Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 108C (2017) 2318–2322

Procedia Computer Science

International Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland

SW-SGD: The Sliding Window Stochastic Gradient Descent Algorithm

Imen Chakroun[1,3], Tom Haber[2,3], Thomas J. Ashby[1,3]

[1] imec, Kapeldreef 75, B-3001 Leuven, Belgium,
ashby@imec.be imen.chakroun@imec.be
[2] Expertise Centrum for Digital Media,
Wetenschapspark 2, 3590 Diepenbeek, Belgium tom.haber@uhasselt.be
[3] ExaScience Life Lab,
Kapeldreef 75, B-3001 Leuven, Belgium,

Abstract

Stochastic Gradient Descent (SGD, or 1-SGD in our notation) is probably the most popular family of optimisation algorithms used in machine learning on large data sets due to its ability to optimise efficiently with respect to the number of complete training set data touches (epochs) used. Various authors have worked on data or model parallelism for SGD, but there is little work on how SGD fits with memory hierarchies ubiquitous in HPC machines. Standard practice suggests randomising the order of training points and streaming the whole set through the learner, which results in extremely low temporal locality of access to the training set and thus, when dealing with large data sets, makes minimal use of the small, fast layers of memory in an HPC memory hierarchy. Mini-batch SGD with batch size n (n-SGD) is often used to control the noise on the gradient and make convergence smoother and more easy to identify, but this can reduce the learning efficiency wrt. epochs when compared to 1-SGD whilst also having the same extremely low temporal locality. In this paper we introduce Sliding Window SGD (SW-SGD) which uses temporal locality of training point access in an attempt to combine the advantages of 1-SGD (epoch efficiency) with n-SGD (smoother convergence and easier identification of convergence) by leveraging HPC memory hierarchies. We give initial results on part of the Pascal dataset that show that memory hierarchies can be used to improve SGD performance.

Peer-review under responsibility of the scientific committee of the International Conference on Computational Science

Keywords: SGD, sliding window, machine learning, SVM, logistic regression

1 Introduction

Many supervised machine learning algorithms be formulated in terms of a mathematical optimization problem where the key challenge is to identify parameters of a given model so that the number of miss-classified points is minimized. The learned model is used in the operational phase to make the predictions for the unlabelled points. In order to capture the quality of the

1877-0509
Peer-review under responsibility of the scientific committee of the International Conference on Computational Science
10.1016/j.procs.2017.05.082

model, a cost (also called loss) function calculating the error between the predicted and the real values on the training set is used. The further away the prediction is from the real value of a training point, the larger the value of the cost function for that point is. The goal of training the model is to minimize the total cost over the whole training set.

First order gradient-based optimization methods are popular for minimising this cost. They attempt to find the values of the model parameters (also called the weights vector) that minimize the cost function when they cannot be calculated analytically. An iteration of these methods usually has a form similar to Equation 1 which shows that the method iteratively takes small steps (α being the step size) in the direction of the negative gradient of the cost function:

$$w_{k+1} = w_k - \alpha_k \frac{\partial C(w_k)}{\partial w_k} \tag{1}$$

where α_k is the step size at step k, w_k is the weight vector at step k, and C is the cost of all training points for a particular weight vector.

Two simple but effective methods for gradient-based optimization are stochastic gradient descent (SGD) and batch gradient descent (GD). The difference between both methods is the size of the sample to consider for computing the gradient. While the batch method uses all training examples (i.e. the whole training set) to calculate the gradient at each step, the stochastic gradient method chooses a single training point at random. This difference in the number of updates per entire training set read leads to two extremes of exploration behaviour: batch gradient descent can be very slow and expensive as we need to calculate the gradients for the whole training set to perform one update. The convergence is however smoother and termination is more easily detectable. SGD is less expensive but suffers from noisy steps and consequently hard to spot termination.

A variant of the SGD method called mini-batch SGD (n-SGD) considers a reasonably small group of training points for computing the gradient at each step, and tries to balance training set access efficiency against noise; the size of the mini-batch n must usually be specified by the user. The random selection of which training point to use at a given step for SGD is typically implemented as a random shuffling of the order of the training vectors rather than genuine random training point selection. The shuffling can be done once before the algorithm is run, or after every epoch. In the following, we sometimes use 1-SGD to denote normal one point SGD in the same notation as n-SGD to avoid ambiguity.

An entire touch of the training set is called an *epoch*. While having different access patterns and updates per epoch, SGD and n-SGD have a common characteristic which is low temporal locality of access to the training set. Each training point is used once, and then not used again until all the other training points have been visited. This means that a cache layer (e.g. DRAM caching for data on disk, or SRAM caching for data in DRAM) in the memory hierarchy of a modern HPC computer system will have little benefit for the algorithm unless all the training points fit inside that cache.

