Reviewing the article

SA-SDCA review and UPDA proposal

Pavel Plyusnin
Moscow Institute of Physics and Technology
pavel@plyus.pw

ABSTRACT

Stochastic Dual Coordinate Ascent is one of the most efficient ways to solve the family of regularized empirical risk mininimization problems. This paper provides a review of the article entitled "Scaling Up Stochastic Dual Coordinate Ascent" by K.Tran, S.Hosseini, L.Xiao, T.Finley and M.Bilenko [3]. It reveals positive and negative aspects of the article. Also, in this paper another parallel version of the algorithm, named as UPDA, is introduced. UPDA seems to be efficient and strong algorithm, which can tackle out even problems with several local minimums, what other descent based methods can not do.

KEYWORDS

SDCA, SA-SDCA, Machine learning, Yandex

ACM Reference Format:
Pavel Plyusnin. 2018. Reviewing the article: SA-SDCA review and UPDA proposal. In *Proceedings of SA-SDCA review and UPDA proposal*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Efficient linear learning is an important task in machine learning. Although, there are more precise methods, such as neural networks, tree forests, boosting techniques and so on, most of them are uninterpretable. Moreover, these complicated methods often learns too long. That is why linear models are still widely used: despite the fact that linear regression can build only linear separating surfaces, their performance in big-data is pretty good. This result is explained by high overall dimensionality, examples are often very sparse with only few non-zero features. So, this fact, together with the facts that linear models are fast and scalable, make it to be a very popular method.
The goal of linear methods is to minimalize regularized empirical loss of linear predictors

$$P(\omega) = \frac{1}{n}\sum_{i=1}^{n}\phi_i(\omega^T x_i) + R(\omega) \tag{1}$$

[3] examines in detail only L2 regularization, claiming that L1 regularization can be solved the same way and referring to the other article [2]. The minimization task (1) can be solved in many ways and in [3] authors propose to solve a dual problem (2) and present SA-SDCA algorithm.

$$D(\alpha) = \frac{1}{n}\sum_{i=1}^{n} -\phi_i^*(-\alpha_i) - \frac{\lambda}{2}\left\|\frac{1}{\lambda n}\sum_{i=1}^{n}\alpha_i x_i\right\|_2^2 \tag{2}$$

In the first part of the article the detailed review of [3] is given. In the second part a new approach to solving problem (1) is proposed.

2 REVIEW OF [3]

2.1 Positive sides

The article trying to solve an important task that is still relevant. Most of the proposals are proven in the limits of their assumptions, which are rather reasonable and correct (I will say more about the assumptions later). The given proofs are correct. Moreover, the article does not only propose an parallelization of SDCA algorithm, but also considers about its practical implementation: it introduces a technique based on decoupling the data input interfaces, and implementing them for disk-based data with block-wise compression and indexing on top of multi-threaded, buffered I/O.
The paper isn't overextended by mathematical calculations and equations, it has a positive effect on readability and accessibility. The work is carried out to the end: the algorithm performance is illustrated in several graphs.

2.2 Drawbacks

Around the same time of publication of the under consideration paper, another one was published [1]. In this article the same problem is solved by a similar idea, so it will be reasonable to compare these two articles, considering one in the context of another.
Now let's turn to the drawbacks of original paper:

- **There is no proven convergence analysis of SA-SDCA**
  As is known, the solution of primal task corresponds to the solution of dual. However, it is true only when SDCA algorithm is performed sequentially, in a step-by-step way, that is, only in a single thread case. The authors proved the convergence of their A-SDCA with Theorem 3, but also it was shown, that A-SDCA does not convergence asymptotically to the optimal solution. Therefore, it means that the A-SDCA reaches only suboptimal level. This explained by the failure of condition (3), which is necessary for reaching the optimal point.

  $$\omega(\alpha) = \frac{1}{\lambda n}\sum_{i=1}^{n}\alpha_i x_i \tag{3}$$

  In the SA-SDCA algorithm authors periodically recalculate ω in accordance with (3), considering that this forced equality

SA-SDCA review and UPDA proposal, April 2018, Moscow, Russia

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$0
https://doi.org/10.1145/nnnnnnn.nnnnnnn

will result in proper convergence. But this belief remains only a belief, without any evidence of correctness of such a replacement, although it is obvious that this measure is not equivalent to original equation (3). Correctness of SA-SDCA algorithm is confirmed only by empirical findings. It can be said, that the paper does not have any error analysis.

