An Investigation into HOGWILD!*

Abhijit Chowdhary
ac6361@nyu.edu

New York University — May 21, 2020

Contents

*See https://github.com/abhijit-c/HOGWILD.

1 Introduction

Despite being the subject of much modern excitement, the ideas of gradient descent date all the way back to Cauchy in 1847. And although it's main application has been in the solution of recent big-data optimization problems, stochastic gradient has been around since the 1940s, formally by Robbins and Monro in 1951. In the last two decades, however, modern hardware has begun to see a tapering off of Moore's law, and has begun to expand out in a distributed fashion with multicore processors and GPUs; naturally the question becomes: In order to take advantage of the strengths of modern hardware, how can we parallelize a stochastic gradient method?

2 HOGWILD!

Prior to 2011, parallel stochastic gradient methods had been introduced, but most suffered from poor scaling due to the necessity of locks. A naive implementation could look like:

Algorithm 1 Very Naive Parallel Stochastic Gradient

Require: Number of data points N, seperable loss function $f = \sum_{e \in E} f_e(x_e)$, Initial x.

for epoch $= 1 \to$ MAX_EPOCHS **do**
  #pragma omp parallel for
  for $k = 1 \to N$ **do**
    Choose i uniformly from $\{1, \ldots, |E|\}$.
    #pragma omp critical
      Read current parameters x.
      Compute $\nabla f_i(x)$.
      $x \leftarrow x - \eta \nabla f_i(x)$.
  end for
end for

Note that the version presented above is one with a fixed number of iterations, as the discussion of stopping criteria seems to be similar to that of the stochastic gradient method, and for extremely large data sets, is often heuristic. But it's clear here that such an algorithm would only effectively be parallelizing the unform sample of i in $\{1, \ldots, |E|\}$, and it's overall parallel efficiency would likely be poor. Technically, you can improve the above by replacing the critical section with selective locks on components of x based on the sparsity pattern of $\nabla f_i(x)$, but because the process of acquiring locks is much more expensive than floating point arithmetic, this helps little.

However, in 2011, the article "HOGWILD!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent" by Niu et al. [NRRW11] proposed a very simple solution to this problem. Remove the locks![1]

[1] Apparently this was discovered by accident by Feng Niu, one of the original paper's authors, when he was debugging stochastic gradient method code. I wish my troubleshooting was nearly as effective... [Rec14]

Algorithm 2 HOGWILD!: Asynchronous Stochastic Gradient with replacement

Require: Number of data points N, seperable loss function $f = \sum_{e \in E} f_e(x_e)$, Initial x.

for epoch $= 1 \to$ MAX_EPOCHS **do**
  #pragma omp parallel for
  for $k = 1 \to N$ **do**
    Choose i uniformly from $\{1, \ldots, |E|\}$.
    Read current parameters x.
    Compute $\nabla f_i(x)$.
    $x \leftarrow x - \eta \nabla f_i(x)$. ▷ Must be done atomically
  end for
end for

and should we want to sample without replacement the algorithm is easily modified to:

Algorithm 3 HOGWILD!: Asynchronous Stochastic Gradient without replacement

Require: Number of data points N, seperable loss function $f = \sum_{e \in E} f_e(x_e)$, Initial x.

for epoch $= 1 \to$ MAX_EPOCHS **do**
  Let P be a random permutation of $\{1, \ldots, |E|\}$. ▷ i.e. a Fisher-Yates Shuffle.
  #pragma omp parallel for
  for $k = 1 \to N$ **do**
    $i \leftarrow P[k]$.
    Read current parameters x.
    Compute $\nabla f_i(x)$.
    $x \leftarrow x - \eta \nabla f_i(x)$. ▷ Must be done atomically
  end for
end for

It should be noted that although the formal OMP locks have been removed, atomic operations are still required in order to prevent mutual exclusion. However, no guards have been placed to prevent a thread from overwriting another's computation midway through, and it's not obvious as to why such a race condition wouldn't destroy the performance of the Stochastic Gradient method. However, with certain assumptions one can show that HOGWILD! behaves roughly like a noisy stochastic gradient method, and thus shares its convergence properties.

3 Convergence Analysis

Given that the result of HOGWILD!, in the absence of noise generated by the asynchronous updates of x, henceforth denoted *asynchronous noise*, is equivalent to a stochastic gradient method, we should expect convergence rates similar to those of stochastic gradient, should noise be small. Take for example the typical linear least squares loss function $f(x) = \frac{1}{2n}\|Dx - b\|_2^2$, where $x \in \mathbb{R}^n$ and $D \in \mathbb{R}^{n \times n}$ a diagonal matrix. Writing this as a sum of each data entry:

$$f(x) = \frac{1}{2n}\sum_{k=1}^{n}(D_{ii}x_i - b_i)^2 = \frac{1}{n}\sum_{k=1}^{n} f_i(x) \implies (\nabla f_i(x))_k = \begin{cases} D_{ii}(D_{ii}x_i - b_i), & k = i \\ 0, & k = 0 \end{cases}$$

Because our $\nabla f_i(x)$'s only have a single entry in their own component, we can see that as long as no other thread is working on component i simultaneously, no asynchronous noise will be generated; should we use HOGWILD! without replacement then this is guaranteed. In the original paper [NRRW11], sparsity of the vector $\nabla f_i(x)$ was required in order to guarantee convergence, but as we'll see later, this isn't always necessary.

