Master Thesis

Communication Filters for Distributed Optimization with the Parameter Server

Yipei Wang

2015 May

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Alexander J. Smola, Chair
Geoffrey J. Gordon
Bhiksha Raj

Abstract

One of the bottlenecks for distributed machine learning system is the huge communication overhead. In this thesis, we explore the filtering idea to tackle this problem and narrow down the scope to study distributed optimization under parameter server framework. We first discuss the intuition in designing filters based on the redundancy hidden in optimization algorithms. We then derived convergence conditions in applying filters. Based on these analysis, we provide efficient algorithms using sampling and randomized rounding techniques. For applying filters in practice, we also discuss strategies in filtering jointly and the scalability influence. The filters are experimentally proved to be able to reduce communication cost significantly without affecting the learning performance.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter introduces the background and motivation of the thesis. In section 1.1, we will introduce the communication bottleneck for distributed learning system and explain the fundamental idea for using filters. In this thesis, we narrow down the scope to explore the idea of filtering for distributed optimization algorithms under the parameter server framework. So an overview of parameter server is given in section 1.2. In section 1.3, I will explain the importance of optimization algorithms in solving machine learning problem and implementations of distributed optimization algorithms under parameter server framework.

1.1 Communication Bottleneck for Distributed Learning System

To solve machine learning problem on massive scale of data , a lot of effort has been spent in designing distributed learning systems, e.g. YahooLDA [1], MLI based on spark[2], GraphLab [3], and parameter server [4] [5] [6]. One common issue for these distributed learning system is huge amount of communication overhead. This is because we leverage the statistics over the distributed data to update the model and update is usually iteratively performed for many times.

However, only limited network bandwidth is available in a data center. A typical networking structure of data center (Figure 1.1) help us understand why the network bandwidth is extremely limited. The communication among machines within local group have maximally 10GB/s network bandwidth, which is much lower than 100GB/s memory bandwidth. The communication among groups have even more limited resource since the 20GB/s network bandwidth are shared by all machines in the connected groups. Moreover, the network bandwidth is usually shared by all running applications. As for cloud computing platform, it become even worse, e.g. we typically get only 1GB/s or worse using spot instances on AWS in practice.

With limited network bandwidth, a large amount of communication overhead would significantly hurt the efficiency of the system. Therefore, the huge communication cost has become one of the main bottlenecks for distributed learning system.

One direction to address the problem is to compress the communicated data. Beyond traditional information lossless compression techniques, we can take advantage of the hidden redundancy in learning algorithms to further reduce the communication cost with lossy compression

methods.

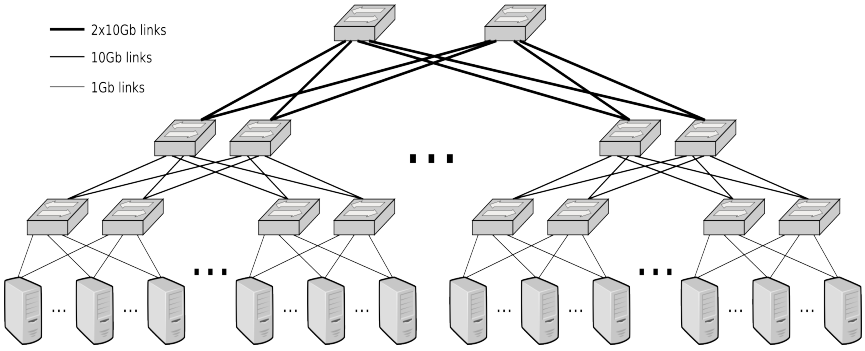


Figure 1.1: Network Structure in data center

1.2 Parameter Server Framework

We will deploy the filtering algorithms on an open sourced parameter server [5]. The reason we pick this system is because it provides a platform to facilitate development of efficient distributed machine learning algorithms. Compared to other systems [6] [7] [1] [8], it is claimed to enjoy much better scalability in terms of both parameters and nodes.

To help the reader better understand the system, I will briefly review its architecture. An instance consists of a server group, many worker nodes and a scheduler node. The scheduler is responsible for assigning workloads and monitoring progress. It organized the servers and workers to collaborate in executing algorithms. Each server node maintains a portion of the shared model parameters. The shared model parameters are stored as sorted (key, value) pairs. Each worker only loads a data partition. It will computes the local statistics and communicate with server nodes to retrieve and update the shared parameters. Data exchange is achieved view pull and push operation. A worker node can push (key, value) pairs to servers, or pull the corresponding values from servers.

One important reason we deploy on this system is because it provides schemes to support efficient communications. We briefly summarize the schemes designed for this purpose.

1. The system offers a scheme to implement flexible consistency model, rather than a specific consistency model [6] [2] [7]. Asynchronous ranged based communication provided in the system can address the performance bottleneck arised from synchronization requirement. A bounded-delay optimization algorithm is designed by taking advantage of this scheme.
2. The system supports applying communication filters when exchanging data. It implements key caching and compression filters, which would not lose any information and can be used for any application. In[9], KKT filter is developed for solving L1 regularized problem. It can significantly reduce the communication cost by leveraging the sparsity induced by L1 regularizer. More details will be covered in chapter 2.

1.3 Distributed Optimization and Implementation on Parameter Server

Distributed optimization is a key tool for solving large-scale machine learning problem[**?**] [**?**] [3] [8]. A large group of machine learning problems can be formulated as a general form:

$$w^* = \operatorname*{argmin}_{w \in \Omega} \phi(w),\ where\ \phi(w) = \frac{1}{n} \sum_{i=1}^{n} \phi_i(w) \tag{1.1}$$

For example, in the popular risk minimization framework, we are seeking for model to minimize the loss function on training data $(x_i, y_i), i = 1, ..., n$ (x_i is feature, y_i is label). The loss function $l(x_i, y_i, w)$ is usually defined by classification error or regression error. We also often include a regularization term $\Omega(w)$ in the objective function to avoid over-fitting. So the problem can be formulated as:

$$w^* = \operatorname*{argmin}_{w} \sum_{i=1}^{n} l(x_i, y_i, w) + \Omega(w) \tag{1.2}$$

After we understand the importance of distributed optimization algorithms, let us see how to implement the algorithms on parameter server. Here we show the implementation of standard distributed subgradient descent algorithm. In each iteration, worker nodes request the recent parameters from server group and compute the local subgradients. Then, they send the subgradients to server group. The server group will aggregate these local gradients and update the parameters. From this example, we can clearly see that there is huge amount of communication among worker nodes and server nodes during one iteration for sending the local gradients and updated model parameters.

Algorithm 1 Distributed Subgradient Descent

Worker r = 1, ..., m :

load training data $\{y_{ik}, x_{ik}\}_{k=1}^{n_r}$
Pull from the servers the model parameters w_r^0
for t = 1 to T **do**
    Compute local gradient $g_r^t = \sum_{k=1}^{n_r} \partial l(x_{ik}, y_{ik}, w_r^{(t)})$
    push g_r^t to servers
    pull w_r^{t+1} from servers
end

Servers:

for t=1 to T **do**
    aggregate $g^t \leftarrow \sum_{r=1}^{m} g_r^t$
    $w^{t+1} \leftarrow w^t - \eta(g^t + \partial\Omega(w^{(t)}))$
end

1.4 Bounded Delay Blockwise First-order Optimization Algorithm

The optimization algorithm we experiment with is outlined in Algorithm 2, which is initially proposed in [9]. This algorithm takes advantage of the opportunities provided by the parameter server framework in handling high-dimensional sparse data. It differs from standard distributed optimization algorithms in several perspectives.

1. Asynchronous update over iterations are allowed with a bounded delay constrain. This significantly reduce the time wasted on synchronization in each iteration without affecting the convergence.
2. it adopts blockwise gradient. This makes is easier to adjust the learning rate for high dimensional data. Also, the blockwise update is less insenstive to the influence of asynchronous update.
3. the clients compute both gradients and preconditioners. By using preconditioner to reweight the learning rate, it significantly speed up the convergence of first-order method.

Algorithm 2 Delayed Block Distributed Blockwise Gradient Method

Client r = 1, ..., m :
load training data $\{y_{ik}, x_{ik}\}_{k=1}^{n_r}$
pull from the servers the model parameters w_r^0
for t = 1 to T **do**
  Compute local gradient g_r^t and local preconditioner u_r^t
  push g_r^t, u_r^t to servers
  pull w_r^{t+1} from servers
end
Servers :
for t=1 to T **do**
  aggregate $g^t \leftarrow \sum_{r=1}^{m} g_r^t$
  aggregate $u^t \leftarrow \sum_{r=1}^{m} U_r^t$
  Regularized update to compute w^{t+1}
end

The regularized update step in the Algorithm is performed on servers depends on the regularization term.

For L1 norm, we applies projected gradient algorithm:

$$w^{t+1} = Prox_{\gamma}[w^t - \gamma * g^t]$$

where

$$Prox_{\gamma}[x] = argmin_y h(y) + \frac{1}{2\gamma}\|x - y\|^2$$

Here $h(y) = \|y\|_1$ and the projection is realized by soft-thresholding. γ is the reweighted learning rate by leveraging the preconditioner: $\gamma = \eta[u^t]^{-1}$, where η is a fixed learning rate.

For L2 norm, we applies standard gradient descent update:

$$w^{t+1} = w^t - \gamma(g^t + 2 * \lambda * w^t)$$

$$\gamma = \eta[u^t + 2 * \lambda]^{-1}$$

Chapter 2

Related Work

2.1 Prior work for reducing communication cost

There are mainly two directions to tackle the problem. One is to modify the algorithm so that the request for data exchanging is reduced [10] [11] [6]. These methods are usually designed for specific problems. Another direction is to compress the communicated data. In general purpose distributed system, the lossless compression technique is widely used for the communication. Learning algorithms usually possess additional information about communicated data, which can be used for lossy compression. Pioneered by [5], they proposed filters designed by heuristics and demonstrate their efficacy. Compared to the first direction, the filtering method is easier to be generalized to various algorithms. In this thesis, we will continue the exploration of designing filters.

