A Monte Carlo Method for Image Classification Using SVM

Emanouil Atanassov[0000-0002-7442-7096], Aneta Karaivanova, Sofiya Ivanovska, Mariya Durchova[0000-0003-2548-282X]

Institute of Information and Communication Technologies, Bulgarian Academy of Sciences, Acad. G. Bonchev str., Block 25A, Sofia 1113, Bulgaria
emanouil@parallel.bas.bg, anet@ parallel.bas.bg, sofia@ parallel.bas.bg, mabs@ parallel.bas.bg

Abstract. Support Vector Machines are a widely used tool in Machine Learning. They have some important advantages with regards to the more popular Deep Neural Networks. For the problem of image classification, multiple SVMs may be used and the issue of finding the best hyperparameters adds additional complexity and increases the overall computational time required. Our goal is to develop and study Monte Carlo algorithms that allow faster discovery of good hyperparameters and training of the SVMs, without impacting negatively the final accuracy of the models. We also employ GPUs and parallel computing in order to achieve good utilisation of the capabilities of the available hardware. In this paper we describe our methods, provide implementation details and show numerical results, achieved on the publicly available Architectural Heritage Elements image Dataset.

Keywords: Deep Neural Networks, Monte Carlo method, Image Classification, Support Vector Machines.

1 Introduction

Support Vector Machines (SVMs) were introduced many years ago by Vladimir Vapnik and his colleagues (Boser, 1992).Since then we have seen many refinements and enhancements of the concept, but they all attempt to maintain some of the advantages of the method. One particularly useful feature of the method is that it seeks global optimum and its practical implementations mostly find this global optimum. Through the use of the so-called kernel trick, the method achieves further flexibility and becomes applicable in many real world scenarios.

In our work we concentrated on the problem of image classification, which is important in the overall area of the use of digital media in the service of understanding and preserving natural heritage. Although the problem of image classification is not the most suitable for application of SVMs (see in (Kroese, 2019; Snoek, 2012)), we will demonstrate how by using Monte Carlo one can achieve acceptable accuracy relatively fast, even without applying more complex techniques (Mikhailov, 1992).

Mathematically, the linear SVMs can be thought of as defined by the weights w_i and bias *b* that can be used to classify samples based on the sign of the function

$$f\ (x) = w^T x + b$$

so that positive signs correspond to one class and negative signs correspond to the other class. The weights and biases are determined by minimizing a loss function, which also incorporates a regularization constant *L*.

Our desire for the known samples is to have absolute values of the function *f* greater than 1, although this is not guaranteed.

Usually the loss function uses L_1 norm, since this naturally leads to zero weights for most of the known samples. However, substantial theoretical and practical use has been found for the case of using the L_2 norm, where the method is usually called Least Squares SVM (LS-SVM). Training LS-SVM models is more straightforward from mathematical point of view, since it leads to a linear algebra problem.

However, both L_1 and L_2 formulations have drawbacks in many practical applications, because the real-world problems are usually non-linear. One solution to this is the so-called kernel trick, where the usual scalar product is replaced with a suitable function $K\ (x\ ,\ y)$. In most cases it expected for $K\ (x_i\ ,\ x_j)$ to be positive semi-definite, but in practice sometimes this condition is not met. A simplified version of the method can be obtained if we consider only weights, with zero bias. For simplicity we consider this case, although our method is applicable also for the general case. Without the bias, the weights are obtained by solving the linear system

$$(K\ +\ \lambda\, I\ )w\ =\ C,$$

where *C* is a column vector with *1* for the samples in the first class and *-1* for the samples in the second class. So far we considered only binary classification problems. A solution to the multi-class classification problem can be obtained in many ways, for example by combining the out of multiple pairwise SVMs. The most popular are *one-vs-all* or *one-vs-one* techniques. However, the training of multiple models becomes more expensive, that is why methods that can speed-up the convergence are interesting.

Monte Carlo methods are an established technique for solving problems where the unknown quantity can represented as the mathematical expectation of some random variable, so that by sampling this variable one can obtain estimates of the true value by averaging. In the domain of linear algebra the Monte Carlo methods have been used for solving linear systems, estimating eigenvalues or inverting matrices (see, e.g., (Alexandrov, 2005; Dimov, 1999)). In machine learning they have wide applicability since the problems at hand are inherently stochastic.