As a baseline of comparison, we first state a result on the convergence of the stochastic gradient method:

Theorem 3.1. *(Convergence of the Stochastic Gradient Method [BCN16]) Let $F : \mathbb{R}^d \to \mathbb{R}$ be an objective function we're seeking to minimize. We can write this as either an expected risk $F(x) = \mathbb{E}_\xi f(x, \xi)$, or an empirical risk $F(x) = \frac{1}{n}\sum_{k=1}^n f_k(x)$. Under the assumptions:*

(1) F is continuously differentiable and ∇F is Lipschitz continuous with Lipshitz constant L, i.e.

$$\left\|\nabla F(x) - \nabla F(y)\right\| \leq L\|x - y\|, \forall x, y \in \mathbb{R}^d$$

(2) F is strongly convex with constant c, i.e.

$$F(x) \geq F(y) + \nabla F(y)^T(x - y) + \frac{1}{2}c\|x - y\|^2, \forall x, y \in \mathbb{R}^d$$

(3) F is bounded below over the region explored by stochastic gradient method.

(4) In expectation, the vector $-\nabla f(x_k, \xi_k)$ is a descent direction for F with norm bounded by it's own norm. That is: $\exists \mu_G \geq \mu > 0$ such that $\forall k \in \mathbb{N}$:

$$\nabla F(x_k)^T \mathbb{E}\left[\nabla f(x_k, \xi_k)\right] \geq \mu\left\|\nabla F(x_k)\right\|^2 \text{ and } \left\|\mathbb{E}\left[\nabla f(x_k, \xi_k)\right]\right\| \geq \mu_G\left\|\nabla F(x_k)\right\|$$

(5) $\exists M, M_V \geq 0$ such that $\forall k \in \mathbb{N}$:

$$\mathrm{Var}\left[\nabla f(x_k, \xi_k)\right] \leq M + M_V\left\|\nabla F(x_k)\right\|^2$$

Then, assuming a fixed stepsize α (a.k.a. learning rate), satisfying $\alpha \in (0, \mu/LM_g]$, we have:

$$\mathbb{E}\left[F(x_k) - F_*\right] \leq \frac{\alpha LM}{2c\mu} + (1 - \alpha c\mu)^{k-1}\left(F(x_1) - F_* - \frac{\alpha LM}{2c\mu}\right)$$

and if we instead choose α diminishing, i.e. let $\alpha_k = \frac{\beta}{\gamma + k}$ where $\beta > 1/c\mu, \gamma > 0$ are chose such that $\alpha_1 \leq \mu/LM_G$, then:

$$\mathbb{E}\left[F(x_k) - F_*\right] \leq \frac{1}{\gamma + k}\max\left\{\frac{\beta^2 LM}{2(\beta c\mu - 1)}, (\gamma + 1)(F(x) - F_*)\right\}$$

The result is technical, and very long to prove, so I just refer to the article [BCN16] for it. Regardless, the last result in the above theorem is what we strive for in *Hogwild!*: convergence in $\mathcal{O}(1/k)$ time.

3.1 Theoretical Results

HOGWILD!'s original article [NRRW11] manages to prove that the asynchronous noise generated by HOGWILD!, under certain sparsity assumptions of the f_i's, converged at the same rate as seen in the stochastic gradient method. However, a newer article from 2015 by De Sa et al. [SZOR15] presented a new framework in the context of martingale theory, which drops said sparsity assumptions and generalizes to certain non-convex formualations. Given that I just learned about martingale theory essentially two weeks ago in Basic Probability, and that much of my synthetic tests are on dense datasets, I feel compelled to present this argument instead.

The following is a cleaned up summary of the arguments presented in the article, except I focus on just demonstrating how martingale theory can be used to generalize convergence arguments for a sequential stochastic gradient method to the asynchronous case, and cut out the generalization required for the non-convex situation.

3.1.1 Initial Machinery

First, recall the definition of a martingale:

Definition 3.1. *[JP04] A sequence of random variables $(X_n)_{n\geq 0}$ is called a martingale, or an $(\mathcal{F}_n)$-martingale, if*

(i) $\mathbb{E}\left[|X_n|\right] < \infty, \forall n.$

(ii) X_n *is* $\mathcal{F}_n$ *measurable,* $\forall n.$

(iii) $\mathbb{E}\left[X_n \mid \mathcal{F}_m\right] = X_m$ *a.s.,* $\forall m \leq n.$

furthermore a supermartingale (submartingale) is one satisfying (i), (ii) exactly, and (iii) with $\leq$ $(\geq)$ *instead.*

Using this, the idea is to model our convergence with a non-negative supermartingale $W_t(x_t, \ldots, x_0)$ which is a function of the previous stochastic gradient iterates. These W_t, when used in the theory later, will be associated with specific stochastic algorithms, as an example below we'll see it applied to serial stochastic gradient. However, given such a supermartingale, and given a bounded stopping time B (in literature known as a horizon), if our stochastic gradient iterates are written as $x_{t+1} = x_t - \eta \nabla f_t(x_t)$, where η is the learning rate, and f_t denotes the random function chosen at time step t, then we see that condition (iii) in the above definition implies that:

$$W_{t+1}(x_t - \eta \nabla f_t(x_t), x_t, \ldots, x_0) \leq W_t(x_t, \ldots, x_0), \forall t \leq B$$

which certainly makes sense if our $-\nabla f_t(x_t)$ is a sufficient search direction. Furthermore, letting our success region be denoted as $S = B_\varepsilon(x^*)$, where x^* is the minimizer of our optimization problem, if we impose that if $x_t \notin S, \forall t \leq T$ then:

$$W_T(x_T, \ldots, x_0) \geq T$$

then we call W_t a *rate supermartingale*. To simplify notation, let F_t be the event where $\nexists t \leq T$ such that $x_t \in S$.