2.2 Communication Filters

We will introduce the filters already developed in parameter server. This part helps the reader get a full view of communication filters and understand the contribution of the thesis.

The filters can be categorized into lossless filters and lossy filters. Lossless filters can be applied in general distributed applications. Here Key caching and snappy compression techniques are viewed as lossless filters. Details about these filters are discussed in the rest of this section.

2.2.1 Lossless Filters

Key Caching Filter (Lossless) When sending parameters, a range of (key, value) pairs need to be communicated. It is likely that the range of keys are shared among many communications and only the values are changed. If so, the receiver can cache the keys so that the sender only send the values with a key signature. Avoiding sending keys can double the network bandwidth.
Snappy Compression Filter (Lossless) It uses a lossless data compression library, Snappy[12], which is widely used in distributed system due to its fast speed. Snappy compress the data by making use of the repeated substring, so it can compress 0x0000 and 0x1111 effectively. When compressing numeric data, smaller dynamic range achieves higher compression rate.

2.2.2 Lossy Filters

Random skip filter It applies on local gradients and random skip some elements. For the remained elements, it reweight by multiply $\frac{1}{q}$, where q is the random skip probability.
Significantly-modified filter It only pushes entries that have changed by more than a threshold since synchronized last time.
KKT filter Inspired by active set methods, KKT filter reduces the range of active parameters by taking advantage of the characteristics of the proximal operator using l1 norm. For L1 regularized problem, the projection is executed by soft-thresholding. Suppose the objective function is in the form $f(w) + \lambda\|w\|_1$, $f(w)$ is differentialable, the proximal operation is described in equation(2) and demonstrated in Fig 2. From the figure, we can see that only big enough updates from the smoothing part of the objective function can finally affect the parameters. Since w is initialized from 0 and the update step should be decreasing for convergent case, heuristic rules can be developed to adaptively select the active set. When we conduct the algorithm only on a subset of features selected by the active set, both the computation load and communication overhead are reduced.

$$Prox(w) = \begin{cases} w - \lambda, & w \in (\lambda, +\infty) \\ 0 & w \in [-\lambda, \lambda] \\ w + \lambda & w \in (-\infty, \lambda) \end{cases} \tag{2.1}$$

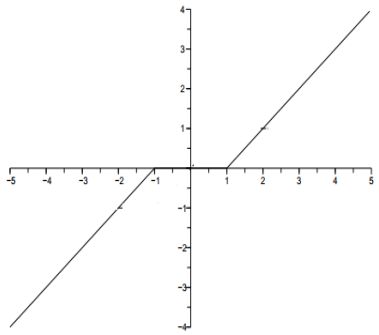


Figure 2.1: Soft-thresholding

[9] also provides a proof for the convergence using significant modified filter and random skip filter in the bounded delay blockwise proximal gradient algorithm. To reach a stationary point in expectation with lipschitz continuity assumption, it requires the significantly-modified filter to pull weights with threshold $O(1/t)$ and the random skip filter generates unbiased estimation. However, there is lack of general principles for designing filters. We plan to address the issue in chapter 3.

Chapter 3

Designing Lossy Communication Filters

Prior work only provides several examples of user specified filters. There is lack of general principles to guide the design and use of lossy filters. Therefore, we seek to address this problem in this thesis.

In section 3.1, we analyze the communicated data for general optimization algorithms and the redundancy hidden in the data. In section 3.2, we introduce the intuition for designing filters and investigate their possibility by analyzing the characteristics of various types of data. In section 3.3, we derive the conditions for convergence of a class of first order algorithms when filters applied. Based on the analysis, we also discuss about several issues related with applying filters in practice in the end, including the influence of scalability and filtering jointly.

3.1 Data Exchange Analysis

To analyze the common properties in data exchange over various distributed optimization algorithms, we introduce a unified framework . Many optimization algorithms can be viewed as solving sequential subproblems[13] in equation 3.1. The intuition is to use a quadratic model to approximate the true objective function within a small region. The quality of the approximation would influence the progress we make in each step and thus the convergence speed.

$$w_{t+1} = \operatorname*{argmin}_{w \in \Delta} f(w_t) + b^T(w - w_t) + \frac{1}{2}(w - w_t)^T A(w - w_t) \tag{3.1}$$

Table 1 summarized the parameters of the quadratic model for various standard optimization algorithms. The quadratic model is constructed by using first/second order Taylor expansion. Calculating and storing the inverse of hessian matrix directly is extremely expansive for high dimensional problem. Therefore, various quasi-Newton methods are studied to approximate the Hessian (or its inverse directly) by building up from changes in historical gradients.

In the parameter server framework, data is distributed over workers and the model parameters are maintained by the server group. So when solving the above subproblem in a distributed version on parameter server, two types of data flow is unavoidable: (1) each client has to access to the recent model parameters w_t. (2) the local statistics to characterize the quadratic model. That is, if A or b is a function over data, the server group has to collect the function value

	A	b
Subgradient Descent	$diag[\eta_t]$	$\partial f(w)$
Projected Gradient Descent	$diag[\eta_t]$	$\bigtriangledown f(w)$
Online/Stochastic Gradient Descent	$diag[\eta_t]$	$\bigtriangledown f_t(w)$
Newton*	hessian matrix	$\bigtriangledown f(w)$
Quasi-Newton*	approximated hessian matrix	$\bigtriangledown f(w)$

Table 3.1: Summary of various line search algorithms. * means it is second order method, others means first order method. η_t is a vector to adjust the learning rate coordinately in iterate t. If using preconditioner, it is incorporated into η_t for reweighting the original learning rate.

computed on each data partition from the workers. For both types, the size of communicated data is proportional to the feature dimension and dominates the data exchange cost, especially in dealing with high dimensional problem.

There is redundancy in these communicated data due to the characteristics of optimization algorithms. The redundancy is mainly due to two reasons.

(1)there is usually no need to stop the optimization until achieving extremely small value since the objective function is created from empirical loss. Therefore, the accurate model can be redundant due to excess risk.

(2)most algorithms are designed by solving series of subproblems, in which the model approximates the function within a region. It can be very inaccurate when the current point is far away from the optimal point. Therefore, keeping accurate update step derived by solving the subproblem is redundant.

Since [9] has shown that first order method with preconditioner adjusted learning rate runs faster than L-BFGS method, for the simplicity of analysis, we focus on studying first order method in this thesis. For Quasi-Newton method, we also need to consider the introduced error from estimating hessian matrix from filtered gradients.

3.2 Intuition

Intuitively, there are two directions to reduce communication overhead. One is to directly avoid sending some entries, which is equivalent to sparsify to communicated data. Sampling or sketching techniques are suitable for designing filters of this direction. The other direction is to send less precised entries since it takes less memory to store a coarser representation. Quantization techniques are suitable for designing this type of filters.

To explore the possibility of designing filters of these directions, we analyze the characteristics for various data types and discuss the hidden redundancy that we can leverage for designing. In general, there are mainly two reasons

For (sub)gradients (b), we first introduced a unified representation (equation 3.3) mentioned in [14] to analyze its general characteristics across various loss functions . In equation 3.3, each row in X is a sample, Y is a column vector to store the corresponding labels.

$$\partial_w R_{emp}(w) = (\partial l(f, Y)^T X)^T, \ where \ f = Xw \tag{3.2}$$

For designing sparsification filter, one idea is to omitting smaller values. For entries with smaller values, there are two possible scenarios. One scenario is that the current model well captures the correlation between predictions and corresponding features. The other is that the appearance of corresponding features has low frequency. Therefore, if we omit some elements belonging to these scenarios, the model may actually benefit from avoiding overfitting. Another idea is to randomly skip some entries. We will discuss the intuition behind this type of filter in chapter 6.

Quantization filter is also applicable here. This is because subgradients mainly decide the search direction, which is unchanged with lower precision representation.

For A, here we only consider the first order case. If we use preconditioner, a n dimensional vector has to be communicated. Sparsifying the preconditioner vector may lead to zero division, so this type of filter is improper here. Quantization Filter is worth to explore because preconditioners are originally computed by approximation and thus insensitive for representation precision.

For model parameters, one observation is that only a fraction of model parameters are significantly changed after each iteration. Therefore, sparsification type filter can be applied to select only sending the significantly changed entries. Quantization filter is also applicable. More accurate model is only guaranteed to achieve smaller training error.

3.3 Condition for Convergence

Previous sections provide enough intuitions for designing filter algorithms. However, the lossy filters would unavoidably introduce error during the optimization. We need to quantify this influence so that we can derive conditions for filters to ensure the convergence.

We first analyze the influence of filters for reaching stationary point. The objective function is not necessary to be convex. Then, for convex problem, we further derive the difference between objective function value achieved by the solution and that by the optimal point. Finally, we can derive the condition for filters to ensure the convergence based on the above analysis. We also give cases satisfied the condition, which are useful for designing filters in practice.

The proof in [9] reflects that the effect of filters and bounded delay asynchronized update can be decorrelated. Also, the blockwise update is mainly for speeding up the first order method for large scale problem and would not effect the convergence properties with certain assumptions. Therefore, for simplicity, we only conduct analysis on the standard algorithms. The effect of filters is applicable for asynchronous blockwise update setting.