Various authors have looked at data or model parallelism for SGD to be able to benefit from the parallelism available in HPC architectures; see [4] for a particular example and an entry to the literature. However, how to improve the interaction of SGD with memory hierarchies, another important architectural feature of HPC machines, is largely overlooked, despite the need to deal with large data sets and the importance of exploiting memory hierarchies to achieve performance. Sliding Window SGD (SW-SGD), a gradient descent optimization algorithm which aims to increase the locality of training point accesses while combining the advantages of SGD and n-SGD in terms of epoch efficiency and smoother convergence respectively.

The remainder of this paper is as follows: in Section 2 the stochastic gradient descent and

the mini batch stochastic gradient descent are described. The sliding window SGD is introduced in Section 3. Experimental results and associated discussion are presented in Section 4. Some conclusions and perspectives on this work are drawn in Section 5.

2 Stochastic Gradient Descent

SGD is popular and has proven to achieve state of-the-art performance on a variety of machine learning tasks such as in [5] and [3]. SGD includes a parameter called the learning rate to define the length of the next step to take when moving forward in the direction of the gradient. Choosing a proper learning rate is a research field in itself and several contributions exist in this context. Generally speaking, a learning rate that is too small leads to slow convergence, while a learning rate that is too large can hinder convergence and cause the loss function to fluctuate around the minimum or even to diverge. In this work, we have opted for a learning rate that is adjusted during the training according to a pre-defined schedule similarly to [5]. The step size is however bigger for n-SGD than for SGD since the noise on the gradient for one point SGD makes it harder to take larger steps without risking preventing convergence.

In n-SGD, the model is updated based on small groups of training samples called mini-batches. In this case, the batch size is set to n, being the number of points to consider in the computation of the gradient. The advantages of using mini-batch gradient descent are two fold: (1) it allows the model to converge nearly as quick as SGD (in terms of time) while (2) having convergence nearly as smooth as GD.

3 Sliding window SGD

The accesses to data items in the cache have much lower latency and much higher bandwidth than accesses to data items in the large memory. Modern HPC systems consist of many layers of memory in a hierarchy, where each pair of adjacent layers has this relationship. If the training set is larger that the cache capacity, then there is no benefit from the cache memory when accessing the data in the large memory by streaming the whole training set and using each item once during an epoch, in the manner that SGD, n-SGD and GD do. Due to the higher bandwidth, the CPU can access potentially many training points in the cache in the time that it takes for a new training point to be fully loaded from the large memory into the cache. SW-SGD is motivated by this observation.

The gradient in SW-SGD is computed using new training points that have just been loaded from the large memory, along with some number of training points that are still in the cache. These extra training points in the cache are essentially free to use, due to the cache effect and the fact that accessing them uses otherwise dead time whilst waiting for new points to load into the cache. Using these extra points in a gradient calculation should provide some extra smoothing similar to the extra points used in n-SGD. Ideally the smoothing would be as effective as that in n-SGD, and SW-SGD would achieve lower noise whilst having the same data touch efficiency as 1 point SGD (after accounting for pipeline-fill effects).

The points used by SW-SGD to calculate gradients are sketched in Figure 1 for a version of the algorithm that uses batch size 1 to update the cache at each step, and cache size 4. For SW-SGD, all but the newest vector in each iteration is available (for free) from the cache.

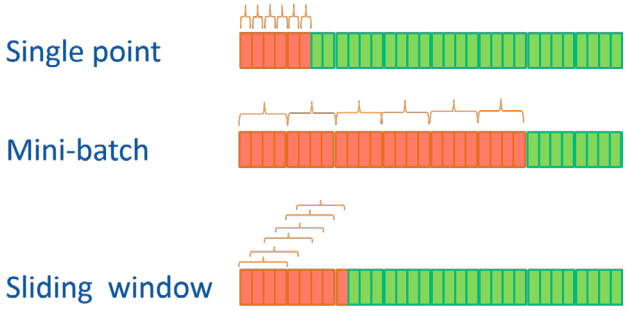

Figure 1: Comparing the data touched with six iterations of 1-SGD, 4-SGD (mini batch) and SW-SGD with cache size 4. The training points accessed at each iteration are under a bracket. The total number of training points accessed after 6 iterations are marked in red.

4 Experiments

4.1 Experimental results

The experiments presented in this section have been conducted on a benchmark dataset extracted from the Pascal challenge on large scale hierarchical classification [1] which is a text classification challenge. The data set is composed of 250,000 training points and 250,000 test points. The used model is a LogLoss support vector machine (SVM) trained with gradient descent. The initialization of the step size is set based on a pre-training phase applied to a subset of the complete dataset such as performed in [5].

Preliminary experiments ran on the Pascal dataset and comparing the loss function obtained with one point SGD and with n-SGD using different batch sizes, have shown that using a mini batch size of 25 converges faster than the other batch sizes. Therefore, for the rest of the experiments a batch size of 25 data points is considered when speaking about the n-SGD.