A good example of the power of this machinery is proving a convergence bound on the serial version of stochastic gradient: using (iii) of definition 3.1, one can see that considering F_T:

$$\mathbb{E}\left[W_0(x_0)\right] \underbrace{\geq} \mathbb{E}\left[W_T\right] = \mathbb{P}\left[F_t\right] \underbrace{\mathbb{P}\left[F_T\right] \mathbb{E}\left[W_T | F_T\right]}_{W_T \geq T} + \underbrace{\mathbb{P}\left[F_T^c\right] \mathbb{E}\left[W_T | F_T^c\right]}_{W_T \geq 0} \geq \mathbb{P}\left[F_T\right] T$$

where the first inequality is by Doob's optional sampling theorem. Thus for a simple serial stochastic gradient method: $\mathbb{P}\left[F_t\right] = \mathbb{E}\left[W_0\right]/T$.

Now that we can characterize serial stochastic gradient in this model, we need a method to analyze asynchronous noise. Recall that since we've guaranteed in the description of HOGWILD! that writes to the iteration variable x_t are done atomically, the only race condition possible is when updates to entries of x_t are interleaved with either the other thread's updates or its reads on x_t.

Indeed, when going to update the ith component of x_{t+1}, the variable used to compute the gradient may have long since changed, making our iteration look more like $x_{t+1} = x_t - \nabla f_t(v_t)$ where v_t's entries were the entries of some previous iterate x. Let $\tau_{i,t}$ denote the lag for the update of the ith component of x_{t+1}, i.e. $(v_t)_i = (x_{t-\tau_{i,t}})_i$. Then we recognize that this lag for each component i (supposing the computer hasn't crashed) must be bounded; let τ' be the the maximum over i of such bounds and let $\tau = \mathbb{E}\left[\tau'\right]$. This is known in literature as the *worst-case expected delay*.

Finally, we need one last definition, mainly one of convinience, to proceed onward. This describes the main conditions upon a rate supermartingale necessary to prove that the asynchronous noise error is irrelevant to the convergence rate.

Definition 3.2. *An algorithm with associated rate supermartingale W is (H, R, ξ)-bounded if the following conditions hold.*

(1) W must be Lipschitz continuous in the current iterate with parameter H, i.e.

$$\left\|W_t(u, x_{t-1}, \ldots, x_0) - W_t(v, x_{t-1}, \ldots, x_0)\right\| \leq H\|u - v\|, \forall t, u, v, x_t, \ldots, x_0.$$

(2) ∇f must be Lipschitz continuous in expectation with parameter R, i.e.

$$\mathbb{E}\left[||\nabla f(x) - \nabla f(y)||\right] \leq R\|u - v\|$$

(3) The expected magnitude of the update must be bounded by ξ, i.e.

$$\mathbb{E}\left[||\nabla f(x)||\right] \leq \xi$$

Note that these look very familiar to the conditions in the stochastic gradient method convergence theorem (theorem 3.1).

3.1.2 Convergence of HOGWILD!

Finally, now that all of the machinery has been defined, we can get to the main result:

Theorem 3.2. *Suppose we have an asynchronous stochastic algorithm with associated rate supermartingale W which is (H, R, ξ) bounded with horizon B. Furthermore, assume that $HR\xi\tau < 1$; then $\forall T \leq B$:*

$$\mathbb{P}\left[F_T\right] \leq \frac{\mathbb{E}\left[W(0, x_0)\right]}{(1 - HR\xi\tau)T}$$

Before we prove it, we briefly discuss how to use this theorem in practice. Suppose we have a loss function f which we want to use HOGWILD! on to minimize. Then first we need to obtain a rate supermartingale proving the problem[2], determine H, R, ξ such that W is (H, R, ξ)-bounded, and then apply this theorem to get a proper rate of convergence. This is a very powerful theorem, as we'll be able to state later, given strongly convex f and with the other required bounds, we can blanket prove that HOGWILD! works on them. Furthermore, for nicer non-convex problems, such as the low-rank least-squared matrix completion problem presented in the paper, it's easy to derive a proper rate supermartingale as well.

Now, with respect to the proof, we only outline it because it's mainly just a repeated application of the above bounds we have in order to lower bound away noise terms, and then it follows the path outlined in the serial stochastic gradient method proof.

