LOCO: Distributing Ridge Regression with Random Projections

Brian McWilliams[†*] **Christina Heinze**[‡*] **Nicolai Meinshausen**[‡]

Gabriel Krummenacher[†] **Hastagiri P. Vanchinathan**[†]

[†]Department of Computer Science [‡]Seminar für Statistik
ETH Zürich, Switzerland
heinzec@student.ethz.ch
meinshausen@stat.math.ethz.ch
{mcbrian|gabriel.krummenacher|hastagiri}@inf.ethz.ch

Abstract

We propose LOCO, a distributed algorithm which solves large-scale ridge regression. LOCO randomly assigns variables to different processing units which do not communicate. Important dependencies between variables are preserved using random projections which are cheap to compute. We show that LOCO has bounded approximation error compared to the exact ridge regression solution in the fixed design setting. Experimentally, in addition to obtaining significant speedups LOCO achieves the same predictive accuracy as standard ridge regression.

1 Introduction

In the last few years there has been great interest in solving large-scale optimization and estimation problems. Since multi-core architectures and powerful computing clusters have become commonplace, parallelization has naturally emerged as a technique to leverage these resources to enable increasingly large problems to be solved quickly.

Two obvious questions arise: (1) how should the data and processing tasks be distributed among processing units (workers) and (2) how and what should each worker communicate. The choice of learning algorithm affects both of these points. Stochastic gradient descent (SGD) methods are suited to parallelization over the rows (observations) of the data [24]. However, synchronization of results to ensure each worker is updating the current gradient becomes expensive. This has motivated recent asynchronous approaches to parallel SGD [8, 15]. Parallelizing over the columns (features) of the data has been proposed for coordinate descent optimization which is commonly used to solve ℓ_1 penalized problems [7]. To ensure convergence, the inherent dependence between features is typically assumed to be small between blocks which are operated on in parallel. Fewer methods have been proposed for distributed optimization on a cluster of machines where communication costs must be considered [17].

Specifically, we limit our focus to ℓ_2 penalized linear regression for large-scale estimation tasks when the size of the data is such that a single multi-core processor is insufficiently fast. We therefore wish to distribute the problem in a way which allows computation to be shared across many machines which do not share memory. Ideally this is done in such a way that synchronization and

[*]Joint first author.

communication between workers are kept to a minimum. Another setting which motivates distributed estimation is one of privacy preservation. In this framework, no single worker may see all of the features and so even when the data size is not massive, sharing memory and data between workers is not permitted.

Randomized dimensionality reduction based on the Johnson-Lindenstrauss lemma has emerged as a way to quickly obtain provably good approximations to a variety of learning tasks [3]. Notably, structured random projections have been used to speed up approximate kernel expansions [12] and linear regression. For the latter, it can be shown that the least squares solution computed on a random projection of either the row [14] or column [11, 13] space of the data matrix results in a solution which is close to optimal. An obvious downside to dimensionality reduction is that the solution obtained is no longer in the original space. Therefore, the estimated coefficients are difficult to interpret with respect to the observed features – a task often as important as prediction accuracy. Furthermore, in order to compute the projection, a single machine is assumed to have access to the entire dataset.

In this work we propose and analyze LOCO, a simple, LOw-COmmunication distributed algorithm to approximately solve ℓ_2 penalized least squares which crucially requires no synchronization between workers. LOCO assigns features to workers by randomly partitioning the data into K blocks (alternatively, this may be part of the problem specification). In each block, a small number of cheaply computed random projections are used to approximate the contribution from the remaining columns of the data. Each worker then simply optimizes the objective independently on this compressed dataset, the size of which is proportional to the size of the random projection and the total number of workers. The solution vector returned by LOCO is constructed by collecting the estimates for the respective unprojected "raw" features from each worker such that it lies in the original space.

Outline and Contribution. In §2 we formally describe our estimation problem and the distributed setting which we consider. We also give a brief introduction to random projections, in particular the Subsampled Randomized Hadamard Transform (SRHT). In §3 we describe LOCO, our algorithm for distributed ridge regression. In §4 we show in the fixed design setting that the error between the coefficients estimated by LOCO and the optimal ridge regression coefficients is bounded, under natural assumptions about the problem setting. Importantly, unlike other approaches to parallelizing or distributing optimization, we make *no* assumptions on sparsity in the data. In §5 we place our contribution in context of recently proposed related approaches to distributed optimization. In §6 we provide implementation details and empirical evaluation of our algorithm on a large-scale simulated dataset. LOCO typically exhibits near-linear speedups with the number of workers with little loss in prediction accuracy.

