Journal of Machine Learning Research 20 (2019) 1-49 Submitted 6/16; Revised 9/18; Published 3/19

The Common-directions Method for Regularized Empirical Risk Minimization

Po-Wei Wang POWEIW@CS.CMU.EDU
Machine Learning Department
Carnegie Mellon University
Pittsburgh, PA 15213, USA
Ching-pei Lee CHING-PEI@CS.WISC.EDU
Department of Computer Sciences
University of Wisconsin–Madison
Madison, WI 53706-1613, USA
Chih-Jen Lin CJLIN@CSIE.NTU.EDU.TW
Department of Computer Science
National Taiwan University
Taipei 106, Taiwan

Editor: Tong Zhang

Abstract

State-of-the-art first- and second-order optimization methods are able to achieve either fast global linear convergence rates or quadratic convergence, but not both of them. In this work, we propose an interpolation between first- and second-order methods for regularized empirical risk minimization that exploits the problem structure to efficiently combine multiple update directions. Our method attains both optimal global linear convergence rate for first-order methods, and local quadratic convergence. Experimental results show that our method outperforms state-of-the-art first- and second-order optimization methods in terms of the number of data accesses, while is competitive in training time.

1. Introduction

Consider the general problem

$$\min_{\boldsymbol{w}\in\mathbb{R}^n} \quad f(\boldsymbol{w}), \tag{1}$$

where $f(\boldsymbol{w})$ is convex and continuously differentiable. Most iterative optimization methods generate iterates by using the first- and second-order derivatives. After a new step is generated, the previous steps and related calculation results are usually considered outdated and thus discarded. In this work, we show that we can reuse previous descent directions in later iterations to reduce computation and to achieve both linear and quadratic convergence.

The idea behind our proposed method is simple. We store all previous directions, and approximately solve the subproblem

$$\min_{\boldsymbol{t}\in\mathbb{R}^m} \quad f(\boldsymbol{w}+P\boldsymbol{t}), \tag{2}$$

where the columns of $P = [\boldsymbol{p}_1, \ldots, \boldsymbol{p}_m]$ form an orthonormal basis of the span of the previous update directions. The resulting direction $P\boldsymbol{t}$ is then used to update the current iterate $\boldsymbol{w}$ for the next iteration. Utilizing all previous directions obtained from the first iteration on seems to be computationally and spatially expensive, especially in the later stage of this method. However, we note that for regularized empirical risk minimization (ERM) of linear models, the method comes with little additional cost by wisely caching the inner products between these $\boldsymbol{p}_i$ and the training data. We will exemplify how to use this caching strategy to significantly accelerate the computation in our method for ERM. Experimental results show that our method outperforms state of the art.

There are other optimization methods that also reuse previous update directions. For example, the heavy-ball method (Polyak, 1964), the (nonlinear) conjugate gradient method (Fletcher and Reeves, 1964), and Nesterov's accelerated gradient method (Nesterov, 1983, 2013) reuse the previous direction $\boldsymbol{w}_k - \boldsymbol{w}_{k-1}$ together with the current gradient to decide the update direction at the k-th iteration. On the other hand, quasi-Newton methods such as BFGS (Dennis and Moré, 1977) and L-BFGS (Liu and Nocedal, 1989) store previous gradients to construct an approximation of the Hessian. More details of these methods and empirical comparisons are provided in later sections.

This paper is organized as follows. We describe the details of our algorithm in Section 2. We prove global linear convergence and local quadratic convergence. Section 3 illustrates important techniques for accelerating the computation when solving ERM problems. We then extend the proposed method to a more general framework and obtain a better linear convergence rate in Section 4. Related works are discussed in Section 5. Experimental results in Section 6 show that our method outperforms state-of-the-art methods empirically in terms of the number of data accesses, and is competitive in terms of training time. We then consider some interesting extensions in Section 7. Section 8 concludes the paper.

Programs used for experiments and a supplementary file including additional results can be found at `http://www.csie.ntu.edu.tw/~cjlin/papers/commdir/`.

2. Common-directions Method

Before starting the description of our algorithm, we first specify the type of problems we consider. For all our theoretical results except Theorem 6, we require that $f(\boldsymbol{w})$ satisfies Assumption 1.

Assumption 1 *The objective function $f(\boldsymbol{w})$ is differentiable, ρ Lipschitz smooth and σ strongly convex with some constants $\rho \geq \sigma > 0$.*[1] *That is,*

$$\|\nabla f(\boldsymbol{u}) - \nabla f(\boldsymbol{v})\| \leq \rho \|\boldsymbol{u} - \boldsymbol{v}\|, \quad \forall \boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^n, \tag{3}$$

and

$$f(\boldsymbol{u}) - f(\boldsymbol{v}) - \nabla f(\boldsymbol{v})^\top (\boldsymbol{u} - \boldsymbol{v}) \geq \frac{\sigma}{2} \|\boldsymbol{u} - \boldsymbol{v}\|^2, \quad \forall \boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^n. \tag{4}$$

For some theoretical results, we further need that $f(\boldsymbol{w})$ satisfies Assumption 2.

1. The requirement $\rho \geq \sigma$ comes from the conditions (3) and (4).

Algorithm 1: A framework for the common-directions method

Given $\boldsymbol{w}_0$, compute $\nabla f(\boldsymbol{w}_0)$;
$P = [\nabla f(\boldsymbol{w}_0)/\|\nabla f(\boldsymbol{w}_0)\|]$;
for *k=0,1,... in the outer loop* **do**
  if $\nabla f(\boldsymbol{w}_k) = \mathbf{0}$, or some stopping condition is satisfied **then**
    stop;
  end
  Let $\boldsymbol{t}_k = -(P^\top \nabla^2 f(\boldsymbol{w}_k) P)^{-1} P^\top \nabla f(\boldsymbol{w}_k)$;
  Backtracking line search on $f(\boldsymbol{w}_k + \theta P \boldsymbol{t}_k)$ by Algorithm 2 to obtain θ_k;
  $\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + \theta_k P \boldsymbol{t}_k$;
  Compute $\nabla f(\boldsymbol{w}_{k+1})$;
  Let $\boldsymbol{p} = \nabla f(\boldsymbol{w}_{k+1}) - P(P^\top \nabla f(\boldsymbol{w}_{k+1}))$;
  if $\boldsymbol{p} \neq \mathbf{0}$ **then**
    $P = [P,\ \boldsymbol{p}/\|\boldsymbol{p}\|]$;
  end
end

Algorithm 2: Backtracking line search

Given $\beta \in (0,1)$, $\lambda > 0$, the current iterate $\boldsymbol{w}$, a descent direction $P\boldsymbol{t}$;
Let $\theta := 1$;
while $f(\boldsymbol{w}) - f(\boldsymbol{w} + \theta P \boldsymbol{t}) < \frac{\lambda}{2}\theta^2 \|P\boldsymbol{t}\|^2$ **do**
  $\theta := \beta\theta$;
end

Assumption 2 *The objective function $f(\boldsymbol{w})$ is twice-differentiable, and the Hessian of $f(\boldsymbol{w})$ is M Lipschitz continuous with some constant $M > 0$. That is,*

$$\|\nabla^2 f(\boldsymbol{u}) - \nabla^2 f(\boldsymbol{v})\| \leq M\|\boldsymbol{u} - \boldsymbol{v}\|, \quad \forall \boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^n.$$

When minimizing a function, we usually only work on derivatives of the current iterate. For example, a gradient descent step can be derived from searching along the negative gradient direction, and a Newton step can be expressed by the current gradient and Hessian. However, after obtaining the update direction, stale derivatives are usually discarded, which can be inefficient. A simple idea is to store all previous directions $\boldsymbol{p}_j$ in $P = [\boldsymbol{p}_1, \ldots, \boldsymbol{p}_m]$ and consider working on the linear span of $\boldsymbol{p}_j$. If possible, we search for the best linear combination $P\boldsymbol{t}$ such that $f(\boldsymbol{w} + P\boldsymbol{t})$ is the smallest; see the sub-problem (2). However, in practice (2) may be difficult to be solved, so following past optimization development, we consider the second-order Taylor approximation of $f(\boldsymbol{w})$ as

$$\tilde{f}(\boldsymbol{u} \mid \boldsymbol{w}) \equiv f(\boldsymbol{w}) + \nabla f(\boldsymbol{w})^\top (\boldsymbol{u} - \boldsymbol{w}) + \frac{1}{2}(\boldsymbol{u} - \boldsymbol{w})^\top \nabla^2 f(\boldsymbol{w})(\boldsymbol{u} - \boldsymbol{w}), \tag{5}$$

where $\nabla^2 f(\boldsymbol{w})$ is the Hessian at $\boldsymbol{w}$. Then, a descent direction $\boldsymbol{d} = P\boldsymbol{t}$ can be found by optimizing $\tilde{f}$ along all the directions $\boldsymbol{p}_j$

$$\min_{\boldsymbol{t}} \quad \tilde{f}(\boldsymbol{w} + \sum_{j=1}^{m} t_j \boldsymbol{p}_j \mid \boldsymbol{w}) = (P^\top \nabla f(\boldsymbol{w}))^\top \boldsymbol{t} + \frac{1}{2}\boldsymbol{t}^\top (P^\top \nabla^2 f(\boldsymbol{w}) P)\boldsymbol{t}. \tag{6}$$

Problem (6) can be solved by the following linear system.

$$(P^\top \nabla^2 f(\boldsymbol{w}) P)\boldsymbol{t} = -P^\top \nabla f(\boldsymbol{w}).$$

Say we have stored m previous directions in $P \in \mathbb{R}^{m\times n}$, where $m \ll n$. Problem (6) has m variables but if instead we consider the Newton direction by solving the problem in (5), the number of variables is n. Therefore, solving (6) should be easier than finding a Newton direction if the construction of $P\nabla^2 f(\boldsymbol{w})P^\top$ is not too expensive. In Section 3, we will show that when the Hessian is structured, the subproblem (6) can be solved easily.