Proof. (i) With W defined exactly as in the serial case, it's not a rate supermartingale. Instead, from it we construct rate supermartingale V_t, where $\forall t, x$ where x_u not converged $\forall u < t$:

$$V_t(x_t, \ldots, x_0) = W_t(x_t, \ldots, x_0) - \underbrace{HR\xi\tau t}_{(1)} + \underbrace{HR\sum_{k=1}^{\infty}\|x_{t-k+1} - x_{t-k}\|\sum_{m=k}^{\infty}\mathbb{P}\left[\tau' \geq m\right]}_{(2)}$$

where (1) allows for longer iteration counts (as HOGWILD needs to allow for given noise corruption), and (2) measures distance between recent iterates. Otherwise if x_u is converged, then we let $V_t(x_t, \ldots, x_0) = V_u(x_u, \ldots, x_0)$. Basically, we've defined V a stopped process.

(ii) Show V_t is a rate supermartingale for HOGWILD!

(iii) Using a similar process to the serial stochastic gradient proof, show that $\mathbb{E}\left[V_T\right] \leq \mathbb{E}\left[V_0\right] = \mathbb{E}\left[W_0\right]$, then using the law of total expectation on this with F_T, and recalling that $\mathbb{E}\left[W_T | F_T\right] \geq T$, we recieve the desired result.

□

Given this, the general theorem for the convex case is:

Theorem 3.3. *Consider trying to minimize f, which is:*

[2] In the article, De Sa et al. describes this as being no more difficult than proving serial convergence, but the proof (in the convex case) doesn't seem to have any similarity between it and serial convergence, so I can't validate this claim.

- *Strongly convex with parameter c.*
- *∇f_k Continuously differentiable in $||\cdot||_1$ with Lipschitz constant L.*
- *Upper bounded second moment of the gradient by M^2*
- *Success criteria $\|x - x^*\|^2 \leq \varepsilon$, for some $\varepsilon > 0$.*

Then we can construct rate supermartingale W_t such that it's (H, R, ξ) bounded with $H = 2\sqrt{\varepsilon}(2\eta c\varepsilon - \eta^2 M^2)^{-1}, R = \eta L, \xi = \eta M$. Then choosing step size η (for some $\nu \in (0,1)$):

$$\eta = \frac{c\varepsilon\nu}{M^2 + 2LM\tau\sqrt{\varepsilon}}$$

then we recieve:

$$\mathbb{P}\left[F_T\right] \leq \frac{M^2 + 2LM\tau\sqrt{\varepsilon}}{c^2\varepsilon\nu T} \log\left(e\|x_0 - x^*\|^2 \varepsilon^{-1}\right)$$

3.2 Numerical Experiments

Now that we've finally slogged through the presentation of the theory, we can produce a few numerical experiments. Take, for example, the linear least squares regression problem with $A \in \mathbb{R}^{n\times m}, x \in \mathbb{R}^m$ and $b \in \mathbb{R}^n$:

$$f(x) = \frac{1}{2}\|Ax - b\|_2^2 = \sum_{k=1}^{n} \underbrace{\frac{1}{2}(A_i x - b_i)^2}_{f_i(x)}$$

This is a particularily interesting example, as when we examing the stochastic descent direction of this loss function:

$$\nabla f_i(x) = A_{i*}^T(A_{i*}x - b_i), \text{ where } A = \begin{bmatrix} A_{1*} \\ \vdots \\ A_{n*} \end{bmatrix}$$

we notice that the sparsity pattern of $\nabla f_i(x)$ is entirely controlled by by the sparsity pattern of the rows of A. This makes this problem particularily convinient when trying to understand the effect of sparsity on the asynchronous noise generated by HOGWILD!. The original paper [NRRW11]'s results required strict assumptions on the sparsity pattern of $\nabla f_i(x)$ in order to guarantee convergence, but as we saw in Theorem 3.3, as long as certain regularity properties are satisfied, there's no need for such an assumption. Indeed f does satisfy the above, assuming that A is of full rank[3]. Therefore, when A is both sparse and dense, we should see a $\mathcal{O}(1/k)$ convergence rate, and indeed see Figure 1 for the validation of that.

[3]One subtle note is that the second condition is actually equivalent to having upper bounded maximal eigenvalue. I think we proved this back when we were discussing Nesterov's method, and also can be found at this stackexchange https://math.stackexchange.com/a/1699082/245618.

