A functional approximation based distributed learning algorithm

Dhruv Mahajan
Microsoft Research
Bangalore, India
dhrumaha@microsoft.com

S. Sathiya Keerthi
CISL
Microsoft, Mountain View
keerthi@microsoft.com

S. Sundararajan
Microsoft Research
Bangalore, India
ssrajan@microsoft.com

Léon Bottou
Microsoft Research
New York, USA
leonbo@microsoft.com

ABSTRACT

Scalable machine learning over big data stored on a cluster of commodity machines with significant communication costs has become important in recent years. In this paper we give a novel approach to the distributed training of linear classifiers (involving smooth losses and L_2 regularization) that is designed to reduce communication costs. At each iteration, the nodes minimize approximate objective functions; then the resulting minimizers are combined to form a descent direction to move. Our approach gives a lot of freedom in the formation of the approximate objective function as well as in the choice of methods to solve them. The method is shown to have $O(\log(1/\epsilon))$ time convergence. The method can be viewed as an iterative parameter mixing method. A special instantiation yields a parallel stochastic gradient descent method with strong convergence. When communication times between nodes are large, our method is much faster than the SQM method [7], which computes function and gradient values in a distributed fashion.

Categories and Subject Descriptors

I.5.2 [**Pattern Recognition**]: Design Methodology—*Classifier design and evaluation*

General Terms

Algorithms, Performance, Experimentation

1. INTRODUCTION

In recent years, machine learning over big data has become an important problem, not only in web related applications, but also more commonly in other applications, e.g., in the data mining over huge amounts of user logs. The data in such applications are usually collected and stored in a decentralized fashion over a cluster of commodity machines (nodes) where communication times between nodes is significantly large. In such a setting it is natural for the examples to be partitioned over the nodes. The development of efficient distributed machine learning algorithms that minimize communication between nodes is an important problem.

In this paper we consider the distributed batch training of linear classifiers in which: (a) both, the number of examples and the number of features are large; (b) the data matrix is sparse; (c) the examples are partitioned over the nodes; (d) the loss function is convex and differentiable; and, (e) the L_2 regularizer is employed. This problem involves the large scale unconstrained minimization of a convex, differentiable objective function $f(w)$ where w is the weight vector. The minimization is usually performed using an iterative descent method in which an iteration starts from a point w^r, computes a direction d^r that satisfies

$$\textbf{sufficient angle of descent: } \angle(-g^r, d^r) \leq \theta \qquad (1)$$

where $g^r = g(w^r)$, $g(w) = \nabla f(w)$, $\angle a, b$ is the angle between vectors a and b, and $0 \leq \theta < \pi/2$, and then performs a line search along the direction d^r to find the next point, $w^{r+1} = w^r + td^r$. Let $w^\star = \arg\min_w f(w)$. A key side contribution of this paper is the proof that, when f is convex and satisfies some additional weak assumptions, the method has global linear rate of convergence (*glrc*)[1] and so it finds a point w^r satisfying $f(w^r) - f(w^\star) \leq \epsilon$ in $O(log(1/\epsilon))$ iterations. *The main theme of this paper is that the flexibility offered by this method with good convergence properties allows us to build a class of useful distributed learning methods.*

Take one of the most effective distributed methods, viz., SQM (Statistical Query Model) [7, 1], which is a batch, gradient-based descent method. The gradient is computed in a distributed way with each node computing the gradient component corresponding to its set of examples. This is followed by an aggregation of the components. We are interested in systems in which the communication time between nodes is large relative to the computation time in each node.[2] In such a scenario, it is useful to ask: **Q1.** *Can we do more computation in each node so that the number of communication passes is decreased, thus reducing the total computing time?*

There have been some efforts in the literature to reduce the amount of communication. In these methods, the current w^r is first passed on to all the nodes. Then, each node p forms an approximation $\tilde{f}_p$ of f using only its examples, followed by several optimization iterations (local passes over its examples) to decrease $\tilde{f}_p$ and reach a point w_p. The

[1]We say a method has *glrc* if $\exists\ 0 < \delta < 1$ such that $(f(w^{r+1}) - f(w^\star)) \leq \delta(f(w^r) - f(w^\star))\ \forall r$.

[2]This is the case when feature dimension is huge. Many applications gain performance when the feature space is expanded, say, via feature combinations, explicit expansion of nonlinear kernels etc.

w_p $\forall p$ are averaged to form the next iterate w^{r+1}. One can stop after just one major iteration (going from $r = 0$ to $r = 1$); such a method is referred to as *parameter mixing (PM)* [17]. Alternatively, one can do many major iterations; such a method is referred to as *iterative parameter mixing (IPM)* [9]. Convergence theory for such methods is inadequate [17, 18], which prompts us to ask: **Q2.** *Is it possible to devise an IPM method that produces* $\{w^r\} \to w^\star$*?*

For large scale learning on a single machine, it is now well established that example-wise methods[3] such as stochastic gradient descent (SGD) and its variations [3, 11] and dual coordinate ascent [10] are much faster than batch gradient-based methods. However, example-wise methods are inherently sequential. If one employs a method such as SGD as the local optimizer for $\tilde{f}_p$ in PM/IPM, the result is, in essence, a parallel SGD method. However, convergence theory for such a method is limited, even that requiring a complicated analysis [26]. Thus, we ask: **Q3.** *Can we form a parallel SGD method with strong convergence properties?*

We make a novel and simple use of the iterative descent method mentioned at the beginning of this section to design a distributed algorithm that answers Q1-Q3 positively. The main idea is to use distributed computation for generating a good search direction d^r and not just for forming the gradient as in SQM. At iteration r, let us say each node p has the current iterate w^r and the gradient g^r. This information can be used together with the examples in the node to form a function $\hat{f}_p(\cdot)$ that approximates $f(\cdot)$ and satisfies $\nabla \hat{f}_p(w^r) = g^r$. One simple and effective suggestion is:

$$\hat{f}_p(w) = f_p(w) + (g^r - \nabla f_p(w^r)) \cdot (w - w^r) \qquad (2)$$

where f_p is the part of f that does not depend on examples outside node p. In section 3 we give other suggestions for forming $\hat{f}_p$. Now $\hat{f}_p$ can be optimized within node p using any method $\mathcal{M}$ which has *glrc*, e.g., Trust region method, L-BFGS, etc. There is no need to optimize $\hat{f}_p$ fully. We show (see section 3) that, in a constant number of local passes over examples in node p, an approximate minimizer w_p of $\hat{f}_p$ can be found such that the direction $d_p = w_p - w^r$ satisfies the sufficient angle of descent condition, (1). The set of directions generated in the nodes, $\{d_p\}$ can be averaged to form the overall direction d^r for iteration r. Note that d^r also satisfies (1). The result is an overall distributed method that finds a point w satisfying $f(w) - f(w^\star) \leq \epsilon$ in $O(\log(1/\epsilon))$ time. This answers **Q2**.

The method also reduces the number of distributed passes over the examples compared with SQM, thus also answering **Q1**. The intuition here is that, if each $\hat{f}_p$ is a good approximation of f, then d^r will be a good global direction for minimizing f at w^r, and so the method will move towards $w^\star$ much faster than SQM. As one special instantiation of our distributed method, we can use, for the local optimization method $\mathcal{M}$, any variation of SGD with *glrc* (in expectation), e.g., the one in Johnson & Zhang [11]. For this case, in a related work we showed that our method has $O(\log(1/\epsilon))$ time convergence in a probabilistic sense [15]. The result is a strongly convergent parallel SGD method, which answers **Q3**. An interesting side observation is that, the single machine version of this instantiation is very close to the variance-reducing SGD method in Johnson & Zhang [11].