2 Problem Setting

Given a matrix of features $\mathbf{X} \in \mathbb{R}^{n \times p}$ and a corresponding vector of responses, $Y \in \mathbb{R}^n$ where the dimensionality p and sample size n are very large. We consider solving problems of the form

$$\min_{\boldsymbol{\beta} \in \mathbb{R}^p} L(\boldsymbol{\beta}) := \min_{\boldsymbol{\beta} \in \mathbb{R}^p} f(\boldsymbol{\beta}) + \mathcal{R}(\boldsymbol{\beta}) \tag{1}$$

Here, f is a smooth convex loss and $\mathcal{R}$ is a convex regularizer which is seperable, i.e. $\mathcal{R}(\boldsymbol{\beta}) = \sum_{j=1}^{p} r_j(\boldsymbol{\beta})$. In this work we will concentrate on the case of ridge regression, where $f(\boldsymbol{\beta}) = n^{-1}\|Y - \mathbf{X}\boldsymbol{\beta}\|^2$ is the squared error loss and $\mathcal{R}(\boldsymbol{\beta}) = \lambda\|\boldsymbol{\beta}\|^2$ is the ridge penalty.[1] Ridge regression has a closed-form solution $\widehat{\boldsymbol{\beta}}^{\text{rr}} = \left(\mathbf{X}^\top\mathbf{X} + n\lambda\mathbf{I}\right)^{-1}\mathbf{X}^\top Y$, but clearly when the dimensionality of the data is large, constructing and inverting the covariance matrix is prohibitively expensive. When the number of samples is very large, ridge regression is usually solved using stochastic gradient descent (SGD) or coordinate descent [19].

Distributing ridge regression. We now consider the case where we distribute the features across K different workers. Formally, let $\mathcal{P}$ be the set of indices $1, \ldots, p$. We partition this into K non-overlapping subsets $\mathcal{P}_1, \ldots, \mathcal{P}_K$ of equal size, $\tau = p/K$ so $\mathcal{P} = \bigcup_{k=1}^{K} \mathcal{P}_k$ and $|\mathcal{P}_1| = |\mathcal{P}_2|, \ldots, = |\mathcal{P}_K| = \tau$. This is for simplicity of notation only, in general the partitions can be of different sizes.

[1] Throughout, $\|\cdot\|$ refers to the Euclidean norm for vectors and the spectral norm for matrices.

A naive attempt at parallelizing (1) would simply be solving the minimization problem on each subset of features $\mathcal{P}_k$ independently. However, without sparsity in the dataset to guide the partitioning process, important dependencies between features in different blocks would not be preserved.

We can rewrite (1) making explicit the contribution from block k. Letting $\mathbf{X}_k \in \mathbb{R}^{n\times\tau}$ be the sub-matrix whose columns correspond to the coordinates in $\mathcal{P}_k$ (the "raw" features of block k) and $\mathbf{X}_{(-k)} \in \mathbb{R}^{n\times(p-\tau)}$ be the remaining columns of $\mathbf{X}$, we have

$$L(\boldsymbol{\beta}) = n^{-1}\|Y - \mathbf{X}_k\boldsymbol{\beta}_{\text{raw}} - \mathbf{X}_{(-k)}\boldsymbol{\beta}_{(-k)}\|^2 + \lambda\|\boldsymbol{\beta}_{\text{raw}}\|^2 + \lambda\|\boldsymbol{\beta}_{(-k)}\|^2. \tag{2}$$

The idea behind our approach is to replace $\mathbf{X}_{(-k)}$ in each block with a low-dimensional approximation. Since the regularizer is separable across blocks, we only require that the contribution from $\mathbf{X}_{(-k)}\boldsymbol{\beta}_{(-k)}$ to $f(\boldsymbol{\beta})$ is preserved.

