Journal of Machine Learning Research 21 (2020) 1-5 Submitted 2/19; Revised 3/20; Published 4/20

ThunderGBM: Fast GBDTs and Random Forests on GPUs

Zeyi Wen[§] zeyi.wen@uwa.edu.au
Hanfeng Liu[†], **Jiashuai Shi**[‡] {kurt.liuhf, shijiashuai}@gmail.com
Qinbin Li[†], **Bingsheng He**[†] {qinbin,hebs}@comp.nus.edu.sg
Jian Chen[‡] ellachen@scut.edu.cn
[§] *Dept. of Computer Science and Software Engineering, Uni. of Western Australia, 6009, Australia*
[†] *School of Computing, National University of Singapore, 117418, Singapore*
[‡] *School of Software Engineering, South China University of Technology, Guangzhou, 510006, China*

Editor: Alexandre Gramfort

Abstract

Gradient Boosting Decision Trees (GBDTs) and Random Forests (RFs) have been used in many real-world applications. They are often a standard recipe for building state-of-the-art solutions to machine learning and data mining problems. However, training and prediction are very expensive computationally for large and high dimensional problems. This article presents an efficient and open source software toolkit called *ThunderGBM* which exploits the high-performance Graphics Processing Units (GPUs) for GBDTs and RFs. ThunderGBM supports classification, regression and ranking, and can run on single or multiple GPUs of a machine. Our experimental results show that ThunderGBM outperforms the existing libraries while producing similar models, and can handle high dimensional problems where existing GPU-based libraries fail. Documentation, examples, and more details about ThunderGBM are available at `https://github.com/xtra-computing/thundergbm`.

Keywords: Gradient Boosting Decision Trees, Random Forests, GPUs, Efficiency

1. Introduction

Gradient Boosting Decision Trees (GBDTs) and Random Forests (RFs) are widely used in advertising systems, spam filtering, sales prediction, medical data analysis, and image labeling (Chen and Guestrin, 2016; Goodman et al., 2016; Nowozin et al., 2013). GBDTs[1] have won many awards in recent Kaggle data science competitions. However, training GBDTs is often very time-consuming, especially for large and high dimensional problems. GPUs have been used to accelerate many real-world applications (Dittamo and Cisternino, 2008), due to their abundant computing cores and high memory bandwidth. In this article, we propose a GPU-based software tool called *ThunderGBM* to improve the efficiency of GBDTs. ThunderGBM supports binary and multi-class classification, regression and ranking. ThunderGBM supports the Python interface, and can run on single or multiple GPUs of a machine. Experimental results show that ThunderGBM is faster than XGBoost, LightGBM (Ke et al., 2017) and CatBoost (Prokhorenkova et al., 2018), while producing similar models on the data sets tested. Moreover, ThunderGBM can handle high dimensional prob-

1. For ease of presentation, we use "GBDTs" rather than mentioning both GBDTs and RFs.

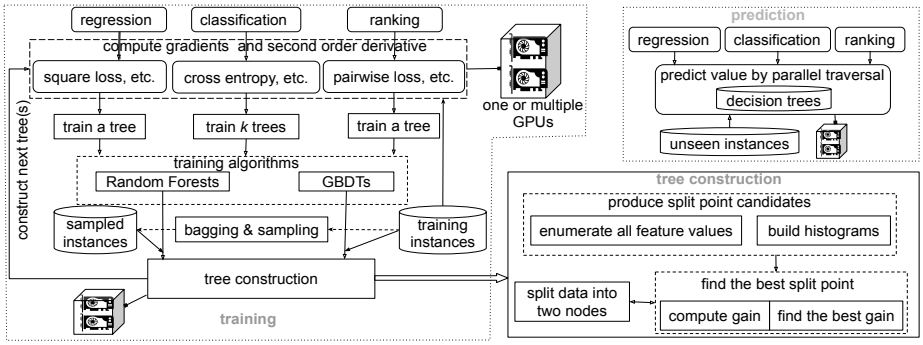

Figure 1: Overview of training and prediction in ThunderGBM.

lems where those existing libraries fail. The key reason for the improvement is that existing libraries are CPU-oriented and use the GPU to accelerate only a part of GBDTs and/or require additional cost and data structures to support both CPU and GPU implementations, whereas ThunderGBM is GPU-oriented and maximizes GPU usage.

2. Overview and Design of ThunderGBM

Figure 1 shows the overview and software abstraction of ThunderGBM. The training algorithms for different tasks (i.e., classification, regression and ranking) are built on top of a generic tree construction module. This software abstraction allows us to concentrate on optimizing the performance of tree constructions. Different tasks only require different ways of computing the derivatives of the loss functions. Notably, the multi-class classification task requires training k trees where k is the number of classes (Bengio et al., 2010; Chen and Guestrin, 2016), while regression and ranking only require training one tree per iteration. The prediction module is relatively simple, and is essentially computing predicted values by concurrent tree traversal and aggregating the predicted values of the trees on GPUs. Here, we focus on the training on a single GPU. More details about using multiple GPUs and the prediction are in the supplementary file (Wen et al., 2019a). We develop a series of optimizations for the training. For each module that leverages GPU accelerations, we propose efficient parallel algorithmic design as well as effective GPU-aware optimizations. The techniques are used to support two major components in ThunderGBM: (i) computing the gradients and second order derivatives, and (ii) tree construction.