From the above concept, we propose the common-directions method. In each iteration of this method, given the orthonormal basis P of the span of the past directions, we solve the subproblem (6) with respect to $\boldsymbol{t}$ to obtain the update direction $P\boldsymbol{t}$. A line search along $P\boldsymbol{t}$ is performed to guarantee the strict decrease of the function value. After updating the iterate $\boldsymbol{w}$, a new direction $-\nabla f(\boldsymbol{w})$ is then taken to enlarge the orthonormal basis P by Gram-Schmidt orthogonalization. A description of the common-directions method is given in Algorithm 1, and the line-search sub-routine is described in Algorithm 2. Note that other choices of the directions being added to P are also possible, but for the ease of description and analysis, we confine our choice to the current steepest descent direction.

We define the following quantities before stating and proving the theorems. We denote the condition number of f by

$$\kappa \equiv \frac{\rho}{\sigma},$$

and define an ϵ-accurate solution of f to be a point $\boldsymbol{w}$ such that

$$f(\boldsymbol{w}) - f(\boldsymbol{w}^*) \leq \epsilon,$$

where $\boldsymbol{w}^*$ is the optimal solution of f. Note that under Assumption 1, problem (1) possesses a unique optimal solution. The following theorems present theoretical convergence of the proposed method. In particular, we show that the line search sub-routine has finite termination, and our method is globally linearly convergent and locally quadratically convergent.

Theorem 3 *Under Assumption 1, every time being called by Algorithm 1, Algorithm 2 terminates within* $\lceil \log_\beta(\beta\sigma/(\rho+\lambda))\rceil$ *steps, and Algorithm 1 converges Q-linearly with iteration complexity* $O(\kappa^3 \log(1/\epsilon))$ *to achieve an* ϵ*-accurate solution. Further, if Algorithm 2 always stops at step size* 1*, the iteration complexity becomes* $O(\kappa \log(1/\epsilon))$*.*

In practice, we observed that the common-direction method usually accepts step size 1.

Theorem 4 *Under Assumptions 1 and 2, Algorithm 1 converges quadratically after certain iteration.*

The proofs are presented in Appendix A.

3. Application on Regularized Empirical Risk Minimization Problems

For most optimization problems, computing the $n \times n$ Hessian matrix, or its inverse, whose cost is $O(n^3)$, is much more expensive than obtaining the function value and the n-dimensional gradient vector. To tackle this problem, different approaches have been considered, and in some cases the problem structure can be utilized to reduce the cost. We are interested in solving machine learning problems, and thus we will focus our discussion on differentiable regularized ERM problems. More specifically, given training feature-label pairs $(\boldsymbol{x}_i, y_i)$, $i = 1, \ldots, l$, with $\boldsymbol{x}_i \in \mathbb{R}^n$, $y_i \in \mathbb{R}$, and a parameter $C > 0$, we consider L2-regularized ERM problems

$$\min_{\boldsymbol{w}\in\mathbb{R}^n} \quad f(\boldsymbol{w}), \quad \text{where } f(\boldsymbol{w}) \equiv \frac{1}{2}\boldsymbol{w}^\top \boldsymbol{w} + C\sum_{i=1}^{l} \xi(y_i; \boldsymbol{w}^\top \boldsymbol{x}_i), \tag{7}$$

and ξ is a loss function convex in $\boldsymbol{w}^\top \boldsymbol{x}_i$. The first term in (7) makes f be σ strongly convex with $\sigma \geq 1$, and the choice of loss functions ensures that $\nabla f(\boldsymbol{w})$ is Lipschitz continuous, though the corresponding constant ρ may be unknown. Therefore, Assumption 1 is satisfied.

For problem (7), by defining

$$X \equiv \begin{pmatrix} \boldsymbol{x}_1^\top \\ \vdots \\ \boldsymbol{x}_l^\top \end{pmatrix},$$

we can write the gradient as

$$\nabla f(\boldsymbol{w}) = \boldsymbol{w} + X^\top \boldsymbol{v}_{\boldsymbol{w}}, \text{ where } (v_{\boldsymbol{w}})_i = C\frac{\partial}{\partial z}\xi(y_i; z) \mid_{z=\boldsymbol{w}^\top \boldsymbol{x}_i}, i = 1, \ldots, l, \tag{8}$$

and if $f(\boldsymbol{w})$ is twice-differentiable, the Hessian is

$$\nabla^2 f(\boldsymbol{w}) \equiv I + X^\top D_{\boldsymbol{w}} X, \tag{9}$$

where I is the identity matrix and $D_{\boldsymbol{w}}$ is a diagonal matrix with

$$(D_{\boldsymbol{w}})_{ii} \equiv C\frac{\partial^2}{\partial z^2}\xi(y_i; z) \mid_{z=\boldsymbol{w}^\top \boldsymbol{x}_i}, i = 1, \ldots, l. \tag{10}$$

The structure of (9) allows efficient computations to use the Hessian in various ways. Our method utilizes it to incorporate the Hessian-vector products with low cost as discussed in the following sections.

Some problems of the form (7), such as logistic regression and ridge regression, also satisfy Assumption 2. The loss of ridge regression is

$$\xi(y_i; \boldsymbol{w}^\top \boldsymbol{x}_i) = (y_i - \boldsymbol{w}^\top \boldsymbol{x}_i)^2.$$

It is clearly twice-differentiable with the diagonal matrix $D_{\boldsymbol{w}}$ defined in (10) being the identity matrix. Clearly, this loss satisfies Assumption 2 with any $M \geq 0$ as the Hessian is a constant. The loss of logistic regression is

$$\xi(y_i; \boldsymbol{w}^\top \boldsymbol{x}_i) = \log(1 + \exp(-y_i \boldsymbol{w}^\top \boldsymbol{x}_i)),$$

where $y_i \in \{-1, 1\}$. Because its third derivative is upper-bounded, by applying first-order Taylor expansion on the Hessian, one can see that Assumption 2 holds. On the other hand, problems that satisfy Assumption 1 only, like squared-hinge loss support vector classification/regression (Boser et al., 1992; Vapnik, 1995), can also be solved by our algorithm, although in this case we can only expect linear convergence.[2]

3.1 Subproblem of Common Directions

At the k-th iteration, given the matrix $P_k = [\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_m] \in \mathbb{R}^{n\times m}$ with orthogonal columns, we consider solving (6) along $P_k\boldsymbol{t}$

$$\min_{\boldsymbol{t}\in\mathbb{R}^m} \quad \tilde{f}(\boldsymbol{w}_k + P_k\boldsymbol{t} \mid \boldsymbol{w}_k). \tag{11}$$

The gradient and the Hessian of (11) with respect to the variable $\boldsymbol{t}$ at $\boldsymbol{t} = \boldsymbol{0}$ are respectively

$$\nabla_{\boldsymbol{t}} \tilde{f}(\boldsymbol{w}_k + P_k\boldsymbol{t}) \mid_{\boldsymbol{t}=\boldsymbol{0}} = P_k^\top \nabla f(\boldsymbol{w}_k) \tag{12}$$

and

$$\nabla^2_{\boldsymbol{t},\boldsymbol{t}} \tilde{f}(\boldsymbol{w}_k + P_k\boldsymbol{t}) \mid_{\boldsymbol{t}=\boldsymbol{0}} = P_k^\top \nabla^2 f(\boldsymbol{w}_k) P_k = P_k^\top (I + X^\top D_{\boldsymbol{w}_k} X) P_k = I + P_k^\top X^\top D_{\boldsymbol{w}_k} X P_k. \tag{13}$$

Observe the term XP_k. The trick to make the computation efficient is to store this term in memory to avoid repetitive computations of $X\boldsymbol{p}_j$ for all previous directions $\boldsymbol{p}_j$. If XP_k, $\nabla f(\boldsymbol{w}_k)$, and $D_{\boldsymbol{w}_k}$ are available, the construction of (12)-(13) involves the following cost.

- $P_k^\top \nabla f(\boldsymbol{w}_k)$: $O(mn)$,
- $(XP_k)^\top D_{\boldsymbol{w}_k} (XP_k)$: $O(lm^2)$.

The optimal solution $\boldsymbol{t}$ of (11) can be obtained by solving the following linear system

$$\left(I + P_k^\top X^\top D_{\boldsymbol{w}_k} X P_k\right) \boldsymbol{t} = -P_k^\top \nabla f(\boldsymbol{w}_k) \tag{14}$$

in $O(m^3)$ cost. Note that the Hessian with respect to $\boldsymbol{t}$ is always invertible because the term I in (13) ensures its positive definiteness.