Let $\tilde{\mathbf{X}}_k \in \mathbb{R}^{n\times(K-1)\tau_{subs}}$ be the matrix whose columns are a low-dimensional approximation to $\mathbf{X}_{(-k)}$, i.e. to the columns of $\mathbf{X}$ not in $\mathbf{X}_k$, and $\tau_{subs} \ll \tau$. Note that each of the other $K-1$ blocks provides an approximation of its respective τ raw features of size τ_{subs}. We shall call the columns in $\tilde{\mathbf{X}}_k$ the "random" features of block k. Defining the sub-problem that worker k solves as

$$L_k(\boldsymbol{\beta}_k) = n^{-1}\|Y - \mathbf{X}_k\boldsymbol{\beta}_{\text{raw}} - \tilde{\mathbf{X}}_k\boldsymbol{\beta}_{k,\text{rp}}\|^2 + \lambda\|\boldsymbol{\beta}_{\text{raw}}\|^2 + \lambda\|\boldsymbol{\beta}_{k,\text{rp}}\|^2, \tag{3}$$

we require the approximation $\tilde{\mathbf{X}}_k$ to be such that the risk of the estimator which minimizes eq. (3) is similar to the risk of the minimizer of eq. (2) (we formalize this in §4). In order to achieve this we construct the approximation using random projections which we briefly describe below.

Johnson-Lindenstrauss projections. Johnson-Lindenstrauss (J-L) projections are low-dimensional embeddings $\boldsymbol{\Pi} : \mathbb{R}^{\tau} \to \mathbb{R}^{\tau_{subs}}$ which preserve – up to a small distortion – pairwise ℓ_2 distances between vectors according to the J-L lemma (see e.g. [3]). Typically, $\boldsymbol{\Pi} \in \mathbb{R}^{\tau\times\tau_{subs}}$ is constructed to be a nearly-orthogonal matrix with entries drawn at random from a sub-gaussian distribution [1].

We concentrate on the class of *structured* random projections, among which the Subsampled Randomized Hadamard Transform (SRHT) has received particular recent attention [6, 20]. The SHRT consists of a preconditioning step after which τ_{subs} columns of the new matrix are subsampled uniformly at random. In more detail, it consists of a projection matrix, $\boldsymbol{\Pi} = \sqrt{\tau/\tau_{subs}}\mathbf{DHS}$ [6,9] with the definitions:

- $\mathbf{S} \in \mathbb{R}^{\tau\times\tau_{subs}}$ is a subsampling matrix.
- $\mathbf{D} \in \mathbb{R}^{\tau\times\tau}$ is a diagonal matrix whose entries are drawn independently from $\{-1, 1\}$.
- $\mathbf{H} \in \mathbb{R}^{\tau\times\tau}$ is a normalized Walsh-Hadamard matrix[2] which is defined recursively as

$$\mathbf{H}_\tau = \begin{bmatrix} \mathbf{H}_{\tau/2} & \mathbf{H}_{\tau/2} \\ \mathbf{H}_{\tau/2} & -\mathbf{H}_{\tau/2} \end{bmatrix}, \quad \mathbf{H}_2 = \begin{bmatrix} +1 & +1 \\ +1 & -1 \end{bmatrix}.$$

  We set $\mathbf{H} = \frac{1}{\sqrt{\tau}}\mathbf{H}_\tau$ so it has orthonormal columns.

The SRHT has similar ℓ_2 distance preserving properties as sub-gaussian random projections but has the added benefit of a fast $O(\tau \log \tau)$ matrix-vector product due to its recursive definition. Note that the SRHT can also be combined with an i.i.d. Gaussian random vector to obtain a Gaussian matrix with approximately independent entries in the same computational time [12].

For moderately sized problems, random projections have been used to reduce the dimensionality of the data prior to performing OLS [11] and ridge regression [13]. However after projection, the solution vector is in the compressed space and so interpretability of coefficients is lost. Furthermore, for large problems the running time of the SRHT presents a large constant overhead.

3 Algorithm

Our procedure LOCO for distributed ridge regression is presented in Algorithm 1. We describe the steps in more detail below.

[2]For the Hadamard transform, τ must be a power of two but other transforms exist (e.g. DCT, DFT) with similar theoretical guarantees and no restriction on τ.

