Sources of Variability in Large-scale Machine Learning Systems

Damien Lefortier[1,2], **Anthony Truchet**[1], **and Maarten de Rijke**[2]
[1] Criteo, Inc.
[2] University of Amsterdam
{d.lefortier, a.truchet}@criteo.com, derijke@uva.nl

Abstract

We investigate sources of variability of a state-of-the-art distributed machine learning system for learning click and conversion prediction models for display advertising. We focus on three main sources of variability: asynchronous updates in the learning algorithm, downsampling of the data, and the non-deterministic order of examples received by each learning instance. We observe that some sources of variability can lead to significant differences between the models obtained and cause issues for, e.g., regression testing, debugging, and offline evaluation. We present effective solutions to stabilize the system and remove these sources of variability, thus fully solving the issues related to regression testing and to debugging. Moreover, we discuss potential limitations of this stabilization for drawing conclusions, in which case we may want to take the variability produced by the machine learning system into account in confidence intervals.

1 Introduction

Machine learning (ML) is being used at a broad range of technology companies, for solving challenging tasks such as, e.g., recommendation of content or products [1], click prediction for display advertising [2, 3], and for optimizing search engines [4]. Increasingly, machine learning is being used on very large datasets [5]. To learn models for predicting clicks or conversions at Criteo, we rely on a large amount of historical data [6, 7], but we also need to frequently refresh our models to learn from the latest data and maximize performance [3]. Below, we first describe our Hadoop-based terascale implementation of stochastic gradient descent (SGD) for sparse problems, a widely used algorithm for large-scale machine learning [8, 9], and of limited-memory BFGS [10] (L-BFGS), which we use sequentially for learning our models [6]. Our implementation of SGD extends HOGWILD! style SGD [11] and combines it with Hadoop AllReduce to provide distribution through parameter averaging. We use AllReduce for distributing L-BFGS through gradient aggregation [12].

We focus on the following questions. *What are the sources of variability in our system? To which extent do they impact its performance, and if the impact is significant, how should we deal with it?* Large-scale ML is often asynchronous in order to learn from very large datasets in a reasonable amount of time [11, 13, 14]: a first source of variability is due to asynchronous updates at the learning-level [15]. A second source arises at the system-level and comes from the fact that a state-of-the-art distributed ML

system typically does not ensure that the samples (or simply their ordering) processed by each machine remains the same from one run to another; this can even impact the model learned by synchronous systems that use SGD [12, 6].

Variability is a real concern for large-scale ML. First, the system is less reproducible, which makes engineering aspects such as regression testing or debugging even more challenging than usual. Indeed, when, for example, moving the learning system used in production to a new cluster, it is critical (to say the least) to be able to ensure that the system still behaves as before (as many things could go wrong when during such a migration that could lead to small changes in the models learned). But if the system is not exactly reproducible, then it is very challenging to move fast and with confidence in such cases. Variability is critical to large-scale machine learning systems as even small degradations in the machine learning code can lead to slightly less efficient models but large financial losses. Finally, variability also makes it harder to draw conclusions when doing offline experiments (does this feature bring an improvement beyond model and metric noise?). As one is typically looking for small increments of a few percentage points [3, 16], variability is a real concern when improving the models, especially for large-scale ML systems as they increase with the size of the dataset (and are insignificant on a small dataset).

Below, we show that the *median relative difference* of normalized log loss [3] across all sources of variability can reach 0.066%, which is significant considering that the model we use for running our experiment (a click prediction model) is only one of the models we use for predicting the value of a display opportunity.[1] We present effective solutions to stabilize the system and remove these sources of variability, thus fully solving the issues related to regression testing and to debugging. Moreover, we discuss potential limitations of this stabilization for drawing conclusions, in which case we may want to take the variability produced by the machine learning system into account in confidence intervals.

2 Large-scale machine learning with SGD and L-BFGS

We consider an effective terascale learning system using stochastic gradient descent (SGD) [8, 9] followed by limited-memory BFGS (L-BFGS) [6]. We refer to [12] for details and only briefly present a case study of our implementation of SGD for large-scale machine learning on Hadoop, both multi-threaded and distributed (not detailed in [12]). We consider a version of SGD well adapted to sparse problems, where most features are not present for any given example thus making the input dataset sparse, and which are quite common [17, 9]. Indeed, as an example, in [17] tens of or hundreds of *active* (present) features per example are reported on average, but 100 billion unique features overall are typical use cases for Sibyl at Google. The recent click log data set open sourced by Criteo [7] is also highly sparse with millions of unique features and only hundreds of active features per examples.