In this paper we are going to study one Monte Carlo technique that can be applied in solving the SVM training problem (with non-linear kernel) and how it can be implemented efficiently using HPC. In the next section we describe our algorithm and give details about its implementation using widely use software libraries. Our practical problem is from the domain of digital heritage, more precisely the classification of images of architectural type. Then we discuss numerical and timing results. Our directions for future work and conclusions are provided in the last section.

2 Base Version of the Monte Carlo Method for SVM Training

In the theory of using some stochastic approaches in the training of SVMs there are many techniques that are known and used. The Stochastic Gradient Descent is a well known technique in the training of both SVMs and Artificial Neural Networks.
There are some sophisticated approaches like the one in (Rudi, 2017), however, they require some numerical stability and may become computationally heavy with the increase in the number of samples.

Here we consider a simpler Monte Carlo approach, where we solve multiple smaller systems, formed by using randomly chosen subsets of the samples, and then aggregate the coefficients that were obtained. More precisely, suppose that we have N samples and we decide to use m smaller models. We use m groups of approximately $\frac{N}{m}$ samples each. It is not so important if the number of samples in each group is equal or not. We could even allow for overlapping of the groups, if we form each group independent of the others. Each group k produces some weights w_k .

By averaging the contributions of all models we obtain a Monte Carlo aggregate model.

Let's evaluate the amount of computations that are necessary for obtaining such an aggregate model. In the regular case, we need N^2 kernel evaluations and then we need to solve a linear system, which would require approximately $\frac{N^3}{3}$ computations, if a Gaussian elimination is used. If we split into m sub-models, we need only $\frac{N^2}{m}$ kernel evaluations and we solve the m systems with a total of approximately $\frac{N^3}{3m^2}$ computations. Therefore, depending on the difficulty of evaluating the kernel, we have advantage of either m or m^2 in speed. We should also consider the fact that the memory requirement of the Monte Carlo method would be about m^2 smaller, since we can train these models one at a time. This is important in Big Data problems, which usually are thought to allow only linear SVM with acceptable speed and memory requirements.
While the accuracy of a regular Monte Carlo method is usually proportional to the inverse square root of the number of samples, for classification problems we can not expect the accuracy measured as classification error percentage to obey this law. We observed that the accuracy converges to the accuracy of the full method and the difference is relatively small, within a few percent. Even if in some situation a difference of 1-2 percent can be important, we can still use the Monte Carlo method during the hyper-parameter search phase, where lots of training problems should be solved and evaluated, and then apply the full method with the best hyper-parameters, see for example are given in (Claesen, 2015; Cristianini, 2020; Fraccaroli, 2020).

3 HPC Implementation of the Monte Carlo SVM Training Method

In this paper we concentrate on the practical problem of image classification. As usually the image datasets may come with different resolutions of the images, they need to be read in memory and resized before the training can proceed. For this task we tried two software libraries - ***boost/gil*** and ***opencv***. In order to speed-up the data processing phase of the training, we used OpenMP to parallelise the cycle over the list of images to be read. We had some technical problem with ***boost/gil***, which did not work correctly with our OpenMP code, therefore we could only use ***opencv***.

In the next figure (Fig. 1) one can see how changing the number of threads by using the environment variable ***OMP_NUM_THREADS*** impacts the overall data loading time.

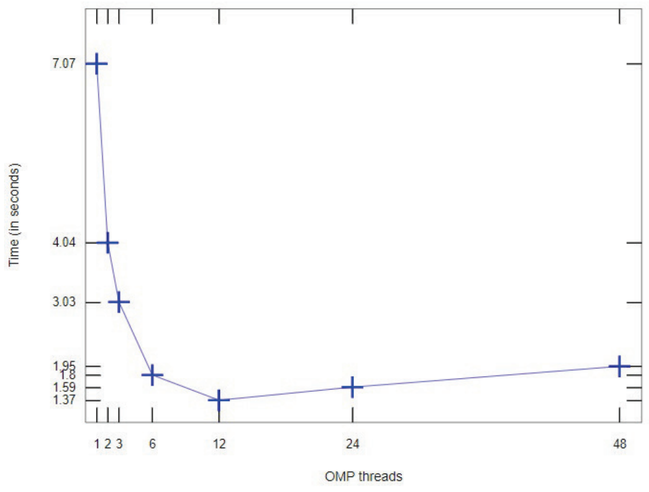

Fig. 1. Time for loading the images using different number of OMP threads.