Algorithm 1 LOCO

Input: Data: $\mathbf{X}$, Y, Number of blocks: K, Parameters: τ_{subs}, λ

1: Partition $\mathcal{P} = \{1, \ldots, p\}$ into K subsets $\mathcal{P}_1, \ldots, \mathcal{P}_K$ of equal size, τ.
2: **for each** worker $k \in \{1, \ldots K\}$ **in parallel do**
3: Compute and send random projection $\widehat{\mathbf{X}}_k = \mathbf{X}_k \mathbf{\Pi}_k$.
4: Construct $\bar{\mathbf{X}}_k$
5: $\bar{\boldsymbol{\beta}}_k \leftarrow \texttt{SolveRidge}(\bar{\mathbf{X}}_k, Y, \lambda)$
6: $\widehat{\boldsymbol{\beta}}_k = \left[\bar{\boldsymbol{\beta}}_k\right]_{1:\tau}$
7: **end for**

Output: Solution vector: $\widehat{\boldsymbol{\beta}} = \left[\widehat{\boldsymbol{\beta}}_1, \ldots, \widehat{\boldsymbol{\beta}}_K\right]$

Input. As well as the usual regularization parameter λ, LOCO requires the specification of the number of workers K and the random projection dimension τ_{subs}.

Steps 1 & 3. We first randomly partition the coordinates into K subsets. Then each worker computes a random projection, via the SRHT, of its respective block which we denote by $\widehat{\mathbf{X}}_k = \mathbf{X}_k \mathbf{\Pi}_k \in \mathbb{R}^{n \times \tau_{subs}}$.

Step 4. Each worker k constructs the matrix

$$\bar{\mathbf{X}}_k \in \mathbb{R}^{n \times (\tau + (K-1)\tau_{subs})} = \left[\mathbf{X}_k, \left[\widehat{\mathbf{X}}_{k'}\right]_{k' \neq k}\right]$$

which is the column-wise concatenation of the raw feature matrix $\mathbf{X}_k$ and the random approximations from all other blocks.[3] Without loss of generality the raw features will always occupy the first τ columns of $\bar{\mathbf{X}}_k$. The last $(K-1)\tau_{subs}$ columns of $\bar{\mathbf{X}}_k$ are a good approximation of the $(K-1)$ blocks of the full data matrix not in $\mathbf{X}_k$ and so solving (1) using $\bar{\mathbf{X}}_k$ obtains a solution which is close to the optimal solution using $\mathbf{X}$. We make this explicit in §4.

Steps 5 & 6. The function $\texttt{SolveRidge}(\bar{\mathbf{X}}_k, Y, \lambda)$ returns a vector

$$\bar{\boldsymbol{\beta}}_k \in \mathbb{R}^{\tau + (K-1)\tau_{subs}} = \arg\min_{\boldsymbol{\beta}_k} n^{-1} \|Y - \bar{\mathbf{X}}_k \boldsymbol{\beta}_k\|^2 + \lambda \|\boldsymbol{\beta}_k\|^2 \tag{4}$$

In practice, any fast algorithm which returns an accurate solution to eq. (4) can be used here. The final solution vector $\widehat{\boldsymbol{\beta}}$ is the concatenation of the first τ coordinates of each $\bar{\boldsymbol{\beta}}_k$ and so lives in the same space as the original data.

Computational, memory and communication costs. The cost of computing a fast random projection of the τ features in each block is $O(\tau \log \tau_{subs})$. Assuming a solver which scales linearly with the problem dimension (i.e. stochastic gradient descent) is used in $\texttt{SolveRidge}(\bar{\mathbf{X}}_k, Y, \lambda)$, the part of the computational cost which is dependent on the dimension scales with $O(\tau \log \tau_{subs} + \tau + (K-1)\tau_{subs})$.

Each machine only needs to store a copy of its block of raw features and a random projection of the remaining features which is $O(\tau + (K-1)\tau_{subs})$. This is substantially smaller than the original dimensionality p. Each worker must communicate its random projection once to all other workers (or to a shared location where the other workers can read it). Aside from this there is no further communication between workers. The small size of the projection ensures that for appropriately sized problems, each worker is able to store its relevant features in local memory.

