TensorForest: Scalable Random Forests on TensorFlow

Thomas Colthurst, Gilbert Hendry, Zachary Nado, D. Sculley
Google Inc.
{thomaswc, gilberth, znado, dsculley}@google.com

Abstract

We present TensorForest, a highly scalable open-sourced system built on top of TensorFlow for the training and evaluation of random forests. TensorForest achieves scalability by combining a variant of the online Hoeffding Tree algorithm with the extremely randomized approach, and by using TensorFlow's native support for distributed computation. This paper describes TensorForest's architecture, analyzes several alternatives to the Hoeffding bound for per-node split determination, reports performance on a selection of large and small public datasets, and demonstrates the benefit of tight integration with the larger TensorFlow platform.

1 Introduction

One of the well-known "secrets" of applied machine learning has been the remarkable effectiveness of random forests [1] as a general purpose method. random forests have consistently placed well in empirical benchmarks for more than a decade, including small-scale studies [2], studies with higher dimensional data [3], and numerous Kaggle competitions. However, despite their success, they have received relatively little attention in recent years. Meanwhile, deep neural networks and their supporting open-source platforms like TensorFlow [4] have grown extremely popular.

We decided to create a scalable random forest implementation within TensorFlow to ensure that TensorFlow users are not locked in to a narrow set of algorithms. We also suspected that pushing the limits of scalability of random forests might show continued gains in performance. Finally, this integration pairs random forests with a wide range of dimensionality reduction methods in TensorFlow, such as those created by the intermediate layers of deep neural networks.

TensorForest is implemented as a set of decision trees learned in an online, greedy fashion in the manner of Hoeffding Trees [5]. However, we found that the classical Hoeffding bound was insufficiently tight, so we explored and report on a number of alternative bounds. We also exploit the extremely randomized trees [6] methodology for improved scalability.

Open Source. TensorForest is available open source as part of the main TensorFlow distribution.

Related Work. There are a number of open-source random forest implementations available; most of these, such as the one in scikit-learn, operate with batch training rather than online training, and scale poorly to very large data sets. One large-scale distributed approach to decision tree training used a MapReduce framework [7] in iterative fashion to perform the full batch (non-online) greedy algorithm, but was limited to some degree by MapReduce's lack of support for streaming data. The most similar open-source package to TensorForest is the SAMOA framework, which ports the online decision trees of the MOA package to a distributed stream processing engine [8]. However, the tight integration of TensorForest within the larger TensorFlow framework gives additional benefit, allowing users to cleanly combine random forest methods with additional methods such as dimensionality reduction using learned deep embeddings.

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

1. **Init**: Tree = [root], Fertile = {root}, Stats(root) = 0, Splits(root) = [].
2. For each batch $B = \{x_i, y_i\}$ of training data,
   (a) Compute the leaf assignment l_i of each x_i.
   (b) Update the leaf statistics Stats(l_i) based on x_i, y_i.
   (c) For each fertile leaf (i.e., l_i in the set Fertile), if |Splits(l_i)| $<$ max_splits, pick a feature j at random and add the split (j, x_{ij}) to Splits(l_i).
   (d) Otherwise, if l_i is fertile and |Splits(l_i)| = max_splits, update the split statistics for l_i using x_i, y_i.
   (e) Calculate the fertile leaves that are finished (see below). For every non-stale finished leaf, turn it into an internal node with its best scoring split, remove it from Fertile, and add its two children to Tree as leaves.
   (f) If |Fertile| $<$ max_fertile, add the max_fertile $-$ |Fertile| leaves with the highest weighted leaf scores to Fertile and initialize their Splits and split statistics.

Until |Tree| = max_nodes or |Tree| stays the same for max_batches_to_grow batches.

Figure 1: TensorForest's core algorithm for training a single tree.

2 Algorithm

Our basic algorithm (Figure 1) is a combination of Hoeffding Trees and extremely randomized trees. Following Hoeffding Trees, we prefer to find a "good enough" splitting rule quickly rather than the optimal split more slowly; specifically, we terminate the search for the best split after looking at a fixed number of samples. Similarly, we adopt the extremely randomized approach of considering only a sample of candidate split points.

Taken together, those two changes bound the number of training samples required to split a node. An additional change, initially proposed in [9], is required to bound the amount of memory used by the algorithm as a whole to a reasonable amount: the number of nodes collecting split statistics (i.e., "fertile" nodes) at any given time is limited. Once this limit is reached, new leaves can only become fertile when a fertile node becomes finished; the next fertile node is picked using the weighted leaf score, which is just the regular leaf score multiplied by the number of samples seen so far by the leaf.