In summary, the paper makes the following contributions.

1. For convex f we establish *glrc* for a general iterative descent method.

2. We propose a distributed learning algorithm that: (a) converges in $O(\log(1/\epsilon))$ time, thus leading to an IPM method with strong convergence; (b) is more efficient than SQM when communication times are high; and (c) flexible in terms of the local optimization method $\mathcal{M}$ that can be used in the nodes.

3. We give an effective parallelization of SGD with good theoretical support and make connections with a recently proposed variance-reducing SGD method.

Experiments validate our theory as well as show the benefits of our method for large dimensional datasets where communication is the bottleneck. We conclude with a discussion on unexplored possibilities for extending our distributed learning method in section 5.

2. BASIC DESCENT METHOD

Let $f \in \mathcal{C}^1$, the class of continuously differentiable functions[4], f be convex, and the gradient g satisfy the following assumptions.

A1. g is Lipschitz continuous, i.e., $\exists\ L > 0$ such that $\|g(w) - g(\tilde{w})\| \leq L\|w - \tilde{w}\| \quad \forall\ w, \tilde{w}$.

A2. $\exists\ \sigma > 0$ such that $(g(w) - g(\tilde{w})) \cdot (w - \tilde{w}) \geq \sigma\|w - \tilde{w}\|^2 \quad \forall\ w, \tilde{w}$.

A1 and A2 are essentially second order conditions: if f happens to be twice continuously differentiable, then L and σ can be viewed as upper and lower bounds on the eigenvalues of the Hessian of f. A convex function f is said to be σ- strongly convex if $f(w) - \frac{\sigma}{2}\|w\|^2$ is convex. In machine learning, all convex risk functionals in $\mathcal{C}^1$ having the L_2 regularization term, $\frac{\lambda}{2}\|w\|^2$ are σ- strongly convex with $\sigma = \lambda$. It can be shown [22] that, if f is σ-strongly convex, then f satisfies assumption A2.

Let $f^r = f(w^r)$, $g^r = g(w^r)$ and $w^{r+1} = w^r + td^r$. Consider the following standard line search conditions.

$$\textbf{Armijo:} \quad f^{r+1} \leq f^r + \alpha g^r \cdot (w^{r+1} - w^r) \qquad (3)$$

$$\textbf{Wolfe:} \quad g^{r+1} \cdot d^r \geq \beta g^r \cdot d^r \qquad (4)$$

where $0 < \alpha < \beta < 1$.

Algorithm 1: Descent method for f

Choose w^0;
for $r = 0, 1 \ldots$ **do**
1. Exit if $g^r = 0$;
2. Choose a direction d^r satisfying (1);
3. Do line search to choose $t > 0$ so that $w^{r+1} = w^r + td^r$ satisfies the Armijo-Wolfe conditions (3) and (4);

end

Let us now consider the general descent method in Algorithm 1 for minimizing f. The following result shows that

[3]These methods update w after scanning each example.

[4]It would be interesting future work to extend all the theory developed in this paper to non-differentiable convex functions, using sub-gradients.

the algorithm is well-posed. A proof is given in the appendix B.

Lemma 1. Suppose $g^r \cdot d^r < 0$. Then $\{t : (3) \text{ and } (4) \text{ hold for } w^{r+1} = w^r + td^r\} = [t_\beta, t_\alpha]$, where $0 < t_\beta < t_\alpha$, and t_β, t_α are the unique roots of

$$g(w^r + t_\beta d^r) \cdot d^r = \beta g^r \cdot d^r, \tag{5}$$

$$f(w^r + t_\alpha d^r) = f^r + t_\alpha \alpha g^r \cdot d^r, \;\; t_\alpha > 0. \tag{6}$$

Theorem 2. Let $w^\star = \arg\min_w f(w)$ and $f^\star = f(w^\star)$.[5] Then $\{w^r\} \to w^\star$. Also, we have *glrc*, i.e., $\exists\ \delta$ satisfying $0 < \delta < 1$ such that $(f^{r+1} - f^\star) \leq \delta\,(f^r - f^\star)\ \forall\ r \geq 0$, and, $f^r - f^\star \leq \epsilon$ is reached after at most $\frac{\log((f^0 - f^\star)/\epsilon)}{\log(1/\delta)}$ iterations. An upper bound on δ is $(1 - 2\alpha(1-\beta)\frac{\sigma^2}{L^2}\cos^2\theta)$.

A proof of Theorem 2 is given in the appendix B. If one is interested only in proving convergence, it is easy to establish under the assumptions made; such theory goes back to the classical works of Wolfe [24, 25]. But proving *glrc* is harder. There exist proofs for special cases such as the gradient descent method [5]. The *glrc* result in Wang & Lin [23] is only applicable to descent methods that are "close" (see equations (7) and (8) in [23]) to the gradient descent method. Though Theorem 2 is not entirely surprising, as far as we know, such a result does not exist in the literature.

It is important to note that the rate of convergence indicated by the upper bound on δ given in Theorem 2 is pessimistic since it is given for a very general descent algorithm that includes plain batch gradient descent which is known to have a slow rate of convergence. Depending on the method used for choosing d^r the actual rate of convergence can be a lot better. For example, we observe very good rates for our distributed method; see section 4.

3. DISTRIBUTED TRAINING

Let $\{x_i, y_i\}$ be the training set associated with a binary classification problem ($y_i \in \{1, -1\}$). Consider a linear classification model, $y = \text{sgn}(w^T x)$. Let $l(w \cdot x_i, y_i)$ be a continuously differentiable loss function that has Lipschitz continuous gradient. This allows us to consider loss functions such as least squares, logistic loss and squared hinge loss. Hinge loss is not covered by our theory since it is non-differentiable.

Suppose the data is distributed in P nodes. Let: I_p be the set of indices i such that (x_i, y_i) sits in the p-th node; $L_p(w) = \sum_{i \in I_p} l(w; x_i, y_i)$ be the total loss associated with node p; and, $L(w) = \sum_p L_p(w)$ be the total loss over all nodes. Our aim is to to minimize the regularized risk functional $f(w)$ given by

$$f(w) = \frac{\lambda}{2}\|w\|^2 + L(w) = \frac{\lambda}{2}\|w\|^2 + \sum_p L_p(w) \tag{7}$$

where $\lambda > 0$ is the regularization constant. It is easy to check that $g = \nabla f$ is Lipschitz continuous.

Our distributed method is based on the descent method in Algorithm 1. We use a master-slave architecture.[6] Let the examples be partitioned over P slave nodes. Distributed computing is used to compute the gradient g^r as well as the direction d^r. In the r-th iteration, let us say that the master has the current w^r and gradient g^r. One can communicate these to all P (slave) nodes. The direction d^r is formed as follows. Each node p constructs an approximation of $f(w)$ using only information that is available in that node, call it $\hat{f}_p(w)$, and (approximately) optimizes it (starting from w^r) to get the point w_p. Let $d_p = w_p - w^r$. Then d^r is chosen to be any convex combination of $d_p\ \forall p$.

Our method offers great flexibility in choosing $\hat{f}_p$ and the method used to optimize it. We only require $\hat{f}_p$ to satisfy the following.

A3. $\hat{f}_p$ is σ-strongly convex, has Lipschitz continuous gradient and satisfies *gradient consistency at* w^r: $\nabla \hat{f}_p(w^r) = g^r$.