4 Analysis

Theorem 1, presented in this section, states that in the fixed design setting the coefficients estimated by LOCO are close to the full ridge regression solution. This result applies to the case where the

[3] Alternatively, when $\mathbf{\Pi}$ is defined explicitly, summing the $\tau \to \tau_{subs}-$dimensional random projections of $K-1$ blocks is equivalent to computing the $(p-\tau) \to \tau_{subs}-$dimensional random projection in one go which allows for a much smaller dimensional but also less accurate representation.

random features in $\tilde{\mathbf{X}}_k$ result from concatenating the SRHT projections of all other blocks and throughout we shall assume that the columns of $\mathbf{X}$ and $\bar{\mathbf{X}}_k$ are standardized.

Consider the model

$$Y = \mathbf{X}\boldsymbol{\beta}^* + \boldsymbol{\varepsilon}, \tag{5}$$

with fixed $\mathbf{X} \in \mathbb{R}^{n\times p}$ and true parameter vector $\boldsymbol{\beta}^* \in \mathbb{R}^p$. The errors ε_i, $i = 1, \ldots, n$ have zero mean, are independent and their variances are bounded by $\sigma^2 > 0$. Let $\widehat{\boldsymbol{\beta}}^{\text{rr}}$ denote the ridge estimate for $\boldsymbol{\beta}^*$, so $\widehat{\boldsymbol{\beta}}^{\text{rr}}$ is the solution which results from solving the ridge regression problem in the original space, stated in eq. (2).

In order to formulate our result, we require the following risk function.

Definition 1 (Risk)**.** *Let $\widehat{\mathbf{b}}$ be an estimator for $\boldsymbol{\beta}^*$ and define the risk of $\widehat{\mathbf{b}}$ with fitted values $\widehat{Y} = \mathbf{X}\widehat{\mathbf{b}} \in \mathbb{R}^n$ as*

$$R(\mathbf{X}\widehat{\mathbf{b}}) = n^{-1}\mathbb{E}_\varepsilon\|\mathbf{X}\boldsymbol{\beta}^* - \mathbf{X}\widehat{\mathbf{b}}\|^2.$$

Before we state our main theorem, we make the natural assumption that the main contribution to the ℓ_2 norm of the true parameter vector – i.e. most of the important signal – lies in the direction of the first J principal components of $\mathbf{X}$. This merely formalizes the conditions under which ridge regression yields good results. Since ridge regression applies more shrinkage in directions associated with smaller eigenvalues [10], if this assumption does not hold we might expect a different estimator to be more appropriate. Furthermore, we shall assume that the ridge constraint is active.

We now present Theorem 1 which states that the expected difference between the coefficients $\widehat{\boldsymbol{\beta}}$ returned by LOCO and the full ridge regression solution is bounded.

Theorem 1. *Under mild assumptions on the problem setting, $\exists\, n_0(\xi)$ for all $\xi > K(\delta + (p-\tau)/e^r)$ such that for all $n \geq n_0$ with probability at least $1 - \xi$*

$$\mathbb{E}_\varepsilon(\|\widehat{\boldsymbol{\beta}}^{rr} - \widehat{\boldsymbol{\beta}}\|^2) \leq \frac{5K}{c\lambda_J}\left(\frac{1}{(1-\rho)^2} - 1\right) R(\mathbf{X}\widehat{\boldsymbol{\beta}}^{rr})$$

where $\rho = C\sqrt{\frac{r\log(2r/\delta)}{(K-1)\tau_{subs}}}$, $r = rank(\mathbf{X})$, λ_J denotes the J^{th} largest non-zero eigenvalue of the covariance matrix and $R(\mathbf{X}\widehat{\boldsymbol{\beta}}^{rr})$ is the risk of the ridge estimator (see e.g. [13]). The expectation is conditional on the random projection as the uncertainty coming from the SRHT is captured in the probability with which the statement holds.