We first introduce some notations. Suppose we solve 3.4 by first order optimization algorithms.

$$\min_{w} f(w) + h(w) \tag{3.3}$$

Classical first order optimization algorithms includes gradient descent, projected gradient and subgradient descent. Their distributed versions can be outlined as Algorithm 3 in appendix A. The only difference happens in updating the model parameters.

To describe the effect of filters, we denote the received vector with filters applied during communication as $x + \sigma$. Here the effect of applying filter is represented as adding noise σ. To

distinguish various filters, we denote the effect when applying filters on local gradient during iterate t sent from worker i as $\sigma_{g_{t,i}}$. And we denote the effect when applying filters on model parameters during iterate t sent to worker i as $\sigma_{w_{t,i}}$. In iterate t, the actual aggregated gradients we collect from workers is

$$\sum_{i=1}^{m} \bigtriangledown_i f(w_t + \sigma_{w_{t,i}}) + \sigma_{g_{t,i}}$$

For the convenience of derivation, we introduce the notation

$$\xi_t = \sum_{i=1}^{m} \bigtriangledown_i f(w_t + \sigma_{w_{t,i}}) - \bigtriangledown_i f(w_t) + \sigma_{g_{t,i}}$$

For simplicity, our analysis conditions on that the filters are unbiased estimators (basic condition). We usually use randomized algorithm to prevent the algorithm getting stuck and it is natural to assume that the algorithm would not introduce additional bias.
Basic Condition The filters applied on local gradients or model parameters in any iterate satisfy

$$W[\sigma] = 0$$

For filter applied on model parameters, it is sufficiently small so that

$$\bigtriangledown_i f(w + \sigma) - \bigtriangledown_i f(w) \approx \bigtriangledown_i^2 f(w)\sigma$$

Lemma 1 With basic condition for filters satisfied, we approximately have $E[\xi_t] = 0$. For the variance, we have

$$Var[\xi_t] = E[\xi_t^2] \leq \sum_{i=1}^{m} CVar[\sigma_{w_{t,i}}] + Var[\sigma_{g_{t,i}}]$$

To conduct convergence analysis, we need to have assumptions about the objective function. The lipschitz continuity will be used in many cases, so the definition is given below.

Definition (Lipschitz Continuity) $\bigtriangledown f(w)$ is lipschitz continuious if there exists constant L such that $\| \bigtriangledown f(x) - \bigtriangledown f(y)\| \leq L\|x - y\|, \forall x,\ y$.

We first analyze the influence of filters for reaching stationary point using first order method. Here $f(w)$ can be non convex. Since subgradient not always exists for non convex function, we only analyze the case for projected gradient descent and gradient descent algorithms.

Theorem 1 For the cases described below, with filters satisfying the basic condition applied, we have

$$\epsilon \sum_{t=1}^{T} \|\Delta_t\| \leq E[F(w_1) - F(w_{T+1})] + \sum_{t=1}^{T} \eta_t Var[\xi_t]$$

Case 1 If $F(w) = f(w) + h(w)$ is twice differentialable and $\bigtriangledown F(w)$ has lipschitz continuity with constant L, $h(w)$ is convex, $f(w)$ is not necessary convex, using gradient descent algorithm.
Case 2 If $f(w)$ is twice differentialable and $\bigtriangledown f(w)$ has lipschitz continuity with constant L,

$h(w)$ is convex but not smooth, $f(w)$ can be convex or nonconvex, using projected gradient descent algorithm.

The proof is shown in appendix A. Projected gradient descent can be viewed as a generalized gradient descent algorithm, where the "gradient" includes the influence from projection. So it is reasonable that they have bound with same form. From the theorem, we can see that the item $\frac{1}{T}\sum_{t=1}^{T}\eta_t Var[\xi_t]$ need to be bounded so that $\lim_{t\to+\infty}\Delta_t = 0$.

For strongly convex problem. it is guaranteed to find the global minimizer if a stationary point is reached. Denote w_{best} as the optimal solution found by an algorithm, w^* as the true optimal solution. We are interested in studying the influence of filters on the bound $F(w_{best})-F(w^*)$.

Theorem 2.A For the following cases, applying filters satisfied the basic condition, we have

$$E[F(w_{best})] - F(w^*) \leq \frac{1}{T}(\frac{R^2}{2t} + \sum_{t=1}^{T}(\frac{\eta_t}{2} + \frac{L\eta_t^2}{2})Var[\xi_t])$$

Case 1 if both $f(w)$and $h(w)$ are convex and twice differientialable, $\bigtriangledown F(w) = f(w) + h(w)$ has lipschitz continuity with constant L, using gradient descent algorithm.
Case 2 $f(w)$ is convex and twice differientialable, $\bigtriangledown f(w)$ has lipschitz continuity with constant L, $h(w)$ is convex but non-smooth, using projected gradient descent algorithm.

Theorem 2.B If both $f(w)$ and $h(w)$ are convex, $f(w)$ is twice differientialable, using subgradient descent algorithm, we have

$$E[F(w_{best}) - F(w^*)] \leq \frac{1}{2tT}(R^2 + \sum_{t=1}^{T}\eta_t^2\|\partial F(w_t)\|) + \frac{1}{T}(\sum_{t=1}^{T}\frac{\eta_t}{2}Var[\xi_t])$$

We can see that the introduced noise by applying filters is reflected on the bound by an additional item, which is proportional to the sum of $Var[\xi_t]$. However, this might not be a bad thing. There is no need to optimize to death. This is because the objective function is created by using empirical loss and the best model is to minimize the true loss.

From the theorems, we can see that the filters need to satisfy the condition that $\sum_{t=1}^{T} Var[\xi_t]$ is bounded. To give more practical guide for filter design, we give two conditions below, which make the filter satisfy the bound condition.

Iterate t on i coordinate, denote $\Delta_{t,i} = w_{t+1,i} - w_{t,i}$ and $\Delta_{t,max} = \|\Delta_{t,i}\|_\infty$.
Condition i $E[\sigma] = 0$, $Var[\sigma] \sim O(\frac{1}{t})$
Condition ii $E[\sigma] = 0$, $Var[\sigma_{w_t}] \leq C\|\Delta_{t,max}\|^2$, C is constant

3.4 Discussion

Based on the theoretical analysis, we discuss several issues about implementation in practice.

For designing filters satisfying convergence conditions, we need to consider how $Var[\sigma]$ is controlled and how to incorporate the condition constraint into the filtering algorithm. For quantization filter, the variance is decided by the resolution. For sparsification filter, the variance on each coordinate is basically proportional to the magnitude of corresponding entry, while the form depends on various algorithms.

Let us take a look at the conditions we provide. Condition i can be implemented easily. However, it might not be very effective in practice. Suppose $Var[\sigma] \leq C\frac{1}{t}$, C is a constant. The bound quickly decrease and the filter might become useless in later iterates. Condition ii requires to track the change of parameters and is more complicated for implementation. But it couples with the algorithm closely, which make it possible that the filter is adjusted adaptively to fit the algorithm.

Beyond the two conditions we discussed, other heuristics can be developed so long as the sum of variance is bounded. For example, only using filters aggressively for the first couple of iterations. People can also develop strategies based on specific application needs.

Chapter 4

Algorithms

In this chapter, we provide filtering algorithms based on the intuitions and conditions discussed before. Section 4.1 introduced basic algorithms we take advantage of for designing various types of filters. Section 4.2 shows the details of filtering algorithm. Section 4.3 will discuss issues related applying the filters in practice, including scalability, filtering jointly.

One issue we need to consider here is the additional CPU cost brought by the filter since filtering algorithms are executed in every communication. For example, advanced sparsification techniques are usually derived by solving another optimization problem, which is improper for sparsification filtering algorithm. Plus, quantization algorithms[**?**] can achieve lowest memory cost by approximating the entropy bound. But this requires to use the data distribution, which is estimated by using more computations. Therefore, we should not go in this way.

4.1 Basic Algorithms Review

4.1.1 Randomized Rounding

Randomized rounding schemes have been widely used in numerical computing [15][16]. To randomized round a scalar w with resolution ϵ, the algorithm is sketched below:
RandomizedRound(w, ϵ):

$m = round(\frac{w}{\epsilon})$

return $\begin{cases} m+1, & with\ prob.\ \frac{w}{\epsilon} - m \\ m, & otherwise \end{cases}$

It has the following statistic properties, which is useful for the convergence condition.

1. $E[RandomizedRound(w, \epsilon)] = w$
2. $Var[RandomizedRound(w, \epsilon)] \propto \epsilon^2$

4.1.2 Priority Sampling

This is a weight sensitive sampling algorithm[17]. To sample k expected elements in vector $w = [w_1, ...w_n]$, the algorithm is sketched in algorithm 3.

Algorithm 3 Priority Sampling

PrioritySample(w, k)
priority_w=GeneratePriority(w);
// Find the (k+1)th highest priority as sampling threshold
τ=FindKthLargestValue(priority_w, K+1);
$\hat{w}$=Sample(w, τ)
Return $\hat{w}$

GeneratePriority(w)
for i= 1 to n **do**
$priority_{w_i} = \frac{|w_i|}{\alpha_i},\ \alpha_i \sim U(0,1).$
end
return priority_w

Sample(w, τ)
for i= 1 to n **do**
$\hat{w}_i = \begin{cases} 0, & if\ priority(w_i) < \tau \\ sgn(w_i) * max(|w_i|, \tau), & else \end{cases}$
end
return $\hat{w}$

It has the following statistic properties.

1. $E[\hat{w}] = w$
2. $Var[\hat{w}_i] = \begin{cases} 0, & |w_i| \geq \tau \\ |w_i|(\tau - |w_i|), & |w_i| < \tau \end{cases}$

4.2 Filter Algorithms

A communication filter is composed of two components, encoding and decoding. For the sender, encoding is executed before sending data. The receiver will execute the decoding step to retrieve the data.