Below we give ways of forming $\hat{f}_p$. The σ-strongly convex condition is easily taken care of by making sure that the L_2 regularizer is a part of $\hat{f}_p$. This condition implies that

$$\hat{f}_p(w_p) \geq \hat{f}_p(w^r) + \nabla \hat{f}_p(w^r) \cdot (w_p - w^r) + \frac{\sigma}{2}\|w_p - w^r\|^2 \tag{8}$$

The gradient consistency condition is motivated by the need to satisfy the angle condition (1). Since w_p is obtained by starting from w^r and optimizing $\hat{f}_p$, it is reasonable to assume that $\hat{f}_p(w_p) < \hat{f}_p(w^r)$. Using these in (8) gives $-g^r \cdot d_p > 0$. Since d^r is a convex combination of the d_p it follows that $-g^r \cdot d^r > 0$. Later we will formalize this to yield (1) precisely.

A natural way of choosing the approximating functional $\hat{f}_p$ is

$$\hat{f}_p(w) = \frac{\lambda}{2}\|w\|^2 + L_p(w) + \hat{L}_p(w) \tag{9}$$

where $\hat{L}_p(w)$ is an approximation of $L(w) - L_p(w) = \sum_{q \neq p} L_q(w)$, but one that does not explicitly require any examples outside node p. To satisfy A3 we only need $\hat{L}_p$ to have Lipschitz continuous gradient; all other conditions are directly satisfied. A simple instance of $\hat{L}_p$ is a linear function constructed using the gradient at w^r:

$$\hat{L}_p(w) = (g^r - \lambda w^r - \nabla L_p(w^r)) \cdot (w - w^r) \tag{10}$$

(The zeroth order term needed to get $f(w^r) = \hat{f}(w^r)$ is omitted because it is a constant that plays no role in the optimization.) There are other ways of forming an approximation $\hat{L}_p(w)$. For example, one could add a second order term, $\frac{1}{2}(w - w^r) \cdot H(w - w^r)$ to the approximation in (10) where H is a positive semi-definite matrix; for H we can use a diagonal approximation or keep a limited history of gradients and form a BFGS approximation of $L - L_p$.

Convergence Theory. The distributed method described above is an instance of Algorithm 1 and so Theorem 2 can be used. However, obtaining d^r requires the determination of the w_p via minimizing $\hat{f}_p$. As already mentioned, it is not necessary for w_p to be the minimizer of $\hat{f}_p$; we only need to find w_p such that the direction $d_p = w_p - w^r$ satisfies (1). The angle θ needs to be chosen right. Let us discuss this first. Let $\hat{w}_p^\star$ be the minimizer of $\hat{f}_p$. It can be shown (see appendix B) that $\angle(\hat{w}_p^\star - w^r, -g^r) \leq \cos^{-1}\frac{\sigma}{L}$. To allow for w_p being an approximation of $\hat{w}_p^\star$, we choose θ such that

$$\frac{\pi}{2} > \theta > \cos^{-1}\frac{\sigma}{L} \tag{11}$$

The following result shows that if an optimizer with *glrc* is used to minimize $\hat{f}_p$, then, only a constant number of iterations is needed to satisfy the sufficient angle of descent

[5] Assumption A2 implies that $w^\star$ is unique.

[6] An *AllReduce* arrangement of nodes [1] may also be used.

condition.

Lemma 3. Assume $g^r \neq 0$. Suppose we minimize $\hat{f}_p$ using an optimizer $\mathcal{M}$ that starts from $v^0 = w^r$ and generates a sequence $\{v^k\}$ having *glrc*, i.e., $\hat{f}_p(v^{k+1}) - \hat{f}_p^\star \leq \delta(\hat{f}_p(v^k) - \hat{f}_p^\star)$, where $\hat{f}_p^\star = \hat{f}_p(\hat{w}_p^\star)$. Then, there exists $\hat{k}$ (which depends only on σ and L) such that $\angle(-g^r, v^k - w^r) \leq \theta \ \forall k \geq \hat{k}$.

Lemma 3 can be combined with Theorem 2 to yield the following convergence theorem.

Theorem 4. Suppose θ satisfies (11), $\mathcal{M}$ is as in Lemma 3 and, in each iteration r and for each p, $\hat{k}$ or more iterations of $\mathcal{M}$ are applied to minimize $\hat{f}_p$ (starting from w^r) and get w_p. Then the distributed method converges to a point w satisfying $f(w) - f(w^\star) \leq \epsilon$ in $O(\log(1/\epsilon))$ time.

Proofs of Lemma 3 and Theorem 4 are given in appendix B.

Practical implementation. Going with the practice in numerical optimization, we replace (1) by the condition, $-g^r \cdot d^r > 0$ and use $\alpha = 10^{-4}$, $\beta = 0.9$ in (3) and (4). We terminate Algorithm 1 when $\|g^r\| \leq \epsilon_g \|g^0\|$ is satisfied at some r. Let us take line search next. On $w = w^r + td^r$, the loss has the form $l(z_i + te_i, y_i)$ where $z_i = w^r \cdot x_i$ and $e_i = d^r \cdot x_i$. Once we have computed $z_i \ \forall i$ and $e_i \ \forall i$, the distributed computation of $f(w^r + td^r)$ and its derivative with respect to t is cheap as it does not involve any computation involving the data, $\{x_i\}$. Thus many t values can be explored cheaply. Since d^r is determined by approximate optimization, $t = 1$ is expected to give a decent starting point. We first identify an interval $[t_1, t_2] \subset [t_\beta, t_\alpha]$ (see Lemma 1) by starting from $t = 1$ and doing forward and backward stepping. Then we check if t_1 or t_2 is the minimizer of $f(w^r + td^r)$ on $[t_1, t_2]$; if not, we do several bracketing steps in (t_1, t_2) to locate the minimizer approximately. Finally, when using method $\mathcal{M}$, we terminate it after a fixed number of steps, $\hat{k}$. Algorithm 2 gives all the steps of the distributed method while also mentioning the distributed communications and computations involved.

Choices for $\mathcal{M}$. There are many good methods having (deterministic) *glrc*: L-BFGS, TRON [13], Primal coordinate descent [6], etc. One could also use methods with *glrc* in the expectation sense (in which case, convergence in Theorem 4 should also be interpreted in some probabilistic sense; see our related work [15] for details). Recently suggested variants of SGD [12, 11] are methods with such convergence. This particular instantiation of our distributed method yields a parallel SGD method with strong convergence properties, which, as already indicated in section 1 (see Q3), fills a gap in the literature. In section 4 we conduct experiments using TRON and the SVRG method in Johnson & Zhang (2013).