2.1 Computing the Gradients and Second Order Derivatives on GPUs

Denoting y_i and $\hat{y}_i$ the true and predicted target value of the i-th training instance, the gradients and second order derivatives are computed using the predicted values and the true values by $g_i = \partial l(y_i, \hat{y}_i)/\partial \hat{y}_i$ and $h_i = \partial^2 l(y_i, \hat{y}_i)/\partial \hat{y}_i^2$. The gradient and second order derivative of the loss function are denoted by g_i and h_i, respectively; $l(y_i, \hat{y}_i)$ denotes the loss function. ThunderGBM supports common loss functions such as mean squared error, cross-entropy and pairwise loss (De Boer et al., 2005; Cao et al., 2007; Lin et al., 2014). More details on loss functions and derivatives are in the supplementary file. Computing g_i and h_i requires the predicted value $\hat{y}_i$ of the i-th training instance, ThunderGBM computes $\hat{y}_i$ based

on the intermediate training results. This is because the training instances are recursively divided into new nodes and are located in the leaf nodes at the end of training each tree. The idea of obtaining the predicted values efficiently is also used in LightGBM. To exploit the massive parallelism of GPUs, we create a sufficient number of threads to efficiently use the GPU resources. Each GPU thread keeps pulling an instance and computes its g and h.

2.2 Tree Construction on GPUs

Tree construction is a key component and time consuming in the GBDT training. We adopt and extend novel optimizations in our previous work (Wen et al., 2018, 2019b) to improve the performance of ThunderGBM. Tree construction contains two key steps: (i) producing the split point candidates, and (ii) finding the best split for each node.

Step (i): ThunderGBM supports two ways of producing the split point candidates: one based on enumeration and the other based on histograms. The former approach requires the feature values of the training instances to be sorted in each tree node, such that it can enumerate all the distinct feature values quickly to serve as the split point candidates. However, the number of split point candidates may be huge for large data sets. The latter approach considers only a fixed number of split point candidates for each feature, and each feature is associated with a histogram containing the statistics of the training instances. Each bin of the histogram contains the values of the accumulated gradients and second order derivatives for all the training instances located in the bin. When using histogram-based training, the data is binned into integer-valued bins, which avoids having to sort the samples at each node, thus leading to significant speed improvement. In ThunderGBM, each histogram is built in two phases. Firstly, a partial histogram is built on the thread block level using shared memory, because a thread block only has accesses to a proportion of gradients and the second order derivatives. Secondly, all the partial histograms of a feature are accumulated to construct the final histogram. ThunderGBM automatically chooses the split point candidate producing strategy based on the data set density, i.e., histograms-based approach for dense data sets and enumeration-based approach for the others. The density is measured by $\frac{\text{total \# of feature values}}{\text{\# of instances} \times \text{\# of dimensions}}$. If the ratio is larger than a threshold, we choose the histogram-based approach; we choose the enumeration-based approach otherwise.

Step (ii): Finding the best split is to look for the split point candidate with the largest gain. The gain (Chen and Guestrin, 2016) of each split point candidate is computed by $gain = \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right]$, where G_L and G_R (resp. H_L and H_R) denote the sum of g_i (resp. h_i) of all the instances in the resulting left and right nodes, respectively; λ is a regularization constant. In ThunderGBM, one GPU thread is dedicated to computing the gain of each split point candidate. The split point candidate with the largest gain is selected as the best split point for the node, which can be computed efficiently by a parallel reduction on GPUs. Once the best split point is obtained, the training instances in a node are divided into two child nodes. For producing the split point candidates by enumeration, ThunderGBM adopts the novel order preserving data partitioning techniques on GPUs proposed in our previous work (Wen et al., 2018), i.e., the feature values of the child nodes can be sorted more efficiently. For producing the split point candidates using histograms, each GPU thread determines which child node an instance should go to based on the best split. ThunderGBM repeats the two steps until a termination condition is met.

data set			on two cpus (sec)			on the gpu (sec)				speedup (on cpus)			speedup (on gpu)		
name	card.	dim.	xgb	lgbm	cat	xgb	lgbm	cat	**ours**	xgb	lgbm	cat	xgb	lgbm	cat
higgs (reg)	11M	28	44.6	22.0	67.9	9.9	12.3	10.1	6.6	6.8	3.3	10.3	1.5	1.9	1.5
log1p (reg)	16K	4M	oom	189	oom	oom	261	oom	25.6	n.a.	7.4	n.a.	n.a.	10.2	n.a.
cifar10 (clf)	50K	3K	521	lerr	lerr	124	lerr	lerr	81.5	6.4	n.a.	n.a.	1.5	n.a.	n.a.
news20 (clf)	16K	62K	287	15.4	oom	109	16.5	oom	5.8	49	2.7	n.a.	18.8	2.8	n.a.
yahoo (rnk)	473K	700	18.8	11	n.a.	2.4	29.4	n.a.	2.4	7.8	4.6	n.a.	1.0	12.3	n.a.