4.2.1 Weight Sensitive Filter

This is a sparsification filter, which is designed based on the intuition that we can safely ignore smaller values in (sub)gradients (mentioned in section 2.2). We incorporate priority sampling while modified the threshold selection step. The only difference than standard priority sampling algorithm is that we set the threshold to be $min(\tau_{k+1}, \gamma\|\Delta_{t-1}\|_\infty)$, where $k = |x|$*sample fraction, x is the vector to be filtered, Δ_{t-1} is the model parameter update in iterate $t-1$. τ_{k+1} use the same notation in section 4.1.2.(In first iterate, threshold is τ_{k+1}).

The filter satisfied the basic condition. The variance for the selected entries are zero. Considering the variance for the zeroed entries:

$$Var[w] = w(\tau - w) \leq 0.25\tau^2, w \leq \tau$$

By modifying the tau to be minFindKthLargestValue(priority_w, K+1), threshold, we can ensure that $Var[w] \leq \|\Delta_t\|^2$. The drawback is that we can not ensure the sparsity degree when the k+1 th largest priority is larger than the given threshold.

When applying the weight sensitive filter in application, the sample fraction p should be manually tuned to achieve the best performance in terms of communication reduction and learning performance. In general, lower sample fraction is more likely to gain more communication reduction.

4.2.2 Rounding Filter

It is a quantization type filter, which is applicable for model parameters and (sub)gradients. Randomized rounding algorithm is applied for convergence condition. The algorithm is sketched here. Denote $w = [w_1, w_2, ...w_n]$.

RoundingFilter(w, n)
 Encode(w, n) //note: $\forall w_i = 0$ is skipped directly
  $a = max(w),\ b = min(w)$
  for i=1 to n
   $w_i^{'} = (w_i - b)/(b - a)$
   $\hat{w}_i = RandomizedRoud(w_i^{'},\ 2^{-n})$
  end
  return $\hat{w}$
 Decode($\hat{w}$, n)
  for i=1 to n
   $w_{recv,i} = 2^n(b - a)\hat{w}_i + a$
  end
  return w_{recv}

The variance for each entry is proportional to $[(b-a) * 2^{-n}]^2$. When filtering gradients, if the range of g_t approximates to $\Delta_{t,max}$, the convergence condition is perfectly fit. But in most cases, since 2^{-2n} is extremely small, slight violation is also fine. When applying on model parameters, the dynamic range will actually increase but still bounded due to the regularization term. Since w is usually extremely small compared to 2^{2n}, and we usually conduct finite number of iterations, a heuristic strategy can be used to achieve the convergence, in which n bits is used for the first couple of iterates and more bits are used later if needed (e.g. if $T|||w|||_\infty^2 * 2^{-2n}$ is more than a threshold).

In filtered vector, every entries only need n bits for storage. "n" usually use 8, 16, 24, 32 bits since byte is the basic unit in CPU. The compression ratio is fixed and equals to N/n, where N is the number of bits for the original representation.

4.2.3 Preconditioner Filter

It is also a quantization filter. Rounding filter tend to transform very small value into zero and thus bring into zero-division risk when applying to preconditioner. Therefore, preconditioner

filter is designed to solve this problem. Based on the intuition that it is only important to capture the magnitude for preconditioner, the algorithm only rounds the significand while keeping the exponent precisely. In this way, the introduced error is proportional to each entries magnitude.(A numeric value w is represented as $w = a * 2^b,\ a \in (0.5, 1)$) Plus, the preconditioners are always non-negative so that we can further reduce the space.

The preconditioner is used to adjust the learning rate. So less accurate representation would not affect the convergence, but it might slow down the convergence. User can choose to increase the bits for storing significand to get more precised representation.

4.3 Summary and Discussion

Filter	Type	Usage	Compression ratio
random skip	lossy (sparsification)	local (sub)gradient	$\frac{1}{P_r}$
weight sensitive filter*	lossy (sparsification)	local (sub)gradient	less than $\frac{1}{P_w}$
rounding filter*	lossy (quantization)	local (sub)gradient, model	N/n
preconditioner filter*	lossy (quantization)	preconditioner	N/(m+8)
KKT filter	lossy (ActiveSet)	L1 regularized	–
snappy compression	lossless	any data	–
key caching	lossless	key range unchanged	–

Table 4.1: Summary of communication filters. * identified algorithms proposed in this thesis. N represents the number of bits for original representation. P_r, P_w represents the sample fraction parameter for the filter.

We summarized the available communication filters in Table 4.1. and briefly explain their type and applicable scenario. Here further discuss two issues when applying these filters in practice.

(1) **Scalability** The filtering algorithms indicate that the added noise by applying filter for each worker depends on the local statistics. So here we quantify the effect by analyzing the bound for variance.

Suppose we use m workers and we use the notation to represent filter effect introduced in chapter 2. For simplicity, we ignore the iterate t index. We introduce the notations for analyzing multiple workers influence.

$$\|g_i\|_\infty = P_{1,i}^2 \|g\|_\infty,\ P_{1,i} \in [0, 1]$$
$$\bigtriangledown^2 f_i(w) = P_{2,i} \bigtriangledown^2 f(w)\ P_{2,i} \in [0, 1]$$

The variance of noise introduced by applying filters on local gradients and model parameters are shown below.

$$
\begin{aligned}
Var[\sigma_g] &= \sum_{i=1}^{m} Var[\sigma_{g,i}] \\
&\leq \sum_{i=1}^{m} \|g_i\|_\infty^2 \\
&= \sum_{i=1}^{m} P_{1,i}^2 \|g\|_\infty^2
\end{aligned}
$$

$$
\begin{aligned}
Var[\sum_{i=1}^{m} \bigtriangledown^2 f_i(w)\sigma_{w,i}] &= \sum_{i=1}^{m} (\bigtriangledown^2 f_i(w))^2 Var[\sigma_{w,i}] \\
&= \sum_{i=1}^{m} P_{2,i}^2 (\bigtriangledown^2 f(w))^2 Var[\sigma_{w,i}]
\end{aligned}
$$

We can see that more workers will lead to smaller bound for the sum of variance since $\sum_{i=1}^{m} p_i^2 \leq 1, \forall p_i \in [0,1]$. Combining the convergence condition analysis, it can converge to optimal point even faster when scale to more workers.

The variance only reflects one perspective of the algorithm property. For sparsification type filter, more workers might lead to the loss of sparsity for the model since each worker selected different subset during the sparsification. We will examine this problem experimentally.

(2) **Joint filter** Single filter cannot achieve the maximal communication reduction. Here we discuss how to combine these filters effectively.

For lossy filter, acitve set type filter (e.g. kkt filter) reduce the feature space for optimization. If it is applied, we should first apply this filter so that other filtering algorithms only need to work on a subset. Then, it is preferred to use sparsification type filter if applied since it can further reduce the working set for other filters (e.g. quantization filter). Lossless filters keep the communicated data unchanged, so we can always further apply them on information processed by all the lossy filters.

The effect on convergence would not be affect if each filter is examined successfully separately. This is because all filters are independent and the main difference when filtering jointly is that it can be applied to a smaller subset of the feature space.

Chapter 5

Experiments

We are interested in studying the following problems experimentally. (1) How much the learning performance is affected when applying filters with different strength in reducing communications? (2) To exercise the full power of filters, we demonstrate the efficacy of applying filters jointly regarding its ability in communication reduction and the affect on learning performance. (3) How the behavior of algorithms change when scale to more machines when using sparsification filter?

5.1 Data

We experimented using 4 binary classification datasets of varing scales, which are listed in Table 5.1. RCV1 includes continuous numeric feature, which generates from word frequency. Kdda and kddb include numeric feature which collapsed to a few values. CTR is a private data randomly sampled from displays within several months for advertisement click prediction. It only has binary features. kddb and CTR data are more sparse, and have long tail characteristics.

dataset	# of samples	feature dimensions	# of non-zero entries
rcv1	0.6M	47k	60M
kdda	8.4M	20M	305M
kddb	20M	29M	360M
ctr	0.1B	1.3 B	10B

Table 5.1: Datasets

5.2 Application

We examine the algorithm using logistic regression problem, which is one of the most popular algorithms in industry when solving large scale learning problems. Given a labeled example (x, y), $y \in \{0, 1\}$, $x_i \in R^d$, the model parameters $w \in R^d$, the loss function in logistic regression is $l(x_i, y_i; w) = log(1 + exp(-y_i < x_i, w >))$. A regularizer will be included to avoid overfitting

problem. So the problem we are solving is formulated in 5.1. We will conduct experiments using $Reg(w) = \|w\|_1$ (L1-regularized) and $Reg(w) = \|w\|_2^2$ (L2-regularized) separately.

$$w^* = \operatorname*{argmin}_w \sum_{i \in TrainSet} l(x_i, y_i, w) + \lambda Reg(w) \tag{5.1}$$

It is worth to mention the difference when using L1 norm and L2 norm as regularizer. In general, L1 norm is able to achieve model with much smaller size due to the sparsity it induced while the learning performance will be inferior to the model trained using L2 norm. L2 regularized model often enjoy more expressive power due to more parameters. Therefore, we verify the algorithm in case using L1 norm and L2 norm separately.

5.3 Setup

The implementation details can be found in appendix. We apply the bounded-delay blockwise first order algorithm to solve the logistic regression problem. The experiments are set up by fixing the maximal delay blocks to 4 and choosing a fixed learning rate by optimizing the convergence speed. For L1 norm, projected gradient descent algorithm is used; For L2 norm, gradient descent algorithm is used. The diagonal Hessian are used as the preconditioner to adjust coordinate learning rate.