Let $\boldsymbol{t}_k$ be the solution of (14). We show that to obtain necessary information for the next iteration, it is sufficient to maintain XP_{k+1} and $X\boldsymbol{w}_{k+1}$. To have the new function value, from (7) we mainly need to compute $X\boldsymbol{w}_{k+1}$. If $X\boldsymbol{w}_k$ is available and the full direction $(XP_k)\boldsymbol{t}_k$ is taken, by

$$X\boldsymbol{w}_{k+1} = X\boldsymbol{w}_k + \theta(XP_k)\boldsymbol{t}_k, \tag{15}$$

with $\theta = 1$, in $O(lm)$ cost $X\boldsymbol{w}_{k+1}$ is obtained. In practice, a step size θ is decided by line search, which involves some minor cost as shown in Section 3.2. The vector $X\boldsymbol{w}_{k+1}$ is also used to construct $\boldsymbol{v}_{\boldsymbol{w}_{k+1}}$ needed for gradient calculation. Then from (8),

$$\nabla f(\boldsymbol{w}_{k+1}) = \boldsymbol{w}_{k+1} + X^\top \boldsymbol{v}_{\boldsymbol{w}_{k+1}}$$

2. If the objective function is not twice differentiable, we can use the generalized Hessian (Mangasarian, 2002) to replace the Hessian in the Newton step, as Assumption 1 guarantees the existence of generalized Hessian.

takes $O(\#\text{nnz})$ cost, where #nnz is the number of non-zero elements in X. We then apply the Gram-Schmidt procedure to check if a new column should be added to P_{k+1}. In practice this procedure is computed by

$$\boldsymbol{p}_{m+1} = (I - P_k P_k^T)\nabla f(\boldsymbol{w}_{k+1}) = \nabla f(\boldsymbol{w}_{k+1}) - P_k(P_k^\top \nabla f(\boldsymbol{w}_{k+1})).$$

The cost is $O(mn)$. If $\boldsymbol{p}_{m+1} \neq \boldsymbol{0}$, to maintain XP_{m+1} after $\boldsymbol{p}_{m+1}$ is added, we must compute $X\boldsymbol{p}_{m+1}$, which costs $O(\#\text{nnz})$. Details of the algorithm is shown in Algorithm 3.

In summary, the cost per iteration is

$$O(lm^2 + mn + m^3 + \#\text{nnz}). \tag{16}$$

Because usually we have $n \gg m$ and $l \gg m$ from the first iteration till reaching a good enough solution, in general #nnz is still the dominant term in (16). Therefore, each of our iterations does not cost much more than a typical iteration in an optimization method that often involves the $O(\#\text{nnz})$ cost.

Further, for the same set of directions P_k, we could solve the subproblem (11) and update $\boldsymbol{w}_k$ multiple times in order to approximately solve (2). Note that the update of $\boldsymbol{w}_k$ does not need to go through the data matrix X. The reason is that the $D_{\boldsymbol{w}}$ matrix and the $\boldsymbol{v}_{\boldsymbol{w}}$ vector rely on $\boldsymbol{w}_k$ only through the inner products $\boldsymbol{w}_k^\top \boldsymbol{x}_i$, and this is a linear combination of XP_k, and thus we can compute it without additional inner product computations with the data matrix X. This idea of solving (11) multiple times will be exploited in Section 4 to achieve the optimal linear convergence rate for first-order methods.

3.2 Line Search on Common Directions

Once the solution $\boldsymbol{t}$ of (14) has been obtained, in $O(mn)$ cost we get the descent direction

$$\boldsymbol{d} = P_k \boldsymbol{t}. \tag{17}$$

Next, we conduct line search in Algorithm 2 for the step size θ in

$$f(\boldsymbol{w}_k + \theta \boldsymbol{d}) = \frac{1}{2}\|\boldsymbol{w}_k + \theta \boldsymbol{d}\|^2 + C\sum_{i=1}^{l} \xi(y_i; \boldsymbol{w}_k^\top \boldsymbol{x}_i + \theta \boldsymbol{d}^\top \boldsymbol{x}_i).$$

The sufficient decrease condition is

$$C\sum_{i=1}^{l} \xi(y_i; \boldsymbol{w}_k^\top \boldsymbol{x}_i) - \theta \boldsymbol{w}_k^\top \boldsymbol{d} - \frac{\theta^2}{2}\|\boldsymbol{d}\|^2 - C\sum_{i=1}^{l} \xi(y_i; \boldsymbol{w}_k^\top \boldsymbol{x}_i + \theta \boldsymbol{d}^\top \boldsymbol{x}_i) \geq \frac{\lambda}{2}\theta^2 \|\boldsymbol{d}\|^2.$$

Because XP_k is maintained, after obtaining $X\boldsymbol{d}$ in $O(lm)$ time by

$$X\boldsymbol{d} = (XP_k)\boldsymbol{t}, \tag{18}$$

and calculating $\boldsymbol{w}_k^\top \boldsymbol{d}$, $\boldsymbol{w}_k^\top \boldsymbol{w}_k$, $\boldsymbol{d}^\top \boldsymbol{d}$ in $O(n)$ time, each line search step would only cost $O(l)$ because $\boldsymbol{w}_k^\top \boldsymbol{x}_i$ is already available in our calculation for the gradient and the Hessian. Thus, we could perform line search in

$$O(lm + mn + l \times \#(\text{line search steps})) \text{ time}. \tag{19}$$

By Theorem 3, #(line search) is bounded by a constant, and thus the complexity does not grow infinitely. After finishing the line search procedure, we already have $\boldsymbol{w}_k^\top \boldsymbol{x}_i + \theta \boldsymbol{d}^\top \boldsymbol{x}_i$ as the next $\boldsymbol{w}_{k+1}^\top \boldsymbol{x}_i$ as shown in (15).

Algorithm 3: The common-directions method for solving the ERM problem (7)

Given $\boldsymbol{w}_0$, compute $\boldsymbol{z} = X\boldsymbol{w}_0$;
Use $\boldsymbol{z}$ to calculate $\boldsymbol{v}_{\boldsymbol{w}_0}$ and $D_{\boldsymbol{w}_0}$ in (8) and (10);
Compute $\nabla f(\boldsymbol{w}_0) = \boldsymbol{w}_0 + X^\top \boldsymbol{v}_{\boldsymbol{w}_0}$ from (8);
$P = [\nabla f(\boldsymbol{w}_0)/\|\nabla f(\boldsymbol{w}_0)\|], U = XP$;
for *k=0,1,... in the oute loop* **do**
  if $\nabla f(\boldsymbol{w}_k) = \mathbf{0}$, or some stopping condition is satisfied **then**
    stop;
  end
  Obtain $\boldsymbol{t}$ by solving (14) with $\boldsymbol{t} = [I + U^\top D_{\boldsymbol{w}_k} U]^{-1}[-P^\top \nabla f(\boldsymbol{w}_k)]$;
  Compute $\hat{\boldsymbol{z}} = U\boldsymbol{t}$;
  Backtracking line search on $f(\boldsymbol{w}_k + \theta P\boldsymbol{t})$ by Algorithm 2 to obtain θ using the values of $\boldsymbol{z}$ and $\hat{\boldsymbol{z}}$ to compute $\xi(y_i; \boldsymbol{z} + \theta\hat{\boldsymbol{z}})$;
  $\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + \theta P\boldsymbol{t}$;
  $\boldsymbol{z} = \boldsymbol{z} + \theta\hat{\boldsymbol{z}}$;
  Use $\boldsymbol{z}$ to calculate $\boldsymbol{v}_{\boldsymbol{w}_{k+1}}$ and $D_{\boldsymbol{w}_{k+1}}$;
  Compute $\nabla f(\boldsymbol{w}_{k+1}) = \boldsymbol{w}_{k+1} + X^\top \boldsymbol{v}_{\boldsymbol{w}_{k+1}}$ from (8);
  Let $\boldsymbol{p} = \nabla f(\boldsymbol{w}_{k+1}) - P(P^\top \nabla f(\boldsymbol{w}_{k+1}))$;
  if $\boldsymbol{p} \neq \mathbf{0}$ **then**
    $P = [P,\ \boldsymbol{p}/\|\boldsymbol{p}\|], U = [U,\ X\boldsymbol{p}/\|\boldsymbol{p}\|]$;
  end
end

3.3 Total Cost of the Algorithm

Based on the analysis in Sections 3.1 and 3.2, the total cost is

$$O(lm^2 + mn + m^3 + \#\text{nnz} + l \times \#(\text{line search steps})) \times \text{iterations}. \tag{20}$$

In Section 5, we will compare the cost with that of some existing approaches.

4. Common-directions Method with Multiple Inner Iterations

In Section 2, we considered solving the second-order Taylor approximation of $f(\boldsymbol{w}_k + P_k\boldsymbol{t})$ with respect to the coefficients $\boldsymbol{t}$ to get a good combination along the directions of P_k. Now we further extend the idea to directly minimize $f(\boldsymbol{w}_k + P_k\boldsymbol{t})$. We first describe the general framework, and then discuss how it can be applied to the special case of solving regularized ERM problems.

4.1 General Framework

We can consider the following convex optimization problem

$$\min_{\boldsymbol{t}} \quad f(\boldsymbol{w}_k + P_k\boldsymbol{t}), \tag{21}$$

and apply any optimization algorithm to solve it. When applying iterative algorithms, we terminate the optimization procedure after the solution is considered good enough.