Table 1: Comparison with XGBoost, LightGBM and CatBoost.

3. Experimental Studies

We conducted experiments on a Linux workstation with two Xeon E5-2640 v4 10 core CPUs, 256GB memory and a Tesla P100 GPU of 12GB memory. The tree depth is set to 6 and the number of trees is 40. More results on experiments and descriptions about the data sets can be found in the supplementary file (Wen et al., 2019a). Five data sets are used here, and regression, classification and ranking are marked with "reg", "clf" and "rnk", respectively. We used the versions of XGBoost, LightGBM and CatBoost on 21 Jul 2019.

The results are shown in Table 1, where "oom" stands for "out of memory", "lerr" stands for "large training error" and "n.a." stands for "not applicable". When the existing libraries are running on CPUs, ThunderGBM is 6.4 to 10x times, 2.7 to 7.4 times, and 10.3 times faster than XGBoost, LightGBM and CatBoost, respectively. When running on GPUs, ThunderGBM is 1 to 10x times, 1.9 to 10 times, and 1.5 times faster than XGBoost, LightGBM and CatBoost, respectively. Moreover, ThunderGBM can handle high dimensional problems (e.g., *log1p*) where the existing libraries fail or run slowly. ThunderGBM also has smaller or comparable errors to the existing libraries (cf. the supplementary file).

4. Conclusion

In this article, we present *ThunderGBM* which supports classification, regression and ranking. ThunderGBM uses the same input command line options and configuration files as XGBoost, and supports the Python interface (e.g., scikit-learn). Our experimental results show that ThunderGBM outperforms the existing libraries while producing similar models, and can handle high dimensional problems where the existing libraries sometimes fail.

Acknowledgments

This work is supported by a MoE AcRF Tier 1 grant (T1 251RES1824) and Tier 2 grant (MOE2017-T2-1-122) in Singapore. Prof. Chen is supported by the Guangdong Basic Applied Research Foundation (2019B1515130001), Guangdong special branch plans young talent with scientific and technological innovation (2016TQ03X445), the Guangzhou science and technology planning project (201904010197) and Natural Science Foundation of Guangdong Province (2016A030313437). Bingsheng He and Jian Chen are corresponding authors. We acknowledge NVIDIA for the hardware donations.

References

Samy Bengio, Jason Weston, and David Grangier. Label embedding trees for large multi-class tasks. In *NeurIPS*, pages 163–171, 2010.

Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: from pairwise approach to listwise approach. In *ICML*, pages 129–136. ACM, 2007.

Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *SIGKDD*, pages 785–794. ACM, 2016.

Pieter-Tjerk De Boer, Dirk P Kroese, Shie Mannor, and Reuven Y Rubinstein. A tutorial on the cross-entropy method. *Annals of Operations Research*, 134(1):19–67, 2005.

Cristian Dittamo and Antonio Cisternino. GPU White paper, 2008.

Katherine E Goodman, Justin Lessler, Sara E Cosgrove, Anthony D Harris, Ebbing Lautenbach, Jennifer H Han, Aaron M Milstone, Colin J Massey, and Pranita D Tamma. A clinical decision tree to predict whether a bacteremic patient is infected with an extended-spectrum β-lactamase–producing organism. *Clinical Infectious Diseases*, 63(7):896–903, 2016.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. LightGBM: A highly efficient gradient boosting decision tree. In *NeurIPS*, pages 3149–3157, 2017.

Guosheng Lin, Chunhua Shen, and Jianxin Wu. Optimizing ranking measures for compact binary code learning. In *ECCV*, pages 613–627. Springer, 2014.

Sebastian Nowozin, Carsten Rother, Shai Bagon, Toby Sharp, Bangpeng Yao, and Pushmeet Kohli. Decision tree fields: An efficient non-parametric random field model for image labeling. In *Decision Forests for Computer Vision and Medical Image Analysis*, pages 295–309. Springer, 2013.

Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. CatBoost: unbiased boosting with categorical features. In *NeurIPS*, pages 6637–6647, 2018.

Zeyi Wen, Bingsheng He, Ramamohanarao Kotagiri, Shengliang Lu, and Jiashuai Shi. Efficient gradient boosted decision tree training on GPUs. In *International Parallel and Distributed Processing Symposium*, pages 234–243. IEEE, 2018.

Zeyi Wen, Hanfeng Liu, Jiashuai Shi, Qinbin Li, Bingsheng He, and Jian Chen. Supplementary material of ThunderGBM: `https://github.com/Xtra-Computing/thundergbm/blob/master/thundergbm-full.pdf`, 2019a.

Zeyi Wen, Jiashuai Shi, Bingsheng He, Jian Chen, Kotagiri Ramamohanarao, and Qinbin Li. Exploiting GPUs for efficient gradient boosting decision tree training. *IEEE Transactions on Parallel and Distributed Systems*, 30(12):2706–2717, 2019b.