For each dataset, it is divided into training data and testing data. The iterations in training model is fixed as 40 based on experience. The experiments are conducted using 8 workers and 2 servers except for scalability experiment. In the baseline, all numeric values are stored and communicated using 64bit double data type. Lossless filters (key caching and snappy compression) are used over all the communications. During each iteration, each worker send the local gradients and preconditioner to server group, then it receives the updated model parameters from server group.

5.4 Experimental results

The first set of experiments attempt to examine how much the learning performance is affected when applying a lossy filter with various strength in communication reduction. For weight sensitive filter, the ability in reducing communication cost is controlled by the sample fraction. The less is the sample faction, the more aggressive is the reduction. For quantization type filter, including rounding filter and preconditioner filter, the strength is scaled by the number of bits of the mapped representation. Fewer bits means more powerful in reducing communication cost.

Figure 5.1 reports the test performance change compared to baseline (AUC change=AUC-AUC(baseline)) when applying a filter. The horizontal axis shows the parameter to scale the strength in communication reduction for the filter. We can observe that the learning performance is not significantly affected (if assuming 1% drop as significant) in most cases. One interesting observation is that the AUC does not consistently decrease as we filter more aggressively, especially the performance is even slightly improved in some scenarios. This is understandable because filtering the communicated data properly may help avoid learning noise in training data

and the model can enjoy better generalization. Of course, lossy filtering also brings the risk of losing useful information, which might lead to performance degradation. The appropriate strength in filtering depends on the data.

To illustrate the full power of lossy filters, we experimentally explore the combination of lossy filters to achieve the maximal communication reduction without significant affect on the learning performance. Here we only experiment with the largest scale data, CTR dataset. In communicating local gradients, the lossy filters we applied include weight sensitive filter and rounding filter. The preconditioner filter is applied and we apply rounding filter in communicating model parameters. For L1-regularized problem, we also further experiment with kkt filter after optimizing all other filters. Except for kkt filter, the strength of lossy filters are optimized by grid search of its parameter.

	communication reduction fraction		AUC change
	worker outgoing	server outgoing	
L1-regularized	0.88	0.5	-2e-3(no kkt: +5e-4)
L2-regularized	0.93	0.75	-1e-4

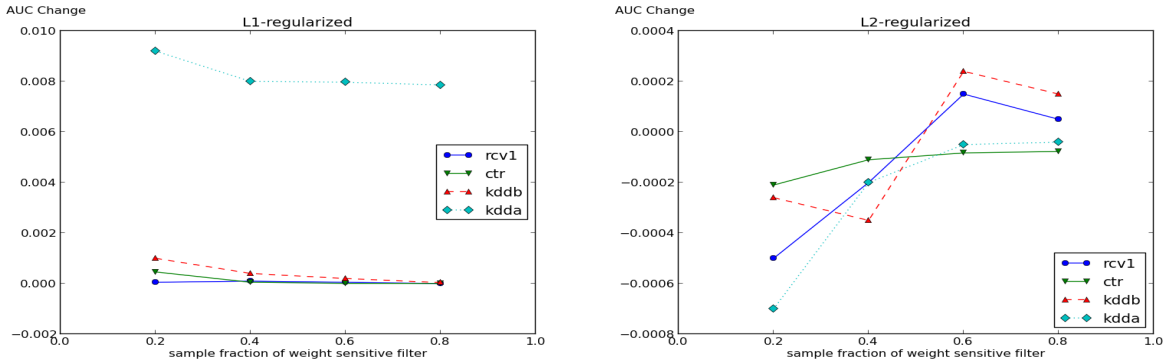
Table 5.2: Best result after applying lossy filters jointly.

Table 5.2 summarized the best result in terms of overall communication cost reduction. The communication is among workers and server group, so we report the outgoing cost of all workers and the server group separately. For both L1-regularized and L2-regularized case, the filter parameters for the best result is: weight sensitive filter use sample fraction of 0.2, rounding filter use 16 bits for filtering either gradients or model parameters, the preconditioner use 8 bits. From the results, we can see that the filters can reduce 90% of the communication overhead in baseline without affect the learning performance. The sparsity of L1-regularized problem lead to less communication overhead than L2-regularized problem, therefore the effect of filters seems less powerful.

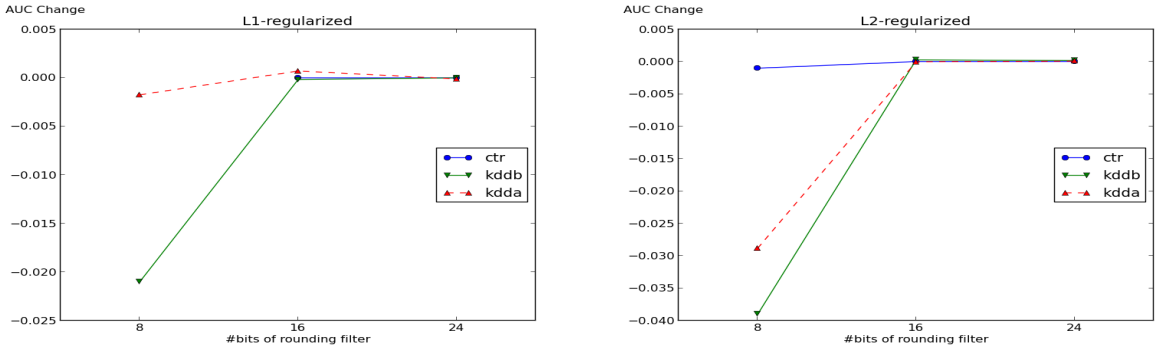
Though exchanging the local statistics and model parameters dominate the overall communications, the cost also include passing keys and other message information. To better understand the efficacy of filters, we also show the cost reduction for communicating gradients, preconditioners and model parameters respectively in Table 5.3.

	gradients	preconditioners	model parameters
L1-regularized (no kkt)	0.895	0.89	0.7
L1-regularized (with kkt)	0.978	0.98	0.73
L2-regularized	0.9	0.94	0.752

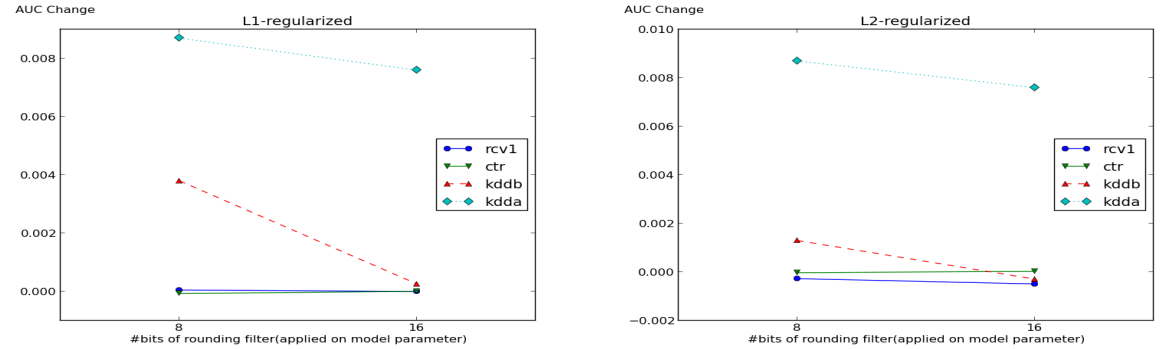
Table 5.3: Communication cost reduction for different data flows.



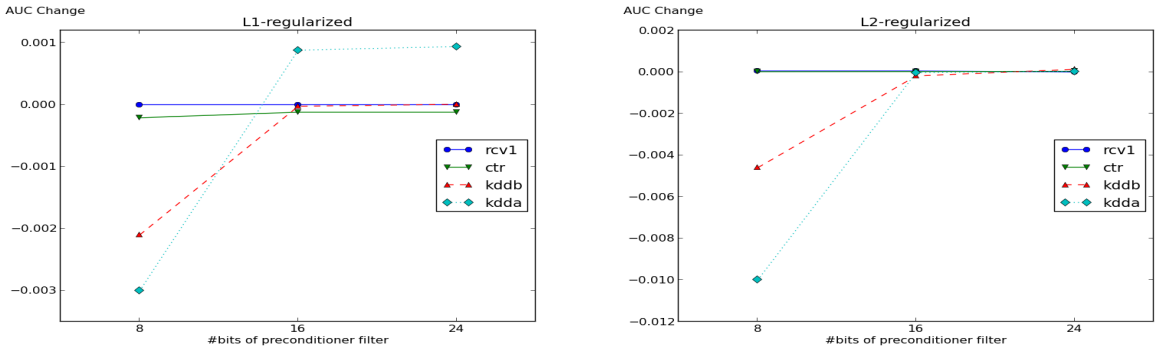
(a) Applying weight sensitive filter in communicating gradients.



(b) Applying rounding filter in communicating gradients.



(c) Applying rounding filter in communicating model parameters.



(d) Applying preconditioner filter.

Figure 5.1: Test Performance change over 4 datasets.

As claimed in chapter 4, the effect of filters in communication reduction is basically additive. It is easy to estimate the reduction ratio of quantization type filter. To help the reader better understand the result of applying filters jointly, we provide the communication reduction result for only using weight sensitive filter on gradients in Figure 5.2. For the sparse logistic regression problem, the non-zero fraction in the filtered vector equals to the sampling fraction most of times. One thing need to mention here is that if ruling out small values would shrink the dynamic range of the vector, snappy compression will achieve higher compression ratio. Here the remain cost fraction will be lower than the sample fraction. This is reflected in the convex shape curve of kddb and ctr dataset, in which the gradients enjoy long tail distribution due to a large fraction of rare features. This additional gain is achieved with larger sample fraction since the filter mainly effects on the tail part in these scenarios.