Connection with SVRG. The connection of our method with the recently proposed SVRG method [11] is interesting. To show this, let us take the $\hat{f}_p$ in (10). Let $n_p = |I_p|$ be the number of examples in node p. Define $\psi_i(w) = n_p l(w \cdot x_i, y_i) + \frac{\lambda}{2}\|w\|^2$. It is easy to check that

$$\nabla \hat{f}_p(w) = \frac{1}{n_p} \sum_{i \in I_p} (\nabla \psi_i(w) - \nabla \psi_i(w^r) + g^r) \tag{12}$$

Algorithm 2: Distributed method for minimizing f
com: communication; *cmp:* = computation; *agg:* aggregation

Choose w^0;
for $r = 0, 1 \ldots$ **do**
  1. Compute g^r (*com:* w^r; *cmp:* Two passes over data; *agg:* g^r); By-product: $\{z_i = w^r \cdot x_i\}$;
  2. Exit if $\|g^r\| \leq \epsilon_g \|g^0\|$;
  3. **for** $p = 1, \ldots, P$ *(in parallel)* **do**
    4. Set $v^0 = w^r$;
    5. **for** $k = 0, 1, \ldots, \hat{k}$ **do**
      6. Find v^{k+1} using one iteration of $\mathcal{M}$;
    end
    7. Set $w_p = v^{\hat{k}+1}$;
  end
  8. Set d^r as any convex combination of $\{w_p\}$ (*agg:* w_p);
  9. Compute $\{e_i = d^r \cdot x_i\}$ (*comm:* d^r; *cmp:* One pass over data);
  10. Do line search to find t (for each t: *comm:* t; *cmp:* l and $\partial l/\partial t$ *agg:* $f(w^r + td^r)$ and its derivative wrt t);
  11. Set $w^{r+1} = w^r + td^r$;
end

Thus, plain SGD updates applied to $\hat{f}_p$ has the form

$$w = w - \eta(\nabla \psi_i(w) - \nabla \psi_i(w^r) + g^r) \tag{13}$$

which is precisely the update in SVRG. In particular, the single node ($P = 1$) implementation of our method using plain SGD updates for optimizing $\hat{f}_p$ is very close to the SVRG method.[7] While Johnson & Zhang [11] motivate the update in terms of variance reduction, we derive it from a functional approximation viewpoint.

Computation-Communication tradeoff. Compared to the SQM method (see section 1), our method does a lot more computation (optimize $\hat{f}_p$) in each node. On the other hand our method reaches a good solution using a much smaller number of outer iterations. Clearly, our method will be attractive for problems with high communication costs, e.g., problems with a large feature dimension. For a given distributed computing environment and specific implementation choices, it is easy to do a rough analysis to understand the conditions in which our method will be more efficient than SQM. Consider a distributed grid of nodes in an *AllReduce* tree. Let us use TRON for implementing SQM and SVRG for $\mathcal{M}$ in our method. Assuming that $T^{\text{outer}}_{\text{SVRG}} < 3.2 T^{outer}_{\text{SQM}}$ (where $T^{\text{outer}}_{\text{SVRG}}$ and T^{outer}_{SQM} are the number of outer iterations required by SQM and our method with SVRG), we can do a rough analysis of the costs of SQM and our method (see appendix A for details) to show that our method will be faster when the following condition is satisfied.

$$\frac{nz}{m} \ll \frac{\gamma P \log_2 P}{2} \frac{T^{\text{outer}}_{\text{SQM}}}{\hat{k}} \tag{14}$$

[7] Note the subtle point that applying SVRG method on $\hat{f}_p$ is different from doing (13), which corresponds to plain SGD. It is the former that assures *glrc* (in expectation).

where: nz is the number of nonzero elements in the data, i.e., $\{x_i\}$; m is the feature dimension; γ is the relative cost of communication to computation (e.g. $100-1000$); P is the number of nodes; and $\hat{k}$ is the number of inner iterations of our method.

4. EXPERIMENTS

In this section, we demonstrate the effectiveness of our method on large dimensional data sets. We first discuss our experimental setup. We then show results to validate the theory proposed in the paper. Finally, we compare our approach with existing distributed machine learning algorithms and clearly demonstrate scenarios under which our method performs better.

4.1 Experimental Setup

We run our experiments on a Hadoop cluster. Since iterations in traditional MapReduce are slower (because of job setup and disk access costs), as in Agarwal et al. [1], we build an AllReduce binary tree between the mappers[8]. The communication bandwidth is $1Gbps$ (gigabits per sec). For functional approximation we use (9) and (10).

Table 1: Datasets

Dataset	Examples	Features	Non-zeros
kdd2010	$8.41M$	$20.21M$	0.31B
url	$1.91M$	$3.23M$	0.22B

Data Sets. We consider two well known large dimensional datasets: *kdd2010* and *url*. Table 3 shows the number of examples, features and nonzero feature values. We use these datasets mainly to illustrate the validity of theory, and its utility to distributed machine learning.

Methods for Comparison. We use the *squared-hinge* loss function with *l2*-regularization for all the experiments. We compare the following methods.

SQM: We use the Trust Region Newton method (TRON) proposed in Lin et al. [13] and, do the gradient and Hessian computations in a parallel manner. We initialize the weight vector to zero and set all the parameters (except regularizer λ) to the values recommended in Lin et al. [13].

HYBRID: We find a local weight vector per node by minimizing the local objective function (based only on the examples in that node) using one epoch of SGD [3]. (The optimal step size is chosen by running SGD on a subset of data.) We then average the weights from all the nodes and use the averaged weight vector to warm start *SQM*. Note that this method is same as that proposed in Agarwal et al. [1] (except that they use the L-BFGS method instead of TRON).

FS-k: Our algorithm with the SVRG method [11] used for solving the local optimization in every iteration. As suggested in [11], we recalculate the batch gradient after every 5 epochs (referred as outer iteration in the local optimization context). We run k outer iterations of SVRG and show results for $k = 8$ and 16.

[8]Note that we do not use the pipelined version and hence we incur an extra multiplicative $logP$ cost in communication.

FT-k: Our algorithm with TRON [13] used for solving the local optimization. We stop the inner optimization after doing k Hessian-vector multiplications. The results are shown for $k = 50$ and 100 for *kdd2010* and $k = 25$ and 50 for *url*.

Evaluation Criteria. We use the Area under Precision-Recall Curve (AUPRC) and relative difference to the optimal function value as the evaluation criteria. The difference is calculated as $(f - f^*)/f^*$ in log scale, where f^* is the optimal function value. We obtained f^* by running the *SQM* algorithm for large number of iterations.

4.2 Results

Comparison with ADMM. The Alternating Direction Method of Multipliers (ADMM) [4], like our method, solves approximate problems in the nodes and iteratively reaches the full batch solution. ADMM has a quadratic proximal term called augmented Lagrangian with penalty parameter ρ. Recently, Deng & Yin [8] proved the linear rate of convergence for ADMM under the assumptions A1 and A2 on ADMM functions. As a result, their analysis also hold for the objective function in (7). They also give an analytical formula to set ρ in order to get the best theoretical linear rate constant. We consider the following two versions of ADMM.

ADMM-R We use the ρ value given by the analytical formula in [8].

ADMM-Adap We start with the value of ρ in ADMM-R and select the best ρ in its neighborhood by running ADMM for 10 iterations and looking at the objective function value. However, this step takes additional time and causes late start of ADMM in the plot (Figure 1).

For both the versions, we use TRON [13] for solving the local optimization. Figure 1 gives a comparison of ADMM with our method *FT-k* on *url* with 100 nodes. The horizontal axis is the number of communication passes. For all the methods we use $TRON$ iterations, $k = 100$. Note that the recommended value of ρ in Deng and Yin [8] makes ADMM an order of magnitude slower than our method. We found that the ρ found by ADMM-R was more than 10^4 times the best ρ value found by ADMM-Adap. Even near this best value, the performance of ADMM was sensitive to the variation of ρ. We also observed that once ADMM-Adap finds the best ρ, it works extremely well. However, a significant amount of time is spent on finding this value, thus making the overall approach slow. Similar observations were made on *kdd2010* and other parameter settings. Moreover, the value of optimal ρ is data or problem dependent. Hence, we do not include ADMM further in our study.

