Libsvm using GPU

Anirudhan J. Rajagopalan
N18824115
ajr619

December 15, 2016

1 Abstract

This project is an attempt to implement the Support Vector Machine [2] classifier using GPUs. We tried two approaches: The first one is to try to port the Sequential Minimal Optimization part to the GPU. The second one is to implement the Gram matrix [3] calculation to GPUs. We end up using the second approach and we show show performance improvements over the conventional approach used by libsvm.

2 Introduction

The most popular implementation of SVM is libsvm [1]. It is used in almost all the traditional machine learning applications. Since it projects data into a higher dimensional space, we were able to solve problems that are not easily classifiable in the lower dimensional space.

Porting a SVM to GPUs will help us use this classifier easily in deep learning models too. Though deep learning already uses various non-linearity to support higher dimensional data, the use of SVM might help us learn the underlying representation without using deeper layers in a deep learning model.

In this project, we explore various ways to parallelize the SVM problem and implement it in GPU.

3 Background Information

SVM is a linear classifier and is mainly successful due to its use of Hilbert space. SVM uses kernel functions which are functions in Reproducing Kernel Hilbert Space (RKHS). Implementation wise, these kernel functions can be as simple as computing a dot product between two set of features.o For the purpose of this project, we will be worrying about these simple RKHS functions. The idea is that, a non-linear lower dimensional data might be separable in a higher dimensional space.

Once, the higher dimensional features are calculated, SVM uses this higher dimensional features to find a classifier that has maximal distance from any of the points from the dataset.

The other part of SVM is the Quadratic Programming part. This part involves actually finding the plane that classifies the points in the dataset while also having a maximum margin from both the data points.

Given l examples $(\bar{x}_1, y_1), (\bar{x}_2, y_2), \ldots, (\bar{x}_l, y_l)$, with $\bar{x}_i \in \mathbb{R}^n$ and $y_i \in \{-1, 1\}$ $\forall i$ where the regularization is controlled by C.

$$\min_{f \in H} C \sum_{i=1}^{l} V(y_i, f(\bar{x}_i)) + \frac{1}{2}||f||_k^2 \tag{1}$$

Here V is the loss function which is typically Hinge loss. Usage of Hinge loss or any approximation of Hing loss such as Huber-Hinge loss helps us find a maximal margin plane that classifies the samples.

Typically we solve for the dual of this actual problem, which can be given as

$$\max_{\alpha \in \mathbb{R}^l} \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \alpha^T K \alpha \tag{2}$$

Subject to: $\sum_{i=1}^{l} y_i \alpha_i = 0$ and $0 \leq \alpha_i \leq C$, $i = 1, \ldots, l$ where $K_{ij} = y_i y j k(\bar{x} i, \bar{x}_j)$ is a kernel function. Solving this requires quadratic programming approaches which gives us the classification function.

$$f(x) = \sum_{i=1}^{l} y_i \alpha_i k(\bar{x}, \bar{x}_i) + b \tag{3}$$

With all the theory mentioned as briefly as possible, we can proceed to find a way to implement this interesting problem using GPUs.

4 Literature Survey

Previous work in this problem and what are the pros and cons of each solution.

5 Proposed solution

6 Experimental Setup

Specs of the machine, problem size, etc.

7 Experimental Result & Analysis

What are your findings. I expect much deeper description that "As we can see X is better than Y".

8 Conclusions

What are your main findings? Can you generalize them?

References

[1] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[2] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[3] Wikipedia. Gramian matrix — wikipedia, the free encyclopedia, 2016. [Online; accessed 16-November-2016].