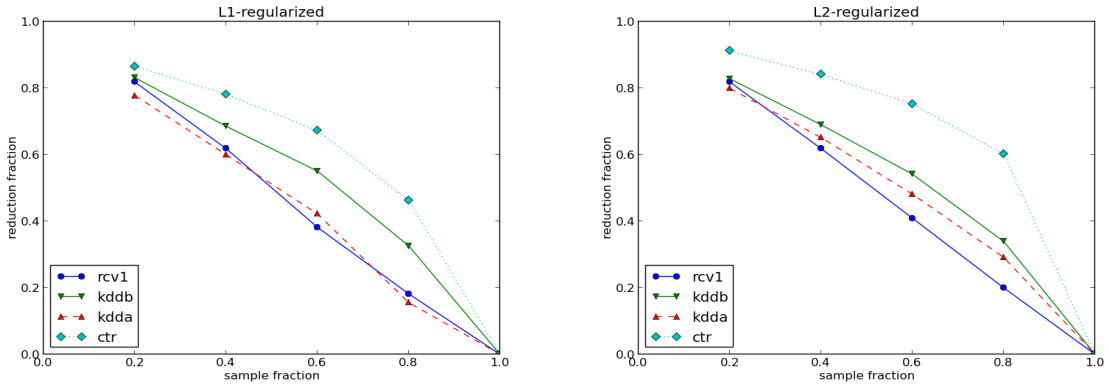
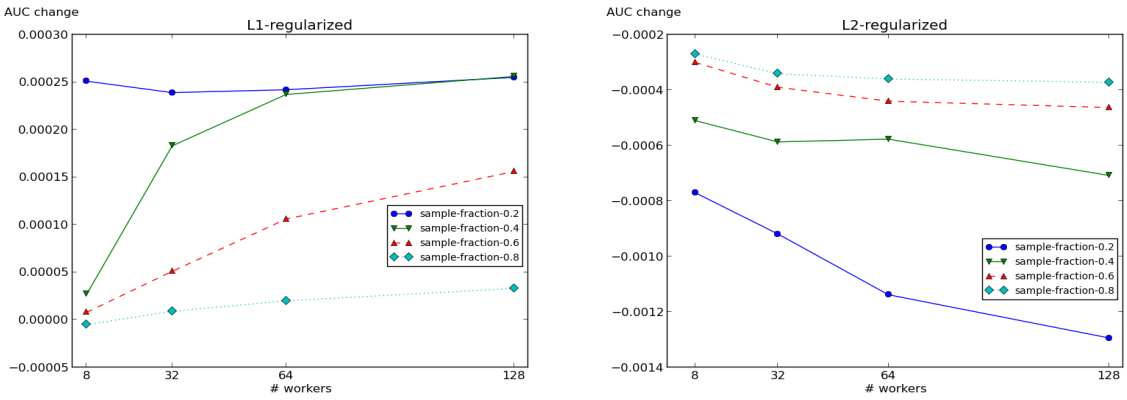


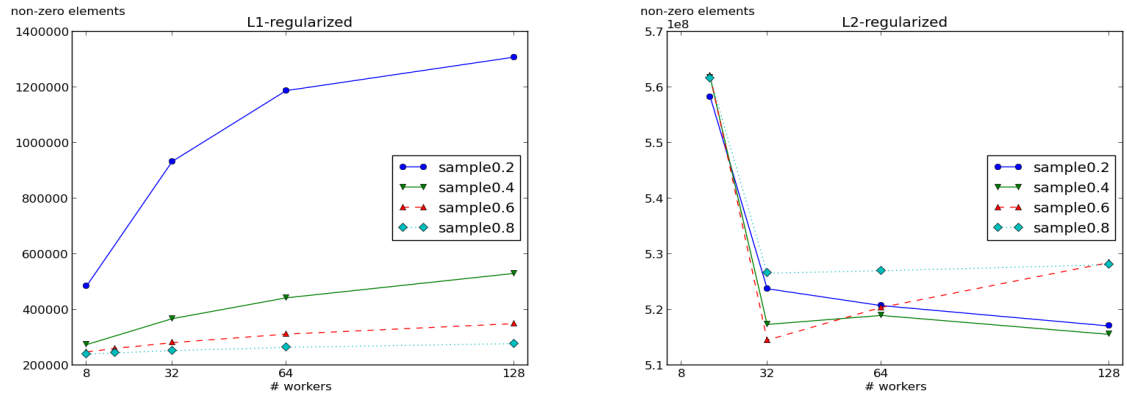
Figure 5.2: Cost reduction of communicating gradients using weight sensitive filter.

Finally, we experimentally examine the scalability influence in using sparsification filter, which cannot be answered by the analysis in chapter 4. The experiment is conducted using AWS EC2 service, with varied number of workers. The number of servers is fixed. We apply weight sensitive filter in sending local gradients and experiment with multiple sample fraction parameters.

Figure 5.3 shows the results regarding learning performance change and the model sparsity. For L1-regularized case, the test AUC slightly increase as the system scales to more workers. At the same time, the model become less sparse. The L1 norm naturally rules out small updates in the projection step (assuming model is initialized as 0). This weakens the influence of applying multiple sparsification filters independently. For L2-regularized case, the test AUC slightly decrease while the model become sparser when we use more workers. In general, the learning performance improves when the model has worse sparsity, and vice versa. This is understandable because the model become more expressive if it has more non-zero parameters.



(a) Test performance change.



(b) Model sparsity.

Figure 5.3: Scalability Test of weight sensitive filter

Chapter 6

Summary and Discussion

We explored the filtering idea to reduce the communication overhead for large scale machine learning problem. We first analyze the characteristics of various optimization algorithms and discuss the intuition in designing filters. We conduct convergence analysis to derive the condition in designing filter and provide algorithms for efficient filtering in practice. We experimentally verify that the lossy filters are able to significantly reduce the communication overhead without affect the learning performance. Surprisingly, we even observe slightly improved performance in some scenarios.

The filter interplays with optimization steps to restrict the learning space in various ways, which might help avoid overfitting and thus explain the improvement. The introduced constraint is related to multiple factors, which makes it hard to conduct explicit analysis. For example, when the filter attempts to filter out smaller values in gradients, it might avoid learning from rare features, or it might perform as an early stopping for a feature which is already well captured in the model. We cannot distinguish these two scenarios when executing filtering. Previous works have analysis on single scenario, like [18] [19] [20] formulate the input noise as regularization, and [21] explains the early stopping can be viewed as regularization.

We only experiment and report the results on logistic regression problem, but these filters can be directly generalized to other applications, e.g. deep learning, matrix factorization, and etc. The filtering ideas can be also applied to problems beyond optimization by making use of the redundancy hidden in the problem, e.g. sampler algorithm. The change of the model sparsity indicates that the model parameters are influenced when weaving filtering algorithm into optimization algorithm. However, it is still an open question that what source of the data or problem can benefit from using certain filter in terms of learning performance.

Appendices

A Algorithm

Algorithm 4 Distributed First order Batch Optimization

Worker r = 1, ..., m :
load training data $\{y_{ik}, x_{ik}\}_{k=1}^{n_r}$
Pull from the servers the model parameters w_r^0
for t = 1 to T **do**
Compute local gradient $g_r^t = \sum_{k=1}^{n_r} \partial l(x_{ik}, y_{ik}, w_r^{(t)})$
push g_r^t to servers
pull w_r^{t+1} from servers
end
Servers:
for t=1 to T **do**
aggregate $g^t \leftarrow \sum_{r=1}^{m} g_r^t$
$w^{t+1} \leftarrow update(w^t, \eta, g^t, \Omega(w^{(t)})$
end

if (sub)gradient descent:
$update(w^t, \eta, g^t, \Omega(w^{(t)})$: **Return** $w^t - \eta(g^t + \partial\Omega(w^{(t)})$
if projected gradient descent:
$update(w^t, \eta, g^t, \Omega(w^{(t)})$: **Return** $Prox_\eta[w^t - \eta g^t]$

B Proof of Theorem 1

For the convenience of derivation, we introduce the following notations. The update step in iterate t (w_t is updated to w_{t+1}) can be represented as:

$$\begin{aligned} \overline{w_{t+1}} &= w_t - \eta_t G_t \\ w_{t+1} &= \overline{w_{t+1}} - \eta_t \xi_t \\ \xi_t &= \sum_{i=1}^{m} \bigtriangledown_i f(w_t + \sigma_{w_{t,i}}) - \bigtriangledown_i f(w_t) + \sigma_{g_{t,i}} \end{aligned} \tag{1}$$

We use (2) in gradient descent algorithm and (3) in projected gradient descent algorithm.

$$G_t = \bigtriangledown F(w_t) \tag{2}$$

$$G_t = \frac{1}{\eta_t}[w_t - Prox_{\eta_t}(w_t - \eta_t \bigtriangledown f(w_t))] \tag{3}$$

Lemma A.1 For projected gradient descent algorithm using update step $w_{t+1} = w_t - \eta_t G_t$, G_t use the form in (2), we have

$$G_t \in \bigtriangledown f(w) + \partial h(w)$$

Proof The prox operator is to solve the problem

$$w_{t+1} = \arg\min_w h(w) + \frac{1}{2\eta_t}\|w_t - \eta_t \bigtriangledown f(w_t)\|$$

w_{t+1} is the optimal solution, so the subgradient of the objective function at w_{t+1} includes 0, that is

$$0 \in \partial h(w_{t+1}) + \frac{1}{\eta_t}(w_{t+1} - w_t + \eta_t \bigtriangledown f(w_t)) \tag{4}$$

From the update step, we know that $G_t = \frac{1}{\eta_t}(w_t - w_{t+1})$. Using this knowledge and rearranging (3) completes the proof.

