Chunking SVM Training Implementation in CUDA

Ing. Josef Michálek (orcus@kky.zcu.cz), Ing. Jan Vaněk Ph.D. (vanekyj@ntis.zcu.cz)
Department of Cybernetics, Faculty of Applied Sciences, University of West Bohemia, Czech Republic

Abstract

Support Vector Machines are one of the most popular tools for classification or regression. There are several open-source SVM training implementations utilizing GPUs, but many of them are now obsolete and don't perform well on modern GPUs. There haven't been much development in this area in the past few years and the goal of this work is to introduce our own implementation of SVM training in CUDA. This poster shows the basic workings of our implementation and compares the training speed and model quality to several well-known open-source SVM training implementations.

Table 1: Used datasets

Name	#train	#test	#features	C	gamma
Epsilon	40,000	10,000	2,000	32	0.0001
Alpha	10,000	50,000	500	512	0.002
Timit	63,881	22,257	39	1	0.025
Adult a9a	32,561	16,281	123	4	0.5
Web w8a	49,749	14,951	300	4	0.5
MNIST even vs odd	60,000	10,000	784	1	0.02
Cov1 Forest	522,911	58,101	54	3	1.0
20 Newsgroups	19,996	19,996	1,335,191	4	0.5
RCV1	20,242	677,399	47,236	4	0.5
Real-Sim	72,309	72,309	20,958	4	0.5

Test Setup

Several **LIBSVM Datasets** and subsets of **Pascal Large Scale Learning Challenge**, **alpha** and **epsilon**, were used. They're available at:
http://largescale.ml.tu-berlin.de
http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

All experiments were performed with **RBF** kernel, since it is the only kernel supported by all implementations. The threshold epsilon was set to 0.001. Used datasets with parameters C and gamma are shown in table 1.

Several open-source GPU accelerated SVM training implementations were chosen and their performance was evaluated on each dataset. Used implementations are:

- **cuSVM** (Austin Carpenter)
  http://patternsonascreen.net/cuSVM.html
  Supports only RBF kernel, has MATLAB interface
- **GPU-SVM** (Brian Catanzaro)
  https://code.google.com/p/gpusvm/
- **GTSVM** (Andrew Cotter, Nathan Srebro, Joseph Keshet)
  http://ttic.uchicago.edu/~cotter/projects/gtsvm/
  Supports RBF, polynomial and sigmoid kernels
  Has command line, MATLAB and C interface
  Supports sparse data
- **MultiSVM** (Sergio. Herrero-Lopez)
  https://code.google.com/p/multisvm/
  Can solve multiclass SVM problems
  Handles binary data
- **WU-SVM** (Stephen Tyree, Jacob R. Gardner, Kilian Q. Weinberger, Kunal Agrawal, John Tran)
  http://machinelearning.wustl.edu/pmwiki.php/Main/Wusvm
- **Our implementation**
  https://github.com/OrcusCZ/SVMbenchmark
  Supports only RBF kernel
  Supports sparse data

Our Implementation

Our SVM training implementation is based on sequential minimal optimization (SMO) algorithm. In SMO algorithm, 2 points are selected from working set and optimised in each iteration. This working set is usually equal to training set. Our implementation uses a working set of specific user-defined size. In one iteration, these steps are performed:

1. **Working set selection**
   Working set of size N is selected according to the method described below. In the first iteration working set is selected using LibSVM's first order heuristic
2. **Local kernel matrix calculation**
   Kernel matrix tile of size N × N for local solver is computed. If possible, already computed rows are copied from kernel matrix cache
3. **Local solver**
   One CUDA kernel uses SMO algorithm to optimize local subproblem. Block size is equal to working set size, because each thread optimizes one element
4. **Kernel cache update**
   Rows of kernel cache belonging to alphas which were modified in the local solver are calculated
5. **Gradient update**
   Gradient has to be updated only for alphas modified in this iteration

From our experiments we found that the fastest convergence can be achieved with this **working set selection method:**

1. Sort all points by how much they violate KKT conditions
2. Pick N/4 first points from both classes
3. Sort points from last iteration's working set by their score in ascending order. Score is a value equal to how many consecutive iterations a particular point was in a working set.
4. Pick enough free vectors to fill new working set
5. If working set not full, pick enough lower bound vectors to fill new working set
6. If working set not full, pick enough upper bound vectors to fill new working set

This working set selection assumes that if a point was selected in working set, it might be selected again in the next iteration. Keeping it in working set for a few iterations takes advantage of already computed kernel values for these points.

Kernel matrix calculation is the most expensive part of the algorithm. Each iteration the kernel values for all points in the working set are calculated at once, which is very efficient on GPGPUs. Local solver does not need to calculate any kernel values.

