On-line learning and stochastic gradient descent

Computational aspects of machine learning

Denys Sobchyshak
Department of Informatics
Technische Universität München
Email: denys.sobchyshak@tum.de

***Abstract*—Throughout the last dacade available data amount has grown ever rapidly, which posed a serious pressure onto existing machine learning algorithms. We will present overview of an approach dealing wth the problem via on-line learning methods, stressing its emergence within the large-scale data setting and showing some of the prallelization ideas as to surpass the inherent sequential nature of on-line optimizers, providing grounds for their scaling onto hundreds of cores in a distributed envornment.**

***Keywords*—On-line learning, stochastic gradient descent, distributed learning systems**

I. Introduction

Even though on-line learning algorithms date back to 1940 when they were first developed by Professor Bernard Widrow and Ted Hoff, his graduate student, under the name ADALINE, which essentially comprises stochastic gradient descent (SGD), their extensive usage and popularity was not as much advocated. With increasing size of analyzed data, more and more attention was drawn to on-line learning, where computational complexity of an algorithm becomes the limiting factor. In this paper we will present an overview of various gradient descent methods and reason in favour of SGD within the large-scale problem setting and in context of asymptotic analysis.

Furthermore, in the last dacade phenomena known as Big Data and Internet of Things have drawn even more attention to rapidly increasing rates of collected data. These phenomena and a result of industry scale data sets growing from Gigabytes to Petabytes of data storage are putting increasing pressure on data processing, where sequential algorithms are no longer sufficient. Under such restrictions we will present an overview of possible parallelization approaches of SGD and guide the reader through pressing problems of parallelization and their potential remedies. We will also take a closer look at work of practitioners within the industry and present some of the latests findings in parallelizing SGD to ensure its scalability to numerious processing units.

II. Online learning approach

Most of the work in theoretical machine learning has been focused on an off-line setting, where one uses a batch of data to produce a predictive model and then apply it to new data. However, this paper focuses on an on-line setting, where predictions are made sequentially based on all previously processed data, which gives rise to the name itself as we process observations one by one. In this setting we start by observing the first example x_1 while predicting its label y_1. After that we observe the true label of y_1 and also the next data point x_2. This process goes on untill we either reach the end of our data set or indefinitelly. Foremost attetion can be drawn to the expectation of improved prediction quality with increasing number of processed data.

Such supervised learning approach perfectly suits the case when data is available sequentially with time, for example, in predicting a stock price given its historic data and suplemented by availability of true price, as compared against the prediction of a learning system. Furthermore, as compared to off-line setting, on-line algorythm is expected to show a drastic decrease in memory usage and processing time untill the first prediction is made.

More formally, a mathematic definition of the learning problem is defined by finding a minimum of an expected risk:

$$E(f) = \int l(f(x), y) dP(z)$$

where function $f \in \mathcal{F}$ minimizes the averaged loss $Q(z, w) = l(f_w(x), y)$ over all data points z. Ultimatelly, we'd prefer to average over the unknown distribution $dP(z)$, however, in practicall setting we perform the calculation over a fixed number of examples $z_1, z_2, ...z_n$, which is referred to as minimization of an empirical risk:

$$E_n(f) = \frac{1}{n} \sum_{i=1}^{n} l(f(x_i), y_i)$$

Let us further consider how one could takle such minimization task with some of the convergence results discussed in [2].

A. Gradient Descent Methods

One of the most common approaches to tackle minimization of empirical risk function is by applying an iterative gradient descent method(GD) where each iteration updates optimization weights w based on the gradient of $E_n(f_w)$:

$$w_{t+1} = w_t - \gamma \frac{1}{n} \sum_{i=1}^{n} \nabla_w Q(z_i, w_t)$$

where γ is a learning rate. When the initial estimate of the optimum w_0 is sufficiently close this method reaches linear convergence described by a residual erro ρ, that is $-\log \rho \sim t$.