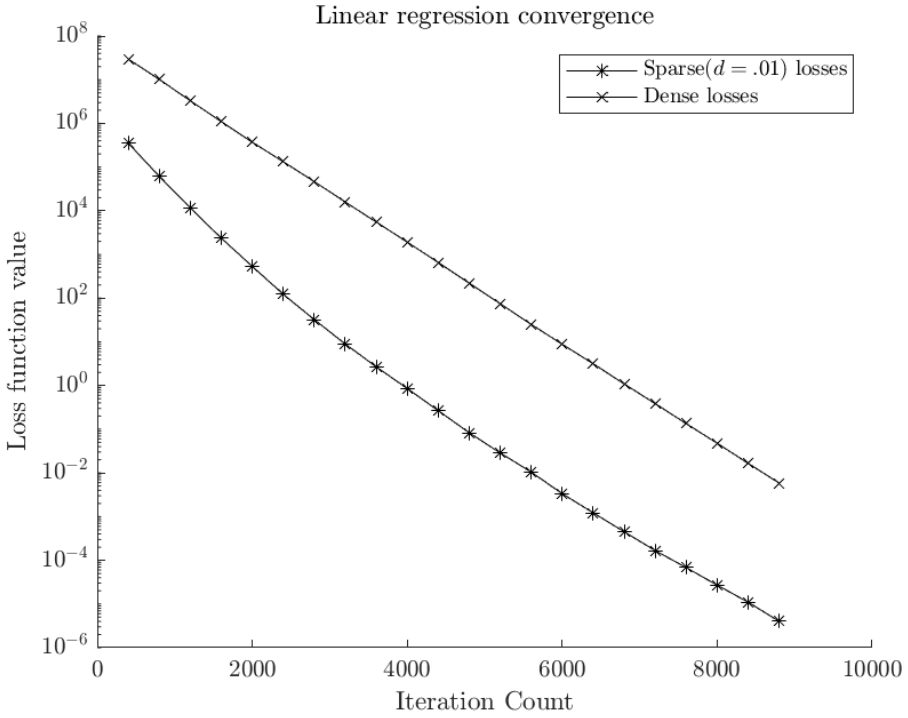


Figure 1: The matrix Sparse($d = 0.1$) is generated by the command `sprand(n,m,.01)`. Playing with the density parameter doesn't change the shape of the convergence line, but note that denser matrices (since the entries are Gaussian) produce a higher initial loss and suffer from more asynchronous noise, and therefore require more iterations.

3.2.1 Asynchronous Noise as a Function of Bandedness

So now that we've seen that, indeed, dense updates enjoy the same convergence properties as sparser ones, we can also investigate the relationship of asynchronous noise as a function of sparsity. Since the sampling of the next data point to be used in a stochastic iterate is done uniformly and independently, we note that in the interest of minimizing asynchronous noise, the exact pattern matters less than the number of non-zero elements. Therefore, an interesting way to represent different sparsity patterns in our linear regression, is in choosing A banded, and then varying the size of that band.

Definition 3.3. *Let $k \geq 1$ and $A \in \mathbb{R}^{n \times n}$. We denote A as a $(k-1)$-banded matrix if k is the maximal integer greater than zero such that either the kth or the $-k$th diagonal are not-identically zero.*

Now consider the operation of HOGWILD! with two simultaneous threads on the loss function defined above, with A k-banded. Then an lower-bound to the probability that these two threads do not share an component of x_i, which they need to read/write from/to, is (assuming sampling with replacement) the probability of picking i_1, i_2 uniformly from $[|k+1, \ldots, n-k|]$ such that $[|i_1 - k, i_1 + k|] \cap [|i_2 - k, i_2 + k|] = \varnothing$. Via a simple counting

argument, we find that this is:

$$\mathbb{P}\left[[|i_1 - k, i_1 + k|] \cap [|i_2 - k, i_2 + k|] = \varnothing\right] = \frac{\big((n-2k) - (2k+1)\big)_+}{n-2k} = \frac{(n-4k-1)_+}{n-2k}$$

where $(\cdot)_+ = \max(0, \cdot)$. This lower-bound gives us an upperbound on the probability that they do share a necessary component, one minus the above. This confirms something we already know, that if $k << n$, then the two threads are very unlikely to have asynchronous noise. However, with even just $k = n/4$, then the above probability is zero, and this is just for $P = 2$. I had wanted to calculate the k taking the above to zero as a function of P, but the analysis quickly becomes intractible for $P \geq 3$, barring a nice counting argument I don't see. Regardless, as long as we can measure asynchronous noise, we can get an idea of how it increases as a function of k.

To construct an experiment to see this, let x_k (k not a component, but an parameter) be the final iterate of HOGWILD!, as applied to the linear regression problem with A k-banded. Then supposing we hold some solution x^* constant among all k, then we can view $f(x_k)$ as a random-variable, whose variance characterizes the amount of asynchronous noise experienced throughout computation, as the sequential algorithm (assuming the random choice of stochastic data points is seeded between intervals) will have $\text{Var}\left[f(x_k)\right] = 0, \forall k$. See Figure 2 for the results of such an experiment.