In Figure 2, different sizes of SW-SGD are compared (left). Besides a mini batch sized 25, old already visited points are also considered in the computation of the gradient at each iteration. The results show that computing the gradient using 25 new points and 50 old points gives the better convergence speed and accuracy. This observation is a result of a comparison with two other configurations: a) 25 new points + 25 old points, and b) 25 new points + 75 old points. The less efficient SW-SGD scenario (in terms of smoothness) namely the combination of 25 old points and 25 new points is compared in Figure 2 (right side) to the 25-SGD (only 25 new points). The results show that the fluctuation in the 25-SGD are higher than the variation observed with SW-SGD and that SW-SGD converges slightly earlier. This shows that SW-SGD does indeed improve smoothness, and our results show that increasing cache size improves the smoothness; SW-SGD with 25 new points + 75 old points can be seen to be significantly better than standard 25-SGD in terms of smoothness when comparing the two parts of Figure 2.

It is important here to remember that using a n-SGD with size of 50 new points is less efficient than using just 25 new points (25 being the best mini batch size found in preliminary experiments). The added value here is therefore brought by the characteristic of considering old visited points in the computation and not because of a bigger batch size.

Note that the experiment described above is phrased in terms of iteration counts, and not in terms of actual machine performance. The paper is indeed investigating the principle of whether SW-SGD can work, rather than the practical implementation details.

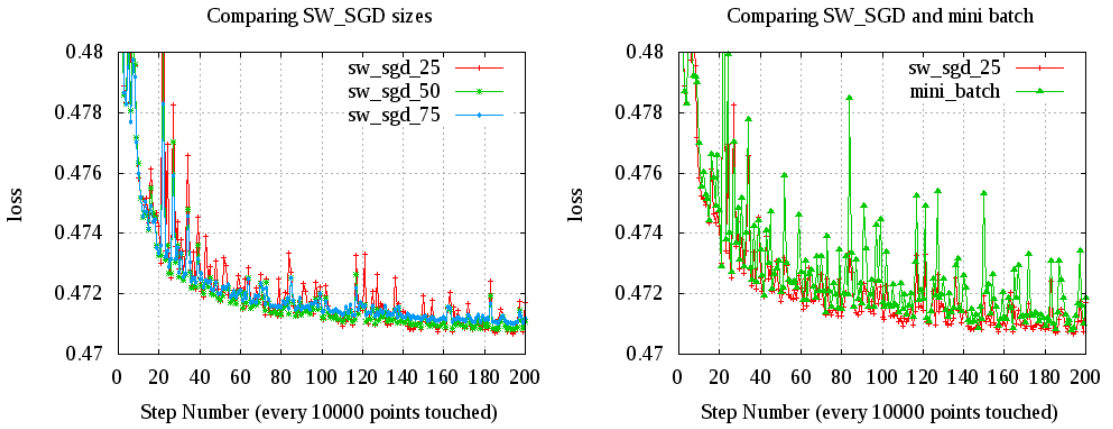

Figure 2: Comparing different sizes of SW-SGD (left) and n-SGD with SW-SGD (right).

5 Conclusion and future work

In this paper, we introduced SW-SGD which adapts SGD to the access characteristics of modern HPC memory to gain extra gradient noise smoothing, hopefully for free. In this way it combines the epoch efficiency of 1-SGD with the lower noise and easier to spot convergence of n-SGD. We compare the approach to 1-SGD and n-SGD in some initial experiments on the Pascal data set. We show that SW-SGD can improve over n-SGD in terms of gradient noise and convergence, for a given number of loads of training points from the large slow memory level. In subsequent work we will expand the experiments to better understand under what circumstances SW-SGD should be used, and how to dimension the cache and implement the algorithm such that the overall performance is better than n-SGD, in terms of total time to solution.

Acknowledgements

This work is funded by the European project ExCAPE [2] which received funding from the European Union's Horizon 2020 Research and Innovation programme under Grant Agreement no. 671555.

References

[1] http://lshtc.iit.demokritos.gr/node/1.

[2] http://www.excape-h2020.eu/

[3] S. Ahn, A. Korattikara, N. Liu, S. Rajan and M. Welling. Large-Scale Distributed Bayesian Matrix Factorization using Stochastic Gradient MCMC. In Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. New York, USA, 2015.

[4] J. Keuper, F-J. Pfreundt. Asynchronous parallel stochastic gradient descent: a numeric core for scalable distributed machine learning algorithms. In Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments. New York, USA, 2015.

[5] L. Bottou: Large-Scale Machine Learning with Stochastic Gradient Descent, Proceedings of the 19th International Conference on Computational Statistics (COMPSTAT'2010), pp. 177–187, Edited by Yves Lechevallier and Gilbert Saporta, Paris, France, August 2010, Springer.