Now if we will turn to the [1] article, we will see a detail error analysis: for example, it is shown, that primal solution w, computed by PASSCoDe-Wild (the alternative algorithm to SA-SDCA), is actually the exact solution of a perturbed problem (Corollary 1). It can be said, that the correctness of a proposed algorithm was better proven in this article than in the previous one. Is such a mathematical meticulousness really necessary – I'm not to judge.

- **Assumption, that the component-wise addition in the line 7 of A-SDCA algorithm is atomic**
  The authors assume that operations in lines 6-8 of their A-SDCA algorithm are indivisible or simultaneous (see equation (12) in paper [3]). Rather reasonable assumption, but this cannot be guaranteed in the implementation. In this case, the authors propose using only one random $v \in e$. Obviously, this seriously reduces the efficiency of the entire algorithm.
- **Strong limitations**
  Also in the article strict constraints are imposed. First of all, the authors introduces smoothness assumptions on the convex losses function, which is a rather severe limitation. Even hinge loss, which is used in SVM, isn't smooth. Furthermore, dataset is required to be sparse enough:

$$\Omega p\tau = O(1/\sqrt{n})$$

  Good that it isn't a strong limitation: this condition can be satisfied by many sparse datasets. But it is worth noting, that in article [1] this assumption is not significant at all.

To sum it up, in the given article the authors propose the good method of parallelization SDCA. Despite some lack of proofs in math, this method performs well in practice. This result is confirmed by many graphs, comparing SA-SDCA with several other state-of-the-art algorithms, so these new techniques (SA-SDCA and block-compressed binary format) have a significant potential in different applications.

2.3 Alternative approaches

For smooth convex functions and large datasets other methods can be used. For example, let's take a look at conjugate gradients. This efficient algorithm converges very quickly: the method guarantees convergence in a finite number of steps, and the required accuracy can be achieved much earlier. Moreover, this algorithm is good in parallelization, it have almost a linear speedup. There are many papers describing parallel variant of this algorithm, so it is a well-researched variant.

Another variant is to use coordinate descent algorithm but for the primal task, not for dual. For example, such research have been conducted by I.Trofimov and A.Genkin in [4]. They proposed the distributed GLMNET algorithm for linear regression both for L1 and L2 regularizers. As it shown in the paper, this algorithm performs fine, and what is most important that it could be easily extended to other regularizers and linear models, not only linear regression can be solved with it.

2.4 Future work

- **Minor additional improvements**
  Of course, the new research, revealing this empirical strong performance of the algorithm, could be conducted. However, I don't think, it so important, because it won't bring anything new, only minor changes. In addition, we can try to generalize the method for L1 regularizer in accordance with [2] and for all loss functions. In this case, new experimental tests should be conducted.
- **Other approaches**
  The given article heavily relies on solving the dual problem. Formulating the problem through its dual and solving this task is claimed to be faster than solving the primal one [1]. But this is not a panacea and maybe it is worth returning to solve the original primal task. Moreover, there are many other approaches, some of them were considered in Alternative approaches section.

3 MY CONTRIBUTION

In all the papers I've seen, researchers try to parallelize the calculation of the same optimization problem. So, the main task in this case is to distribute descent steps between calculators. Unfortunately, there are many problems with synchronization: it is hard to maintain variables in a proper conditions, it's hard for processors to make efficient steps because they can't use the newest state of the system, so they all make iteration from the same starting point and main process often should to average the result, which came from the different calculators.