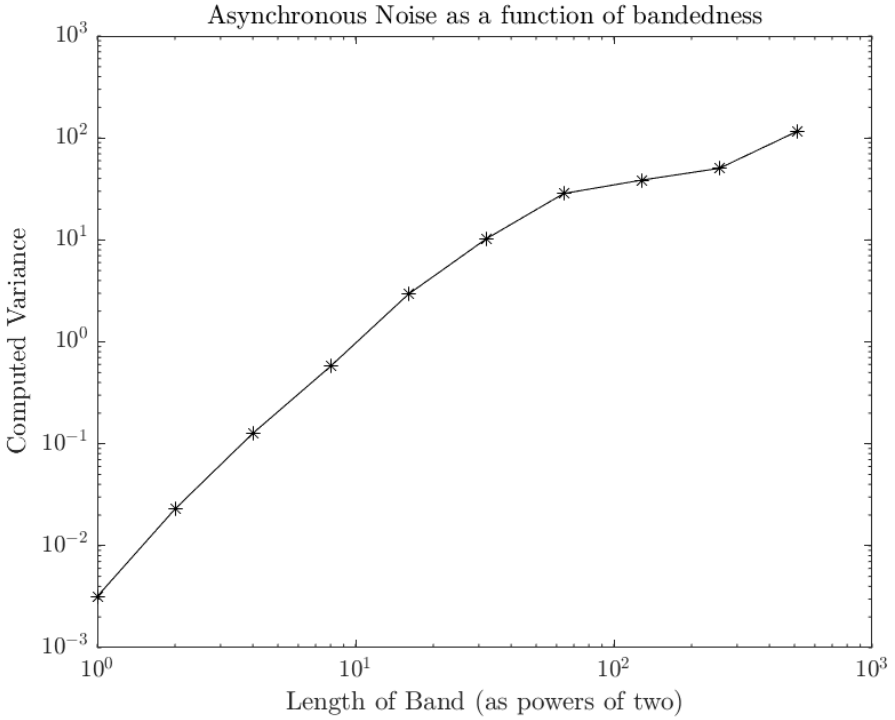


Figure 2

4 Efficiency Analysis

Given that this is a parallel algorithm, we would be remiss if we didn't study it's efficiency. The classical method is an analysis via Amadahl's law, which states that:

$$S_{\text{latency}}(s) = \frac{1}{(1-p) + \frac{p}{s}}$$

where S_{latency} is the theoretical speedup in execution of the whole task, s is the speedup of the portion benefiting from parallel processing, and p is the proportion of execution now being parallelized. Supposing we're working on HOGWILD! with replacement, then the entire computation (minius I/O) is parallelized, so $p = 1$. Furthermore, since the parallel region is just one embarassingly parallel (after justification) loop, $s = P$, where P is the number of processors. Thus, Amadahl's law turns out to be pretty trivial in the case of asynchronous stochastic methods, in that $S_{\text{latency}}(s) = s = P$.

However, in practice this is most-definitely not observed, not in my tests, and neither in the original paper [NRRW11]. See Figure 3 for my scaling results. Specifically for my

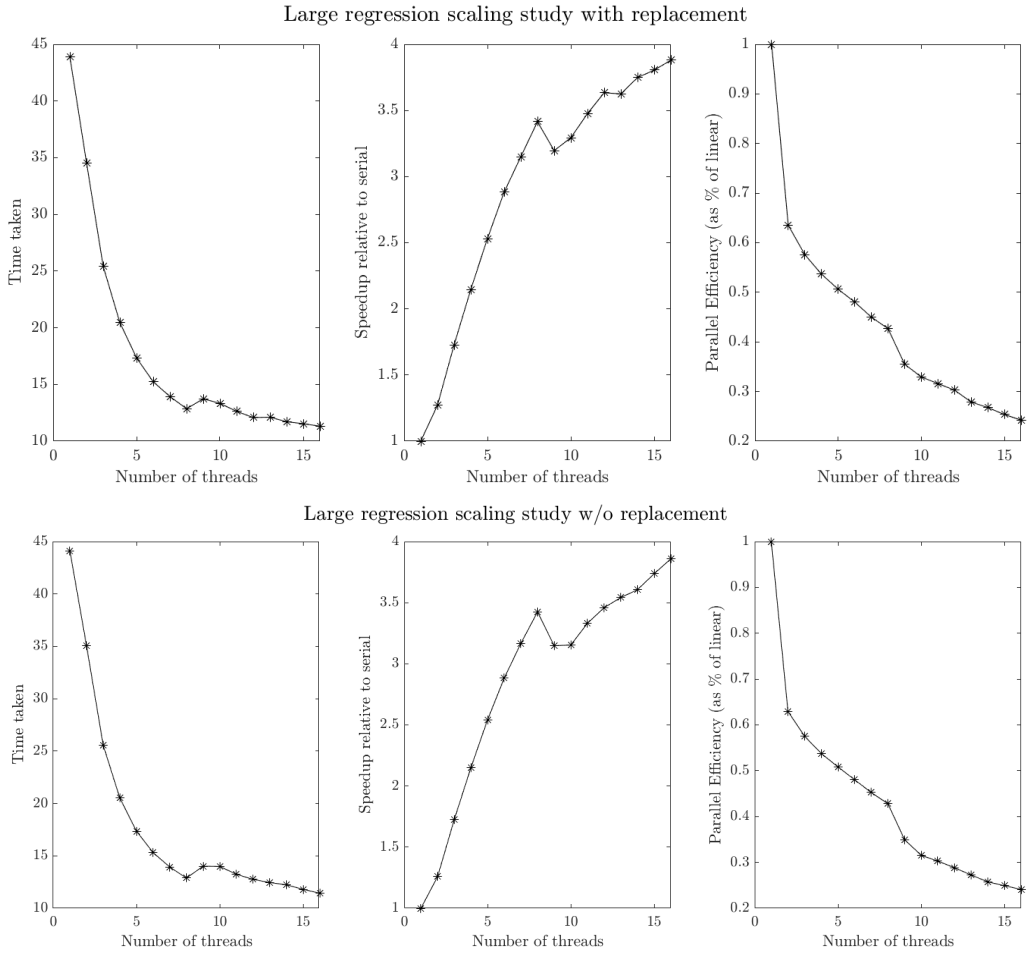


Figure 3

tests, it's not too hard to tell what's breaking the performance. Simply put, we're being

cruel to the cache. Both of these scaling studies were computing over pretty large matrices, $A \in \mathbb{R}^{400^2 \times 400}$, which in just information is 0.5GB large. Furthermore, the column of 400^2 doubles, for my processor the Ryzen 3700X, doesn't even have a tenth of it fit into L1 Cache. Thus for computing the gradient terms, which have a $A_{i*}^T \cdot A_{i*}$ term inside of them, even if we explicity avoid computing the full matrix, we will have a large amount of Cache misses. This is especially given that I, in my haste, wrote my code using Eigen matrices, which upon reflection at this very moment, are column major. Thus, when parallelizing over many cores each utilizing the limited cache space, we suffer many cache-misses.