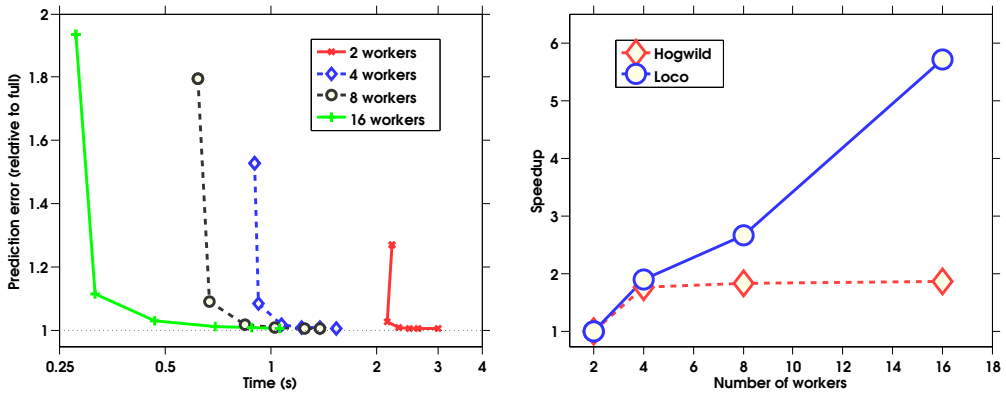
The bound above scales with the number of workers, K and inversely with $(1-\rho)^2$, which measures the quality of the random feature representation. The latter is improved (for a fixed τ_{subs}) by increasing K although this has the additional effect of increasing the computational overhead per worker which scales as $O\left((K-1)\tau_{subs}\right)$.

5 Related work

Recently, several methods have been proposed for parallelizing convex optimization. Among these, HOGWILD! [15], ASYNCDA and ASYNCADAGRAD [8] have shown that large speedups are possible with asynchronous gradient updates when data is sparse. These methods rely on the idea that if the number of non-zero coordinates in each stochastic gradient evaluation is small compared to p, workers updating the same solution vector in parallel will rarely propose conflicting updates. As such each worker is allowed to update the solution asynchronously without the need for locking, provided the delay of any processor is not too great.

Similar requirements on sparsity have been used to parallelize coordinate descent [17]. Other approaches rely on alternative, but related conditions which require the spectral norm of the data – which captures the size of dependencies between features – to be small [7, 18].

Whilst sparsity is a natural and common feature of large datasets, in some fields the data collected is dense with many correlated features. Furthermore, in the setting where $n \ll p$, SGD can converge slowly. Under these conditions we might expect the performance of the above mentioned approaches to suffer. Notably, LOCO makes no assumptions about sparsity since each block sees a representation



(a) (b)

Figure 1: (a) relative prediction error against time and (b) relative speedup comparison between LOCO and HOGWILD! for $p = 131,072$.

of the remaining features such that updates to the individual solution vectors are not independent of the rest of the dataset. LOCO does not require synchronization between workers since each worker may only update its own part of the solution vector. Although a bound on the maximum delay is not required or assumed, the total running time of LOCO will naturally be dictated by the slowest worker.

Most of the above mentioned approaches implement parallelism on a multi-core architecture with shared memory. For distributed optimization, communicating results between workers introduces overhead. Several communication strategies for distributed coordinate descent were discussed in [17]. In contrast, LOCO requires computing random projections for each block and communicating them once and so no additional synchronization or communication are required until prediction time.

Parallel estimation has also been considered in the case of kernel ridge regression [23]. It has been shown that randomly splitting and distributing the samples among workers and averaging their estimates achieves a superlinear speedup whilst retaining optimality up to a number of workers which is problem dependent. Conceptually, LOCO is perhaps most similar to this approach, although clearly the usual i.i.d. assumption on the observations does not hold for partitioning the features.

6 Experimental Results

Implementation details. We implemented LOCO in Python making use of fast packages for random projections and ridge regression. For the random projection we used the DCT implemented in FFTW[4] and the ridge regression solver is implemented in `scikit-learn` which outperforms SGD for $p \gg n$. We ran LOCO on the BRUTUS cluster[5] where each worker "communicates" by writing its random projections to a Lustre file system to which each worker is connected via InfiniBand network and as such enables very fast simultaneous reads and writes. We consider each machine to be a worker and since the cluster is heterogeneous we do not exploit any further local parallelization. For comparison, we modified the loss and gradient computations of HOGWILD![6] to perform ridge regression and ran it on a single Xeon E3-1275 V with 32GB of RAM (4 cores with $2\times$ hyper-threading, a comparable but slightly slower than the test machine of [15]). The difference between setups makes a comparison of absolute timing inaccurate but relative speedup for each method is comparable.

