Weak Consistency and Stochastic Environments

Harmonization of Replicated Machine Learning Models

Tobias Herb
Technical University Berlin
tobias.herb@tu-berlin.de

Tim Jungnickel
Technical University Berlin
tim.jungnickel@tu-berlin.de

Christoph Alt
Technical University Berlin
c.alt@campus.tu-berlin.de

ABSTRACT

Many machine learning (ML) models are of a stochastic nature. We aim to combine the principles of weak consistency with large scale distributed machine learning. We see interesting opportunities in this domain in (1) perceiving parallel ML algorithms based on model replication as a "collaborative task" where local progress on models is instantaneously exchanged and by (2) making this exchange more efficient by exploiting the underlying stochastic nature. Based on this motivation, we extend the notion of consistency for replicated objects with intrinsic stochastic structure and introduce *harmonization* as the reconciliation principle to enable efficient consistency maintenance of these objects. We present as a concrete application the *harmonization of replicated ML models*.

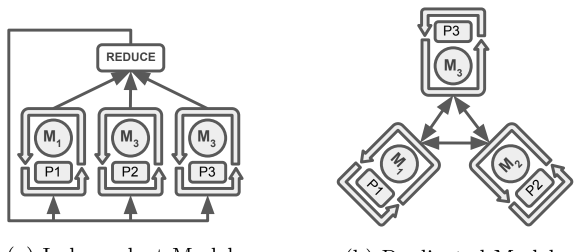
Keywords

Distributed systems, weak consistency, large-scale machine learning, stochastic gradient descent

1. INTRODUCTION

Machine learning (ML) analytics have become the core of many businesses and emerging fields. ML is about inferring a function from training data. The learning algorithm analyzes the training data and produces an inferred function, the so-called model, used to correctly determine unseen data.

However, the speedup in computational power has hit a barrier and in recent years the growth of the amount of data to be analyzed has been vastly larger than the speedup in computation. The capabilities of large scale ML are rather limited by computing time than sample size. The computational complexity of learning algorithms become the critical limiting factor when one envisions very large data sets and motivate the need for efficient techniques in the area of distributed machine learning that fully exploit the inherent properties of the applied algorithms. Stochastic gradient descent (SGD) is an algorithm to perform optimization which



(a) Independent Models (b) Replicated Models
Figure 1: Distributed Training of ML Models.

is widely used for many large scale machine learning algorithms (e.g. linear/logistic regression, linear SVM, matrix factorization, artificial neural networks, ...) offering great performance in large-scale settings [9]. A current challenge in distributed ML is the efficient parallelization of the learning process. Common parallelization approaches of linear models follow the traditional MapReduce schema [4] as visualized in Fig. 1a. Independent (M1 - M3) model instances are trained in an iterative fashion on different data partitions (P1 - P3). After each training iteration, often referred to as *epoch*, a global reduce operation then combines[1] all the independent models. The resulting model is then propagated back. This procedure of independent learning and global model mixing repeats until a given termination criteria[2] is satisfied [3]. We propose a novel approach by combining the principles of weak consistency with the inherent stochastic nature of ML models. For this we change communication pattern by perceiving parallel ML algorithms based on model replication as an âĂİcollaborative taskâĂİ where local progress on replicated model instances is exchanged instantaneously, improving the overall convergence of the learning process, because updates are sooner visible and integrated into parallel model instances [8], as depicted in Fig. 1b. To compensate for the associated communication overhead, we exploit the underlying stochastic nature of ML models by propagating only significant updates that contribute to the global convergence. To apply this concept on a broader scale, we introduce the concept of ***harmonization*** as an reconciliation principle for replicated objects with intrinsic stochastic structure.

2. BACKGROUND

PaPOC'16, April 18-21 2016, London, United Kingdom
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4296-4/16/04...$15.00
DOI: http://dx.doi.org/10.1145/2911151.2911161

[1]in case of linear models, this is accomplished by parameter averaging
[2]e.g. combined model is converged or a fixed number of epochs is reached

2.1 Iterative Convergent Algorithms (ICAs)

Most ML algorithms can be described abstractly as an iterative process of continuous model refinement until a provided termination criteria is satisfied:

```
do
        $\delta \leftarrow \Delta(\Theta^t, \gamma)$
        $\Theta^{t+1} \leftarrow u(\Theta^t, \delta)$
until $T(\Theta^{t+1})$
```

The expression $\Delta(\Theta^t, \gamma)$ computes the *update-step* δ dependent on the dataset γ and the current state of the model Θ^t. The step δ is applied via the update function u on the current state Θ^t to generate the refined state Θ^{t+1}. The steps are repeated until a predefined convergence criteria $T(\Theta^{t+1})$ is fulfilled.

