The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

Efficient Support Vector Machine Training Algorithm on GPUs

Jiashuai Shi,[†‡] Zeyi Wen,[‡] Bingsheng He,[‡] Jian Chen,[†*]
[†]South China University of Technology, Guangzhou, China
[‡]National University of Singapore, Singapore
shijiashuai@gmail.com, wenzy@comp.nus.edu.sg, hebs@comp.nus.edu.sg, ellachen@scut.edu.cn

Abstract

Support Vector Machines (SVMs) are popular for many machine learning tasks. With rapid growth of dataset size, the high cost of training limits the wide use of SVMs. Several SVM implementations on GPUs have been proposed to accelerate SVMs. However, they support only classification (SVC) or regression (SVR). In this work, we propose a simple and effective SVM training algorithm on GPUs which can be used for SVC, SVR and one-class SVM. Initial experiments show that our implementation outperforms existing ones. We are in the process of encapsulating our algorithm into an easy-to-use library which has Python, R and MATLAB interfaces.

Introduction

Support Vector Machines (SVMs) show great generalization performance in various machine learning tasks including classification (SVC), regression (SVR) and distribution estimation (one-class SVM). Sequential Minimal Optimization (SMO) is the most popular algorithm to train SVMs. First proposed by Platt (1999), and improved by Fan et al. (2005), SMO has been implemented in the widely used library LIBSVM (Chang and Lin 2011), making SVMs into a practical algorithm. With rapid growth of dataset size, training SVMs by simply using LIBSVM takes significant amount of time (e.g. days or weeks). Besides, in order to choose hyper-parameters for best performance, SVMs are often trained hundreds of times by using grid-search and cross-validation. The high training cost limits the wide use of SVMs.

To accelerate SVMs, many researchers have applied distributed and parallel techniques to SVMs. Approximation techniques have to be used for distributed SVM training, because training SVMs needs to access the whole training data and the communication cost is extremely high. So majority of the existing studies focus on how to train SVMs on a single machine efficiently. Graphic Processing Units (GPUs) are probably the best hardware to accelerate SVMs. Catanzaro (2008) first proposed to train SVMs by using SMO on GPUs. Cotter (2011) used a clustering method to tailor SMO on GPUs for better memory coalescing access. Vanek (2017) pointed out the limited bandwith of modern GPUs, and implemented OHD-SVM tailored for GPUs.

[*]Jian Chen is the corresponding author.

Though these studies achieve significant speed-up over LIBSVM, they have two major limitations. First, they still do not make efficient use of GPUs. We conducted experiments using the three existing GPU SVM implementations on a mid-range and a high-end GPU, and the results on the mid-range GPU are almost the same as those on the high-end GPU. Second, they only support either SVC or SVR. We summarize the functionalities of existing SVM implementations. Existing GPU SVM implementations are mainly for binary SVM classification, and do not support multi-class classification or regression problems. These drawbacks result in users' unwillingness to use SVMs for their applications.

In this work, we propose a simple and effective parallel SVM training algorithm on GPUs. Our initial experimental results show that our algorithm outperforms existing ones. To help users easily apply GPU SVMs to their applications, we plan to develop an easy-to-use SVM library for GPUs, supporting multi-class SVMs, SVM regression, one-class SVM, cross-validation and various programming interfaces like Python, R and MATLAB.

Support Vector Machine

An instance $\boldsymbol{x}_i$ is attached with label $y_i \in \{+1, -1\}$. n is the number of instances. Training SVMs is to solve a dual optimization problem defined by $\boldsymbol{\alpha}$, where α_i denotes the *weight* of $\boldsymbol{x}_i$, $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is a kernel value computed from a kernel function. The kernel values of all the training instances form a *kernel matrix*. Since the kernel matrix is too large to fit in memory, SMO-type algorithms iteratively select a *working set* W consisting of some instances to optimize.

Technical Challenges

Calculation of kernel value is the most computational expensive part in SVM training. Caching techniques are commonly used to avoid repeat kernel value calculation (Catanzaro, Sundaram, and Keutzer 2008; Chang and Lin 2011). They store individual rows of kernel matrix in cache for reusing. However, on GPUs, we find that iteratively calculating individual rows has low GPU utilization. With a large working set, we can calculate multiple rows of kernel matrix at one time, which is efficient on GPUs. However, using a large working set with the first-order heuristic (Joachims 1998; Cotter, Srebro, and Keshet 2011) brings much more sub-

Table 1: Dataset information and parameters

dataset	#classes	card.	dim.	C	γ
rcv1	2	20,242	47,326	100	0.125
webdata	2	49,479	300	10	0.5
real-sim	2	72,309	20,958	4	0.5
mnist	10	60,000	780	10	0.125
news20	20	15,935	62,061	4	0.5

Table 2: Training time (s) among different implementations

dataset	libsvm-omp	gtsvm	ohd-svm	Ours
rcv1	8.38	49.04	1.96	0.98
webdata	67.51	6.98	3.64	2.37
real-sim	74.21	120.1	7.10	2.81
mnist	429.10	34.47	N/A	26.68
news20	33.77	55.04	N/A	14.92

problems to solve, i.e., slower convergence. To expoilt high-performance GPUs, we should redesign the SMO algorithm.

