Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online
CONFERENCE PAPER ARCHIVE

IFAC-PapersOnLine 48-25 (2015) 292–297

Improving Distributed Stochastic Gradient Descent Estimate via Loss Function Approximation

Alexander A. Senov [*]

[*] *Saint-Petersburg State University, Saint-Petersburg, Russia (e-mail: alexander.senov@gmail.com).*

Abstract: Both problems of learning and optimization in context of huge amounts of data became very important nowadays. Unfortunately even online and parallel optimization methods may fail to handle such amounts of data and fit into time limits. For this case distributed optimization methods may be the only solution. In this paper we consider particular type of optimization problem in distributed setting. We propose an algorithm substantially based on the distributed stochastic gradient descent method proposed in Zinkevich et al. (2010). Finally we experimentally study properties of the proposed algorithm and demonstrate its superiority for particular type of optimization problem.

Keywords: Large Scale Optimization Problems; Stochastic Optimization; Parameter Estimation; Learning Algorithms; Parallel Algorithms; Steepest Descent; Least-Squares Estimation

Both machine learning and optimization became very popular and widely used in today's control applications (see, for example Boyd (2012); Granichin et al. (2014)). It is well known that most part of machine learning problems can be translated to the optimization task of real-valued convex parametrized function $\mathcal{L}(\omega)$, known as *loss function* (Bottou (1998)). Moreover, in most cases such functions are *separable* (Boyd et al. (2011)), i.e. $\mathcal{L}(\omega) = \sum_{i=1}^{N} \ell_i(\omega)$. Thus, optimization of convex separable functions became rather important problem in context of control.
Convex functions optimization problem is very well studied (see Boyd and Vandenberghe (2004) for an extensive overview). Assuming function convexity and differentiability one can choose among many first-order iterative optimization algorithms, e.g. *gradient descent* algorithm also known as *steepest descent* originally proposed in Cauchy (1847). The general idea of first-order iterative optimization algorithms is to gradually improve current estimate of function argmin by moving it on direction opposite to the function gradient. These algorithms calculates gradient of entire $\mathcal{L}$ function, thus computational complexity of each iteration is at least $\mathcal{O}(N)$. Since they may need a considerable number of iterations to achieve desirable accuracy they became inapplicable in case of large N. Such algorithms often mentioned as *batch* methods because they process entire batch of $\{\ell_i\}_{i=1}^{N}$ at each iteration. Recent developments in *randomized* optimization algorithms may reduces computational complexity by special randomization and recurrent estimation techniques Granichin and Polyak (2003) but it do not change the situation radically.
In case of large N so-called *online* optimization algorithms win especial popularity. These algorithms consider only small fixed amount of ℓ_i functions at every iteration, thus reducing iteration complexity to $\mathcal{O}(1)$ in terms of N. Despite the fact that they usually need more iterations in contrast to batch methods the overall complexity often turned out be lower. One of the most popular online optimization algorithm is *stochastic gradient descent* proposed in Davidon (1976) and further extensively developed especially for learning purposed Cun et al. (2004); Bottou (1998).
Occasionally, even online algorithms might fail cause of tight time limits. To accelerate them we can use *parallel* computation and corresponding parallel optimization algorithms. Such parallel algorithms (see, for example Boyd et al. (2011); Bertsekas (1997)) efficiently utilize shared memory and tight processes coupling to communicate between processes intensively thus ensuring great speedup.
Though parallel algorithms provide great speedup on a single machine even them may fail if amount of data exceeds size of whole machine memory and it is impossible to scale single machine performance any more. Fortunately, latest hardware and software developments makes possible to design even *distributed* algorithms — that works on multiple computing *nodes* with wired of wireless connection between them. The only limit of such computational environment is network latency, budget and lack of theoretical approaches. Indeed, despite the fact that hardware and software developed significantly in this area not much work has been done in area of distributed optimization, especially in context of such computational models as *MapReduce* which allows only one inter-node communication phase at the end of computation. Some recent developments in this area include Chu et al. (2007) where authors describe distributed (*multi-core* in their notation) realization of several machine learning algorithms (and implicitly several first-order optimization algorithms) in MapReduce computational model. In Mann et al. (2009) authors solve distributed optimization problem by rather simple but inventive strategy: they solve optimization

⋆

problem independently on each node and in the end joint solution obtained simply averaging across solutions from every node. This method is even more attractive cause it perfectly fits into MapReduce model using only one MapReduce iteration. This solution was further developed in Zinkevich et al. (2010) where authors use stochastic gradient descent on every node with only subset of $\{\ell_i\}_1^N$ and provide strong theoretical analysis of convergence for both original and distributed versions of stochastic gradient.

Despite the great theoretical and experimental analysis in Chu et al. (2007), Mann et al. (2009) and Zinkevich et al. (2010) all these papers exploit the same idea for distributed optimization: first run independent optimization processes on every node, then collect and average obtained parameter estimates. In this paper we replace this last averaging phase with slightly more complex but efficient strategy: we build and approximation $\widehat{\mathcal{L}}$ of function $\mathcal{L}$ using data additionally collected at each node during independent optimization process and take $\operatorname{argmin} \widehat{\mathcal{L}}$ as final parameter estimate.