Therefore, usually (21) is only solved approximately. In particular, the method discussed in Section 2 can be viewed as using only one Newton iteration to loosely solve (21). Because of two levels of iterations, we refer to the process of approximately solving (21) as an outer iteration, while each step in the process as an inner iteration.

Among the different methods to solve (21), we apply multiple inner Newton steps on the variable $\boldsymbol{t}$. That is, we repeatedly minimize the second-order Taylor expansion of (21).

$$\min_{\Delta \boldsymbol{t} \in \mathbb{R}^m} \quad \tilde{f}(\boldsymbol{w}_k + P_k \boldsymbol{t} + P_k \Delta \boldsymbol{t} \mid \boldsymbol{w}_k + P_k \boldsymbol{t}).$$

Because

$$\begin{aligned} &\tilde{f}(\boldsymbol{w}_k + P_k \boldsymbol{t} + P_k \Delta \boldsymbol{t} \mid \boldsymbol{w}_k + P_k \boldsymbol{t}) \\ &= f(\boldsymbol{w}_k + P_k \boldsymbol{t}) + (P_k^\top \nabla f(\boldsymbol{w}_k + P_k \boldsymbol{t}))^\top \Delta \boldsymbol{t} + \frac{1}{2} \Delta \boldsymbol{t}^\top P_k^\top \nabla f(\boldsymbol{w}_k + P_k \boldsymbol{t}) P_k \Delta \boldsymbol{t}, \end{aligned}$$

this amounts to solving the following linear system for $\Delta \boldsymbol{t}$.

$$(P_k^\top \nabla^2 f(\boldsymbol{w}_k + P_k \boldsymbol{t}) P_k) \Delta \boldsymbol{t} = -P_k^\top \nabla f(\boldsymbol{w}_k + P_k \boldsymbol{t}). \tag{22}$$

Each time the solution of the linear system will be our update direction, and then we will conduct backtracking line search to ensure the sufficient function decrease. The detailed algorithm is described in Algorithm 4, where we can see that $\boldsymbol{w}_k + P_k \boldsymbol{t}$ is the iterate updated in the inner iterations.

We now show that this general framework has better global convergence rate than the one discussed in Section 2.

Theorem 5 *Under Assumption 1 and a proper inner stopping condition, given any $\epsilon > 0$, Algorithm 4 converges R-linearly and obtains an ϵ-accurate solution in $O(\sqrt{\kappa} \log(1/\epsilon))$ iterations.*

The proof is in Appendix B.

It has been shown in Nesterov (2003) that $O(\sqrt{\kappa} \log(1/\epsilon))$ is the optimal iteration complexity for methods whose directions are obtained from the span of $\nabla f(\boldsymbol{w}_0), \nabla f(\boldsymbol{w}_1), \cdots$. The inner stopping condition used in proving Theorem 5 is not easy to implement in practice. It is possible to devise a practical one to ensure that this optimal linear convergence rate can be guaranteed. However, as we will see in the empirical results in Section 6, the setting of using a single inner iteration in approximately solving (21) tends to outperform that of using multiple inner iterations in terms of overall training time. In light of this understanding, we do not further explore practical inner stopping conditions.

Another interesting property of Algorithm 4 is that if the optimal solution of (21) is obtained at each outer iteration, then the algorithm can reach the optimum of (7) in finite steps even if neither Assumption 1 nor 2 is satisfied.

Theorem 6 *If at each iteration of Algorithm 4, the optimal solution of* (21) *is obtained, the optimum of* (1) *is reached within n iterations.*

The proof is in Appendix C.

These two theorems show that our algorithm can be treated as an analogy of the linear conjugate gradient method to general nonlinear optimization problems. See more details in Section 5.1

Algorithm 4: Common-directions method with multiple inner iterations

Given $\boldsymbol{w}_0$, compute $\nabla f(\boldsymbol{w}_0)$;
$P = [\nabla f(\boldsymbol{w}_0)/\|\nabla f(\boldsymbol{w}_0)\|]$;
for *k=0,1,... in the outer loop* **do**
  if $\nabla f(\boldsymbol{w}_k) = \mathbf{0}$, or some stopping condition is satisfied **then**
    stop;
  end
  $\boldsymbol{w} = \boldsymbol{w}_k$;
  while an inner stopping condition is not satisfied **do** `// inner loop`
    Obtain $\Delta \boldsymbol{t}$ by solving (22) with

$$(P^\top \nabla^2 f(\boldsymbol{w}) P)\Delta \boldsymbol{t} = -P^\top \nabla f(\boldsymbol{w});$$

    Backtracking line search on $f(\boldsymbol{w} + \theta P \Delta \boldsymbol{t})$ by Algorithm 2 to obtain θ;
    $\boldsymbol{w} = \boldsymbol{w} + \theta P \Delta \boldsymbol{t}$;
  end
  $\boldsymbol{w}_{k+1} = \boldsymbol{w}$;
  Compute $\nabla f(\boldsymbol{w}_{k+1})$;
  Let $\boldsymbol{p} = \nabla f(\boldsymbol{w}_{k+1}) - P(P^\top \nabla f(\boldsymbol{w}_{k+1}))$;
  if $\boldsymbol{p} \neq \mathbf{0}$ **then**
    $P = [P, \boldsymbol{p}/\|\boldsymbol{p}\|]$;
  end
end

4.2 Application on Regularized ERM Problems

For general problems, Algorithm 4 may not be beneficial because the cost of one inner iteration of solving (21) is similar to that of conducting an outer iteration of Algorithm 1, while (1) rather than (21) is what we actually would like to solve. However, for solving the regularized ERM problems (7), we show that one inner iteration of the while-loop in Algorithm 5 can be significantly cheaper than one outer iteration of Algorithm 3. We note that in the complexity analysis (16) for Algorithm 3, the dominant term $O(\#\text{nnz})$ is unavoidable for calculating $\nabla f(\boldsymbol{w}_{k+1})$. A crucial difference here is that this $O(\#\text{nnz})$ term is not needed. In Algorithm 3, the whole $\nabla f(\boldsymbol{w}_{k+1})$ vector is used for the Gram-Schmidt procedure in checking if P_k should be augmented. Instead, here P_k remains the same throughout inner iterations and all we need is $P_k^\top \nabla f(\boldsymbol{w}_k + P_k \boldsymbol{t})$ in (22). When we apply Algorithm 4 to regularized ERM problems, the linear system (22) has the following form.

$$(I + P_k^\top X^\top D_{\boldsymbol{w}_k + P_k \boldsymbol{t}} X P_k)\Delta \boldsymbol{t} = -(P_k^\top (\boldsymbol{w}_k + P_k \boldsymbol{t}) + P_k^\top X^\top \boldsymbol{v}_{\boldsymbol{w}_k + P_k \boldsymbol{t}}). \tag{23}$$

If

$$XP_k \ \text{ and } \ X(\boldsymbol{w}_k + P_k \boldsymbol{t}), \tag{24}$$

are maintained, the right-hand side can be easily calculated by

- $\boldsymbol{v}_{\boldsymbol{w}_k + P_k \boldsymbol{t}}$: $O(l)$,

- $P_k^\top(\boldsymbol{w}_k + P_k\boldsymbol{t})$: $O(mn)$,
- $(P_k^\top X^\top)\boldsymbol{v}_{\boldsymbol{w}_k+P_k\boldsymbol{t}}$: $O(lm)$.

Interestingly, for other operations in Algorithm 3, it has been shown that the two values in (24) can be easily maintained. Here we follow the same setting. A detailed procedure is in Algorithm 5. In summary, the cost per inner iteration is

$$O(lm^2 + mn + m^3). \tag{25}$$

In compared with the cost per outer iteration in Algorithm 3, here we do not have the #nnz term. The main difference comes from that in (23), we use the available XP_k to calculate

$$(P_k^\top X^\top)\boldsymbol{v}_{\boldsymbol{w}_k+P_k\boldsymbol{t}},$$

but in an outer iteration of Algorithm 3, $\nabla f(\boldsymbol{w}_{k+1})$ is calculated by

$$X^\top \boldsymbol{v}_{\boldsymbol{w}_{k+1}},$$

which costs $O(\#\text{nnz})$. Therefore, because in general #nnz is larger than (25), an inner iteration here is cheaper than an outer one in Algorithm 3.

5. Related Works

We compare our algorithm with linear conjugate gradient method for solving a linear system, as well as popular nonlinear optimization methods. The methods being discussed include those considering a linear combination of multiple directions to form the update direction, and those utilizing the Hessian matrix in various ways.

5.1 Linear Conjugate Gradient Method

Linear conjugate gradient (CG) method iteratively solves the following linear system.

$$A\boldsymbol{x} = \boldsymbol{b}, \tag{26}$$

where $A \in \mathbb{R}^{n\times n}$ is symmetric and positive-definite, and $\boldsymbol{b} \in \mathbb{R}^n$. Alternatively, CG can be viewed as an optimization method for the following problem.

$$\min_{\boldsymbol{x}} \quad \frac{1}{2}\boldsymbol{x}^\top A\boldsymbol{x} - \boldsymbol{b}^\top \boldsymbol{x}. \tag{27}$$

The idea behind linear CG is that at each iteration, we generate a new update direction $\boldsymbol{p}_k$ that is conjugate to the existing ones such that

$$\boldsymbol{p}_i^\top A\boldsymbol{p}_j = 0, \ \ \forall i \neq j.$$

Then an exact line search is conducted to decide the next iterate. At each iteration, a matrix-vector product $A\boldsymbol{v}$ for some vector $\boldsymbol{v}$ is conducted to generate the desired step, and this matrix-vector product is the computational dominant part at each CG iteration.