3.1 Naive parallel descent algorithm

I propose a completely different way. In a brief, each process will calculate its own optimization problem with its own starting point. Having done several iterations, the processes send the current solution to the main process without waiting for convergence. The main thread averages the result and initializes the other processes with the new starting points in a neighborhood of the calculated average. The algorithm is shown in Algorithm 1:

Examples of how algorithm works are shown in Figure 1 and Figure 2. Figure 1 shows the calculated result, when repeat loop was executed once, i.e each processor made $k = 5$ steps. As seen, the result after just one iteration is pretty good: it is almost as good as the results from «lucky» threads, which started closer to the optimum point, and isn't spoiled by «unlucky» threads, which started far from optimum point.
Figure 2 demonstrates that average result is often better than the result from single thread. Also, it illustrates 2 iterations of repeat loop.
In addition, it should be noted, that any of the descent methods could be used in line 3. The algorithm may solve primal problem (1), so the simple Stochastic gradient descent (SGD) can be used. Also,

Algorithm 1: Naive parallel descent algorithm (NPDA)

Data: Initial w, n - number of threads, k - number of steps to do on each iteration
Result: Optimal w for problem (1)

repeat
  In the neighborhood of w $O(w)$ *randomly* choose n vectors w_{0_i};
  Each of n threads makes k iterations of chosen method (SDCA) computing w_i, having been initialized with w_{0_i};
  $w \leftarrow \frac{1}{n}\sum_{i=1}^{n} w_i$
until *until convergence*;

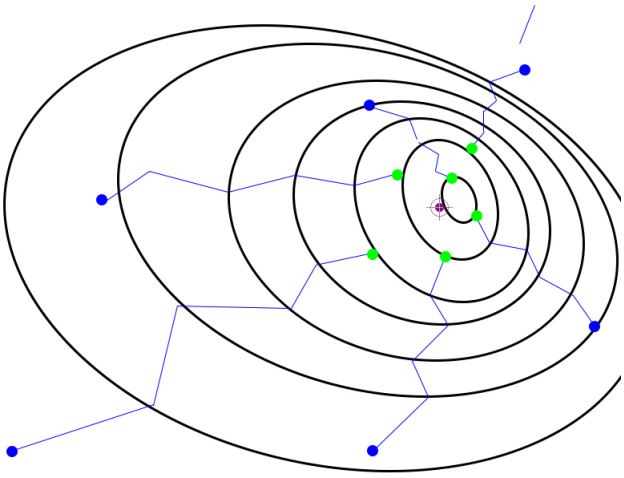

Figure 1: Demonstration of the algorithm on 6 threads. Each thread have made 5 steps, having started from blue point and finished in green one. The result point is marked with red target symbol

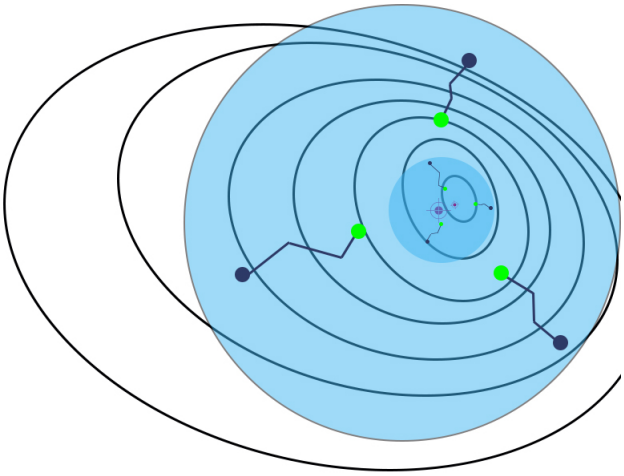

Figure 2: Demonstration of the algorithm on 3 threads. Each thread have made 3 steps, having started from blue point and finished in green one. The result point is marked with red target symbol. The repeat loop is executed twice