A better optimization algorithm can be composed by replasing the learning rate with a positve definite matrix Γ_t which approaches inverse of cost Hessian at optimum:

$$w_{t+1} = w_t - \Gamma_t \frac{1}{n} \sum_{i=1}^{n} \nabla_w Q(z_i, w_t)$$

which is also known as a 2nd order gradient descent method(2GD) and was proven to reach quadratic convergence $-\log\log\rho \sim t$.

B. Stochastic Gradient Descent Methodds

Stochastic setting of GD slightly simplifies the process, instead of computing the gradient of empricial loss function $E_n(f_w)$ exactly, one uses a single randomply picked data point z_t for its estimation:

$$w_{t+1} = w_t - \gamma \nabla_w Q(z_t, w_t)$$

In this simplification we assume that our optimization parameter behaves as a stochastic process $w_t, t = 1, ...,$ which greatly depends on the data sampling and introduces certain amount of noise. Utilization of stochastic approximation framework provided in [1], [2] fully addresses proofs of convergence including cases where loss function is not everywhere differentiable. Results of the aforementioned analysis show that speed of SGD convergence is limited by noisy approximation of the true gradient and is in fact optimal with learning rate $\gamma_t \sim t^{-1}$, at which point expectation of the residual error decereases similarly $E\rho \sim t^{-1}$.

Similar to 2GD one can multiply gradients by positve definite matrix approaching inverse of the Hessian:

$$w_{t+1} = w_t - \gamma_t \Gamma_t \nabla_w Q(z_t, w_t)$$

which yields the 2nd order stochastic gradient descent method. However, such change doesn't decrease the bias introduced by stochastic noise and retains the convergence rate $E\rho \sim t^{-1}$.

C. Sample SGD pseudocode

A simplistic pseudocode of SGD with inplace optimiation parameter update may look as follows:

```
for (x_t, y_t) in data_set:
    loss_fn = f(w, x_t, y_t)
    # compute gradient
    d_loss_fn_wrt_w = ...
    w -= gamma * d_loss_fn_wrt_w
    if <stopping condition is met>:
        return w
```

III. SGD in large-scale setting

Following results presented in [2] for asymptotic analysis of SGD as summarized in TABLE1, where excess error is defined as

$$\mathcal{E} = \mathcal{E}_{app} + \mathcal{E}_{est} + \mathcal{E}_{opt} \sim \mathcal{E}_{app} + (\frac{\log n}{n})^{\alpha} + \rho, \alpha \in [1/2, 1]$$

with $\mathcal{E}_{app}$ measuring how closly functions in our function space can approximate the optimal solution, $\mathcal{E}_{est}$ measuring the effect of minimizing empirical risk instead of the expected risk, $\mathcal{E}_{opp}$ measuring the impact of approximate optimization on expeted risk,

TABLE I
Asymptotic results for various optimization methods

	GD	2GD	SGD	2SGD
Time per iteration:	n	n	1	1
Iterations to accuracy ρ:	$\log\frac{1}{\rho}$	$\log\log\frac{1}{\rho}$	$\frac{1}{\rho}$	$\frac{1}{\rho}$
Time to accuracy ρ:	$n\log\frac{1}{\rho}$	$n\log\log\frac{1}{\rho}$	$\frac{1}{\rho}$	$\frac{1}{\rho}$
Time to excess error $\mathcal{E}$:	$\frac{1}{\mathcal{E}^{1/\alpha}}\log\frac{1}{\mathcal{E}}$	$\frac{1}{\mathcal{E}^{1/\alpha}}\log\frac{1}{\mathcal{E}}\log\log\frac{1}{\mathcal{E}}$	$\frac{1}{\mathcal{E}}$	$\frac{1}{\mathcal{E}}$

one can conclude that wilst SGD and 2SGD are worst optimization algorithms (as depicted in third row of TABLE1), they require less time to reach predefined expected risk (as in fourth row of TABLE1). In other words, in a large-scale setting where an algorithm is restricted by its computing time rather then the amount of data provided, stochastic gradient methods perform asymptotically better.