Apart from ADMM, Bertsekas and Tsitsiklis [2] discuss several other classic optimization methods for separable convex programming, based on proximal and Augmented Lagrangian ideas. ADMM represents the best of these methods. Also, Gauss-Seidel and Jacobi methods given there are related to feature partitioning, which is not the main theme of this paper. Therefore we will not mention these methods any further.

Linear Convergence. To validate linear convergence, we study the variation of $(f - f^*)/f^*$ (in log scale) as a function of the number of communication passes[9]. For our algorithms (*FS-k* and *FT-k*), the number of communication

[9]Note that we do not use the number of outer iterations as

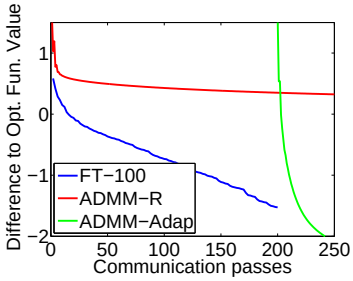
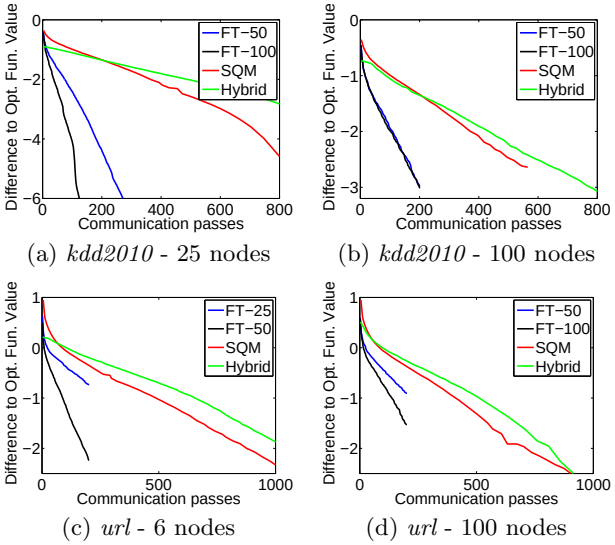


Figure 1: Plot showing comparison with ADMM for *url* with 100 nodes. x-axis is the number of communication passes and y-axis is the relative decrease in function value in $\log_{10}$ scale.



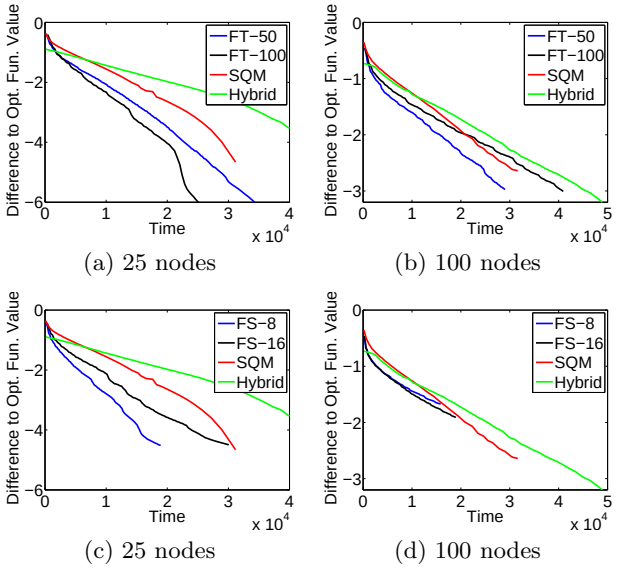
(a) *kdd2010* - 25 nodes (b) *kdd2010* - 100 nodes

(c) *url* - 6 nodes (d) *url* - 100 nodes

Figure 2: Plots showing linear convergence of our method using TRON as the local optimizer. x-axis is the number of communication passes and y-axis is the relative decrease in function value in $\log_{10}$ scale.

passes is just twice the number of outer iterations. From Figure 2, we make the following observations for *FT-k* on the *kdd2010* dataset: (a) the rate of convergence is linear for both $P = 25$ and 100, (b) it is steeper when $P = 25$. This steeper behavior for $P = 25$ is expected because the functional approximation in each node becomes better as the number of nodes decreases. Note that, almost always, the rate of convergence is better in the early stages of the optimization and becomes steady in the end stages. We observed similar linear convergence behavior for *FS-k* also. Note that the slope is dependent on k and remains nearly same when k is sufficiently large and the number of examples per node is small (see for example, the *kdd2010* dataset when $P = 100$).

x-axis because it has different meaning for different methods. For example for *SQM* and *HYBRID* each outer iteration requires different number of passes (Hessian-vector computations) over data and hence different communication also. However, our class of methods requires fixed number of passes over data as well as only 2 communication passes per outer iteration.



(a) 25 nodes (b) 100 nodes

(c) 25 nodes (d) 100 nodes

Figure 3: Plots showing overall linear convergence of our method and comparisons with *SQM* and *HYBRID* for *kdd2010*. x-axis is time (in seconds). Results are shown using with TRON and SVRG for local optimization.

Similar observations hold for the URL dataset as well.

Time Taken. Figure 3 shows the timing results. We observe that there is an optimum value of k for which we get the best result. This is because although the rate of convergence becomes better with increasing k (as discussed above), the computation cost starts increasing and becomes dominant after a certain value of k. Moreover, the optimal k value also decreases with increasing P. This happens because of two reasons. First, the computation cost increases with decreasing number of nodes. As a result the number of inner iterations that we can perform before the computation cost starts dominating the communication cost, decreases. Second, since the functional approximation becomes better as P decreases, we require lesser number of iterations to get a good descent direction. As a result, our approach does well even if k is small. From our experiments, we also observed that at the optimal k, neither communication cost nor computation cost dominates other completely. *Hence, as a rule of thumb, we recommend that the value of k should be chosen (or selected in a range) such that both the costs balance each other.*

Overall, these experiments clearly demonstrate: (a) the flexibility of our distributed algorithm in using any linear convergent local optimization algorithm, (b) a linearly convergent IPM algorithm and (c) a parallel SGD method (with its variants such as SVRG).

Comparison with other methods. For *HYBRID* and *SQM* algorithms, the number of communication passes is equal to the number of Hessian-vector and gradient computations. From Figures 2 and 3, we first see that *HYBRID* performs better than *SQM* due to warm start when the number of iterations are small. However, the performance difference between *HYBRID* and *SQM* decreases with increasing

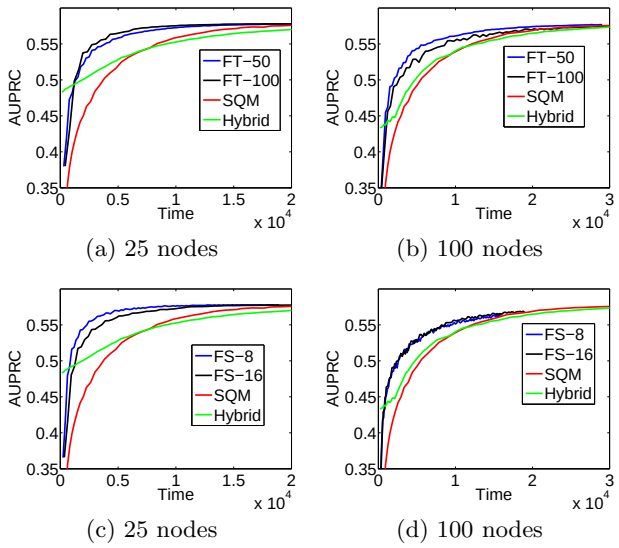
iterations and eventually *SQM* performs better. This behavior is a bit surprising and needs to be investigated.