Fertile leaves are declared finished when one or more of the following occur:

1. The leaf has accumulated split_after_samples samples.
2. The leaf is fertile for more than one entire pass of the data set. Such a leaf is called stale, and is not split.
3. A statistical estimate (see section 4 for details) declares that the current best scoring split has a p or greater chance of being the asymptotically best scoring split, where p is a tuning parameter.

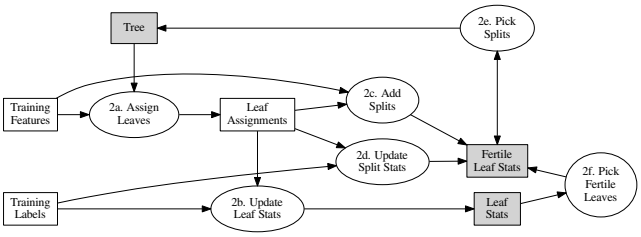
For classification problems, the leaf and split statistics are per-class counts, and the leaf and split scores are both Gini impurity [1]. For regression problems, the leaf and split statistics are the sample counts, sums, and sums of squares, and the leaf and split scores are the variances.

Training a forest is done by training the trees in parallel, in online fashion, relying on the extremely randomized approach to induce diversity. (We do not exploit any intra-tree parallelism in the current implementation).

3 Architecture

TensorFlow is a dataflow language, in which computations are expressed as a graph of operations and stateful variables which communicate via multi-dimensional arrays called tensors. To implement our algorithm in TensorFlow, we chose to represent each tree and each tree's statistics as a handful of tensors, and we created a new TensorFlow operation for each algorithm step (see Figure 2). To give

Figure 2: Dataflow for one iteration of training one tree. Persistent state is in shaded boxes, temporary variables are in clear boxes, and ovals correspond to operations.



just one example, there is a float-valued tensor t for each tree that stores all of its split thresholds, with $t[i]$ being the split threshold for internal node i.

We considered some alternative architectures. We could have chosen the decision tree node as the level of representation, in which case the dataflow graph would have appealingly looked like a tree. However, this representation would have been impractical for large trees because of the overhead incurred in instantiating and evoking each operation, as well as the time required to create the graph in Python. In addition, TensorFlow currently imposes a 2 GB limit on the size of the serialized computation graph, which would have capped forest size substantially in comparison.

Another representation we examined was to represent each tree and its statistics as C `structs` when in memory, and as tensors of binary data when in transit. This would have been easier to program against, and would have had better data locality (all of the data for a single node would be in one place, rather than spread among multiple tensors), but each operation using the tree or statistics would have had to deserialize them from a string-valued tensor, at prohibitive cost.

4 Statistical Bounds

We now more closely examine the issue of how to determine whether, with high confidence, we have seen enough data at a fertile leaf in order to pick the best split. For concreteness we only consider the classification context, where for each split $i \in 1 \dots m$ we have left and right per-class probability count vectors L_i and R_i. We take the splits to be in increasing order of their split score S_i, which is simply the sum of the Gini impurities on the left and the right:

$$S_i = G(L_i) + G(R_i),\ G(C) = 1 - \sum_j \left(\frac{C_j}{N_C}\right)^2,\ N_C = \sum_j C_j \tag{1}$$

In the above equation, and throughout this section, sums will range over the number of classes.

The standard solution to this question is the Hoeffding Tree bound [5], which states that split 1 is asymptotically better than split 2 with probability $1 - \delta$ if

$$|S_2 - S_1| > \sqrt{\frac{R^2 \ln(1/\delta)}{2N}} \tag{2}$$

where R is the range of $|S_2 - S_1|$, which is 2, and $N = N_{L_1} + N_{R_1} = N_{L_2} + N_{R_2}$ is the number of examples seen by the fertile node.

We found two issues with using this bound in our system. First, in real world data, there are frequently two features which are identical or nearly identical. If such features are chosen as splits with the same threshold values, then their split scores will remain the same (and $|S_2 - S_1|$ will be zero) for a very long, possibly infinitely long, time. As a practical solution, we force a split to be chosen after at most split_after_samples training points, breaking ties arbitrarily.