Using a regularization term on sparse data makes the update dense, i.e., all weights are updated for each example, which prevents SGD from fully benefiting from the sparsity — thus making the learning much slower with than without the regularization term. A well-known trick to solve this issue is to do the updates related to the regularization term in a lazy fashion. That is when need, so for active weights only (taking in account the number of updates missed), thus making the update fully sparse [9]; see below. Another well-known trick that builds upon the sparsity is to use HOGWILD!, i.e., lock-free multi-threaded SGD, to speed up the learning [11]. Here, we combine these two approaches on each machine and obtain Algorithm 1, where the w_i are our set of parameters to fit, η is the learning rate, g_i is the gradient of the loss w.r.t. w_i and λ is the regularization parameter. We use L_2 regularization for simplicity and without loss of generality.

[1] As we predict the expected sales amount generated by a user following a display using different sub-models, these variabilities will add up across all models (see Section 3).

Algorithm 1 Lock-free lazy SGD algorithm with L2 regularization running on each machine

nbUpdates $= 0$ ▷ Number of data points updates so far
lastTimeUpdated $= \{0\}$ ▷ Array initialized with zeroes
for each epoch **do**
  parallel for each data point e **do**
    for each *active* weight w_i of e **do**
      $w_i = w_i - \eta(g_i + \lambda w_i * (\text{nbUpdates} - \text{lastTimeUpdated}[w_i]))$
      lastTimeUpdated$[w_i] = $ nbUpdates
    end for
    atomicIncrement(nbUpdates)
  end parallel for
end for

This algorithm can then be distributed over multiple machines [13, 14].[2] Here, we distribute this algorithm using synchronous parameter averaging [18] using Hadoop AllReduce, because it scales well up to the terascale [12, 6] and is easy to implement. We also use AllReduce for distributing L-BFGS through gradient aggregation as usually done [12, 6]. More specifically, our implementation has two stages: a *mapper* step for pre-processing the data (for sampling examples or applying feature transforms) and a *reducer* step for learning using SGD and L-BFGS, which uses Hadoop AllReduce for synchronization between machines.

3 Sources of variability

In this section, we investigate three sources of variability of the system described in Section 2 and how they impact the model obtained. In all our experiments, we use internal data from the production click logs at Criteo from September 2015. We consider a data set of 2.6 billion examples with binary labels (click vs. no-click) and hundreds of millions of unique attribute values. To quantify these sources of variability, we use a click prediction model trained using logistic regression with our system.

We use the *Normalized Log Loss* (NLL) as our offline metric (defined formally below), which shows the relative improvement in *Log Loss* of the model to be evaluated versus a baseline predictor, in our case the average empirical click-through rate (CTR) of the dataset. We denote y_i the binary outcome variable indicating if there was a click or not and p_i the estimated probability of click. This metric was also used in [3] where they call it the *Relative Information Gain* (RIG). For each source of variability, we report the *median relative difference* of this normalized log loss[3] computed on the test set over 15 runs of our system. Each run takes a few hours to complete and we therefore limit ourselves to 15 runs per source of variability in our experiments below.

$$\textbf{Log Loss (LL)} \qquad -\frac{1}{N}\sum_i (y_i \log(p_i) + (1-y_i)\log(1-p_i))$$

$$\textbf{Normalized Log Loss (NLL)} \qquad 1 - \frac{LL_{Model}}{LL_{AverageEmpiricalCTR}}$$

All experiments are run using 40 reducer instances for distributing the learning and using L-BFGS, initialized with SGD, until the optimizer makes little progress. We use the hashing trick [19] to reduce the

[2] See this blog post by John Langford for details: `http://hunch.net/?p=151364`.

[3] This is the median difference between all pairs of runs w.r.t. the mean normalized log loss.

dimensionality of our dataset and to thus reduce the number of parameters to fit from hundreds of millions as indicated above to millions.