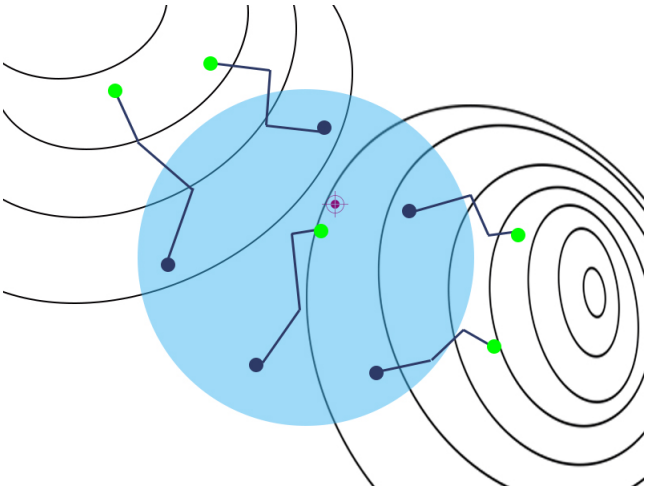

Figure 3: In the neighborhood of starting point there are several local minimums, so different threads may converge to different answers. Average result is very bad

you can reformulate problem through its dual (2) and maximize corresponding function. In this case, SDCA can be used. So that, it does not matter, which method is used and dual or primal problem is solved. Without loss of generality, hereinafter is assumed that primal task is solved (so, the solution for minimization problem is required)
Now let's take a closer look at the NPDA and point out its drawbacks. It helps to introduce the improved algorithm.

(1) **Starting point** The starting point is very important for successfull perfomance of NPDA. In particular, *it should be only one solution for the minimization task near the starting point.* Figure 3 demonstrates what problems occur if this condition is not met. Threads, having started with different starting point (all of them are in neighborhood of initial one), converge to different optimal solutions. After averaging, the algorithm obtains a very bad result point, which does not approach to any of the real solutions. Most likely, after many iterations the algorithm will converge to the one of the solutions, but it will be very slow, not faster than if only one process worked (see next paragraph)
(2) **Radius of neigborhood**

- A high rate of convergence of the algorithm is ensured by the fact, that *all the threads approach the solution from different sides.* After the averaging, the result point is closer to the solution than any previous one. From this fact appears the constraint on the radius, in which new starting points for threads are chosen: the optimal solution should belong the neighborhood of average (and the first initial)

point all the time. Moreover, the closer the real solution is to the center of the neighborhood, the better.
- If the condition above is not satisfied, the method will still converge but without any speed-up: all the threads will make similar calculation, so there will be no difference between one ore more threads. Thats why algorithm converges in previous item (1): after some time, the algorithm will converge to the one of the minimums, but only from the one side. Considering the fact, that method should be good in parallelization, the condition above must be satisfied for efficient work.

3.2 Ultimate parallel descent algorithm

To overcome the problems above, the UPDA is introduced. Its main difference from the Naive one is that it can tackle with several local minimums nearby. After each thread calculated the result vector w_i, the main thread, using some *euristics*, determines, in which directions these vectors have shifted, and from these conclusions groups w_i by the found local minimums. If there are several local minimums, the algorithm processes them in separate and finally chooses the best solution. If *Euristic* isn't too difficult to compute, these changes have not a negative effect on the perfomance. The formal algorithm is presented below:

Algorithm 2: Ultimate parallel descent algorithm (UPDA)

Data: Initial w, n - number of threads, k - number of steps to do on each iteration
Result: Optimal w for problem (1)

1 **Function** UPDA(w, n, k)
2 In the neighborhood of w $O(w)$ *randomly* choose n vectors w_{0_i};
3 Each of n threads makes k iterations of the chosen method (SDCA) computing w_i, having been initialized with w_{0_i};
4 With n vectors w_i, determine the number of the nearest minimums and their w: $Mins \leftarrow Euristic(w_1, \dots, w_n)$;
5 **for** $m \in Mins$ **do** in parallel
6 $w_m \leftarrow \frac{1}{|m|} \sum_{w_i \in m} w_i$;
7 **if** *not convergence* **then**
8 $w_m \leftarrow \text{UPDA}(w_m, \left\lfloor \frac{n}{|Mins|} \right\rfloor, k)$;
9 **return** $\arg \min_{w_m : m \in Mins} P(w_m)$