Algorithm 5: Common-directions method with multiple inner iterations for the ERM problem (7)

Given $\boldsymbol{w}_0$, compute $\boldsymbol{z} = X\boldsymbol{w}_0$;
Use $\boldsymbol{z}$ to compute $\boldsymbol{v}_{\boldsymbol{w}_0}$ and $D_{\boldsymbol{w}_0}$ in (8) and (10);
Compute $\nabla f(\boldsymbol{w}_0) = \boldsymbol{w}_0 + X^\top \boldsymbol{v}_{\boldsymbol{w}_0}$ from (8);
$P = [\nabla f(\boldsymbol{w}_0)/\|\nabla f(\boldsymbol{w}_0)\|]$, $U = XP$;
for *k=0,1,... in the outer loop* **do**
  if $\nabla f(\boldsymbol{w}_k) = \mathbf{0}$, or some stopping condition is satisfied **then**
    stop;
  end
  $\boldsymbol{w} = \boldsymbol{w}_k, \boldsymbol{v}_{\boldsymbol{w}} = \boldsymbol{v}_{\boldsymbol{w}_k}, D_{\boldsymbol{w}} = D_{\boldsymbol{w}_k}$;
  while an inner stopping condition is not satisfied **do** `// inner loop`
    Obtain $\Delta \boldsymbol{t}$ by solving (23) with

$$\Delta \boldsymbol{t} = (I + U^\top D_{\boldsymbol{w}} U)^{-1}(-P^\top \boldsymbol{w} + U^\top \boldsymbol{v}_{\boldsymbol{w}});$$

    Compute $\hat{\boldsymbol{z}} = U\Delta \boldsymbol{t}$;
    Backtracking line search on $f(\boldsymbol{w} + \theta P \Delta \boldsymbol{t})$ by Algorithm 2 to obtain θ, using the values of $\boldsymbol{z}$ and $\hat{\boldsymbol{z}}$ to compute $\xi(y_i; \boldsymbol{z} + \theta\hat{\boldsymbol{z}})$;
    $\boldsymbol{w} = \boldsymbol{w} + \theta P \Delta \boldsymbol{t}$;
    $\boldsymbol{z} = \boldsymbol{z} + \theta(XP)\Delta \boldsymbol{t}$;
    Use $\boldsymbol{z}$ to compute $\boldsymbol{v}_{\boldsymbol{w}}$ and $D_{\boldsymbol{w}}$ in (8) and (10);
  end
  $\boldsymbol{w}_{k+1} = \boldsymbol{w}, \boldsymbol{v}_{\boldsymbol{w}_{k+1}} = \boldsymbol{v}_{\boldsymbol{w}}, D_{\boldsymbol{w}_{k+1}} = D_{\boldsymbol{w}}$;
  Compute $\nabla f(\boldsymbol{w}_{k+1}) = \boldsymbol{w}_{k+1} + X^\top \boldsymbol{v}_{\boldsymbol{w}_{k+1}}$ from (8);
  $\boldsymbol{p} = \nabla f(\boldsymbol{w}_{k+1}) - P(P^\top \nabla f(\boldsymbol{w}_{k+1}))$;
  if $\boldsymbol{p} \neq \mathbf{0}$ **then**
    $P = [P_k, \boldsymbol{p}/\|\boldsymbol{p}\|]$, $U = [U, X\boldsymbol{p}/\|\boldsymbol{p}\|]$;
  end
end

Therefore, the cost per iteration of CG is simply the cost of a matrix-vector product and several $O(n)$ operations on the vectors.

There are two nice properties of linear CG. First, when viewed as a solver for (26), it obtains the exact solution in n steps. Second, when viewed as a solver for (27), its iteration complexity for an ϵ-accurate solution is $O(\sqrt{\kappa}\log(1/\epsilon))$. See, for example, Chapter 5.1 of Nocedal and Wright (2006). Because the method proposed in Section 4 possesses the same two properties in Theorems 5 and 6, it can be viewed as an extension of linear CG to general nonlinear convex optimization.

5.2 First-order Methods and Momentum

For nonlinear convex optimization, the simplest first-order method is gradient descent. In gradient descent, the update direction is always the negative gradient $-\nabla f(\boldsymbol{w})$. It has been

shown in many standard convex optimization books such as Boyd and Vandenberghe (2004); Nocedal and Wright (2006); Nesterov (2003) that with a proper choice of the step size, to obtain an ϵ-accurate solution for a given $\epsilon > 0$, gradient descent requires $O(\kappa \log(1/\epsilon))$ iterations. However, this is far from the aforementioned optimal convergence rate $O(\sqrt{\kappa}\log(1/\epsilon))$ for methods that have their update directions in the span of all previous gradients.

Methods like the heavy-ball method (Polyak, 1964) and nonlinear CG try to improve the convergence by using the "momentum." These methods possess update rules in the form

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \alpha_k \nabla f(\boldsymbol{w}_k) + \beta_k(\boldsymbol{w}_k - \boldsymbol{w}_{k-1}), \tag{28}$$

where α_k and β_k vary in different methods, and the term $\boldsymbol{w}_k - \boldsymbol{w}_{k-1}$ is the so-called momentum. By induction we can easily see that $\boldsymbol{w}_k - \boldsymbol{w}_{k-1}$ in (28) is a linear combination of $\nabla f(\boldsymbol{w}_0), \nabla f(\boldsymbol{w}_1), \ldots, \nabla f(\boldsymbol{w}_{k-1})$, and therefore these methods fall in the category of picking the update direction from the span of the previous gradients. Nonlinear CG has many different variants that consider different strategies based on $\boldsymbol{w}_k$, $\boldsymbol{w}_{k-1}$, $\nabla f(\boldsymbol{w}_k)$, $\nabla f(\boldsymbol{w}_{k-1})$ and even $\nabla^2 f(\boldsymbol{w}_k)$, and all of them reduce to linear CG when the problem is (27). Interested readers are referred to Hager and Zhang (2006) and the references therein. When f is quadratic and strongly convex, it can be shown that both the heavy-ball method with proper parameters (Lessard et al., 2016) and nonlinear CG methods can achieve the optimal convergence rate for first-order methods,[3] but for non-quadratic functions, the situation is different. Some variants of nonlinear CG can be shown to converge asymptotically, but the rate is unclear (Hager and Zhang, 2006, Theorem 5.1). On the other hand, Ghadimi et al. (2014) showed that for problems satisfying Assumption 1, with properly chosen parameters, the heavy-ball method converges globally R-linearly. If in addition the problem is twice-differentiable, Polyak (1987) showed that when the iterate is close enough to the optimum, the local R-linear convergence rate of the heavy-ball method matches the optimal linear convergence rate of first-order methods for a specific choice of α_k and β_k.

Nesterov's accelerated gradient (Nesterov, 1983) considers two sequences $\{\boldsymbol{w}_k\}$ and $\{\boldsymbol{s}_k\}$ with the following update rules.

$$\begin{aligned}
\boldsymbol{w}_{k+1} &\equiv \boldsymbol{s}_k - \frac{1}{\rho}\nabla f(\boldsymbol{s}_k), \\
\boldsymbol{s}_{k+1} &= (1 + \frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1})\boldsymbol{w}_{k+1} - \frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\boldsymbol{w}_k.
\end{aligned}$$

Interestingly, this method uses only the previous gradients in a deterministic way with κ to be the only data-dependent value, but it can guarantee the optimal iteration complexity of first-order methods for problems satisfying Assumption 1. A problem in this method is that we need to know the values ρ and σ in advance, and these values may not be available. To solve this problem, a variant of Nesterov's accelerated gradient is proposed in Nesterov (2013). This algorithm estimates these parameters at each iteration via an iterative procedure similar to backtracking line search, and achieves the same iteration complexity with little additional cost. Note that for all these first-order methods, the main bottleneck is the computation of the gradients, and the cost is proportional to the number of different gradients computed per iteration.

3. Section 1 of Hager and Zhang (2006) suggests that nonlinear CG methods reduce to linear CG when f is quadratic, and the convergence rate follows what we described in Section 5.1.

Our method can be viewed as a generalization of these approaches, in the sense that the same search space for the update direction is considered. The major difference is that these momentum methods are restricted to some deterministic or pre-specified ways of combination, while our method considers combining multiple directions with adaptive weights according to the data that can and will change over iterations. Another difference is that we utilize higher-order derivatives to decide the combination coefficient, so quadratic convergence in Theorem 4 can be obtained.

5.3 Methods Utilizing Multiple Directions

The cutting plane method (Kelley, 1960) and the bundle method (Teo et al., 2010) use all gradients obtained from the first iteration on. They differ from our method from using the previous gradients to construct a piecewise linear function to approximate (the hard-to-compute part of) the actual function.[4] Therefore, the purpose of using previous gradients in these methods and that of ours are different. An advantage of the cutting plane method and the bundle method is the ability to deal with nonsmooth problems by using subgradients. When being applied to problems satisfying Assumption 1, the bundle method converges linearly (Teo et al., 2010). However, optimizing a piecewise linear function with a growing number of pieces is more expensive than dealing with one smooth function. Our method considers easier-to-solve subproblems at each iteration. Moreover, our method converges quadratically while the bundle method and the cutting plane method do not have this property.