Variability from asynchronous updates Algorithm 1 relies on asynchronous updates, which provide a significant speed-up in terms of learning time without significantly degrading the model [11] and are therefore highly beneficial. However, they lead to non-deterministic overwrites, which introduce some variability in the learned model. The median relative difference of NLL for this source of variability reaches 0.048% and 0.052% when training our model on 602M and 2.6B examples, respectively (Table 1, first line of results). We can use one thread to fully remove this variability. One can also get rid of this source of variability by using a synchronous distributed variant of SGD [15], although this would mean a different architecture for our system. We could use L-BFGS only, but it would have to be warm started in another way than with SGD, as L-BFGS is very slow to converge far away from the optimum [12]. All experiments, except those for the first line (of results) have been run with one thread.

Variability from sampling the data In many applications of large-scale machine learning, there is a fundamental class imbalance in the dataset, e.g., in our context, around 200 displayed ads for 1 click. Down-sampling of the dominant class(es) is typically used to solve this problem and ease learning [3]. In this case, the effective dataset received by the learning machines after down-sampling is an arbitrary, implementation-dependent, subset of the dataset. This introduces a fundamental variability in the models learned by the system. In our experiments, we down-sample a large portion of the negative (non-clicked) examples. We use a *deterministic sampling* scheme: for each example, some unique features (e.g., a display ID) are hashed with a salt. This allows us to specifically quantify the variability introduced here by training a model multiple times with different salts. Table 1 shows that the median relative difference of NLL for this source of variability reaches 0.030% and 0.053% when training our model on 602M and 2.6B examples, respectively (third result line in Table 1), which is about the same as with asynchronous updates. Fixing the salt allows us to remove this source of variability, which is instrumental for any reliable regression testing or for quantifying other sources of variability. All experiments, except those for the third result line have been run with the same salt.

Variability from the shuffling of the input At the system-level, the very large datasets we use require distributed storage and learning. We use Hadoop MapReduce, but the following side-effects are generalizable to other distribution frameworks. The mapper instances load and merge the input log files from one of their (HDFS) replicas in a non-deterministic and not controllable way. Then, the log lines are shuffled between mapper and reducer instances according to some *shuffling key*. A default strategy to even the load between reducer instances is to interleave data from all mapper instances to each reducer. This makes both the partitioning of the log lines between the reducer instances and the order in which they are seen by each reducer highly variable. Our system is sensitive to this source of variability with a median relative difference of NLL for this source of variability of 0.049% and 0.056% when training our model on 602M and 2.6B examples, respectively (Table 1, second result line), which is very similar to the results obtained for the two sources of variability above. Note that, as with the two previous sources of variability, the median relative difference of NLL increases slightly with the dataset size. We address this source of variability by using a *deterministic shuffling* scheme: we build the shuffling key for each log line using some unique features. This ensures that, for a given number of reducer instances, and a sampling salt (see above), one reducer will always receive the same effective data in the same order. All experiments except those for the second row of Table 1 use this setup.

Addressing variability Table 1 (fourth result line) shows that, combined, our solutions, as described above, allow us to stabilize the learning system; we do not observe any variability after the proposed changes. In other experiments using the full model (optimizing the expected sales amount generated by a user following a display) [16], we observe that the median relative difference of Utility [20] on the full

Source of variability	Median relative difference of NLL	
Number of examples	602M	2.6B
Variability from async. updates (12 threads)	0.048 %	0.052 %
Variability from shuffling	0.049 %	0.056 %
Variability from sampling	0.030 %	0.053 %
Variability cumulated from all sources	**0.058** %	**0.066** %
Variability with all proposed changes	ϵ	ϵ

Table 1: Impact of the different sources of variability: *median relative difference* of the normalized log loss (NLL) obtained on the test set (of 103M examples) between 15 runs of our system, where $\epsilon < 10^{-9}$ is an insignificant value.

model can reach large values (closer to 1%) across all variabilities before our proposed changes [16]. Our changes allow us to completely stabilize the full model as well. (These experiments are omitted due to space constraints.)

4 Conclusion

We investigated three sources of variability of a state-of-the-art large scale ML system related to concurrency, down-sampling and shuffling.

