High Accuracy SGD Using Low-Precision Arithmetic and Variance Reduction (for Linear Models)

Alana Marzoev
Cornell University
mam655@cornell.edu

Christopher De Sa
Cornell University
cdesa@cs.cornell.edu

ABSTRACT

Stochastic gradient descent (SGD) is a popular algorithm for solving optimization problems, many of which originate from the training of linear models. The performance of SGD is dependent upon its ability to efficiently process large-scale data sets. With this dependency in mind, we consider a new SGD-style algorithm, referred to as HALP, which utilizes low-precision arithmetic and variance reduction techniques. We are interested in this algorithm because of its potential to unlock the benefits of low-precision arithmetic, such as increased computational throughput and a reduced memory footprint, without being limited by quantization error or a constrained representational domain. To begin to benchmark these benefits on CPUs, we implement an 8-bit version of this algorithm, and run preliminary experiments measuring its convergence per iteration and wall-clock time.

1 INTRODUCTION

A multitude of machine learning tasks can be represented as optimization problems. These optimization problems can be solved using a popular algorithm, stochastic gradient descent (SGD). The performance of SGD is largely contingent on its ability to quickly and efficiently process large-scale data sets, while simultaneously producing highly accurate results. Efficiency, here, is considered in multiple contexts, and includes energy, hardware, and statistical efficiency.

Low-precision arithmetic has been utilized in place of floating point computation in prior attempts at improving the hardware and energy efficiency in SGD. A low-precision implementation enjoys a myriad of benefits over the traditional SGD equivalent. These benefits include an increase in computational throughput, a reduction of the memory footprint, and a decrease in the energy consumption associated with running the SGD algorithm [2]. However, these benefits come at a cost: this technique is ultimately restricted by quantization (rounding) bias, as well as the fundamental accuracy limitations of a constrained representational domain, which uses fewer bits than that of the training examples. This domain may therefore may be insufficient for storing the entirety of the true solution in full precision, instead storing only the closest representable model.

Although many applications do not possess stringent accuracy requirements, others do, which motivates the development of a high-accuracy, high-efficiency SGD algorithm. To this end, we consider a new SGD-style algorithm, optimized for linear models, that was developed as part of a larger collaboration with the individuals mentioned in Section 4. The scope of this paper is focused on the details of the 8-bit implementation of the algorithm and preliminary experimental results. This algorithm leverages both low-precision arithmetic and variance reduction techniques [1] in such a way that helps overcomes the aforementioned limitations associated with low-precision arithmetic.

In this paper, we experimentally show that for a specific problem type, this **h**igh **a**ccuracy, **l**ow **p**recision algorithm, referred to from here on out as HALP, converges at a linear rate to arbitrary levels of accuracy. We also begin the process of benchmarking the benefits of low-precision arithmetic seen in HALP, specifically when running the algorithm on CPUs. We approach this evaluation by comparing the wall clock times of 8-bit HALP with Scikit-learn's logistic regression implementation [4], specifically observing the time it takes for both implementations to reach similar accuracies on a specific task, using distance to optimum as the metric of accuracy.

Algorithm 1 HALP: High-Accuracy Low-Precision Optimization for Linear Models

given: N loss functions ∇l_i and training examples x_i, number of epochs K, epoch length T, step size α, and initial iterate $\tilde{w}_0$.
given: number of low-precision-representation bits b.
given: low-precision data representation ■
$z_{0,T} \rightarrow 0$.
for $k = 1$ **to** K **do**
  $\tilde{w}_k \leftarrow \tilde{w}_{k-1} + z_{k-1,T}$
  for $i = 1$ **to** N **do**
    $\tilde{\phi}_{k,i} \leftarrow {x_i}^T \tilde{w}_k = \tilde{\phi}_{k-1,i} + {x_i}^T z_{k-1,T}$
  end for
  $\tilde{g}_k \leftarrow \nabla f(\tilde{w}_k) = \frac{1}{N} \sum_{i=1}^{N} l_i'(\tilde{\phi}_{k,i}) x_i$
  $\tilde{s}_k \leftarrow \frac{\tilde{g}_k}{\mu(2^{b-1}-1)}$
  re-scale: ■ $= (\tilde{s}_k, b)$
  choose: low-precision intermediate representation ■
  choose: low-precision intermediate representation ■ = ■ × ■

  require: domain(■) ⊆ domain(■)
  $\tilde{h}_k \leftarrow Q_{■}(\alpha \tilde{g}_k)$
  $z_{k,0} \leftarrow 0$
  for $t = 1$ **to** T **do**
    sample i uniformly from $\{1, \ldots, N\}$
    $n \leftarrow Q_{■}\left(\alpha \left(l_i'(\tilde{\phi}_{k,i} + {x_i}^T z_{k,t-1}) - l_i'(\tilde{\phi}_{k,i})\right)\right) x_i$
    $z_{k,t} \leftarrow Q_{■}\left(z_{k,t-1} - n - \tilde{h}_k\right)$
  end for
end for
return $\tilde{w}_K + z_{K,T}$

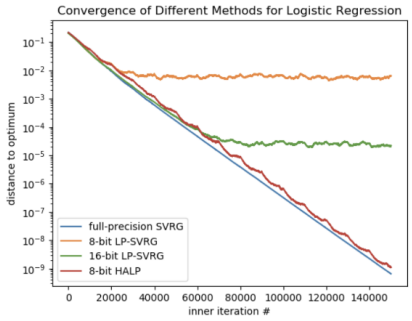

Figure 1: Convergence rate comparison of low-precision SVRG, full-precision SVRG, 8-bit HALP, and 16-bit HALP.