IV. Parallelization of SGD

Unlike batch GD, SGD has inherently sequential nature, which essentially limits how one can parallelize the algorithm. However, pressing industry demands have led researchers to find feasible ways of achieving comparable convergence rates, as compared against its sequential counterpart, medium to high scalability on multiple compute units, which not seldomly include both GPUs and CPUs, while keeping both computational complexity and data trasfer low. We will further describe several parallelization ideas and their pitfalls as compared to the method described in [3].

A. Distributed subgradient approach

This approach essentially distributes parallelization of gradient computation among several machines, which hold part of the data, and subsequent agregation of the result on a master machine. While showing great linear scalability relative to amount of data and log-linearrelative to number of computers, this approach suffers from excessive communication overhead which results from multiple passes through data by MapReduce. In addition, since several MapReduce iterations are required to ensure fault tolerance, this approach generally doesn't fit into a setting where computation units are not tightly coupled, as in multiple machines compared to multicore approach.

B. Distributed convex solver approach

This approach is comprised of a relatively simple idea of performing a minibatch optimization by means of breaking down a small subset of the data set into several mini-batches and then solving each and every problem on a separate machine for further agregation of the solutions and averaging obtaind values. Most notable part of this idea is that MapReduce needs to perform only one pass, which dramatically reduces overall communication between machines. Despite the fact that this approach significantly reduces variance relative to the sequential verion, the bias introduced by stochastic nature of the algorithm doesn't reduce at all, as compared to sequential version. Moreover, this approach requires a batch-solver to run on every single machine, thus making the whole algorithm rather computationaly complex. More importantly, analysis of given approach showed that the convergence of the method strongly depend on degree of strong convexity of regularization.

C. Distributed SGD approach

A balance between MapReduce communication overhead and valid asymptotic analysis can be achied by modyfing the distributed convex solver approach to incorporate a SGD minimizer. More precicely, each processor would carry out a SGD on the set of loss functions $Q_i(w)$ with a fixed learning rate γ for T steps as described in Algorithm 1.

```
Algorithm 1: $SGD(Q_1, \dots, Q_m, T, \gamma, w_0)$
for $t = 1$ to $T$ do
  Draw $j \in (1 \dots m)$ uniformly at random.
  $w_{t+1} = w_t - \gamma * dQ_j(w_t)$
end for
return $w_T$.
```

To agregate computed parameters one would use master routine. Overall pseudocode looks as presented in Algorithm 2.

```
Algorithm 2: SimuParallelSGD(Examples: $Q_1, \dots, Q_m$, $\gamma$, Machines: $k$)
Define $T = [m/k]$
Randomly partition the examples, giving $T$ examples to each machine
for all $i \in 1 \dots, k$ parallel do
  Randomly shuffle the data on machine $i$
  Initialize $w_{i,0} = 0$
  for all $t \in 1, \dots, T$: do
    Gether the $t$th example on the $i$th machine, $Q_{i,t}$
    $w_{i,t+1} = w_{i,t} - \gamma d_w Q_i(w_{i,t})$
    end for
  end for
Agregate from all computers $v = \frac{1}{k} \sum_{i=1}^{k} w_{i,t+1}$ and return $v$
```

While this approach in a nutshel is rather simple, its mathematical analysis is nontrivial and can be found in original paper [3].

D. Empirical results of distributed SGD approach

To demonstrate scalability potential of the algorithm we can take a look at experimental results obtained in [3] where number of training instances per machine was compared to a relative RMSE on the test set as executed on 1, 10 and 100 machines as in Fig.1 and also to a normalized objective function as in Fig.2. All tests were performed on a $\sim$3M examples databse of emails with $\sim$785M features in the whole data set, as resulted after heshing.

It is worth noticing that scaling of 1 to 10 machines gives much better improvement as the one from 10 to 100 machines, which signifies that the proposed algorithm scales poorly to a highly distributed scenario.