Quasi-Newton methods also consider previous momentums and gradients, but the purpose is to approximate the real Hessian matrix via these vectors in order to achieve superlinear convergence. Among different quasi-Newton methods, the BFGS method (Dennis and Moré, 1977) and its limited-memory variant, L-BFGS (Liu and Nocedal, 1989), are the most widely used ones. It is known that the BFGS method admits local superlinear convergence and converges globally; see, for example, (Nocedal and Wright, 2006, Chapter 6). By applying a proper line search procedure, L-BFGS has global linear convergence for problems satisfying Assumption 1.[5] Note that because older information are discarded, L-BFGS does not have superlinear convergence, but it has the advantage of less spatial cost and is more suitable for large-scale problems.

At the k-th iteration for any $k \geq 1$, given the current iterate $\boldsymbol{w}_k$, BFGS solves the following quadratic problem.

$$\min_{\boldsymbol{p}_k} \quad \nabla f(\boldsymbol{w}_k)^\top \boldsymbol{p}_k + \frac{1}{2}(\boldsymbol{p}_k)^\top B_k \boldsymbol{p}_k, \tag{29}$$

where B_k is a positive definite symmetric approximation of $\nabla^2 f(\boldsymbol{w}_k)$ such that

$$B_k^{-1} = (I - \mu_{k-1}\boldsymbol{u}_{k-1}\boldsymbol{s}_{k-1}^\top)^\top B_{k-1}^{-1}(I - \mu_{k-1}\boldsymbol{u}_{k-1}\boldsymbol{s}_{k-1}^\top) + \mu_{k-1}\boldsymbol{s}_{k-1}\boldsymbol{s}_{k-1}^\top, \tag{30}$$

4. It is known in functional analysis that any convex function can be approximated by piecewise linear functions to any precision.
5. Note that the proof that L-BFGS converges globally linearly utilized the property of using a bounded number of historical vectors and thus does not apply to BFGS.

with

$$\mu_{k-1} \equiv \frac{1}{\boldsymbol{u}_{k-1}^\top \boldsymbol{s}_{k-1}}, \quad \boldsymbol{s}_{k-1} = \boldsymbol{w}_k - \boldsymbol{w}_{k-1}, \quad \boldsymbol{u}_{k-1} = \nabla f(\boldsymbol{w}_k) - \nabla f(\boldsymbol{w}_{k-1}).$$

Note that although B_0 can be decided arbitrarily, the most common choice is

$$B_0 = \beta I \tag{31}$$

for some $\beta > 0$. The solution of (29) is clearly

$$\boldsymbol{p}_k = -B_k^{-1} \nabla f(\boldsymbol{w}_k). \tag{32}$$

If (31) is used, this operation can be conducted cheaply with only $O(kn)$ cost by a sequence of vector-vector operations obtained from expanding (30).[6] After deciding $\boldsymbol{p}_k$, a line search procedure is conducted to find a suitable step size θ_k to ensure that the sequence $\{\boldsymbol{w}_k\}$ converges to the optimal solution of f.

With the choice of (31), by induction the vector

$$\boldsymbol{s}_k = \alpha_{k+1} \boldsymbol{p}_k \tag{33}$$

is a linear combination of $\nabla f(\boldsymbol{w}_0), \ldots, \nabla f(\boldsymbol{w}_k)$ for all k.[7] Thus, we can see that $\boldsymbol{p}_k \in \texttt{span}\{\nabla f(\boldsymbol{w}_0), \ldots, f(\boldsymbol{w}_k)\}$ and thus the BFGS algorithm only utilizes the previous gradients, but is able to achieve superlinear convergence. Interestingly, our algorithm utilizes the same information as BFGS, but because we also consider the Hessian to search for the best update direction in $\texttt{span}\{\nabla f(\boldsymbol{w}_0), \ldots, \nabla f(\boldsymbol{w}_k)\}$, our method is better than BFGS for being able to achieve quadratic convergence.

A major drawback of the BFGS algorithm is that it requires storing many vectors (or an $O(n^2)$ dense matrix) in memory. The memory consumption can be prohibitively expensive after iterations. This issue also happens in our method, but we notice that in practice, it is not severe. One reason is that the memory capacity is large enough to store the required information unless the number of iterations is high, and both our method and BFGS converge fast enough such that the memory capacity is not reached. For extremely large data, the memory issue may be faced, but this issue is less problematic in our method than in BFGS. The reason is that at each iteration of BFGS, two new vectors are added to memory, while in our method, at most one vector is added.

To handle the memory consumption problem of BFGS, its limited-memory variant, L-BFGS (Liu and Nocedal, 1989), is proposed. L-BFGS uses a different setting of B_k^{-1}. Given an integer $m > 0$, at the k-th iteration, L-BFGS only uses $\boldsymbol{s}_i, \boldsymbol{u}_i, i = k-m, k-m+1, \ldots, k-1$ and $\nabla f(\boldsymbol{w}_k)$ to construct the approximate Hessian inverse. For L-BFGS, in the recursive computation of (30), the matrix B_{k-m}^{-1} can be replaced by any matrix. According

6. If B_0 has neither sparsity nor some special structures, the cost of computing (32) at each iteration is $O(n^2)$ by directly maintaining (30), and it does not fall in the category of algorithms that only use gradients anymore because the vector $B_0^{-1}\nabla f(\boldsymbol{w})$ is considered.

7. By continually expanding (32) using (31) and (30), we can see that $\boldsymbol{p}_k$ is a linear combination of $\boldsymbol{s}_i, \boldsymbol{u}_i, i = 1, \ldots, k-1$ and $\nabla f(\boldsymbol{w}_k)$. By induction, if all $\boldsymbol{s}_i$ are linear combinations of previous gradients, then with the definition of $\boldsymbol{u}_i$, $\boldsymbol{p}_k$ is also a linear combination of previous gradients. The induction is finished by that $\boldsymbol{s}_k$ is a multiple of $\boldsymbol{p}_k$ in (33).

to (Nocedal and Wright, 2006, Chapter 7), a choice that has been shown to be practically effective is

$$\frac{\boldsymbol{s}_{k-1}^T \boldsymbol{u}_{k-1}}{\|\boldsymbol{u}_{k-1}\|^2} I. \tag{34}$$

This is also the choice we use in our implementation for empirical comparisons. As mentioned earlier, L-BFGS has only linear but not superlinear convergence. Nevertheless, it might outperform BFGS practically because the computation cost per iteration and the memory access amount are both lower. Moreover, it is shown in Bubeck et al. (2015) that L-BFGS outperforms those methods mentioned in Section 5.2 empirically, although the coefficient of its linear convergence is not clear.

5.4 Newton Method

At each iteration, given the current iterate $\boldsymbol{w}$, the Newton method obtains the update direction by solving

$$\min_{\boldsymbol{p}\in\mathbb{R}^n} \quad \boldsymbol{p}^\top \nabla f(\boldsymbol{w}) + \frac{1}{2}\boldsymbol{p}^\top \nabla^2 f(\boldsymbol{w})\boldsymbol{p}. \tag{35}$$

To obtain the solution of (35), the following n by n linear system is solved.

$$\nabla^2 f(\boldsymbol{w})\boldsymbol{p} = -\nabla f(\boldsymbol{w}) \quad \Rightarrow \quad \boldsymbol{p} = -\nabla^2 f(\boldsymbol{w})^{-1}\nabla f(\boldsymbol{w}). \tag{36}$$

Then a line search procedure is conducted to ensure the sufficient function decrease. It is known that for twice-differentiable functions satisfying Assumption 2, the Newton method with line search has local quadratic convergence (Boyd and Vandenberghe, 2004; Nocedal and Wright, 2006). However, The computational cost per Newton iteration for directly solving the linear system (36) is as expensive as $O(n^3)$. Moreover, solving (36) precisely may not be beneficial when the current iterate is far from optimum. Therefore, truncated Newton methods that utilize, for example, the iterative linear CG method we discussed in Section 5.1 to approximately solve (36) is often used. In general, the computational cost per CG iteration is $O(n^2)$ for conducting the Hessian-vector product provided the Hessian is available. When we only want to solve the problem approximately, usually the required CG iteration is much smaller than n, making the computation of a truncated Newton step much cheaper than that of a full Newton step. It is known that truncated Newton steps combined with a line search procedure or trust region techniques retain local quadratic convergence for problems satisfying Assumption 2 and have superlinear convergence for problems satisfying Assumption 1, if the residual of the solution to (36) goes to zero as the number of truncated Newton iterations goes to infinity. See, for example, Lin and Moré (1999) and (Nocedal and Wright, 2006, Theorem 7.2).

