a 2D convolution with 32 kernels of spatial size 5x5). Accuracies are reported on a separate testing set. For all tree models except the multilayer forest, the rule to determine the tree output is a linear model with 10 classes that is randomly initialized. Hence, a linear classifier, which can be considered a tree with 0 splits, is used as a baseline. For the multilayer forest, the rule is a randomly initialized linear model that compresses the input into one output and applies a RELU nonlinearity before passing the output to the next layer.

It is observed in these preliminary experiments that the performance of trees was only comparable to the baseline linear model. One concern that has been raised in a previous work [10] is the observation that soft decision trees tend to get stuck on plateaus in which one or few of the internal nodes always assign almost all the probability to one of its sub-trees, which causes the gradient of the logistic for the decision to tend to zero. This would cause all of the data to route through a small subset of rules, thereby eliminating much of the potential expressiveness of the tree model and potentially defaulting to the linear rule in a single leaf. In these preliminary experiments however, it was difficult to tell whether the limitation in performance was due to this phenomenon or the lack of epochs used to train each model due to time constraints.

Of note, the best performing ensembles based on these experiments were the multilayered forest

and boosted trees. When they consist of shallow trees, both ensembles avoid the problem of multiplying consecutive sigmoids together. The increase in model performance with the increasing number of trees in the multilayered forest is similar to the behavior of increasing the number of neurons in a densely connected layer as individual trees function analogously to single neurons. The increase in model performance associated with boosting trees is likely due to the summation of additional gradient information of multiple models allowing more rapid optimization over a single epoch, all else equal. Although similar to boosted decision trees in that the prediction of individual trees are summed together, here, each model is learned simultaneously as opposed to sequentially with the previous model fixed. For boosted trees especially, the addition of feature learning through convolutions had a strong positive impact on model performance.

5 Discussion

This report investigates the feasibility of optimizing trees inferred using the Randomized Tessellation sampling algorithm with the hierarchical representation learning and gradient descent optimization of deep neural networks. The original hypothesis of this project was to show that the usage of a decision tree initialized neural network can achieve similar performance to deep learning models while using fewer parameters on the LUNA16 dataset. In that direction, the scope of the work here shows that a visual codebook using a multilayered forest is potentially feasible as verified on the MNIST dataset. In addition, the performance of boosted trees as an end-layer classifier is of potential interest for other prediction tasks.

Future work would be to determine whether the learned splits induce any meaningful partitions of the dataspace such that similar examples have similar decision node outputs. In addition, a more unified growing-training algorithm that duplicates a rule among the split leaves may be useful to investigate. Currently, the linear models are independently initialized for each leaf. However, it may make more sense to optimize a tree with a single leaf, induce a split and clone the rule for both leaves, and then re-optimize with the added split. The split may be validated for usefulness before deciding to keep it or terminate the growing process.

6 Acknowledgements

Supported in part by an Alpha Omega Alpha Carolyn L. Kuckein Student Research Fellowship.

References

[1] P. Kontschieder, M. Fiterau, A. Criminisi, and S. Rota Bulo, "Deep neural decision forests," in *Proceedings of the IEEE international conference on computer vision*, pp. 1467–1475, 2015.

[2] A. Suárez and J. F. Lutsko, "Globally optimal fuzzy decision trees for classification and regression," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 12, pp. 1297–1311, 1999.

[3] G. Biau, E. Scornet, and J. Welbl, "Neural Random Forests," *arXiv e-prints*, p. arXiv:1604.07143, Apr 2016.

[4] R. Tanno, K. Arulkumaran, D. Alexander, A. Criminisi, and A. Nori, "Adaptive neural trees," in *Proceedings of the 36th International Conference on Machine Learning* (K. Chaudhuri and R. Salakhutdinov, eds.), vol. 97 of *Proceedings of Machine Learning Research*, (Long Beach, California, USA), pp. 6166–6175, PMLR, 09–15 Jun 2019.

[5] B. Lakshminarayanan, D. M. Roy, and Y. Whye Teh, "Mondrian Forests: Efficient Online Random Forests," *arXiv e-prints*, p. arXiv:1406.2673, Jun 2014.

[6] S. Ge, S. Wang, Y. Whye Teh, L. Wang, and L. T. Elliott, "Random Tessellation Forests," *arXiv e-prints*, p. arXiv:1906.05440, Jun 2019.

[7] M. I. Jordan and R. A. Jacobs, "Hierarchical mixtures of experts and the em algorithm," *Neural computation*, vol. 6, no. 2, pp. 181–214, 1994.

[8] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010.

[9] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," *arXiv preprint arXiv:1512.01274*, 2015.

[10] N. Frosst and G. E. Hinton, "Distilling a neural network into a soft decision tree," *CoRR*, vol. abs/1711.09784, 2017.