Second, both *FS-k* and *FT-k* need significantly less communication passes (3−5 times) than *HYBRID* to reach moderately small relative error (say 10^{-3}). In this case, our algorithms perform better in terms of time also. Note that as seen in Figure 4, this is sufficient to get a good AUPRC performance; also, our algorithms (both *FT-k* and *FS-k*) reach the stable performance much quicker than other algorithms. This clearly illustrates the usefulness of our distributed algorithm when communication cost is the bottleneck.

One other important point to note is: *HYBRID* and *SQM* start performing better when a very small relative error (e.g., 10^{-6}) is desirable. This behavior can be explained as follows: In the beginning of the optimization, our functional approximation gives a good global view to all the nodes. As a result, we perform better than *SQM* and *HYBRID* by doing multiple inner iterations on this global approximation. However, closer to the optimum, the function curvature starts dominating the rate of convergence. Since *SQM* and *HYBRID* have better curvature estimates (available via global Hessian) they start performing better near the optimal solution. Hence, in summary, our approach has good global convergence but slow local movement (i.e., near the optimal solution) while *SQM* and *HYBRID* have slow global convergence but good local movement. Although theoretically one can incorporate second order functional approximation in our approach also, effectively communicating the Hessian information can be challenging. In future, we would like to incorporate ideas from Quasi-newton algorithms like L-BFGS [14] in our functional approximation and develop hybrid algorithms that switch to *SQM* at some point in our method.

Computation and Communication Costs Table 2 shows the ratio of computational cost to communication cost for three different settings of nodes and datasets for all the methods. Note that the ratio is extremely small for *HYBRID* and *SQM*. Hence, communication cost dominates the time for these two methods. On the other hand, both the costs are well balanced for the different settings of our method. Note that ratio varies in the range of $0.625 - 2.845$. This clearly shows that our approach trades-off computation with communication, while significantly reducing the number of outer iterations (Figure 2) and time (Figure 3).

To conclude, our functional approximation based distributed learning algorithm is flexible and fills several gaps in the literature. We have demonstrated that our algorithms work well when (a) the number of features is very large, (b) the functional approximation is good, and (c) moderately small relative objective function error is desired. We expect to come up with better functional approximations and hybrid algorithms in the near future that does well under all conditions.



(a) 25 nodes (b) 100 nodes

(c) 25 nodes (d) 100 nodes

Figure 4: Plots showing AUPRC metric for our method, *SQM* and *HYBRID* for *kdd2010*. *x*-axis is time (in seconds).

Table 2: Comp./Comm. cost for various methods.

FT-50	*FT-100*	*FS-8*	*FS-16*	*HYB.*	*SQM*
kdd2010, 100 nodes					
1.640	2.456	0.625	0.905	0.032	0.036
kdd2010, 25 nodes					
2.119	2.274	1.691	2.869	0.052	0.054
url, 100 nodes					
1.729	2.845	0.746	1.135	0.036	0.034

5. DISCUSSION

In this section, we discuss briefly, other different distributed settings made possible by our algorithm. The aim is to show the flexibility and generality of our approach while ensuring *glrc*.

Section 3 considered example partitioning where examples are distributed across the nodes. First, it is worth mentioning that, due to the gradient consistency condition, *partitioning* is not a necessary constraint; our theory allows examples to be resampled, i.e., each example is allowed to be a part of any number of nodes arbitrarily. For example, to reduce the number of outer iterations, it helps to have more examples in each node.

Second, the theory proposed in section 3 holds for feature partitioning also. Suppose, in each node p we restrict ourselves to a subset of features, $J_p \subset \{1, \ldots, d\}$, i.e., include the constraint, $w_p \in \{w : w(j) = w^r(j) \ \ \forall r \notin J_p\}$, where $w(j)$ denotes the weight of the j^{th} feature. Note that we do not need $\{J_p\}$ to form a partition. This is useful since important features can be included in all the nodes.

Gradient sub-consistency. Given w^r and J_p we say that $\hat{f}_p(w)$ has gradient sub-consistency with f at w^r on J_p if $\frac{\partial f}{\partial w(j)}(w^r) = \frac{\partial \hat{f}}{\partial w(j)}(w^r) \ \ \forall\, j \in J_p$.

Under the above condition, we can modify the algorithm proposed in Section 3 to come up with a feature decomposition algorithm with *glrc*.

Several feature decomposition based approaches [21, 20] have been proposed in the literature. The one closest to our method is the work by Patrikkson on a synchronized parallel algorithm [20] which extends a generic cost approximation algorithm [19] that is similar to our functional approximation. The sub-problems on the partitions are solved in parallel. Although the objective function is not assumed to be convex, the cost approximation is required to satisfy a monotone property, implying that the approximation is convex. The algorithm only has asymptotic linear rate of convergence and it requires the feature partitions to be disjoint. In contrast, our method has *glrc* and works even if features overlap in partitions. Moreover, there does not exist any counterpart of our example partitioning based distributed algorithm discussed in section 3.

Recently Mairal [16] has developed an algorithm called MISO. The main idea of MISO (which is in the spirit of the EM algorithm) is to build majorization approximations with good properties so that line search can be avoided, which is interesting. MISO is a serial method. Developing a distributed version of MISO is an interesting future direction; but, given that line search is inexpensive communication-wise, it is unclear if such a method would give great benefits.

Our approach can be easily generalized to joint example-feature partitioning as well as non-convex setting. The exact details of all the extensions mentioned above and related experiments are left for future work.

6. CONCLUSION

To conclude, we have proposed a novel functional approximation based distributed algorithm with provable global linear rate of convergence. The algorithm is general and flexible in the sense of allowing different local approximations at the node level, different algorithms for optimizing the local approximation, early stopping and general data usage in the nodes.

7. APPENDIX A: COMPLEXITY ANALYSIS

Let us use the notations of section 3 given around (14). We define the overall cost of any distributed algorithm as

$$[(c_1\frac{nz}{p}+c_2m)T^{\text{inner}}+c_3\gamma m\log_2 P]T^{\text{outer}}, \tag{15}$$

where T^{outer} is the number of outer iterations, T^{inner} is the number of inner iterations at each node before communication happens and c_1 and c_2 denote the number of passes over the data and m-dimensional dot products per inner iteration respectively. For communication, we assume an AllReduce binary tree as described in Agarwal et al [1] without pipelining. As a result, we get a multiplicative factor of log_2P in our cost. γ is the ratio of computation to communication speed. For sparse datasets γ is very large. c_3 is the number of m-dimensional vectors (gradients, Hessian-vector computations etc.) we need to communicate.

Table 3: Value of cost parameters

Method	c_1	c_2	c_3	T^{inner}
SQM	2	$\approx 5-10$	1	1
Our	1.2	0.2	2	$\hat{k}$

The values of different parameters for SQM implemented using TRON and our approach implemented using SVRG are given in Table 3. $T^{\text{outer}}_{\text{SQM}}$ is the number of overall conjugate gradient iterations plus gradient computations.