The second issue is that the Hoeffding bounds are quite loose when compared to the true probabilities of split 2 asymptotically having a lower Gini impurity than split 1. Table 1 reports the Hoeffding bounds and the true probabilities for a selection of examples. For the table, we algebraically manipulate the usual presentation of the Hoeffding bound in equation 2 into the equivalent

$$1 - \delta < 1 - e^{-2N|S_2 - S_1|^2 R^{-2}} \tag{3}$$

Table 1: The true value of the probability of the second best split asymptotically having a better Gini impurity than the first best split, as compared to the Hoeffding bound, a Chebyshev bound, and a bootstrap estimate. The true value $P(G_1 > G_2)$ were derived by integrating equation 4, and the Hoeffding bound was used in the form shown in equation 3. The bootstrap estimate was computed using 100k bootstrap samples. The bootstrap estimate agrees with the true value much more tightly than the other bounds.

Split 1 Counts	Split 2 Counts	True $P(G_1 > G_2)$	Hoeffding Tree bound	Chevyshev bound	Bootstrap estimate
4; 1	3; 2	0.324	0.983	5.5	0.35
5; 0	3; 2	0.113	0.861	1.125	0.15
10; 2	7; 5	0.116	0.869	1.2	0.13
12; 0	7; 5	0.008	0.465	0.325	0.01
87; 12	82; 17	0.161	0.792	2.07	0.17
99; 0	55; 44	4.46e-17	1.45e-05	0.027	0

which gives a bound on the probability $1 - \delta$.

The true probabilities $P(G_1 > G_2)$ were derived as follows: let split 1 and split 2 have left count vectors L_1 and L_2. For simplicity, we ignore the right sides of the splits and assume instead that $\sum_j L_{1j} = \sum_j L_{2j}$; any bound violations in this situation can then be converted into a full bound violation by letting $R_1 = L_1$ and $R_2 = L_2$. If the asymptotic class probabilities induced by splits 1 and 2 are $(l_1, 1 - l_1)$ and $(l_2, 2 - l_2)$, then we may then obtain the true probability $P(G_1 > G_2)$ by integrating

$$P(l_1, l_2 | L_1, L_2) = \frac{{l_1}^{L_{11}}(1 - l_1)^{L_{12}}{l_2}^{L_{21}}(1 - l_2)^{L_{22}}}{B(L_{11} + 1, L_{12} + 1)B(L_{21} + 1, L_{22} + 1)} \tag{4}$$

over the region defined by $G(l_1, 1 - l_1) > G(l_2, 2 - l_2)$, where $B(\alpha, \beta)$ is the beta function.

Because of this lack of tightness in the Hoeffding bound, we have explored a number of alternative methods for determining when to split a fertile node. First, we derived an alternative statistical bound based on the multivariate Chebyshev Inequality

$$P(|X - \mu| < \epsilon) > 1 - \frac{\text{tr}(\Sigma)}{\epsilon^2} \tag{5}$$

for a random vector X with mean μ and covariance matrix Σ. To apply it, we consider $X = (l_1, r_1, l_2, r_2)$ where as before l_i and r_i are the unknown asymptotic per-class frequencies induced by split i on the left and right respectively. X is thus the Cartesian product of four Dirichlet variables, and has mean

$$\mu = \left(\frac{L_1 + \mathbf{1}}{\sum_i L_{1i} + 1}, \frac{R_1 + \mathbf{1}}{\sum_i R_{1i} + 1}, \frac{L_2 + \mathbf{1}}{\sum_i L_{2i} + 1}, \frac{R_2 + \mathbf{1}}{\sum_i R_{2i} + 1} \right) \tag{6}$$

where $\mathbf{1}$ is the vector of all 1's. The trace of the covariance matrix $\text{tr}(\Sigma)$ can be decomposed as $\text{tr}(\Sigma) = \text{tr}(\Sigma_{L_1}) + \text{tr}(\Sigma_{R_1}) + \text{tr}(\Sigma_{L_2}) + \text{tr}(\Sigma_{R_2})$ where the trace of the covariance matrix associated with the counts C is

$$\text{tr}(\Sigma_C) = \frac{c^2 - \sum_i (C_i + 1)^2}{c^2(c + 1)}, \; c = \sum_i (C_i + 1) \tag{7}$$

The final variable ϵ is the distance between μ and the nearest point (a, b, c, d) to it on the surface defined by $G(a) + G(b) = G(c) + G(d), a \cdot \mathbf{1} = 1, b \cdot \mathbf{1} = 1, c \cdot \mathbf{1} = 1, d \cdot \mathbf{1} = 1$; this point can be found by applying Lagrange multipliers and solving a quadratic equation. As Table 1 shows, however, the resulting Chebyshev upper bound is much too loose to be useful in practice.