Simulated data. We consider a large-scale simulated problem. The data is generated from a Gaussian distribution with mean zero and a block-wise covariance matrix such that the features are *not* independent and the block structure is not known to the algorithm *a priori*. Specifically, the pairwise

[4] http://www.fftw.org

[5] `http://en.wikipedia.org/wiki/Brutus_cluster`

[6] Code available from: `http://hazy.cs.wisc.edu/hazy/victor/Hogwild/`

within-block correlation is 0.8. The matrix of simulated data is fully dense, so there is no sparsity that could be exploited for speedups which directly handicaps HOGWILD!.

The scenario we consider is $n = 1000$, $p = 131,072$ ($200M$ non-zeros, testing set is the same size as the training set) and has rank $r = 20$. The purpose of this experiment is to compare LOCO against HOGWILD! in a setting where memory is not a limiting factor and so HOGWILD! can comfortably parallelize the problem and run on a single machine. We aim to compare the speedup of both methods when the data is dense. The behaviour of HOGWILD! should also be similar to the behaviour of ASYNCDA [8].

According to Theorem 1, increasing τ_{subs} will improve the prediction error but also increase the computational time. To give a meaningful comparison between the methods we vary τ_{subs} and report the time spent training the model and the prediction error achieved in figure 1(a). Since for different number of workers, $\tau = p/K$ is different, the random projection dimension, τ_{subs} is chosen relative to τ, i.e. $0 \leq \tau_{subs} \leq 0.1\tau$. These statistics are averaged over 5 trials and are shown for $K = \{2, 4, 8, 16\}$. As a baseline we indicate the results for a standard ridge regression solver (i.e. $K = 1$). The prediction error is normalized by this result to obtain relative prediction error.

Figure 1(a) shows that a test error comparable to full ridge regression (horizontal dotted line at 1) is obtained with very low-dimensional random projections, even for $K = 16$. The largest error for each setting occurs when $\tau_{subs} = 0$, i.e. the dataset is partitioned but no random features are added, and increases as K increases. This highlights the importance of accounting for dependencies between blocks of features and justifies our random projection approach to distributing features across workers.

Figure 1(b) compares the relative speedup for increasing K for LOCO and HOGWILD!. HOGWILD! exhibits linear speedup for up to 4 threads but no speedup when more are added. LOCO exhibits an almost linear speedup with the number of workers – with $K = 16$ we obtain a $6\times$ speedup over $K = 2$. The absolute running time of LOCO was faster and the timings for HOGWILD! ignore a large constant overhead for file loading.

We must stress that this experiment is designed so that the theoretical assumptions underpinning HOGWILD! are violated. Clearly if the data were very sparse, HOGWILD! would exhibit large speedups as investigated in [8, 15] . However, for very large datasets it would still be limited by the number of cores and amount of memory available to a single machine. In such scenarios it may be advantageous to combine LOCO and HOGWILD!.

7 Discussion

In this work we have presented LOCO, a simple algorithm for distributed ridge regression – requiring minimal communication and no synchronization – based on random projections. We have shown theoretically and empirically that LOCO achieves small additional error compared with the optimal ridge regression solution. It obtains near linear speedups with the number of workers without making any additional assumptions about sparsity in the data. Furthermore, in principle any fast ridge regression solver can be used in conjunction with LOCO to further speed up learning in each individual worker according to whichever computing architecture or data assumptions apply. It should be noted that LOCO is a complementary rather than competing approach to methods such as HOGWILD!. In particular, a large but sparse problem could be solved on a cluster of multi-core machines extremely quickly by combining LOCO with HOGWILD!.

Further Work. Although currently our results are specific for ridge regression, we expect that the same principles can be generalized to other convex optimization problems.

As mentioned in the introduction, distributed optimization – where no single worker sees all of the data – is a natural paradigm when preserving privacy is required. Additionally, the class of J-L projections that we use have been shown to preserve differential privacy [5]. We aim to explore the connection between LOCO and privacy aware learning.