We observed that all sources of variability yield roughly similar median differences of NLL according to our experiments ($\approx 0.05\%$). The first source of variability takes its root in a trade-off between efficiency and determinism of concurrent execution. The second one is fundamental: it can only be arbitrarily frozen. For the third source, which is accidental to the distribution framework used, we proposed a specific solution. We showed that, together, our solutions fully stabilize the system, thus solving the issues related to regression testing and debugging.

While the proposed solution for the accidental variability caused by shuffling is effective, the freezing of the *fundamental* variability caused by down-sampling merely hides this variability. However, by hiding fundamental sources of variability, we may become over-confident in our offline experiments and we may therefore want to take into account the variability produced by the ML system in our confidence intervals. The formalization of this process is left as future work.

Acknowledgments We would like to thank our colleagues Olivier Koch, Etienne Sanson, and Loic Le Bel for their useful comments on early versions of this paper.

References

[1] R. M. Bell and Y. Koren, "Lessons from the netflix prize challenge," *ACM SIGKDD Explorations Newsletter*, vol. 9, no. 2, pp. 75–79, 2007.

[2] C. Li, Y. Lu, Q. Mei, D. Wang, and S. Pandey, "Click-through prediction for advertising in twitter timeline," in *KDD '15*, pp. 1959–1968, ACM, 2015.

[3] X. He, J. Pan, O. Jin, T. Xu, B. Liu, T. Xu, Y. Shi, A. Atallah, R. Herbrich, S. Bowers, *et al.*, "Practical lessons from predicting clicks on ads at facebook," in *KDD '14*, pp. 1–9, ACM, 2014.

[4] T. Joachims, "Optimizing search engines using clickthrough data," in *KDD '02*, pp. 133–142, ACM, 2002.

[5] J. Leskovec, A. Rajaraman, and J. D. Ullman, *Mining of Massive Datasets*. Cambridge University Press, 2014.

[6] O. Chapelle, E. Manavoglu, and R. Rosales, "Simple and scalable response prediction for display advertising," *ACM Trans. Int. Systems and Technology*, vol. 5, no. 4, p. 61, 2014.

[7] "Criteo Releases Industry's Largest-Ever Dataset for Machine Learning." `http://www.criteo.com/news/press-releases/2015/06/criteo-releases-industrys-largest-ever-dataset/`, 2015.

[8] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *COMPSTAT '10*, pp. 177–186, Springer, 2010.

[9] J. Lin and A. Kolcz, "Large-scale machine learning at twitter," in *SIGMOD '12*, pp. 793–804, ACM, 2012.

[10] D. C. Liu and J. Nocedal, "On the limited memory bfgs method for large scale optimization," *Mathematical programming*, vol. 45, no. 1-3, pp. 503–528, 1989.

[11] B. Recht, C. Re, S. Wright, and F. Niu, "Hogwild: A lock-free approach to parallelizing stochastic gradient descent," in *NIPS '11*, pp. 693–701, 2011.

[12] A. Agarwal, O. Chapelle, M. Dudík, and J. Langford, "A reliable effective terascale linear learning system," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1111–1133, 2014.

[13] M. Li, L. Zhou, Z. Yang, A. Li, F. Xia, D. G. Andersen, and A. Smola, "Parameter server for distributed machine learning," in *Big Learning NIPS Workshop*, 2013.

[14] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le, *et al.*, "Large scale distributed deep networks," in *NIPS '12*, pp. 1223–1231, 2012.

[15] F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu, "1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns," in *Interspeech '14*, 2014.

[16] F. Vasile and D. Lefortier, "Cost-sensitive learning for bidding in online advertising auctions," in *Machine Learning for eCommerce (NIPS 2015 Workshop)*, 2015.

[17] T. Chandra, E. Ie, K. Goldman, T. L. Llinares, J. McFadden, F. Pereira, J. Redstone, T. Shaked, and Y. Singer, "Sibyl: a system for large scale machine learning," in *DSN '14*, 2014.

[18] R. McDonald, K. Hall, and G. Mann, "Distributed training strategies for the structured perceptron," in *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 456–464, Association for Computational Linguistics, 2010.

[19] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg, "Feature hashing for large scale multitask learning," in *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 1113–1120, ACM, 2009.

[20] O. Chapelle, "Offline evaluation of response prediction in online advertising auctions," in *Proceedings of the 24th International Conference on World Wide Web Companion*, pp. 919–922, International World Wide Web Conferences Steering Committee, 2015.