It is entirely possible that we are seeing a bottleneck from the filesystem reading part or some interaction between the OpenMP parallelisation and ***opencv***. In any case, we see that the optimum performance is achieved for 12 CPU cores.

Our system is equipped with dual CPUs Intel Xeon Gold 5118 with 24 physical cores and 48 logical cores (hyper-threading enabled). It also has a powerful Nvidia V100 GPU, which was useful in the next phases of the computation (NVIDIA, 2017).
For the kernel we used simple Gaussian kernel, which uses the Euclidian distance between the images. Although this specific kernel can be computed efficiently through some well-known linear algebra tricks, we implemented a GPU kernel, which gives us flexibility to add more complex processing in the future.

The most important issue that we observed in the development of this kernel was that when implemented a straightforward cycle to compute distances, one has to take into account memory coalescence. The best performance was obtained when the second

matrix that we used internally was transposed in order to make sure that threads that are close together (from one warp) do not fetch memory locations that are far apart.
In the next figure (Fig. 2) one can see the difference in performance when the number of threads in the kernel invocation are varied.

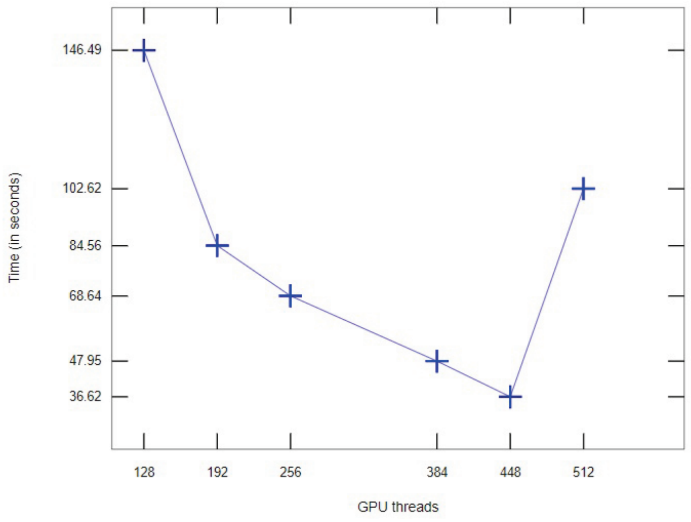

Fig. 2. Time for the kernel computation on the GPU when the number of threads are varied.

Note that many times it is not possible to use high number of threads because of the resulting register pressure. In this case the simplicity of the kernel decreased the possibility of this problem happening. Nevertheless, we see that the optimum performance is achieved when using 448 threads, as performance drops with 512 or more threads.

After the computation of the kernel we need to solve the corresponding linear system. Although we used MKL and MKL-enabled version of ***armadillo*** for the usual CPU-based computations, we invoked the Nvidia BLAS subroutines for solving the linear systems entirely on the GPU.

Since the results of the kernel computations are obtained first at the GPU, we could bypass the memory copy back and forth between host and GPU. As the CPU is free during the tasks of solving linear systems, in a more complex software system it could be used for other computations.

4 Numerical Results

We tested our approach mainly using the Architectural Heritage Elements Dataset (AHE_Dataset) dataset (Llamas, 2017; Abed, 2020), although we also checked the ***Imagenet*** dataset, with similar results. As we stated above, we used our approach in the determination of the best hyper-parameters of the model (the constant in the Gaussian kernel and the regularisation constant), which was performed by a simple grid-search with exponential step size, as is usual in this domain. For each pair of images, we used the best hyper-parameters. We can see in the next tables (Table 1 and Table 2) how the accuracy changes with the change in the number of samples *m* in the Monte Carlo method.

Table 1. Percentage accurately classified (higher is better)

Models	1	2	3	4	9
Accuracy	**84.26**	82.43	84.13	80.90	79.14

Table 2. Percentage correctly classified using one-vs-one method (higher is better)

Models	1	2	3	4	9
Accuracy	62.20	60.93	**62.92**	60.13	59.41

In Table 1 we show the average accuracy for pairs of categories.

One can see that the improvement in accuracy when using the full dataset, e.g., without Monte Carlo, is relatively small compared with the significant speed advantage in the order of at least *m* of the Monte Carlo method, and even some of the Monte Carlo tests show better performance. The users can determine themselves what kind of decrease in accuracy is acceptable for the particular problem.