Finally, [22] established bounds on the minimum amount of communication necessary for a distributed estimation task to achieve minimax optimal risk. It would be interesting investigate how LOCO fits into this framework.

References

[1] D. Achlioptas. Database-friendly random projections: Johnson-Lindenstrauss with binary coins. *Journal of Computer and System Sciences*, 2003.

[2] Alekh Agarwal and John C Duchi. Distributed delayed stochastic optimization. In *NIPS*, pages 873–881, 2011.

[3] Nir Ailon and Bernard Chazelle. The fast johnson-lindenstrauss transform and approximate nearest neighbors. *SIAM Journal on Computing*, 39(1):302–322, 2009.

[4] Francis Bach. Sharp analysis of low-rank kernel matrix approximations. *arXiv preprint arXiv:1208.2015*, 2012.

[5] Jeremiah Blocki, Avrim Blum, Anupam Datta, and Or Sheffet. The johnson-lindenstrauss transform itself preserves differential privacy. In *Foundations of Computer Science (FOCS), 2012 IEEE 53rd Annual Symposium on*, pages 410–419. IEEE, 2012.

[6] Christos Boutsidis and Alex Gittens. Improved matrix algorithms via the Subsampled Randomized Hadamard Transform. 2012. arXiv:1204.0062v4 [cs.DS].

[7] Joseph K. Bradley, Aapo Kyrola, Danny Bickson, and Carlos Guestrin. Parallel coordinate descent for l_1-regularized loss minimization. In *International Conference on Machine Learning*, 2011.

[8] John C. Duchi, Michael I. Jordan, and H. Brendan McMahan. Estimation, optimization, and parallelism when data is sparse. In *Advances in Neural Information Processing Systems*, 2013.

[9] N Halko, P G Martinsson, and J A Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):217–288, 2011.

[10] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2009.

[11] Ata Kabán. New bounds on compressive linear least squares regression. In *Artificial Intelligence and Statistics*, 2014.

[12] Quoc Le, Tamas Sarlos, and Alex Smola. Fastfood approximating kernel expansions in loglinear time. In *ICML*, 2013.

[13] Yichao Lu, Paramveer Dhillon, Dean P Foster, and Lyle Ungar. Faster ridge regression via the subsampled randomized hadamard transform. In *Advances in Neural Information Processing Systems 26*, pages 369–377, 2013.

[14] Michael W Mahoney. Randomized algorithms for matrices and data. April 2011. arXiv:1104.5557v3 [cs.DS].

[15] Feng Niu, Benjamin Recht, Christopher Ré, and Stephen J. Wright. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. In *NIPS*, 2011.

[16] Zhimin Peng, Ming Yan, and Wotao Yin. Parallel and distributed sparse optimization. In *Preprint*, 2013.

[17] Peter Richtarik and Martin Takac. Distributed coordinate descent method for learning with big data. In *Preprint*, 2013.

[18] Chad Scherrer, Ambuj Tewari, Mahantesh Halappanavar, and David Haglin. Feature clustering for accelerating parallel coordinate descent. In *Advances in Neural Information Processing Systems (NIPS)*, 2012.

[19] Shai Shalev-Shwartz and Tong Zhang. Stochastic dual coordinate ascent methods for regularized loss. *The Journal of Machine Learning Research*, 14(1):567–599, 2013.

[20] Joel A Tropp. Improved analysis of the subsampled randomized Hadamard transform. November 2010. arXiv:1011.1595v4 [math.NA].

[21] Joel A Tropp. User-friendly tail bounds for sums of random matrices. April 2010. arXiv:1004.4389v7 [math.PR].

[22] Yuchen Zhang, John Duchi, Michael Jordan, and Martin J Wainwright. Information-theoretic lower bounds for distributed statistical estimation with communication constraints. In *Advances in Neural Information Processing Systems*, pages 2328–2336, 2013.

[23] Yuchen Zhang, John C Duchi, and Martin J Wainwright. Divide and conquer kernel ridge regression: A distributed algorithm with minimax optimal rates. *arXiv preprint arXiv:1305.5029*, 2013.

[24] Martin Zinkevich, Markus Weimer, Alexander J Smola, and Lihong Li. Parallelized stochastic gradient descent. In *NIPS*, volume 4, page 4, 2010.