2.2 Data- and Model-Parallelism

Distributed parallelization is made possible by replicating the computation of the *update-step* Δ. Usually, a distinction between two common variants is made:

Data-Parallelism: Update-steps are computed in parallel on assigned fractions of the **partitioned dataset** γ and either applied on a **shared** or **replicated model** Θ.

Model-Parallelism: Update-steps are computed in parallel on a **replicated dataset** γ and locally applied on an assigned fraction of the **partitioned model** Θ.

Our approach is based on the *Data-Parallelism* variant.

2.3 Distributed Execution of ICAs

We structure the execution of an epoch within a *parallel* ICA-instance in three logical phases:

(A) **Computation:** Compute the *refined model* locally.

(B) **Communication:** Perform communication to make the *local progress* visible to the other parallel ICA-instances.

(C) **Coordination:** Perform optional global coordination logic among the parallel ICA-instances to preserve *ML algorithm specific properties* [2].

The semantics of phases (A) and (B) do not require a happened-before ordering. Hence an *overlapping computation and communication phase* is allowed.

3. CONSISTENCY IN STOCHASTIC ENVIRONMENTS

Inspired by the *continuous consistency model* of [6], we loosely define *data consistency* in stochastic environments as replicated entities with intrinsic stochastic structure that are *bounded* tolerant to deviations in *numerical values* and *replica staleness* among each other. Based on this, we define ***harmonization*** as the reconciliation principle to maintain consistency of the replicated entities within application specific bounds. The exact specification and implementation of these bounds are highly application dependent [5]. Being aware of these boundaries and carefully exploring them can lead to efficient consistency maintenance solutions. For instance, stochastic properties can be exploited by reducing *consistency preserving* communication to a legal (i.e. correctness preserving) minimum. A further way that leverages the stochastic properties of replicated objects is given in the next section. In this scenario, we additionally claim to increase the global convergence for many distributed machine learning algorithms by including the *staleness deviation* in the replica reconciliation.

4. HARMONIZATION OF REPLICATED ML MODELS

We introduce harmonized replicated ML models as a concrete realization of our harmonization concept. We focus on linear models that assume the output y can be expressed as a linear combination with the input attributes (x_1, x_2, ...). Both, the output and input attributes are expected to be numeric. The parameters of the underlying model represent the coefficients of the linear combination.

The fundamental idea is based on: (1) the overlapping of computation- and communication phase to enable instantaneous propagation of updated parameters, (2) selection of relevant parameter updates which contribute to global convergence and (3) the asynchronous integration of updated parameters, based on epoch-weighted averaging inspired by [1]. We claim that the immediate exchange of updated parameters increases the global convergence, because learning progress is available to all parallel instances as soon as possible. Secondly, the immediate and continuous propagation of refined parameters leads to a constant utilization of the network resource compared to communications bursts at the end of epochs. Our approach is based on data-parallel machine learning, which implies a replicated model and a data set partition assigned to each distributed ICA-instance. High delayed update propagation lead to a decrease of the convergence rate [8]. To keep propagation latencies as low as possible, a high network coupling degree among the ICA-instances is required,

4.1 Communication

To enable the immediate propagation of updated model parameters among parallel ICA-instances, we extend the definition of an ICA in section 2.1 by weaving in send/receive-primitives. The communication primitives are executed asynchronously to avoid interruption of the ICA execution. The

send-primitive is triggered after the local computation in the *Computation and Communication Phase* and consists of: (1) buffering of updated parameters to a reasonable size, depending on network properties and current utilization.
(2) buffer transmission to the other ICA-instances.

The **receive-primitive** provides remote updates for further processing.

4.2 Harmonization Functions

The harmonization functions are tightly coupled to the send/receive primitives and can be seen as an intermediate layer among the connected ICA-instances. The concrete harmonization mechanic is realized via: (1) a filter function applied on sender side to propagate only significant updates and (2) a merge function applied at the corresponding receiver side to integrate remote updates.

ML Model: We refine the notion of a ML model by giving Θ internal structure via an ordered set of parameters. We access the model parameters by the position e.g., $\Theta(n-1)$ refers to the last parameter in Θ if the model covers n parameters.