Our Implementation

We describe an efficient SVM training algorithm on GPUs in this section. The following three steps of our algorithm are implemented on GPUs in a highly parallel way.

Select a working set: The process of calculating kernel values is the performance bottleneck in SVM training. We select a large working set using first-order heuristic for better GPU utilization (Cotter, Srebro, and Keshet 2011).

Naively using a large working set results in low training efficiency. Only a few α_is are updated after the subproblem is solved. In other words, many rows of kernel matrix are wasted. To address this problem, we propose to use a *working set buffer*, similar to (Vanek, Michalek, and Psutka 2017), but much simpler. Except for filling up whole working set in the first iteration, we select $\frac{|W|}{2}$ instances in the following iterations, and replace the *oldest* instances in the working set. In this way, only $\frac{|W|}{2}$ rows of kernel matrix need to be calculated in each iteration. The use of working set buffer also leads to faster convergence, because the instances already in working set are more likely to be selected in next iterations.

Precomputation of kernel matrix: Instead of using cache, we precompute the rows of kernel matrix needed to solve the subproblem and store them in GPU memory. Note that only $\frac{|W|}{2} \times n$ kernel values need to be calculated because of working set buffer. With large working set, the precomputation is very efficient on GPUs.

Solving a subproblem: Given a working set and precomputed kernel matrix, the subproblem can be solved efficiently on GPUs. We implement the improved SMO (Fan, Chen, and Lin 2005) to solve the subproblem. Each GPU thread is assigned to optimize one instance. After the subproblem is solved, an indicator variable will be transfer to CPU to check if the optimization problem is optimal.

Experiments

We compare our implementation with LIBSVM with OpenMP, gtSVM (Cotter, Srebro, and Keshet 2011) and OHD-SVM (Vanek, Michalek, and Psutka 2017). GPUSVM (Catanzaro, Sundaram, and Keutzer 2008) is excluded because gtSVM reports better results. The datasets are from LIBSVM website. We choose commonly used Gaussian kernel. The regularization parameter C and Gaussian kernel parameter γ are chosen by grid-seach. The experiments were conducted in a workstation with two Xeon E5-2640v4, 256GB memory and a Tesla P100 12GB GPU.

Since all the four approaches rely on SMO algorithm, the models trained from the four are the same. Experimental results in Table 2 show that our implementation is consistently faster than others, whenever the dataset is small or large, sparse or dense. The working set buffer brings faster convergence, resulting ours siginificant speed-up over gtSVM. The simplicity of kernel value reusing without cache makes our algorithm faster than OHD-SVM, because using cache has much more overhead.

Conclusion

In this work, we propose a simple and effective algorithm to train SVM on GPUs, which can also be used to solve SVR and one-class SVM problems. Our initial experiments show that our algorithm outperforms existing ones.

Acknowledgement

This work is supported by a MoE AcRF Tier 1 grant (T1 251RES1610) in Singapore and Special Planning Project of Guangdong Province in China (Grant No. 609055894069).

References

Catanzaro, B.; Sundaram, N.; and Keutzer, K. 2008. Fast support vector machine training and classification on graphics processors. In *Proceedings of the 25th international conference on Machine learning*, 104–111. ACM.

Chang, C.-C., and Lin, C.-J. 2011. Libsvm: a library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)* 2(3):27.

Cotter, A.; Srebro, N.; and Keshet, J. 2011. A gpu-tailored approach for training kernelized svms. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, 805–813. ACM.

Fan, R.-E.; Chen, P.-H.; and Lin, C.-J. 2005. Working set selection using second order information for training support vector machines. *Journal of machine learning research* 6(Dec):1889–1918.

Joachims, T. 1998. Making large-scale svm learning practical. Technical report, SFB 475: Komplexitätsreduktion in Multivariaten Datenstrukturen, Universität Dortmund.

Platt, J. C. 1999. Fast training of SVMs using Sequential Minimal Optimization. In *Advances in kernel methods*. MIT Press. 185–208.

Vanek, J.; Michalek, J.; and Psutka, J. 2017. A gpu-architecture optimized hierarchical decomposition algorithm for support vector machine training. *IEEE Transactions on Parallel and Distributed Systems*.