The paper is organized as follows. In section 1 we cover some basics and previous results required for further explanation. Then in section 2 we formulate the particular distributed optimization problem statement. In section 3 we propose an algorithm aimed to solve the proposed problem. Finally, in section 4 we illustrate some of its properties and perform comparative analysis with algorithm proposed in Zinkevich et al. (2010) on modelled data.

1. PRELIMINARIES

In this section we briefly introduce some methods and concepts necessary for further explanation.

1.1 Notation remarks

We use small light symbols x for scalars and indexes (except parameter vector ω), small bold symbols $\boldsymbol{x}$ for vectors, capital light symbols X for constants and sets (except parameter matrix Θ), capital bold symbols X for matrices. Specifically we denote K as number of nodes, T as number of iterations, λ as iterative optimization step size, B as size of mini-batch

1.2 Separable stochastic quadratic function

This entire paper is concentrated on particular type of functions: separable stochastic quadratic real-valued function $\mathcal{L}$ parametrized by $\omega \in \mathbb{R}^d$. By term *separable* we mean that $\mathcal{L}$ can be represented as follows

$$\mathcal{L}(\omega) = \frac{1}{N}\sum_{i=1}^{N} \ell(\omega). \tag{1}$$

By term *quadratic* we mean that every ℓ_i is quadratic in its argument vector ω (so as $\mathcal{L}$). By term *stochastic* we mean that that every function ℓ_i is noised with some stochastic nature, i.e $\ell_i(\omega) = \ell(\omega) + \varepsilon_i^{\ell}(\omega)$, where $\ell(\omega)$ is some standard quadratic function and $\varepsilon_i^{\ell}(\omega)$ is some zero-mean finite-variance random function (i.e. it is a random variable somehow dependent on ω).

This choice of particular loss function type may seem unobvious. However, it can be easily explained. Consider linear regression problem (described in section 1.5) with input vectors $\boldsymbol{x}_i$, output scalars y_i and unknown parameter ω. The loss *prediction error* of this model would be equal to

$$\ell_i^{lr} = (y_i - \boldsymbol{x}_i^T\omega)^2 = \omega^T\left(\boldsymbol{x}_i\boldsymbol{x}_i^T\right)\omega - 2y_i\boldsymbol{x}_i^T\omega + y_i^2 = \dot{\omega}^T\mathbf{Z}_i\dot{\omega},$$

which clearly has the same form as ℓ_i: $\ell_i^{lr}(\omega) = \ell^{lr}(\omega) + \varepsilon_i^{lr}(\omega)$, where

$$\ell^{lr}(\omega) = \dot{\omega}^T \mathrm{E}_{x,\varepsilon}[\mathbf{Z}]\,\dot{\omega},$$
$$\varepsilon_i^{lr}(\omega) = \dot{\omega}^T\left(\mathbf{Z}_i - \mathrm{E}_{x,\varepsilon}[\mathbf{Z}]\right)\dot{\omega}.$$

1.3 Stochastic gradient descent

Stochastic gradient descent (SGD) became a very popular optimization technology in past decades, especially in machine learning and control areas by virtue of both its simplicity and effectivity. It was originally proposed in Davidon (1976) for least squares estimation and further developed particularly for online learning purposes Cun et al. (2004); Bottou (1998). SGD aimed to solve optimization problem of stochastic separable functions in following way. Assuming ℓ_i convexity and using $\min_\omega \ell = \mathrm{E}\min_\omega \ell_i$, at each iteration we randomly chose one ℓ_j among $\{\ell_i\}_1^N$ and shift current parameter (arg min) estimate opposite to gradient direction $\partial_\omega \ell_j$. Stochastic gradient descent pseudo code listed in 1.

Algorithm 1 SGD($\{\ell_1,\ldots,\ell_N\}, T, \lambda, \omega_0$)
```
for $t \leftarrow 1$ to $T$ do
    Draw $j \in \{1,\ldots,N\}$ uniformly at random
    $\omega_t \leftarrow \omega_{t-1} - \lambda\,\partial_\omega \ell_j(\omega_{t-1})$
end for
return $\omega_T$
```

Mini-batch stochastic gradient descent (mbSGD) algorithm use the same strategy, but at each iteration it randomly choose a subset J of size B of functions $\{\ell_i\}_1^N$ and average gradient among gradients of functions from J. Its pseudo code presented in listing 2.

Algorithm 2 mbSGD($\{\ell_1,\ldots,\ell_N\}, T, \lambda, \omega_0, B$)
```
for $t \leftarrow 1$ to $T$ do
    Draw $J \subset \{1,\ldots,N\}$ of size $|J| = B$ uniformly at random
    $\omega_t \leftarrow \omega_{t-1} - \lambda\frac{1}{B}\sum_{j\in J}\partial_\omega \ell_j(\omega_{t-1})$
end for
return $\omega_T$
```

1.4 Distributed stochastic gradient descent

In this subsection we rephrase an algorithm proposed in Dekel et al. (2012) our work is substantially based on. Algorithm pseudo code listed in 3. Worth noting that in Zinkevich et al. (2010)) authors assume that dataset size is so large that we can choose arbitrary N to provide enough observations for T iterations on K machines with batch size B. It means that we can first specify number of iterations T, batch size B and number of machines K and then choose appropriate N to met condition $T = \lfloor N/K/B \rfloor$. We will stick to this assumption later on.