The example of its work is illustrated in Figure 4.
There are still many uncertainties and questions. They all are objectives for future works:

(1) **How to choose the initializing point**. In case of only one local minimum NPDA is good enough and performs faster than UPDA
(2) **How to choose the radius of neighborhood**. It is obvious, that radius should shrink with each iteration. But at what speed? Moreover, the constraint (2) in the previous chapter should be satisfied too. The explicit formula must be designed

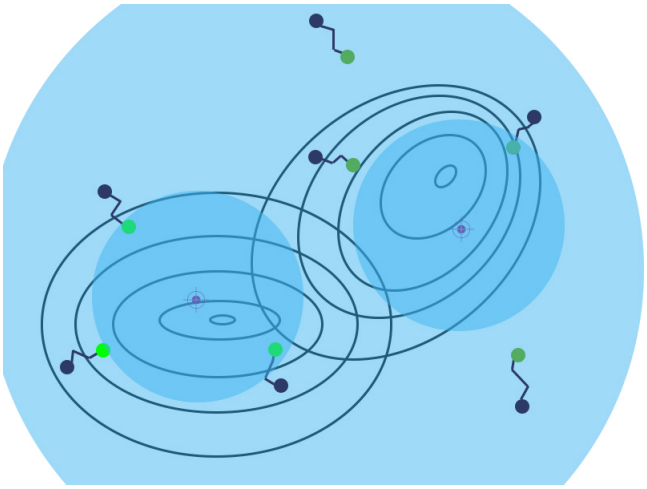

Figure 4: In the neighborhood of starting point there are several local minimums, different threads converges to different answers. Euristic succesfully distinguish 2 local minimums, so next iterations will be in the neighborhood of 2 different starting points for different solutions

(3) **How to choose start points for different processes (line 2)**. Will it be a good idea to choose them in a random way? Or it is worth taking points strictly on the border of the neighborhood. Moreover, maybe they should be maximally distant from each other. Or maybe the neigborhood should be divided by n sectors, and random point should be chosen from the each sector. There are many options to choose them and the better way should be found.
(4) **Parameter** k. k is a hyperparameter of the method. So its best value or some recomendations should be found.
(5) **Number of threads** n. What is the optimal number of processes? It's obvious that speed-up improvement won't rise infinitely by increasing n. Quite possibly, the optimal number of threads n is 2, 3, or the dimension of the feature space, increased by 1.
(6) **Proof of work**. Finally, the efficent work of the algorithm should be proven in an experimental way. It should be tested on different data sets and its perfomance should be compared with other state-of-the-art methods. Moreover, convergence of the algorithm should be proven in theory.

4 CONCLUSION

SDCA is a very efficient method to solve the family of regularized empirical risk minimization problems, that is why there are many attempts to parallelize it. The [3] article proposed an efficient generalizing of this algorithm. In my paper another version of parallel SDCA is proposed, moreover this modification can use any other descent method and can tackle with several local solutions, but it should be researched better.

REFERENCES

[1] C.-J. Hsieh, H.-F. Yu, and I. S. Dhillon. PASSCoDe:Parallel asynchronous stochastic dual coordinate descent. In *ICML*, 2015.

[2] S. Shalev-Shwartz and T. Zhang. Proximal stochatic dual coordinate ascent. arXiv:1211.2772, November 2012.
[3] K.Tran, S.Hosseini, L.Xiao, T.Finley and M.Bilenko. Scaling Up Stochastic Dual Coordinate Ascent. In *KDD'15*, August 10-13, 2015
[4] I.Trofimov and A.Genkin Distributed Coordinate Descent for L1-regularized Logistic Regression. arXiv:1411.6520, November 2014