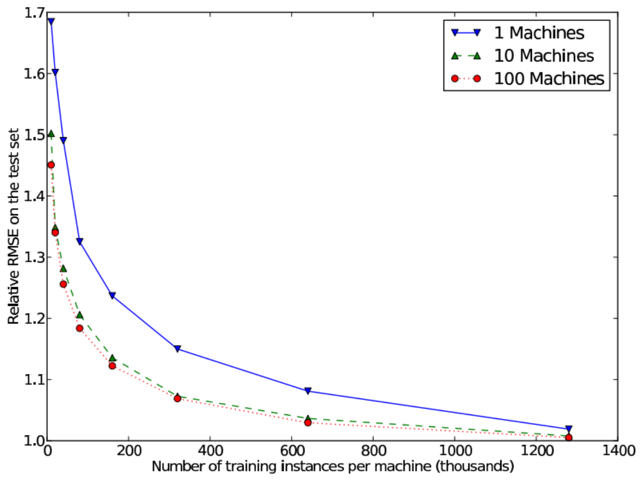

Fig. 1. Relative Test-RMSE with $\gamma = 1e^{-3}$

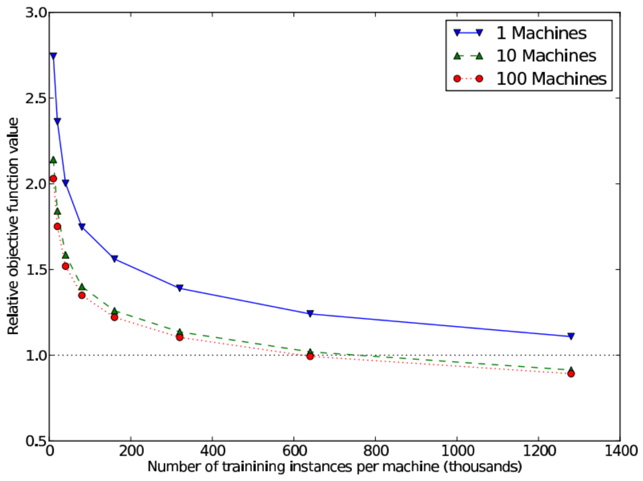

Fig. 2. Relative train error using Huber loss $\gamma = 1e^{-3}$

V. Asynchronous SGD

While provinding good results, the parallization approach described in the previous section has several drawbacks. Namely, it suffers from parameter locking, which is excersised when parameters are being averaged, which prevents further processing of data by slave SGD machines. Moreover, even though the use of MapReduce in this setting does not result in high communication costs, in general it is recommended to avoid its usage in a large-scale numerically intensive applications, due to its inability to coop with iterative nature of solvers and also due to its bandwidth overhead, which is resulted from fault tolerance redundacies. For example, one can experience a potential drop from 12GB/s throughput in a multicore shared memory to only tens of MB/s. Thus we advocate two approaches which do not use MapReduce and perform asynchronous parameter updates as follows.

A. Hogwild!

In order to eliminate the overhead caused by parameter variable locking, presented algorithm encorporates a simple idea of using high throughput memory shared across all compute units where optimization parameter w is to be stored. Any processor can update the parameter at any given time. Even though such approach might seem excessively prone to parameter overwrites, it works extremely well in case where data access is sparce, meaning that each SGD modifies only a small part of the optimization parameter. Particularly, when our goal is to minimize a loss function

$$f(w) = \sum_{e \in E} f_e(w_e)$$

where e denotes a small subset of $1, \dots, n$ and w denotes values of parameter vector w indexed by e, the sparcity of loss function means that $|E|$ and n are very large while each individual f_e influences only few values of the whole parameter w. This consept is illustrated in more detail with precise mathematical examples of particular cost functions in [4].

Hogwild! implementation running on each processor would look as in Algorithm 3.