Lemma A.2 If a function $f(w)$ is twice differientiable and satisfies lipschiz continuity: $\exists L, s.t. \| \bigtriangledown f(x) - \bigtriangledown f(y)\| \leq L\|x - y\|, \forall x, y$. We have

$$f(y) \leq f(x) + < \bigtriangledown f(x), y - x > + \frac{L}{2}\|x - y\|, \forall x, y$$

Proof The function is twice differentialable and the first order derivative satisfy lipschiz continuity. So by definition of second order derivative, we can obtain $\bigtriangledown^2 f(x) \leq L, \forall x$. Combing with Talor expansion, we can complete the proof.

$$\begin{aligned} f(x) &= f(y) + < \bigtriangledown f(x), y - x > + \frac{1}{2} \bigtriangledown^2 f(u)\|y - x\|^2,\ u \in [min(x, y), max(x, y)] \\ &\leq f(y) + < \bigtriangledown f(x), y - x > + \frac{L}{2}\|y - x\|^2, \end{aligned} \tag{5}$$

Proof for Theorem 1.A Based on Lemma A.2, $F(w)$ satisfy its condition and we have

$$F(w_{t+1}) - F(w_t) \leq < \bigtriangledown F(w_t), w_{t+1} - w_t > + \frac{L}{2}\|w_{t+1} - w_t\|^2 \tag{6}$$

From (1),(2), we have $\bigtriangledown F(w_t) = \frac{1}{\eta_t}(w_t - w_{t+1}) - \xi_t$. Substituting this into the above inequation, we can obtain

$$\begin{aligned} F(w_{t+1}) - F(w_t) &\leq -\frac{1}{\eta_t}\|w_{t+1} - w_t\|^2 - < \xi_t, w_{t+1} - w_t > + \frac{L}{2}\|w_{t+1} - w_t\|^2 \\ &= (\frac{L}{2} - \frac{1}{\eta_t})\|\Delta_t\|^2 - < \xi_t, -\eta_t(\bigtriangledown F(w_t) + \xi_t) > \end{aligned} \tag{7}$$

Combining with Lemma1, where we approximately have $E[\xi_t] = 0$, $Var(\xi) = E[\|\xi\|^2]$, we can get

$$E[F(w_{t+1}) - F(w_t)] \leq (\frac{L}{2} - \frac{1}{\eta_t})\|\Delta_t\|^2 + \eta_t Var[\xi_t] \tag{8}$$

Summing over t and rearranging yields

$$E[F(w_{T+1}) - F(w_1)] - \sum_{t=1}^{T} \eta_t Var[\xi_t] \leq \sum_{t=1}^{T} (\frac{L}{2} - \frac{1}{\eta_t}) \|\Delta_t\| \tag{9}$$

If the learning rate η_t is pick up properly and $\frac{L}{2} - \frac{1}{\eta_t} \leq 0$, we can get

$$\sum_{t=1}^{T} (\frac{1}{\eta_t} - \frac{L}{2}) \|\Delta_t\| \leq E[F(w_1) - F(w_{T+1})] + \sum_{t=1}^{T} \eta_t Var[\xi_t] \tag{10}$$

If $\exists \epsilon > 0, s.t. \frac{1}{\eta_t} - \frac{L}{2} \geq \epsilon$, we get the conclusion in the theorem.

Proof for Theorem 1.B Using lemma A.2, we have

$$f(w_{t+1}) - f(w_t) \leq < \bigtriangledown f(w_t), w_{t+1} - w_t > + \frac{L}{2} \|w_{t+1} - w_t\|^2 \tag{11}$$

By the definition of subgradient, we have

$$h(w_{t+1}) - h(w_t) \leq < \partial h(w_{t+1}), w_{t+1} - w_t > \tag{12}$$

Combining (11) and (12), we can get

$$F(w_{t+1}) - F(w_t) \leq < \bigtriangledown f(w_t) + \partial h(w_{t+1}), w_{t+1} - w_t > + \frac{L}{2} \|w_{t+1} - w_t\|^2 \tag{13}$$

From Lemma A.1, we know that $G_t \in \bigtriangledown f(w_t) + \partial h(w_{t+1})$. From (1), $G_t = \frac{1}{\eta_t}(w_t - w_{t+1}) - \xi_t$. Therefore, we can get a bound for $F(w_{t+1}) - F(w_t)$ which has exactly the same form as (7) derived for proving Theorem 1. Following the steps in proving theorem 1, we can get the conclusion.

C Proof of Theorem 2

We use the same notation in proving Theorem 1. Here we also analyze the subgradient method, so in equation (1) of Appendix A, we use

$$G_t = \partial F(w_t) \tag{14}$$

The key idea for the proof is that in each iterate, $\overline{w_{t+1}}$ and w_t satisfy the relation in the standard algorithm, so we can derive the relation for algorithm with filters by integrating the difference introduced through using w_{t+1}.

Proof for Theorem 2.A Using Lemma A.2, we have

$$\begin{aligned} F(w_{t+1}) &\leq F(w_t) + < \bigtriangledown F(w_t), w_{t+1} - w_t > + \frac{L}{2} \|w_{t+1} - w_t\|^2 \\ &= F(w_t) + < \bigtriangledown F(w_t), -\eta_t(\bigtriangledown F(w_t) + \xi_t) > + \frac{L}{2} {\eta_t}^2 \| \bigtriangledown F(w_t) + \xi_t\|^2 \end{aligned} \tag{15}$$

Based on Lemma 1, we can get (16), where the expectation is taken for the filters applied during iterate t.

$$\begin{aligned} E[F(w_{t+1})] &\leq F(w_t) - \eta_t(1 - \frac{L\eta_t}{2})\| \bigtriangledown F(w_t)\|^2 + \frac{L}{2}{\eta_t}^2\|\xi_t\|^2 \\ &\leq F(w_t) - \frac{\eta_t}{2}\| \bigtriangledown F(w_t)\|^2 + \frac{L}{2}{\eta_t}^2\|\xi_t\|^2,\ if\ \eta_t \leq \frac{1}{L} \end{aligned} \tag{16}$$

Based on the convexity of $f(w)$, we have

$$F(w_t) \leq F(w^*) + < \bigtriangledown F(w_t), w_t - w^* > \tag{17}$$

Combining (16) and (17) we can get

$$\begin{aligned} E[F(w_{t+1}) - F(w^*)] \leq & < \bigtriangledown F(w_t), w_t - w^* > -\frac{\eta_t}{2}\| \bigtriangledown F(w_t)\|^2 + \frac{L}{2}{\eta_t}^2 E[\|\xi_t\|^2] \\ &= \frac{1}{2\eta_t}[\|w_t - w^*\|^2 - \|\overline{w_{t+1}} - w^*\|^2] + \frac{L}{2}{\eta_t}^2 E[\|\xi_t\|^2] \\ &= \frac{1}{2\eta_t}[\|w_t - w^*\|^2 - \|w_{t+1} - w^*\|^2 + \|w_{t+1} - w^*\|^2 - \|\overline{w_{t+1}} - w^*\|^2] + \frac{L}{2}{\eta_t}^2 E[\|\xi_t\|^2] \end{aligned} \tag{18}$$

We define $\rho_t = \|w_{t+1} - w^*\|^2 - \|\overline{w_{t+1}} - w^*\|^2$. Suppose $\eta_t \geq t$, summing over t, we can get

$$\sum_{t=1}^{T} E[(F(w_{t+1}) - F(w^*))] \leq \frac{1}{2t}\|w_1 - w^*\|^2 + \frac{1}{2\eta_t}\sum_{t=1}^{T}\rho_t + \sum_{t=1}^{T}\frac{L}{2}{\eta_t}^2 E[\|\xi_t\|^2] \tag{19}$$

For ρ_t which captures part of the influence of filters, we have

$$\begin{aligned} E[\rho_t] &= E[< w_{t+1} - \overline{w_{t+1}}, w_{t+1} + \overline{w_{t+1}} - 2w^* >] \\ &= E[\eta_t\xi_t, 2\overline{w_{t+1}} - \eta_t\xi_t - 2w^*] \\ &= \eta_t^2 E[\|\xi_t\|^2] \end{aligned} \tag{20}$$

Combining (19) and (20), let $R = \|w_1 - w^*\|$, using lemma 1, we get

$$\sum_{t=1}^{T} E[(F(w_{t+1}) - F(w^*))] \leq \frac{R^2}{2t} + \sum_{t=1}^{T}(\frac{\eta_t}{2} + \frac{L\eta_t^2}{2})Var[\xi_t] \tag{21}$$

$$E[F(w_{best})] - F(w^*) \leq \frac{1}{T}(\frac{R^2}{2t} + \sum_{t=1}^{T}(\frac{\eta_t}{2} + \frac{L\eta_t^2}{2})Var[\xi_t]) \tag{22}$$

Lemma B.2 To solve $\min_w f(w) + h(w)$ with projected gradient descent algorithm, $f(w)$ is convex and twice diferientialbale, $h(w)$ is convex. We use update step $w_{t+1} = w_t - \eta_t G_t$, $G_t \in \bigtriangledown f(w_t) + \partial h(w_t)$ in iterate t (based on Lemma A.1). Suppose W^* is the optimal solution, we have

$$f(w_{t+1}) \leq f(w^*) + \frac{1}{2\eta_t}(\|w_t - w^*\|^2 - \|w_{t+1} - w^*\|^2)$$