As mentioned above, the cost of directly computing the Hessian-vector product is $O(n^2)$ both spatially and computationally if we store the Hessian matrix explicitly. However, as Keerthi and DeCoste (2005); Lin et al. (2008) proposed, for ERM problems in (7), if we calculate the product between Hessian and a given vector $\boldsymbol{v}$ by

$$\nabla^2 f(\boldsymbol{w})\boldsymbol{v} = (I + X^\top D_{\boldsymbol{w}} X)\boldsymbol{v} = \boldsymbol{v} + \left(X^\top \left(D_{\boldsymbol{w}} \left(X\boldsymbol{v}\right)\right)\right), \tag{37}$$

this computation costs only $O(\#\text{nnz} + n)$ per CG iteration, where #nnz is at most ln but is usually much smaller for sparse data. Further, no storage space is needed for the

operation in (37). On the other hand, explicitly forming the Hessian matrix in this case costs $O(\#\text{nnz}^2)$. Therefore, for regularized ERM problems, the CG approach without explicitly computing the Hessian matrix is more preferable. In this sense, truncated Newton methods costs

$$O((\#\text{nnz} + n) \times \text{CG iterations}) \tag{38}$$

per Newton iteration. A comparison between (38) and (20) shows that by a careful utilization of the problem structure of ERM, we have a cheaper cost of running one Newton iteration to approximately minimize a sub-problem over a subspace. The cost of our Newton iteration is only about that of one CG iteration here. We will examine the relative training speed between the truncated Newton method and our algorithm by experiments in Section 6.

6. Experiments

We examine the empirical speed of our algorithm for solving two L2-regularized ERM problems. The first one is logistic regression, in which $\xi(\cdot)$ in (7) is defined by

$$\xi(y; z) \equiv \log\left(1 + \exp\left(-yz\right)\right),$$

and the second one is the squared-hinge loss problem, where $\xi(\cdot)$ is

$$\xi(y; z) \equiv \max\left\{1 - yz, 0\right\}^2.$$

In both cases, $y \in \{-1, 1\}$. We first compare the approach of conducting only one inner iteration discussed in Section 2 with that of conducting multiple inner iterations illustrated in Section 4 using the logistic regression problems. The relative efficiency between our methods and the methods reviewed in Section 5 is then examined on both loss functions.

6.1 Experiment Settings

We consider the data sets listed in Table 1 with three different choices of parameters $C = \{10^{-3}, 1, 10^3\}$ to examine the situation of different condition numbers. All data sets except yahoo-japan and yahoo-korea are publicly available.[8] We use the relative distance to the optimal objective value, which is defined below, to compare different optimization approaches.

$$\left|\frac{f(\boldsymbol{w}) - f(\boldsymbol{w}^*)}{f(\boldsymbol{w}^*)}\right|,$$

where $\boldsymbol{w}^*$ is the optimal solution obtained by running our algorithm long enough. All experiments are conducted on a 64-bit machine with Intel Xeon 2.0 GHz CPU (E5504), 4MB cache, and 32GB memory.

We consider two different criteria for comparison. The first one is the empirical training time. This is the most important criterion in single-machine training for most users. We also consider the number of data passes, which is proportional to the number of iterations[9] and

8. `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets`.
9. For the Newton method, this is the number of inner iterations in CG plus the number of Newton iterations.

Data	Training size (l)	Features (n)	Density (#nnz/(ln))
a9a	32,561	123	11.2757%
covtype	581,012	54	21.9979%
news20	19,996	1,355,191	0.0336%
url	2,396,130	3,231,961	0.0036%
epsilon	400,000	2000	100.0000%
webspam	350,000	16,609,143	0.0224%
yahoo-japan	140,963	832,026	0.0160%
yahoo-korea	368,444	3,053,939	0.0111%
rcv1t	677,399	47,236	0.1549%
KDD2010-b	19,264,097	29,890,095	0.0001%

Table 1: Statistics of data sets.

is thus an indicator of iteration complexity. This criterion can also serve as a training time estimate for scenarios that the cost is highly dependent to the number of passes through the data, such as distributed machine learning, or disk-level classification (Yu et al., 2012). In addition, this criterion excludes the effect of implementation differences.

6.2 Using Single or Multiple Inner Iterations

We compare on logistic regression problems the performance of the two variants of our algorithms, namely the approach of using only a single inner iteration and the variant of using multiple inner iterations, in Figures 1-6. Note that the result of KDD2010-b with $C = 10^3$ is not shown because the training time is too long. For the backtracking line search procedure in Algorithm 2, we set $\beta = 0.4, \lambda = 0.25$. The solution to the linear system (14) is obtained by a Cholesky factorization with partial pivoting. For the multiple inner iteration variant, we use the following heuristic stopping condition for the inner loop in Algorithm 4.

$$\|P_k \nabla f(\boldsymbol{w})\| \leq \min(0.1\|\nabla f(\boldsymbol{w}_k)\|, \|\nabla f(\boldsymbol{w}_k)\|^2).$$

We can see that the two approaches perform quite similarly in both criteria, while the single inner iteration approach is slightly better in the training time. The figures of data passes indicate that using a single inner iteration has similar function value decrease per iteration to using multiple inner iterations, so the additional cost of multiple inner iterations sometimes results in slightly longer training time. Therefore, we use the single iteration variant in later experiments of comparing with existing algorithms.

6.3 Comparison Between Different Methods

We compare the following different algorithms for training L2-regularized smooth empirical risk minimization problems.

- CommDir: our common-directions method with single inner iteration.

- Line search truncated Newton method (NEWTON): at each Newton iteration, (35) is approximately solved by linear CG, until CG generates an approximate solution $\boldsymbol{p}$ of (35) that satisfies
$$\|\nabla f(\boldsymbol{w}) - \nabla^2 f(\boldsymbol{w})\boldsymbol{p}\| \ \leq 0.1\|\nabla f(\boldsymbol{w})\|.$$
We follow the discussion in Section 5 to efficiently compute the Hessian-vector products in $O(n + \#\text{nnz})$ time. The implementation is modified from that in LIBLINEAR (Fan et al., 2008).

- L-BFGS: We implement the L-BFGS algorithm by considering different numbers of history states $m = 5, 10, 15, 20, 30$.[10] For the first step, where no historical information is available, we follow the implementation by the authors of Liu and Nocedal (1989) to take the initial step size for the gradient direction as $1/\|\nabla f(\boldsymbol{w}_0)\|$. For later iterations, we take the matrix defined in (34) as B_{k-m}^{-1}. In general $m = 30$ performs the best, so we report results of using this number of states.

- BFGS: we use our L-BFGS implementation and set m to infinity to obtain a BFGS solver.

- Nesterov's accelerated gradient (AG): we consider the approach proposed by Nesterov (2013) that adaptively estimates the parameters σ and ρ by a procedure similar to backtracking line search. We take the initial estimation to be $\sigma = 1$ and $\rho = 1/Cl$.

For NEWTON, L-BFGS and BFGS that require a line search procedure to ensure the convergence, we follow our method to apply Algorithm 2 with the same parameters $\beta = 0.4$ and $\lambda = 0.25$ used in Section 6.2. We exploit the trick discussed in Section 3.2 to efficiently conduct this line search procedure. Note that for all methods except AG, evaluating $X^T\boldsymbol{v}_{\boldsymbol{w}}$ in (8) for gradient at each iteration requires one data pass, and the line search procedure requires another one to maintain $X\boldsymbol{w}$. For NEWTON, each CG iteration requires one data pass. For AG, each inner iteration for adaptively estimating σ and ρ requires two data passes to compute related values. All algorithms are implemented in C++.

The results on logistic regression problems are shown in Figures 7-9 for running time and Figures 10-12 for number of data passes. Note that the result of KDD2010-b with $C = 10^3$ is not shown because the training time is too long. We can see that our method has the fewest data passes in all settings and all data sets. This observation suggests the usefulness of our method in the scenario that the data passes are expensive. The number of data passes is significantly reduced because we use the ideas in Section 3 to store the inner products XP_k to avoid redundant data accesses. In terms of training time, our method is among the fastest. The exceptions are a9a and rcv1t whose l are much larger than n, and the largest data set KDD2010-b, which from Table 1 is highly sparse. From the result of data passes, we see that our method requires fewer iterations than other methods, in the cost of the $O(lm^2 + mn)$ operations per iteration in addition to the $O(\#\text{nnz})$ cost required by all methods to conduct one data pass; see (20). When $O(lm^2 + mn) \ll O(\#\text{nnz})$, this additional cost is negligible and hence the advantage of fewer iterations dominates.

10. We also tried the implementation by the authors of Liu and Nocedal (1989) at `http://users.iems.northwestern.edu/~nocedal/lbfgs.html`, but found that it is not as efficient as ours. Note that their implementation does not utilize the efficient line search procedure discussed in Section 3.2.

However, when n is relatively small, or when the data set is highly sparse, the $O(lm^2 + mn)$ cost is significant in compared with $O(\#\text{nnz})$, so our method may be slower for these data sets.

We then show results on squared-hinge loss problems in Figures 13-15 for running time and in Figures 16-18 for data passes. Similar trends of superiority of our method in data passes are still observed in squared-hinge loss problems, but in general the squared-hinge loss problems are harder to optimize for all methods. A possible reason is that logistic loss is infinitely continuously differentiable but squared-hinge loss is only once continuously differentiable. This difficulty in optimization of squared-hinge loss problems is reflected in running time as well. Since our common-directions method has cost that grows superlinearly with the number of iterations passed, its relative running time performance is also worse on squared-hinge loss problems than that on logistic regression problems.