Algorithm 3: Hogwild! update for individual processors

```
loop    Sample $e$ uniformly at random from $E$
    Read current state $w_e$ and evaluate $G_e(x)$
    for $v \in e$ do $w_v = w_v - \gamma b_v^T G_e(w)$
end loop
```

where $G = (V, E)$ is a hypergraph induced by $f(w)$ whose nodes are the individual components of w and where each subvector w_e induces an edge in the graph $e \in E$ and b_v is equal to 1 on the vth component and 0 otherwise. It is important to note that the processor modifies only values in e leaving all others untouched. Such approach does not require a locking mechanism on most modern hardware, such as GPUs or general purpose multicore CPUs.

Analytical results for convergence of the algorithm are rather non-trivial and can be found with exhaustive explanations in [4]. Important to note, however, that the algorithm converges in nearly the same number of iterations ans its sequential counterpart and provided experimental results which outperformed theoretical analysis. In terms of scalability it reaches near linear speedup. Some of the experimental results as compared to almost idential rooundrobin implementation with exception of gradient updates can be found in Fig.3. Those results present a comparison of wall clock time while parallelized over 10 cores, both algorithms were implemented in c++ and were ran on the same hardware setup with a dual Xeon X650 CPUs (6 cores each x 2 hyperthreading) and 24GB of RAM.

B. Downpour SGD

Unlike previous examples, this parallelization approach was used to train a deep neural netowrk with image recognition research project [5] and was developed as part of DistBelief

					HOGWILD!			ROUND ROBIN		
type	data set	size (GB)	ρ	Δ	time (s)	train error	test error	time (s)	train error	test error
SVM	RCV1	0.9	0.44	1.0	9.5	0.297	0.339	61.8	0.297	0.339
MC	Netflix	1.5	2.5e-3	2.3e-3	301.0	0.754	0.928	2569.1	0.754	0.927
	KDD	3.9	3.0e-3	1.8e-3	877.5	19.5	22.6	7139.0	19.5	22.6
	Jumbo	30	2.6e-7	1.4e-7	9453.5	0.031	0.013	**N/A**	**N/A**	**N/A**
Cuts	DBLife	3e-3	8.6e-3	4.3e-3	230.0	10.6	**N/A**	413.5	10.5	**N/A**
	Abdomen	18	9.2e-4	9.2e-4	1181.4	3.99	**N/A**	7467.25	3.99	**N/A**

Fig. 3. Comparison of wall clock for various data sets and loss functions for Hogwild! and Roundrobin implementations

framework at Google. However, the ideas used and results obtained are still worth mentioning as they give birth to potential further work.

In a nutshell approach is as follows: the training data is devided into a number of subsets and a copy of the model is ran on each of those subsets. This approach leverages the idea of Hogwild! asynchronous parameter updates in the form of centralized parameter server which is connected to 10 model replicas each holding 1/10 of the parameter w. This is schematically presented on Fig.4. Thus, after the model is ran on the assigned subset of training data, computed gradients are communicated to the central parameter server.

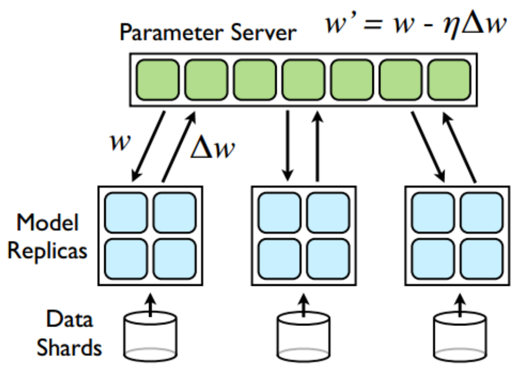

Fig. 4. Model replicas asynchronously fetch parameters w and push gradients ∇w to the parameter server

Downpour SGD approach is more robust to machines failures than synchronous parallelized SGD, because in case of machine failure in synchronous SGD the entire training process is bound to be delayed. However, for asynchronous SGD within a model replica fails, the other model replicas continue to operate and process their updates via parameter server.