Second, we tried a jackknife estimate [10] based on removing 1 sample in all possible ways from the true count vectors. Unfortunately, this often gave unreliable probability estimates. For example, the jackknife estimate of $P(G_1 > G_2)$ for all of the examples in Table 1 is 0.

Finally, we tried a bootstrap estimate [11] of $P(G_1 > G_2)$. This gave better estimates, but at a computational cost. We found that in order to reliably get two decimal places of precision required over 10k bootstrap samples. We also found two tricks that dramatically improved the bootstrap's ability to approximate $P(G_1 > G_2)$: (a) generate the bootstrap samples using the Laplace-smoothed

Table 2: Performance of techniques for determining when to split a fertile node, when compared on training 100 trees of 10k nodes on the MNIST handwritten digits dataset [12]. All three systems picked a split after at most 250 training samples. Both the Hoeffding Tree and bootstrap systems used $p = 0.99$ for the probability of the splitting decision being correct, so for the bootstrap system, $\lceil -\log_2 0.01 \rceil = 7$ bootstrap samples were used. Each training step processes 1k training samples.

Technique	Test Accuracy	# Training Steps
None	0.965	16,000
Hoeffding Tree	0.965	15,000
Bootstrap	0.964	4,000

class frequency estimates (i.e., pick a sample from class j with probability $(C_j + 1)/(\sum_k C_k + 1)$ instead of $C_j/(\sum_k C_k)$), and (b) assign half the weight to $P(G_1 > G_2)$ when the two bootstrap samples had the same Gini impurity. Table 1 shows how the bootstrap estimates obtained in this way generally track the true $P(G_1 > G_2)$.

Table 2 compares three of these techniques on the MNIST dataset. All three give almost exactly the same test set accuracy, but the training time is reduced modestly with the Hoeffding Tree bound and dramatically with the bootstrap technique. In the bootstrap system, we use the standard trick of drawing just enough samples to let us take an action with the desired statistical confidence.

5 Experiments

We ran three sets of experiments, confirming the performance of TensorForest against full batch methods, exploring scalability and the benefit of very large forests, and demonstrating the utility of being integrated with the larger TensorFlow platform.

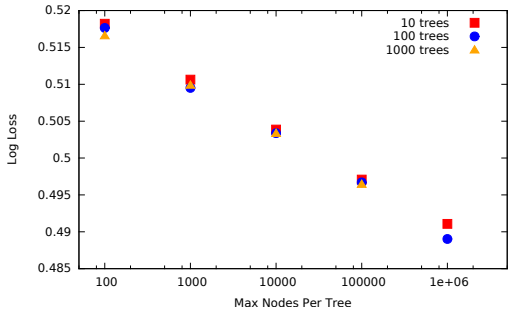
Comparing with scikit-learn. Table 3 compares TensorForest with the ExtraTrees implementation in `scikit-learn` on a selection of datasets from the UCI ML Repository [13]. Both systems use the extremely randomized approach, but ExtraTrees is a batch algorithm while TensorForest is entirely online. For these and the other runs reported on in this section, the forests consist of 100 trees with 10k nodes. The two systems have comparable accuracy, with TensorForest sacrificing a small percentage for its increased scalability. For small data sets, TensorForest does not train more quickly due to overhead from the larger TensorFlow platform. However, for the larger HIGGS data set, TensorForest trains in about one percent of the time taken by `scikit-learn`, even without taking advantage of distributed training. The next experiments show that TensorForest scales up many additional orders of magnitude.

Exploring Larger Forests. One of the more interesting trends from work in deep learning around the year 2010 was the finding that for very large data sets, increasing model capacity leads to

	Data			Accuracy (%) / R^2 Score	
Data Set	# Examples	# Features	# Classes	TensorForest	scikit-learn
Iris	150	4	3	95.6	94.6
Diabetes	442	10	Regression	0.462	0.461
Boston	506	13	Regression	0.793	0.872
Digits	1797	64	10	96.7	97.6
Sensit (Comb.)	78k	100	3	81.0	83.1
Aloi	108k	128	1000	89.8	91.7
rcv1	518k	47,236	53	78.7	81.5
Covertype	581k	54	7	83.0	85.0
HIGGS	11M	28	2	70.9	71.7

Table 3: Accuracy comparison of TensorForest with `scikit-learn`'s ExtraTrees implementation.