Filter-Function: *The filter decides whether an updated parameter $\Theta^{t+1}(p)$ is handed over to the send-primitive or not*:

$$F_\epsilon(\Theta^t(p), \Theta^{t+1}(p)) \quad \triangleq \quad \left|\frac{\Theta^{t+1}(p) - \Theta^t(p)}{\Theta^t(p)}\right| \quad \geq \quad \epsilon$$

This filter implementation rates a parameter as significant for propagation, if its relative change, computed by building the difference between the new value $\Theta^{t+1}(p)$ and the old value $\Theta^t(p)$ divided[3] by the old value, is greater than or equal to a user provided threshold ϵ.

The filter threshold ϵ relates to a individual parameter update. Many updates on a parameter, each for itself falling below the threshold, may in aggregate become significant over time. A simple way to determine these *significant parameter aggregates* is to maintain for each parameter an additional *update-aggregate*, keeping the delta since last propagation. If the update-aggregate exceeds the threshold is the parameter propagated and the update-aggregate invalidated. Remote updates of this parameter results also in a invalidation. This simple extension is limited by the number of parameters, because the additional information lead to a doubling of model sizes. In case of large models can probabilistic data structures be applied, e.g. a variation of the *count-min sketch* [10]. This data structure enables a sub linear growth of model sizes with increasing number of parameters by approximating updates-aggregates for the most frequent updated parameters.

Merge-Function: *The merge function integrates a remote parameter $\Theta^{t_R}(p)$ into the local parameter $\Theta^{t_L}(p)$*:

$$M_\lambda(\Theta^{t_R}(p), \Theta^{t_L}(p)) \triangleq \frac{(t_R - t_L)\,\lambda\,\Theta^{t_R}(p) + \Theta^{t_L}(p)}{2} \rightarrow \Theta^{t_L}(p)$$

This merge implementation performs a weighted averaging between a remote- and the corresponding local parameter. The weighting is applied on the remote parameter by computing first the "epoch-distance" between the remote and local parameter $(t_R - t_L)$ and then regulating the impact of this distance by multiplying the user supplied factor with λ to define the "importance" of the remote parameter. Finally, the arithmetical mean is formed over the *weighted remote-* and the local parameter.

5. FUTURE WORK

We have implemented the presented ML model harmonization in our new distributed ML framework and are currently working on concrete experiments with very large data sets to examine the convergence characteristics of our approach.

It would be interesting from a theoretical machine learning perspective to examine how the user supplied *harmonization hyper-parameters* ϵ, λ could be inferred by a system. For example based on clever sub-sampling of the training data.

With regards to the community that addresses the problem of distributed data consistency, it might be interesting to invest in a theoretical framework where the notion of consistency in stochastic environments is investigated to discover general properties inherent in this specific problem domain.

[3]If $\Theta^t(p) = 0$ we take the absolute change $\left|\Theta^{t+1}(p) - \Theta^t(p)\right|$

6. REFERENCES

[1] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July 2011.

[2] G. Gibson and N. Zeldovich, editors. *2014 USENIX Annual Technical Conference, USENIX ATC '14, Philadelphia, PA, USA, June 19-20, 2014*. USENIX Association, 2014.

[3] R. McDonald, K. Hall, and G. Mann. Distributed training strategies for the structured perceptron. HLT '10, pages 456–464, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[4] X. Meng, J. K. Bradley, B. Yavuz, E. R. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. B. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. J. Franklin, R. Zadeh, M. Zaharia, and A. Talwalkar. Mllib: Machine learning in apache spark. *CoRR*, abs/1505.06807, 2015.

[5] A. S. Tanenbaum and M. v. Steen. *Distributed Systems: Principles and Paradigms (2Nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006.

[6] H. Yu and A. Vahdat. Design and evaluation of a conit-based continuous consistency model for replicated services. *ACM Trans. Comput. Syst.*, 20(3):239–282, Aug. 2002.

[7] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola. Parallelized stochastic gradient descent. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 2595–2603. Curran Associates, Inc., 2010.

[8] Wei Dai, Abhimanu Kumar, Jinliang Wei, Qirong Ho, Garth A. Gibson, and Eric P. Xing. High-performance distributed ML at scale through parameter server consistency models. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pages 79–87, 2015.

[9] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In Yves Lechevallier and Gilbert Saporta, editors, *Proceedings of the 19th International Conference on Computational Statistics (COMPSTAT'2010)*, pages 177–187, Paris, France, August 2010. Springer.

[10] Graham Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, April 2005.