In addition, Downpour SGD uses AdaGrad adaptive learning rates which increase robustness of the whole implementation and tackle problems of stability within deep neural network context. Corresponding learning rate is as follows:

$$\nu_{i,K} = \gamma / \sqrt{\sum_{j=1}^{K} \nabla w_{i,j}^2}$$

One can notice that the learning rates are computed from the summed squared gradients of each parameter and thus are easily implemented locally within each shard.

Some of the experimetal results of algorithm scalability can be seen on Fig.5.

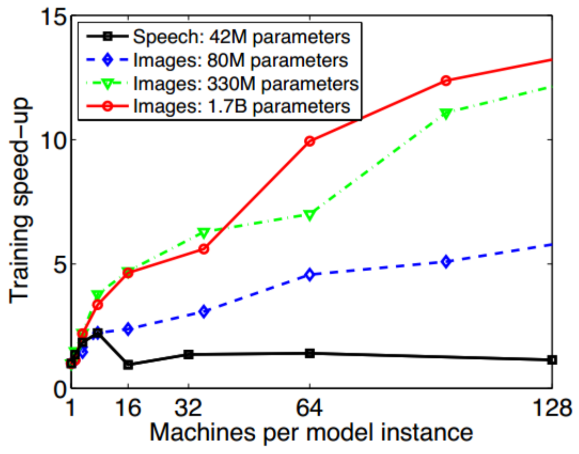

Fig. 5. Training speed-up for four different deep networks as a function of machines allocated to a single DistBelief model instance

VI. Conclusion and further work

Within this paper we have presented and justified the rationale behind using on-line learning algorithms for large-scale problems, we have pointed out advantages and disadvantages of various parallelization approaches and provided some experimental results to supplement our arguments. Even though the reader was guided to conclude that a specific parallelization idea dominates the realm, all of the presented schemes have their drawbacks. Supplying improvement of the latter always comes with a cost and any of the parallelization methods might turn to be an optimal choise for a specific problem setting.

However, in the ultimate scenario where amount of data is assumed to be infinite, as in processing of real-time streaming data, we conclude that the latter approach, namely parallelized asynchronous stochastic gradient descent with adaptive learning rates, should show the best results. This topic presents itself as an intimidating research prospect and is encouraged to be studied separately.

References

[1] L. Bottou, “Online algorithms and stochastic approximations,” in *Online Learning and Neural Networks*, D. Saad, Ed. Cambridge, UK: Cambridge University Press, 1998, revised, oct 2012. [Online]. Available: http://leon.bottou.org/papers/bottou-98x

[2] ——, “Large-scale machine learning with stochastic gradient descent,” in *Proceedings of the 19th International Conference on Computational Statistics (COMPSTAT’2010)*, Y. Lechevallier and G. Saporta, Eds. Paris, France: Springer, 2010, pp. 177–187. [Online]. Available: http://leon.bottou.org/papers/bottou-2010

[3] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola, “Parallelized stochastic gradient descent,” pp. 2595–2603, 2010. [Online]. Available: http://papers.nips.cc/paper/4006-parallelized-stochastic-gradient-descent.pdf

[4] B. Recht, C. Re, S. Wright, and F. Niu, “Hogwild: A lock-free approach to parallelizing stochastic gradient descent,” in *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, Eds. Curran Associates, Inc., 2011, pp. 693–701. [Online]. Available: http://papers.nips.cc/paper/4390-hogwild-a-lock-free-approach-to-parallelizing-stochastic-gradient-descent.pdf

[5] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. aurelio Ranzato, A. Senior, P. Tucker, K. Yang, Q. V. Le, and A. Y. Ng, “Large scale distributed deep networks,” in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1223–1231. [Online]. Available: http://papers.nips.cc/paper/4687-large-scale-distributed-deep-networks.pdf