Since dense dot products are extremely fast and c_3 is a small number for both the approaches, we ignore it for simplicity. Now for our method to have lesser cost than SQM, we can use (15) to get the condition,

$$(1.2\hat{k}T^{\text{outer}}_{\text{SVRG}}-2T^{\text{outer}}_{\text{SQM}})\frac{nz}{P}\leq(T^{\text{outer}}_{\text{SQM}}-2T^{\text{outer}}_{\text{SVRG}})\gamma m\log_2 P \tag{16}$$

Ignoring $T^{\text{outer}}_{\text{SQM}}$ on the left side of this inequality and rearranging, we get the looser condition,

$$\frac{nz}{m}\leq\frac{\gamma P\log_2 P}{\hat{k}}\frac{1}{1.2}(\frac{T^{\text{outer}}_{\text{SQM}}}{T^{\text{outer}}_{\text{SVRG}}}-2) \tag{17}$$

Assuming $T^{\text{outer}}_{\text{SQM}} > 3.2T^{\text{outer}}_{\text{SVRG}}$, we arrive at the final condition in (14).

8. APPENDIX B: PROOFS

8.1 Proofs of the results in section 2

Let us now consider the establishment of the convergence theory given in section 2.

Proof of Lemma 1. Let $\rho(t)=f(w^r+td^r)$ and $\gamma(t)=\rho(t)-\rho(0)-\alpha t\rho'(0)$. Note the following connections with quantities involved in Lemma 1: $\rho(t)=f^{r+1}$, $\rho(0)=f^r$, $\rho'(t)=g^{r+1}\cdot d^r$ and $\gamma(t)=f^{r+1}-f^r-\alpha g^r\cdot(w^{r+1}-w^r)$. (3) corresponds to the condition $\gamma(t)\leq 0$ and (4) corresponds to the condition $\rho'(t)\geq\beta\rho'(0)$.

$\gamma'(t)=\rho'(t)-\alpha\rho'(0)$. $\rho'(0)<0$. ρ' is strictly monotone increasing because, by assumption A2,

$$\rho'(t)-\rho'(\tilde{t})\geq\sigma(t-\tilde{t})\|d^r\|^2\ \ \forall\, t,\tilde{t} \tag{18}$$

This implies that γ' is also strictly monotone increasing and, all four, ρ, ρ', γ' and γ tend to infinity as t tends to infinity.

Let t_β be the point at which $\rho'(t)=\beta\rho'(0)$. Since $\rho'(0)<0$ and ρ' is strictly monotone increasing, t_β is unique and $t_\beta>0$. This validates the definition in (5). Monotonicity of ρ' implies that (4) is satisfied iff $t\geq t_\beta$.

Note that $\gamma(0)=0$ and $\gamma'(0)<0$. Also, since γ' is monotone increasing and $\gamma(t)\to\infty$ as $t\to\infty$, there exists a unique $t_\alpha>0$ such that $\gamma(t_\alpha)=0$, which validates the definition in (6). It is easily checked that $\gamma(t)\leq 0$ iff $t\in[0,t_\alpha]$.

The properties also imply $\gamma'(t_\alpha)>0$, which means $\rho'(t_\alpha)\geq\alpha\rho'(0)$. By the monotonicity of ρ' we get $t_\alpha>t_\beta$, proving the lemma.

Proof of Theorem 2. Using (4) and A1,

$$(\beta-1)g^r\cdot d^r\leq(g^{r+1}-g^r)\cdot d^r\leq Lt\|d^r\|^2 \tag{19}$$

This gives a lower bound on t:

$$t\geq\frac{(1-\beta)}{L\|d^r\|^2}(-g^r\cdot d^r) \tag{20}$$

Using (3), (20) and (1) we get

$$\begin{aligned} f^{r+1} &\leq f^r+\alpha tg^r\cdot d^r \\ &\leq f^r-\frac{\alpha(1-\beta)}{L\|d^r\|^2}(-g^r\cdot d^r)^2 \\ &\leq f^r-\frac{\alpha(1-\beta)}{L}\cos^2\theta\|g^r\|^2 \end{aligned} \tag{21}$$

Subtracting $f^\star$ gives

$$(f^{r+1} - f^\star) \leq (f^r - f^\star) - \frac{\alpha(1-\beta)}{L}\cos^2\theta\|g^r\|^2 \quad (22)$$

A2 together with $g(w^\star) = 0$ implies $\|g^r\|^2 \geq \sigma^2\|w^r - w^\star\|^2$. Also A1 implies $f^r - f^\star \leq \frac{L}{2}\|w^r - w^\star\|^2$ [22]. Using these in (22) gives

$$\begin{aligned}(f^{r+1} - f^\star) &\leq (f^r - f^\star) - 2\alpha(1-\beta)\frac{\sigma^2}{L^2}\cos^2\theta(f^r - f^\star) \\ &\leq (1 - 2\alpha(1-\beta)\frac{\sigma^2}{L^2}\cos^2\theta)(f^r - f^\star) \quad (23)\end{aligned}$$

Let $\delta = (1 - 2\alpha(1-\beta)\frac{\sigma^2}{L^2}\cos^2\theta)$. Clearly $0 < \delta < 1$. Theorem 2 follows.

8.2 Proofs of the results in section 3

Let us now consider the establishment of the convergence theory given in section 3. We begin by establishing that the exact minimizer of $\hat{f}_p$ makes a sufficient angle of descent at w^r.

Lemma 5. Let $\hat{w}_p^\star$ be the minimizer of $\hat{f}_p$. Let $d_p = (\hat{w}_p^\star - w^r)$. Then

$$-g^r \cdot d_p \geq (\sigma/L)\|g^r\|\|d_p\| \quad (24)$$

Proof. First note, using gradient consistency and $\nabla f_p(\hat{w}_p^\star) = 0$ that

$$\|g^r\| = \|\nabla\hat{f}_p(w^r) - \nabla\hat{f}_p(\hat{w}_p^\star)\| \leq L\|d_p\| \quad (25)$$

Now,

$$\begin{aligned}-g^r \cdot d_p &= (\nabla\hat{f}_p(w^r) - \nabla\hat{f}_p(\hat{w}_p^\star))^T(w^r - \hat{w}_p^\star) \\ &\geq \sigma\|d_p\|^2 \\ &= \sigma\|g^r\|\|d_p\|\frac{\|d_p\|}{\|g^r\|} \\ &\geq \frac{\sigma}{L}\|g^r\|\|d_p\| \quad (26)\end{aligned}$$

where the second line comes from σ-strong convexity and the fourth line follows from (25).

Proof of Lemma 3. Let us now turn to the question of approximate stopping and establish Lemma 3. Given θ satisfying (11) let us choose $\zeta \in (0,1)$ such that

$$\frac{\pi}{2} > \theta > \cos^{-1}\frac{\sigma}{L} + \cos^{-1}\zeta \quad (27)$$

By A3 and equations (3.16) and (3.22) in [22], we get

$$\frac{\sigma}{2}\|v - \hat{w}_p^\star\|^2 \leq \hat{f}_p(v) - \hat{f}_p^\star \leq \frac{L}{2}\|v - \hat{w}_p^\star\|^2 \quad (28)$$

After k iterations we have

$$\hat{f}_p(v^k) - \hat{f}_p^\star \leq \delta^k(\hat{f}_p(w^r) - \hat{f}_p^\star) \quad (29)$$

We can use these to get

$$\begin{aligned}\|v^k - \hat{w}_p^\star\|^2 &\leq \frac{2(\hat{f}_p(v^k) - \hat{f}_p^\star)}{\sigma} \\ &\leq \frac{2\delta^k(\hat{f}_p(w^r) - \hat{f}_p^\star)}{\sigma} \\ &\leq \frac{\delta^k L}{\sigma}\|w^r - \hat{w}_p^\star\|^2 \stackrel{\text{def}}{=} (r^k)^2 \quad (30)\end{aligned}$$

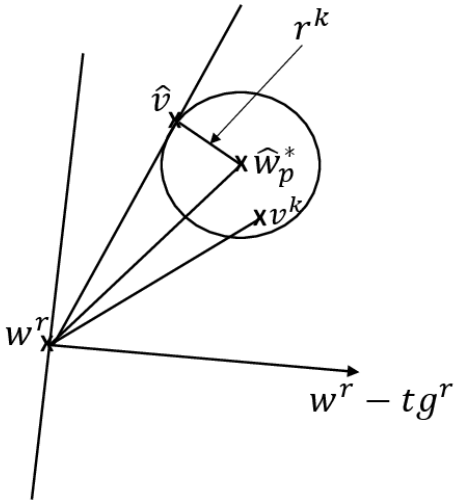


Figure 5: Construction used in the proof of Lemma 3.