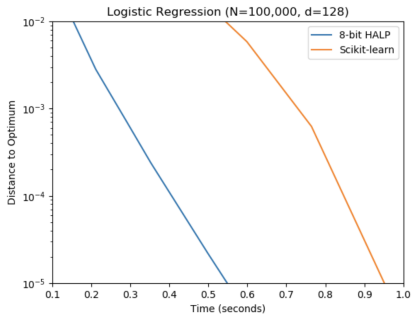

Figure 2: Comparison of distance to optimum vs. wall-clock time of sklearn's logistic regression and HALP.

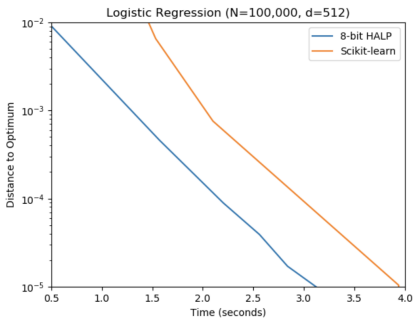

Figure 3: Comparison of distance to optimum vs. wall-clock time of sklearn's logistic regression and HALP.

2 HALP FOR LINEAR MODELS

Many machine learning tasks can be written as a linear model, where for a loss function $l : \mathbb{R} \rightarrow \mathbb{R}$, training examples $x_i \in \mathbb{R}^d$, and component functions $f_i(x)$,

$$f_i(w) = l(w^T x_i)$$

The HALP algorithm discussed here is optimized for such tasks.

Algorithm. We use three different key techniques in the HALP for linear models optimization. The first of these is the utilization of randomized rounding in the quantization process, in which we convert numbers to their equivalent values in different representational domains. A representational number domain is defined by two parameters, the scale factor δ, and the number of bits, b, which we denote denote with a color, and use to represent the following set of numbers:

$$\blacksquare = (\delta, b) = \{-\delta \cdot 2^{b-1}, \ldots, \delta \cdot 2^{b-1} - 1\}$$

The benefits of using randomized rounding are omitted here, as they have been explored in prior work [2].

The second technique, which enables the HALP algorithm to reach arbitrary accuracies, is a novel re-centering and re-scaling mechanism. We use this mechanism in setting the parameters of the low-precision representational domain. As HALP's outer iterate approaches the true value with progressively higher levels of confidence in the later iterations of the algorithm, the range of z that we must optimize over becomes smaller. This prompts us to repeatedly re-scale the domain to be centered on the outer iterates of SVRG, which are increasingly closer to the this true value. This technique is one of the primary distinctions between HALP and "regular" low-precision SVRG.

The third technique used in HALP enables an improvement in hardware efficiency and potential improvements in wall clock time. This technique relies on the realization that the following substitution can be made in the inner loop of the algorithm:

$$l_i'(x_i^T w_{k,t-1}) = l_i'(x_i^T w_k - x_i^T z_{k,t-1})$$

As a result of this change, it is now possible to cache all of these dot products at the beginning of each epoch, preventing any full-precision computation from needing to be done in the inner loop, and facilitating potential performance improvements in HALP.

Implementation. We implemented an 8-bit version of HALP for linear models in C++, and manually vectorized the inner loop of the algorithm using Intel intrinsics instructions. We chose to do this manually as opposed to relying on gcc's automatic vectorization because we were able to get larger speedups in wall clock time with the former, which is a previously explored effect [3]. We restrict the generated input data to not include -128, to prevent sign flips when performing our optimized, low-precision dot products.

3 RESULTS

Setup. We tested 8-bit HALP on a regularized logistic regression problem using randomly generated data. For comparison, we tested Scikit-learn's logistic regression implementation on the same data.

Linear convergence. To verify that HALP converges linearly to arbitrary levels of accuracy, we compared the convergence rates of full-precision SVRG, low-precision SVRG (both 8-bit and 16-bit), and 8-bit HALP. The results of this experiment can be seen in Figure 1: 8-bit HALP converges linearly, tracking full-precision SVRG, while both 8-bit and 16-bit SVRG are unable to follow, hitting noise balls rather than converging all the way to the true solution.

Wall clock time. Our second set of experiments compare the wall clock time to the distance to optimum reached by 8-bit HALP and Scikit-learn's logistic regression implementation, using synthetic training sets with $100,000$ examples. We show that 8-bit HALP outperforms Scikit-learn's implementation for varying model sizes.

4 CONCLUSION & ACKNOWLEDGEMENTS

This paper introduces the 8-bit HALP optimization for linear models, discusses its key enabling techniques, and presents a partial and preliminary experimental evaluation of the algorithm. This work is part of a larger, ongoing collaboration with Megan Leszczynski, Jian Zhang, and Christopher Ré, who we would like to acknowledge for their many contributions.

REFERENCES

[1] Rie Johnson, Tong Zhang. *Accelerating Stochastic Gradient Descent using Predictive Variance Reduction.*

[2] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, Pritish Narayanan. *Deep Learning with Limited Numerical Precision.*

[3] Christopher De Sa, Matthew Feldman, Christopher RÃĺ, and Kunle Olukotun. *Understanding and Optimizing Asynchronous Low-Precision Stochastic Gradient Descent.*

[4] Pedregosa et al. *Scikit-learn: Machine Learning in Python.*