Using Lemma A.1,Lemma A.2, we can derive

$$\begin{aligned} F(w_{t+1}) &\leq f(w_t) + < \bigtriangledown f(w_t), w_{t+1} - w_t > + \frac{L}{2}\|w_{t+1} - w_t\|^2 + h(w_{t+1}) \\ &= f(w_t) + < G_t - \partial h(w_{t+1}), \eta_t(G_t + \xi_t) > + \frac{L}{2}\|\eta_t(G_t + \xi_t)\|^2 + h(w_{t+1}) \end{aligned} \tag{23}$$

Taking the expectation for filters applied in iterate t, we have

$$\begin{aligned} E[F(w_{t+1})] &\leq f(w_t) - \eta_t(1 - \frac{L\eta_t}{2})\|G_t\|^2 + < \partial h(w_{t+1}), w_{t+1} - w_t > + h(w_{t+1}) + \frac{L}{2}\eta_t^2 E[\|\xi_t\|^2] \\ &= f(w_t) - \frac{\eta_t}{2}\|G_t\|^2 - < \partial h(w_{t+1}), w_{t+1} - w_t > + h(w_{t+1}) + \frac{L}{2}\eta_t^2 E[\|\xi_t\|^2],\ if\ \eta_t \leq \frac{1}{L} \end{aligned} \tag{24}$$

The convexity of $f(w)$ and $h(w)$, we have

$$f(w_t) \leq f(w^*) + < \bigtriangledown f(w_t), w_t - w^* > \tag{25}$$

$$h(w_{t+1}) \leq h(w^*) + < \partial h(w_{t+1}), w_{t+1} - w^* > \tag{26}$$

Substituting (25),(26) into (24), we can obtain

$$\begin{aligned} E[F(w_{t+1})] &\leq F(w^*) + < G_t, w_t - w^* > - \frac{\eta_t}{2}\|G_t\|^2 + \frac{L}{2}\eta_t^2 E[\|\xi_t\|^2] \\ &= F(w^*) + \frac{1}{2\eta_t}(\|w_t - w^*\|^2 - \|\overline{w_{t+1}} - w^*\|^2) + \frac{L}{2}\eta_t^2 E[\|\xi_t\|^2] \end{aligned} \tag{27}$$

Here we derived the same form inequation as in gradient descent algorithm in (18). Therefore, proceeding the same steps completes the proof.

Proof for Theorem 2.C Using the update step, we have

$$\begin{aligned} \|\overline{w_{t+1}} - w^*\| &= \|w_t - \eta_t(\partial F(w_t) + \xi_t) - w^*\|^2 \\ &= \|w_t - w^*\|^2 + \|\eta_t(\partial F(w_t))\|^2 - 2 < \eta_t(\partial F(w_t)), w_t - w^* > \end{aligned} \tag{28}$$

Rearranging yields

$$< \partial F(w_t), w_t - w^* > = \frac{1}{2\eta_t}[\|w_t - w^*\|^2 - \|\overline{w_{t+1}} - w^*\|^2 + \|\eta_t \partial F(w_t)\|^2] \tag{29}$$

By the definition of subgradient, we have

$$F(w_t) - F(w^*) \leq < \partial F(w_t), w_t - w^* > \tag{30}$$

Substituting (29) into (30), we have

$$\begin{aligned} E[F(w_t) - F(w^*)] &\leq \frac{1}{2\eta_t}[\|w_t - w^*\|^2 - \|\overline{w_{t+1}} - w^*\|^2 + \|\eta_t \partial F(w_t)\|^2] \\ &= \frac{1}{2\eta_t}[\|w_t - w^*\|^2 - \|w_{t+1} - w^*\|^2] + \frac{1}{2\eta_t}\|\eta_t \partial F(w_t)\|^2 + \frac{1}{2\eta_t}E[\rho_t] \end{aligned} \tag{31}$$

Summing over t, using the analysis result of $E[\rho_t]$, suppose $\eta_t \geq t$ and $\|w_t - w^*\| \leq R$, we have

$$\sum_{t=1}^{T} E[F(w_t) - F(w^*)] \leq \frac{1}{2t}(R^2 + \sum_{t=1}^{T} \eta_t^2 \|\partial F(w_t)\|) + \sum_{t=1}^{T} \frac{\eta_t}{2} Var[\xi_t] \tag{32}$$

$$E[F(w_{best}) - F(w^*)] \leq \frac{1}{2tT}(R^2 + \sum_{t=1}^{T} \eta_t^2 \|\partial F(w_t)\|) + \frac{1}{T}(\sum_{t=1}^{T} \frac{\eta_t}{2} Var[\xi_t]) \tag{33}$$

For standard subgradient algorithm, η_t need to satisfy the condition that $\sum_{t}^{\infty} \eta_t^2$ is bounded, here we use the same condition when applying filters.

D Implementation Details

The algorithms are implemented in C++11 using MPI communication on the open sourced parameter server. `https://github.com/yipeiw/parameter_server/tree/master/src/filter`

There are some tricks we use to minimize the added CPU cost by these filters. In priority sample filter, the division in generating priority for each entry is expensive. We use the pooling trick to reduce the cost. We cache a number of random ratio ($\frac{1}{\alpha}$, $\alpha \sim U(0,1)$) in preprocessing and the priority is generated online by multiplying a number randomly selected from the pool. In randomized rounding algorithm, the division of ϵ is implemented by bit manipulation since it equals to a power of 2.

To be compatible with the key caching filter, we still passing all the values but tag those filtered value either as 0 or as NaN, which can be effectively compressed out by the snappy compression filter.

Bibliography

[1] A. Ahmed, M. Aly, J. Gonzalez, S. M. Narayanamurthy, and A. J. Smola, "Scalable inference in latent variable models," in *Proceedings of the Fifth International Conference on Web Search and Web Data Mining, WSDM 2012, Seattle, WA, USA, February 8-12, 2012*, pp. 123–132, 2012. 1.1, 1.2

[2] H. Karau, "Fast data processing with spark," 2013. 1.1, 1

[3] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, and C. Guestrin, "Graphlab: A distributed framework for machine learning in the cloud," *CoRR*, vol. abs/1107.0922, 2011. 1.1, 1.3

[4] A. J. Smola and S. M. Narayanamurthy, "An architecture for parallel topic models," *PVLDB*, vol. 3, no. 1, pp. 703–710, 2010. 1.1

[5] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, (Broomfield, CO), pp. 583–598, USENIX Association, Oct. 2014. 1.1, 1.2, 2.1

[6] Q. Ho, J. Cipar, H. Cui, S. Lee, J. K. Kim, P. B. Gibbons, G. A. Gibson, G. Ganger, and E. P. Xing, "More effective distributed ml via a stale synchronous parallel parameter server," in *Advances in Neural Information Processing Systems 26* (C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, eds.), pp. 1223–1231, Curran Associates, Inc., 2013. 1.1, 1.2, 1, 2.1

[7] A. Ahmed, N. Shervashidze, S. M. Narayanamurthy, V. Josifovski, and A. J. Smola, "Distributed large-scale natural graph factorization," in *22nd International World Wide Web Conference, WWW '13, Rio de Janeiro, Brazil, May 13-17, 2013*, pp. 37–48, 2013. 1.2, 1

[8] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. W. Senior, P. A. Tucker, K. Yang, and A. Y. Ng, "Large scale distributed deep networks," in *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, pp. 1232–1240, 2012. 1.2, 1.3

[9] M. Li, D. Andersen, A. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," *NIPS*, 2014. 2, 1.4, 2.2.2, 3.1, 3.3

[10] M. Li, T. Zhang, Y. Chen, and A. J. Smola, "Efficient mini-batch training for stochastic optimization," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, (New York, NY, USA), pp. 661–670, ACM, 2014. 2.1

[11] W. Chen, Z. Wang, and J. Zhou, "Large-scale l-bfgs using mapreduce," in *Advances in Neural Information Processing Systems 27* (Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, eds.), pp. 1332–1340, 2014. 2.1

[12] "http://code.google.com/p/snappy," 2.2.1

[13] J. Nocedal and S. J. Wright, *Numerical Optimization*. New York: Springer, 2nd ed., 2006. 3.1

[14] C. H. Teo, S. V. N. Vishwanathan, A. J. Smola, and Q. V. Le, "Bundle methods for regularized risk minimization," *Journal of Machine Learning Research*, vol. 11, pp. 311–365, 2010. 3.2

[15] P. Raghavan and C. D. Thompson, "Randomized rounding: a technique for provably good algorithms and algorithmic proofs," *Combinatorica*, vol. 7, no. 4, pp. 365–374, 1987. 4.1.1

[16] D. Golovin, D. Sculley, H. B. McMahan, and M. Young, "Large-scale learning with less RAM via randomization," in *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, pp. 325–333, 2013. 4.1.1

[17] N. G. Duffield, C. Lund, and M. Thorup, "Priority sampling for estimation of arbitrary subset sums," *J. ACM*, vol. 54, no. 6, 2007. 4.1.2

[18] C. M. Bishop, "Training with noise is equivalent to tikhonov regularization," *Neural Comput.*, vol. 7, pp. 108–116, Jan. 1995. 6

[19] S. Wager, S. Wang, and P. S. Liang, "Dropout training as adaptive regularization," in *Advances in Neural Information Processing Systems 26* (C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, eds.), pp. 351–359, Curran Associates, Inc., 2013. 6

[20] D. P. Helmbold and P. M. Long, "On the inductive bias of dropout," *CoRR*, vol. abs/1412.4736, 2014. 6

[21] G. Federico, M. Jones, and T. Poggio, "Regularization theory and neural networks architectures," *Neural Computation 7*, pp. 21–269, 1995. 6