Figure 3: Log loss as a function of tree size on the Criteo data set (lower is better). These results show increasingly better predictive performance with larger forests; there is not yet a peformance plataeu.



increased predictive performance. Indeed, much of the success of deep models at this time was based on infrastructure allowing increases in model size and training data.

To see if similar trends might hold for random forests, we ran a series of models on the Criteo data set [14]. Criteo is a binary classification problem to predict ad clicks, containing 13 continuous and 26 categorical features with 37M training and 4M evaluation examples. We tested up to 100M-node forests with a range of max nodes and number of trees, with a minimum of 500 samples at each node and a maximum of 1000. Training and evaluation were performed on Google shared resources with 2-20 trees per machine; training time ranged from a few minutes for smaller forests to 2-3 days for larger ones. Note that comparison with full-batch methods on this data is computationally infeasible.

The results in Figure 3 show a clear log-linear trend, with increasingly larger forests giving increasingly better results. Interestingly, even with 100M nodes in the forest we don't yet hit a performance plateau. It seems likely that just as continuing to increase model size for deep networks led to additional wins, the same trend may hold true for random forests. This would be useful given that inference with tree-based methods can be far cheaper in terms of computational cost.

TensorFlow and TensorForest Integration. Tight integration allows TensorForest users to take advantage of the full TensorFlow platform, which may be especially useful for up-stream dimensionality reduction methods including learned deep embeddings. To demonstrate this, we ran a set of experiments on the ILSVRC2012 imagenet data, using the lower levels of a pre-trained version of the publicly available state-of-the-art Inception-ResNet-v2 model [15] as a fixed embedding for input features. The final dropout, fully connected layer, and softmax layer were removed from the deep model (totalling roughly 1.5M parameters); the final averaging layer was used as the embedding. No cropping was done. The current best Top-1 accuracy for this prediction task in this form is 80.1% with the full model, but was 75% as recently as 2015 [15].

The results shown in Table 4 for this preliminary experiment are not intended to establish a new state of the art, but do show that strong predictive accuracy is achievable using a random forest approach with the pre-trained embedding while using much smaller models. Even more salient, setting up and running these experiments was easily possible within a clean, well integrated framework.

Num. Trees	Max Nodes Per Tree	Total Forest Size	Top-1 Accuracy	Top-5 Accuracy
20	2k	40k	64.7	76.8
40	4k	160k	71.6	83.7
50	6k	300k	74.0	85.7
300	5k	1,500k	78.1	92.5

Table 4: Results on ILSVRC2012 image recognition data, using TensorForest with the early layers of a pre-trained Inception-ResNet-v2 model as a fixed embedding for input. The best of these results approach the current state of the art and show the potential utility of random forests integrated within the larger TensorFlow ecosystem.

References

[1] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.

[2] Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on Machine learning*, 2006.

[3] Rich Caruana, Nikos Karampatziakis, and Ainur Yessenalina. An empirical evaluation of supervised learning in high dimensions. In *Proceedings of the 25th international conference on Machine learning*, 2008.

[4] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Gregory S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian J. Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Józefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Gordon Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul A. Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda B. Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016. `http://arxiv.org/abs/1603.04467`. Software available from tensorflow.org.

[5] Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, pages 71–80, New York, NY, USA, 2000. ACM.

[6] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine learning*, 2006.

[7] Biswanath Panda, Joshua S Herbach, Sugato Basu, and Roberto J Bayardo. Planet: massively parallel learning of tree ensembles with mapreduce. *Proceedings of the VLDB Endowment*, 2009.

[8] Gianmarco De Francisci Morales and Albert Bifet. Samoa: Scalable advanced massive online analysis. *Journal of Machine Learning Research*, 2015.

[9] Misha Denil, David Matheson, and Nando De Freitas. Consistency of online random forests. *ICML*, 2013.

[10] M. H. Quenouille. Problems in plane sampling. *Ann. Math. Statist.*, 20(3):355–375, 09 1949.

[11] B. Efron. Bootstrap methods: Another look at the jackknife. *Ann. Statist.*, 7(1):1–26, 01 1979.

[12] Yann LeCun, Corinna Cortes, and Christopher JC Burges. The MNIST database of handwritten digits, 1998. `http://yann.lecun.com/exdb/mnist/`.

[13] M. Lichman. UCI machine learning repository, 2013.

[14] O. Chapelle et al., 2014. `https://www.kaggle.com/c/criteo-display-ad-challenge/data`.

[15] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR*, 2016.