If all pairs of models are trained, they can also be used in a *one-vs-one* type of multi-class classifiers. In Table 2 we can see the accuracy in this case, although it is relatively low. We can see how $m=3$ for the Monte Carlo samples achieved even better accuracy than $m=1$, e.g., the Monte Carlo method is more accurate than the full computation. Our feeling is that a more sophisticated kernel needs to be used before we can get better results in the multi-class problem. Fortunately, the method itself does not depend on the particular kernel choice, so such an extension will be straightforward to implement.

5 Conclusions and Directions for Future Work

The kernel-based methods are currently falling behind in popularity with respect to Deep Neural Networks, especially for the task of image classification. However, they still retain some well-known advantages and there are many use cases where they are applicable. In our work we proved that through the use of Monte Carlo method and partial use of GPGPU computing during the training process, one can speed-up the training and obtain usable SVM models in acceptable time. Thus with relatively small overall computational budget one can still achieve usable accuracy in the classification. Our algorithm has enough flexibility to accommodate more advanced kernels, which would further expand its domain of applicability. We have shown that even in its basic version, it can be applied successfully in the task of classification of objects from the domain of digital cultural heritage. We plan in our future work to test the use of more sophisticated kernels, add data augmentations steps, and use specially designed low-discrepancy sequences in some places where pseudo-random numbers are currently used.

Acknowledgements.

This work has been financed by a grant from CAF America.

We acknowledge the provided access to the e-infrastructure of the Centre for Advanced Computing and Data Processing, with the financial support by the Grant No BG05M2OP001-1.001-0003, financed by the Science and Education for Smart Growth Operational Program (2014-2020) and co-financed by the European Union through the European structural and investment funds.

References

Abed, M. A.-A. (2020). Architectural heritage images classification using deep learning. *Proceedings of the 2nd International Workshop on Visual Pattern.*

Alexandrov, A. K. (2005). Finding the Smallest Eigenvalue by the Inverse Monte Carlo Method with Refinement. *ICCS 2005: Computational Science – ICCS 2005. 3516*, pp. 766-774. Lecture Notes in Computer Science. doi:doi.org/10.1007/11428862_104

Boser, B. E. (1992). A Training Algorithm for Optimal Margin Classifiers. *(COLT'92), Proceedings of the 5th Annual Workshop on Computational Learning Theory.* Pittsburgh, PA: USA: ACM Press. doi:doi.org/10.1145/130385.130401

Claesen, M. &. (2015). *Hyperparameter Search in Machine Learning.* MIC 2015: The XI Metaheuristics International Conference, CoRR, abs/1502.02127. .

Cristianini, N. S.-T. (2020). *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods.* Cambridge University Press, ISBN: 0521780195.

Dimov, I. D. (1999). A new iterative Monte Carlo approach for inverse matrix problem. *Journal of Computational and Applied Mathematic, 92*(1), 15-35.

Fraccaroli, M. L. (2020). Automatic Setting of DNN Hyper-Parameters by Mixing Bayesian Optimization and Tuning Rules. (O. a. Nicosia G. et al. (eds) Machine Learning, Ed.) *Lecture Notes in Computer Science, 12565*. doi:doi.org/10.1007/978-3-030-64583-0_43

Kroese, D. B. (2019). *Data Science and Machine Learning: Mathematical and Statistical Methods.* Chapman and Hall/CRC.

Llamas, J. L.-G.-B. (2017). Classification of Architectural Heritage Images Using Deep Learning Techniques. *Applied Sciences*, 992.

Mikhailov, G. (1992). *Optimization of Weighted Monte Carlo Methods. Springer-Verlag Berlin-Heidelberg.*

NVIDIA. (2017). Retrieved from NVIDIA Tesla V100 GPU Architecture the world's Most Advanced Data Center GPU. White paper http://www.nvidia.com/object/volta-architecture-whitepaper.html.

Rudi, A. C. (2017). FALKON: an optimal large scale kernel method. *NIPS'17: Proceedings of the 31st International Conference on Neural Information Processing Systems*, (pp. 3891–3901).

Snoek, J. L. (2012). Practical Bayesian Optimization of Machine Learning Algorithms. *Advances in Neural Information Processing Systems*, *25*, pp. 2960-2968.

Received: June 15, 2021
Reviewed: July 10, 2021
Finally Accepted: July 25, 2021