To illustrate how the speedup might approach the theoretical rate when such practical performance impediments don't exist, we recognize that the problem is that we're memory bound. Should we instead be CPU bound, say if the necessary data fits comfortably into the L1 Cache divided by the number of threads, but instead requires lots of computation to compute, we should see near optimal rates of convergence. To simulate this while staying in a linear regression model, we can cheat and add a sleep statement at the end of each stochastic update, simulating additional computation. See Figure 4 for the results of such a test.

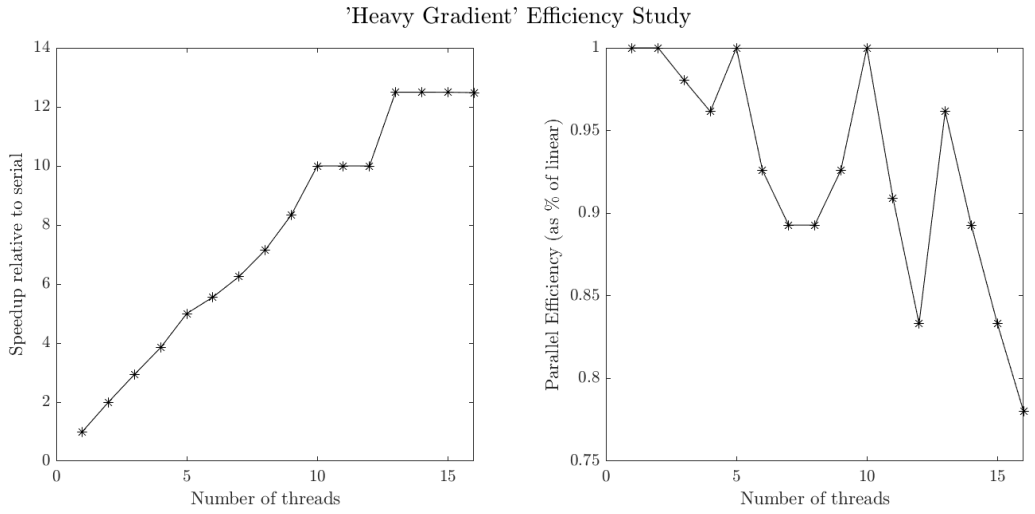
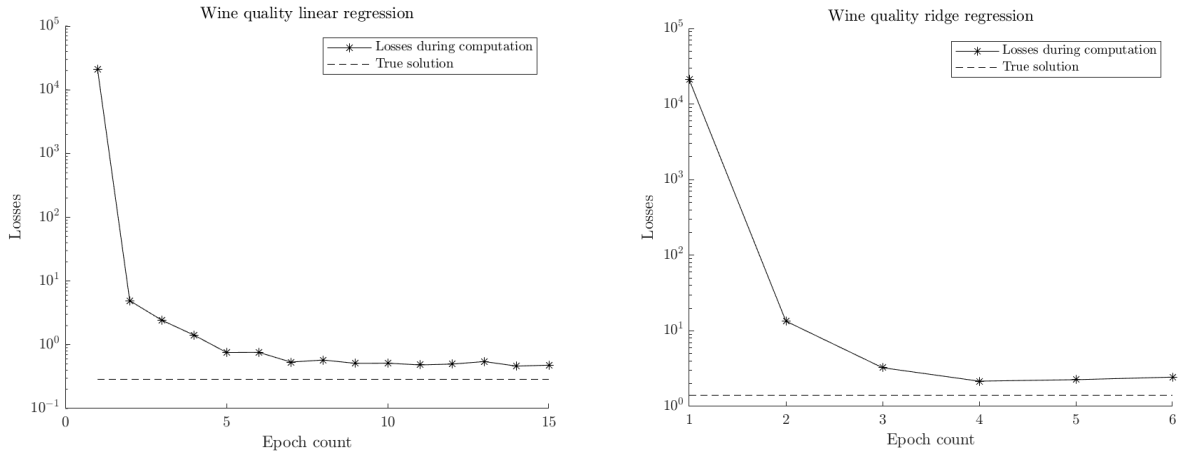


Figure 4

5 Applications: Judging Wine Quality

As a sort of fun application, I found a dataset on UCI's archive from a paper by Cortez et al. [CCA+09] which charted various physicochemical properties of a set of white wine, and had corresponding 'quality' (judged by a committee, the CVRVV) levels for each data point. The reason why I decided to try this dataset, is not only because knowing what makes alcohol quality clearly a very important task, but also because in the original paper, their uncertainty regarding the relevancy of all input variables was noted. That is, this is an interesting opportunity to try a feature selection method: my method of choice this time

was ridge-regression, the ℓ^2 variant of LASSO. See Figures 5a and 5b for the linear and ridge regression variants, respectively.