For now let us assume the following:

$$\|v^k - \hat{w}_p^\star\|^2 \leq \|w^r - \hat{w}_p^\star\|^2 \quad (31)$$

Using (30) note that (31) holds if

$$\frac{\delta^k L}{\sigma} \leq 1 \quad (32)$$

Let S^k be the sphere, $S^k = \{v : \|v - \hat{w}_p^\star\|^2 \leq (r^k)^2\}$. By (30) we have $v^k \in S^k$. See Figure 8.2. Therefore,

$$\phi^k \leq \max_{v \in S^k}\phi(v) \quad (33)$$

where ϕ^k is the angle between $\hat{w}_p^\star - w^r$ and $v^k - w^r$, and $\phi(v)$ is the angle between $v - w^r$ and $\hat{w}_p^\star - w^r$. Given the simple geometry, it is easy to see that $\max_{v\in S^k}\phi(v)$ is attained by a point $\hat{v}$ lying on the boundary of S^k (i.e., $\|\hat{v} - \hat{w}_p^\star\|^2 = (r^k)^2$) and satisfying $(\hat{v} - \hat{w}_p^\star) \perp (\hat{v} - w^r)$. This geometry yields

$$\begin{aligned}\cos^2\phi(\hat{v}) &= \frac{\|\hat{v} - w^r\|^2}{\|\hat{w}_p^\star - w^r\|^2} \\ &= \frac{\|\hat{w}_p^\star - w^r\|^2 - (r^k)^2}{\|\hat{w}_p^\star - w^r\|^2} \\ &= 1 - \frac{(r^k)^2}{\|\hat{w}_p^\star - w^r\|^2} = 1 - \frac{\delta^k L}{\sigma} \quad (34)\end{aligned}$$

Since $\phi^k \leq \phi(\hat{v})$,

$$\cos^2\phi^k \geq 1 - \frac{\delta^k L}{\sigma} \quad (35)$$

Thus, if

$$1 - \frac{\delta^k L}{\sigma} \geq \zeta^2 \quad (36)$$

then

$$\cos\phi^k \geq \zeta \ \ \forall k \geq \hat{k} \quad (37)$$

holds. By (27) this yields $\angle(-g^r, v^k - w^r \leq \theta$, the result needed in Lemma 3. Since $\zeta > 0$, (36) implies (32), so (31)

holds and there is no need to separately satisfy it. Now (36) holds if

$$k \geq \hat{k} \stackrel{\text{def}}{=} \frac{\log(L/(\sigma(1-\zeta^2)))}{\log(1/\delta)} \tag{38}$$

which proves the lemma.

Proof of Theorem 4. It trivially follows from a combination of Lemma 3 and Theorem 2.

9. REFERENCES

[1] A. Agarwal, O. Chapelle, M. Dudik, and J. Langford. A reliable effective terascale linear learning system. In *arXiv:1140.4198*, 2011.

[2] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and distributed computation: Numerical methods*. Athena Scientific, Cambridge, MA, 1997.

[3] L. Bottou. Large-scale machine learning with stochastic gradient descent. In *COMPSTAT'2010*, pages 177–187, 2010.

[4] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, pages 1–122, 2011.

[5] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge University Press, Cambridge, UK, 2004.

[6] K.W. Chang, C.J. Hsieh, and C.J. Lin. Coordinate descent method for large-scale l2-loss linear SVM. *JMLR*, pages 1369–1398, 2008.

[7] C.T. Chu, S.K. Kim, Y.A. Lin, Y.Y. Yu, G. Bradski, A.Y. Ng, and K. Olukotun. Map-reduce for machine learning on multicore. *NIPS*, pages 281–288, 2006.

[8] W. Deng and W. Yin. On the global linear convergence of the generalized alternating direction method of multipliers. *Rice University CAAM Technical Report*, TR12-14, 2012.

[9] K.B. Hall, S. Gilpin, and G. Mann. Mapreduce/bigtable for distributed optimization. In *NIPS Workshop on Leaning on Cores, Clusters, and Clouds*, 2010.

[10] C.J. Hsieh, K.W. Chang, C.J. Lin, S.S. Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear SVM. In *ICML*, pages 408–415, 2008.

[11] R. Johnson and T. Zhang. Accelerating stochastic gradient descent using predictive variance reduction. *NIPS*, 2013.

[12] N. Le Roux, M. Schmidt, and F. Bach. A stochastic gradient method with an exponential convergence rate for strongly convex optimization with finite training sets. In *arXiv*, 2012.

[13] C.J. Lin, R.C. Weng, and S.S. Keerthi. Trust region newton method for large-scale logistic regression. *JMLR*, pages 627–650, 2008.

[14] D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Math. Programming*, 45(3, (Ser. B)):503–528, 1989.

[15] D. Mahajan, S. S. Keerthi, S. Sundararajan, and L. Bottou. A parallel SGD method with strong convergence. *NIPS Workshop on Optimization in Machine Learning*, 2013.

[16] J. Mairal. Optimization with first order surrogate functions. *ICML*, 2013.

[17] G. Mann, R.T. McDonald, M. Mohri, N. Silberman, and D. Walker. Efficient large-scale distributed training of conditional maximum entropy models. In *NIPS*, pages 1231–1239, 2009.

[18] R.T. McDonald, K. Hall, and G. Mann. Distributed training strategies for the structured perceptron. In *HLT-NAACL*, pages 456–464, 2010.

[19] M. Patriksson. Cost approximation: A unified framework of descent algorithms for nonlinear programs. *SIAM J. on Optimization*, 8:561–582, 1998.

[20] M. Patriksson. Decomposition methods for differentiable optimization problems over cartesian product sets. *Comput. Optim. Appl.*, 9:5–42, 1998.

[21] P. Richtárik and M. Takác. Parallel coordinate descent methods for big data optimization. *CoRR*, abs/1212.0873, 2012.

[22] A. Smola and S.V.N. Vishwanathan. *Introduction to Machine Learning*. Cambridge University Press, Cambridge, UK, 2008.

[23] P.W. Wang and C.J. Lin. Iteration complexity of feasible descent methods for convex optimization. *Technical Report, National Taiwan University*, 2013.

[24] P. Wolfe. Convergence conditions for ascent methods. *SIAM Review*, 11:226–235, 1969.

[25] P. Wolfe. Convergence conditions for ascent methods: II: Some corrections. *SIAM Review*, 13:185–188, 1971.

[26] M. Zinkevich, M. Weimer, A. Smola, and L. Li. Parallelized stochastic gradient descent. In *NIPS*, pages 2595–2603, 2010.