Table 2: Trained model accuracy

Dataset	cuSVM	GPU-SVM	GTSVM	MultiSVM	WU-SVM	Ours
Epsilon	88.6 %	88.6 %	88.5 %	88.6 %	88.7 %	88.6 %
Alpha	78.6 %	78.6 %	77.1 %	78.6 %	78.5 %	78.6 %
Timit	87.7 %	87.7 %	87.7 %	87.7 %	87.2 %	87.7 %
Adult a9a	82.7 %	82.7 %	82.7 %	82.7 %	83.5 %	82.7 %
Web w8a	99.4 %	99.4 %	99.4 %	99.4 %	97.8 %	99.4 %
MNIST	98.9 %	98.9 %	98.9 %	98.9 %	98.4 %	98.9 %
Cov1 Forest	84.9 %	84.9 %	63.0 %	84.9 %	83.0 %	84.9 %
20 News.	N/A	N/A	99.9 %	N/A	N/A	99.9 %
RCV1	N/A	N/A	96.5 %	N/A	N/A	96.5 %
Real-Sim	N/A	N/A	99.7 %	.N/A	N/A	99.7 %

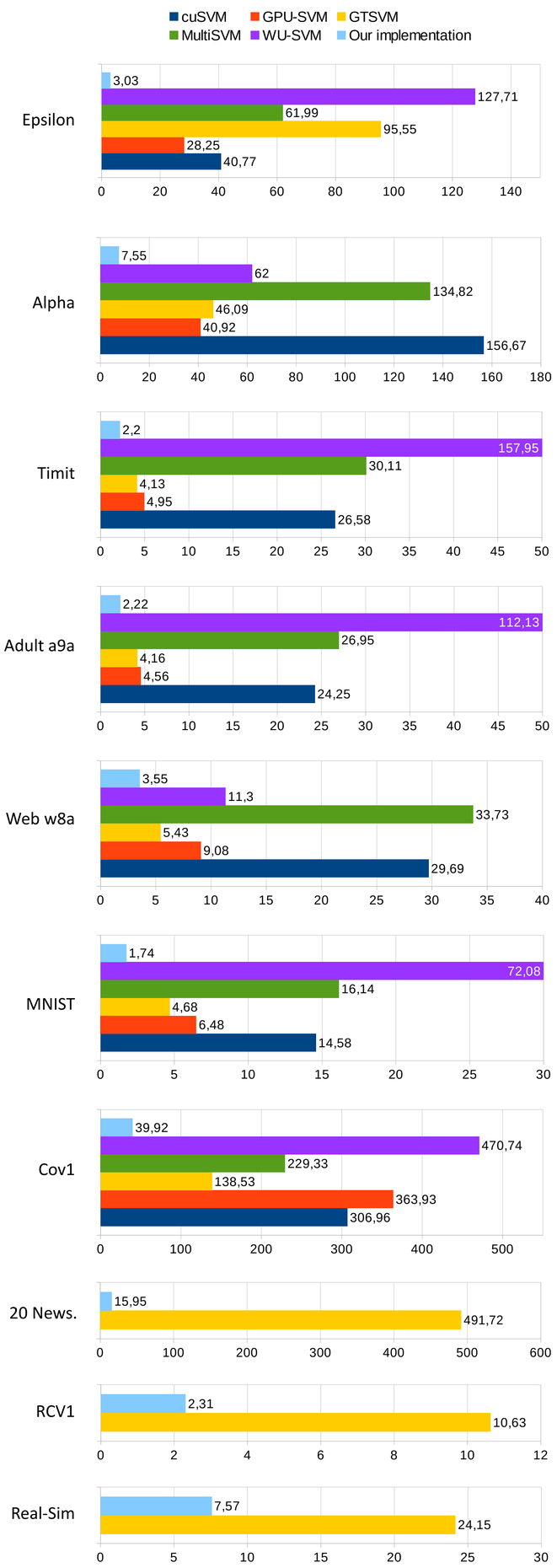
Results

Training time and model accuracy of all implementations were evaluated. Datasets 20 Newsgroups, RCV1 and Real-Sim are sparse and only some implementations were able to train them. N/A in the table denotes that sparse datasets are not supported.

Model accuracy is shown in table 2. Almost all implementations give models of the same quality. GTSVM and WU-SVM trained worse models on some datasets. WU-SVM also trained slightly better models on Epsilon and Adult a9a, but took very long time.

GTSVM has a configuration option specifying whether to use small or large clusters. Both options were evaluated and model with better accuracy was used. With large clusters, GTSVM crashed on out of memory when training dataset 20 Newsgroups.

Below are **charts** showing **training time in seconds** on a PC with **Intel i7-4790K, 32 GB RAM, NVIDIA GTX 980 Ti.**



This work was supported by the Grant Agency of the Czech Republic, project No. GAČR GBP103/12/G084.