(a) (b)

Figure 5: Both converge in rougly four to eight epochs, but it should be noted that neither converges to the true solution. Regardless of how I chose the learning rate, it appeared that both runs of HOGWILD! would get stuck in a noise ball somewhere close to the true solution. This is an important failing of stochastic gradient methods in general, sometimes the noise generated will prevent us from seeing a 'true' solution, and indeed the ridge-regressed version didn't properly feature select, whereas the solution computed via CVX saw the 11th feature to be most weighted (The 11th feature, to my amusement, is alcohol content. Certainly an important part of what makes a good wine...).

6 Conclusions and Future Work

In conclusion, HOGWILD! is a great method to very easily parallelize exist stochastic gradient code. Although if the component gradient computation is memory bound, you're unlikely to see much scaling in speedup, if instead it's CPU bound, you may see near-perfect scaling, which is great. Of course, as seen in the wine-quality tests, because this is a stochastic method after all (and is, in theory, viewed as a noisy approximation to stochastic gradient, which itself can be viewed as a noisy approximation to gradient descent), we have to wary about such noise preventing us from usable results.

For non-convex problems, although martingale theory helps us to prove convergence for a few simple cases, it seems that nothing quite addresses the general case yet, although the existing theory is strong for any convex problem. Personally, something I will keep an eye out in the future for is papers on extending the presented theory to more non-convex objectives, as I found that paper to be an interesting read.

There were a couple interesting directions of research that I didn't have time to look into that I do plan to over the rest of this summer. One paper that I read, but didn't have the time to fully digest was on CYCLADES [PLT+16], which was a version of Asynchronous

stochastic gradient descent which introduces no asynchronous noise via a deferred updating procedure, which was *very* interesting, as most of the theoretical analysis on HOGWILD! centered around showing the noise to be insignificant, and then using the analysis on the stochastic gradient method. There was also another paper by Nguyen et al. [NNvD+18] which managed to show convergence of SGD and therefore HOGWILD! without the bounded gradient assumption, which is obviously a very important relaxation. Finally, I had wanted to implement more types of optimization problems, but I am somewhat uncomfortable with the field as I've never taken a machine learning course before; I intend to self-study a bit and come back to implement a classication style HOGWILD!. Finally, I wanted to explore GPU implementations, but I read a couple of references citing that it wasn't trivial to get large speedups with a GPU, and I decided to leave it till later.

Overall, I'm actually a little bit dissapointed in my progress for this project. I don't want to blame it all entirely on COVID-19[4], as I did lose large chunks of time to failed directions of research in this. For example, the k-banded matrix analysis from the convergence section, was initially spawned from banded matrices arising from finite difference discretizations of PDEs, and I had wanted to try and tackle them from an optimization perspective. But in hindsight this is totally silly, as HOGWILD! is a noisy approximation to a noisy algorithm, it was going to be hard to converge to any real smooth solution, without either running into a noise ball, or running for an absurd number of epochs, destroying the purpose of HOGWILD! anyway. In the end, such matrices are already well-conditioned, so there's no need for this. I had wondered if stochastic DEs would have any applications, but I don't know anything about them and I'm already late so I skipped it. Anyway, I spent way longer than I care to admit on the PDE direction. In addition, about midway through this project my HPC account on Prince here at NYU expired, which is why my tests are only run on a 16 thread machine, which is my home desktop which I built not too long ago. Fighting with the support deparment to stop disabling my Courant account has been problematic my entire undergraduate here, but I think my scaling results were clear anyway.

[4]All of my other classes transitioned to a week-long take-home type of final, which was also made extremely difficult to combat the fact that we could use our notes. I actually think I learned a lot more this way, but the 7 days I spent on my basic probability exam, despite managing an A, was one of the more desperate time of my college life...

References

[BCN16] Léon Bottou, Frank E. Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning, 2016.

[CCA+09] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4):547–553, 2009.

[JP04] Jean Jacod and Philip Protter. *Probability Essentials*. Springer Berlin Heidelberg, 2004.

[NNvD+18] Lam M. Nguyen, Phuong Ha Nguyen, Marten van Dijk, Peter Richtárik, Katya Scheinberg, and Martin Takáč. Sgd and hogwild! convergence without the bounded gradients assumption, 2018.

[NRRW11] Feng Niu, Benjamin Recht, Christopher Re, and Stephen J. Wright. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent, 2011.

[PLT+16] Xinghao Pan, Maximilian Lam, Stephen Tu, Dimitris Papailiopoulos, Ce Zhang, Michael I. Jordan, Kannan Ramchandran, Chris Re, and Benjamin Recht. Cyclades: Conflict-free asynchronous machine learning, 2016.

[Rec14] Benjamin Recht. HOGWILD! for machine learning on multicore. https://youtu.be/l5JqUvTdZts, June 2014.

[SZOR15] Christopher De Sa, Ce Zhang, Kunle Olukotun, and Christopher Ré. Taming the wild: A unified analysis of hogwild!-style algorithms, 2015.