The comparison here between NEWTON and L-BFGS is inconsistent with that in Lin et al. (2008). A further investigation showed that the major difference is from the choice of m. In Lin et al. (2008), $m = 5$ is used, while we observe that setting larger m like 30 here in most cases leads to faster convergence in terms of time, as long as the memory capacity is not an issue. Another minor difference is that each line search iteration of L-BFGS in the experiments of Lin et al. (2008) requires one data pass while we use the idea of caching inner products to accelerate the line search in our experiments. Moreover, in Lin et al. (2008), instead of line search, a trust region approach is considered for the truncated Newton method.

An interesting observation in our experimental results is the impressive performance of BFGS. Although BFGS has better theoretical convergence than L-BFGS, in practice people tend to use L-BFGS because BFGS consumes too much memory. Our experiments indicate that with larger memory and better computation power nowadays, BFGS becomes feasible and can outperform L-BFGS in some cases. However, it is also observed that when both the number of iterations to converge and n are large, such as the cases of webspam and url with $C = 1000$, BFGS still suffers from the memory problem.

7. Discussion

As shown in our experiments, our method has a smaller number of data passes than all other methods. This property implies that our method has at least two potential applications for solving extremely large-scale ERM problems. The first one is disk-level classification when the data cannot fit into memory, and the second one is distributed machine learning. In both tasks, each data pass involves expensive operations like intensive disk I/O or between-machine communication. With fewer data passes, our method is suitable for these tasks to tackle the bottleneck.

Each data pass in disk-level classification requires expensive disk I/O to access data because we cannot store all of them in memory. Existing methods for disk-level classification (Yu et al., 2012; Chang and Roth, 2011) consider solving the dual problem with techniques involving shuffling and partitioning data into blocks. Each block can be fully loaded into memory and these methods work on one block at a time. The random shuffling step gives better convergence, but requires lengthy pre-processing time because of heavy disk I/O. In contrast, our method does not require the random shuffling of data.

To reduce the number of data passes, Chang and Roth (2011) considered caching important instances in memory. Our method also uses ideas similar to caching, while the information being kept in memory is not some training instances, but the inner products XP_k. The number of these inner products is roughly the same as the number of iterations, which is usually not too large. Therefore, the memory consumption of our method tends to be smaller, and hence our method can cache more important information to reduce the data I/O.

For distributed machine learning that uses multiple machines, each data pass requires an expensive communication of a vector proportional to the data size to synchronize information among machines. Usually this step is the bottleneck of distributed machine learning. With fewer data passes and thus fewer rounds of communication, our method can be expected to have shorter training time than other algorithms in distributed environments.

8. Conclusions

In this paper, we propose a new optimization algorithm that utilizes the Hessian information to combine different update directions. Theoretically, our method has global linear and local quadratic convergence, while the cost per iteration is not expensive for solving ERM problems. Empirically, extensive experimental results show that for solving ERM problems, our method is competitive with state-of-the-art methods in terms of training time, but outperforms them in terms of data passes. After the first version of this paper, we have studied possible choices of common directions other than the gradients for distributed training in Lee et al. (2017).

Acknowledgments

The implementation of Nesterov's accelerated gradient is modified from an unreleased C# experimental code provided by Lin Xiao. C.-J. Lin was supported in part by the Ministry of Science and Technology of Taiwan via the grants 107-2221-E-002-167-MY3.

References

Bernhard E. Boser, Isabelle Guyon, and Vladimir Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 144–152. ACM Press, 1992.

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

Sébastien Bubeck. Convex optimization: Algorithms and complexity. *Foundations and Trends in Machine Learning*, 8(3-4):231–357, 2015.

Sébastien Bubeck, Yin Tat Lee, and Mohit Singh. A geometric alternative to Nesterov's accelerated gradient descent. Technical report, 2015. arXiv preprint arXiv:1506.08187.

Kai-Wei Chang and Dan Roth. Selective block minimization for faster convergence of limited memory large-scale linear models. In *Proceedings of the Seventeenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2011.

John E. Dennis, Jr and Jorge J. Moré. Quasi-Newton methods, motivation and theory. *SIAM Review*, 19(1):46–89, 1977.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: a library for large linear classification. *Journal of Machine Learning Research*, 9: 1871–1874, 2008. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/liblinear.pdf`.

Reeves Fletcher and Colin M Reeves. Function minimization by conjugate gradients. *The Computer Journal*, 7(2):149–154, 1964.

Euhanna Ghadimi, Hamid Reza Feyzmahdavian, and Mikael Johansson. Global convergence of the heavy-ball method for convex optimization. Technical report, 2014. arXiv preprint arXiv:1412.7457.

William W Hager and Hongchao Zhang. A survey of nonlinear conjugate gradient methods. *Pacific Journal of Optimization*, 2(1):35–58, 2006.

S. Sathiya Keerthi and Dennis DeCoste. A modified finite Newton method for fast solution of large scale linear SVMs. *Journal of Machine Learning Research*, 6:341–361, 2005.

James E. Kelley, Jr. The cutting-plane method for solving convex programs. *Journal of the Society for Industrial and Applied Mathematics*, 1960.

Ching-Pei Lee, Po-Wei Wang, Weizhu Chen, and Chih-Jen Lin. Limited-memory common-directions method for distributed optimization and its application on empirical risk minimization. In *Proceedings of SIAM International Conference on Data Mining (SDM)*, 2017. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/l-commdir/l-commdir.pdf`.

Laurent Lessard, Benjamin Recht, and Andrew Packard. Analysis and design of optimization algorithms via integral quadratic constraints. *SIAM Journal on Optimization*, 26(1): 57–95, 2016.

Chih-Jen Lin and Jorge J. Moré. Newton's method for large-scale bound constrained problems. *SIAM Journal on Optimization*, 9:1100–1127, 1999.

Chih-Jen Lin, Ruby C. Weng, and S. Sathiya Keerthi. Trust region Newton method for large-scale logistic regression. *Journal of Machine Learning Research*, 9:627–650, 2008. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/logistic.pdf`.

Dong C. Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(1):503–528, 1989.

Olvi L. Mangasarian. A finite Newton method for classification. *Optimization Methods and Software*, 17(5):913–929, 2002.

Yurii E. Nesterov. A method of solving a convex programming problem with convergence rate $o(1/k^2)$. *Soviet Mathematics Doklady*, 27:372–376, 1983.

Yurii E. Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. Kluwer Academic Publishers, 2003.

Yurii E. Nesterov. Gradient methods for minimizing composite functions. *Mathematical Programming*, 140(1):125–161, 2013.

Jorge Nocedal and Stephen Wright. *Numerical Optimization*. Springer, second edition, 2006.

Boris T. Polyak. *Introduction to Optimization*. Translation Series in Mathematics and Engineering. 1987.

Boris Teodorovich Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

Choon Hui Teo, S.V.N. Vishwanathan, Alex Smola, and Quoc V. Le. Bundle methods for regularized risk minimization. *Journal of Machine Learning Research*, 11:311–365, 2010.

Vladimir Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, NY, 1995.

Hsiang-Fu Yu, Cho-Jui Hsieh, Kai-Wei Chang, and Chih-Jen Lin. Large linear classification when data cannot fit in memory. *ACM Transactions on Knowledge Discovery from Data*, 5(4):23:1–23:23, February 2012. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/kdd_disk_decomposition.pdf`.

Appendix A. Analysis of Common-directions Method with Single Inner Iteration

We will show that under Assumptions 1, Algorithm 1 converges linearly by Theorem 3. If in addition Assumption 2 is satisfied, Algorithm 1 converges quadratically by Theorem 4.

Lemma 7 *Let P be an n by m real matrix such that $P^\top P = I$. Then*

$$\|P^\top \boldsymbol{v}\| \leq \|\boldsymbol{v}\|, \ \forall \boldsymbol{v} \in \mathbb{R}^n.$$

Further, equality holds if and only if

$$\boldsymbol{v} \in \mathtt{span}(P) \equiv \{P\boldsymbol{t} \mid \forall \boldsymbol{t} \in \mathbb{R}^m\}.$$

Proof Note that $\boldsymbol{v}$ can be decomposed as

$$\boldsymbol{v} = PP^\top \boldsymbol{v} + (I - PP^\top)\boldsymbol{v}. \tag{39}$$

Taking norm at both sides and applying $P^\top P = I$, there is

$$\|\boldsymbol{v}\|^2 = \|P^\top \boldsymbol{v}\|^2 + \|(I - PP^\top)\boldsymbol{v}\|^2 \geq \|P^\top \boldsymbol{v}\|^2. \tag{40}$$

Thus, the first result follows. For the second result, we know that $\boldsymbol{v}$ is in the span of P if and only if its orthogonal projection to the span of P is itself. Therefore,

$$\boldsymbol{v} \in \mathtt{span}(P) \iff \boldsymbol{v} = PP^\top \boldsymbol{v}.$$

Combining this with (39), we obtain

$$\boldsymbol{v} \in \mathtt{span}(P) \implies (I - PP^\top)\boldsymbol{v} = \boldsymbol{0} \implies \|P^\top \boldsymbol{v}\| = \|\boldsymbol{v}\|.$$

Similarly, with (40), we have

$$\|P^\top \boldsymbol{v}\| = \|\boldsymbol{v}\| \implies \|(I - PP^\top)\boldsymbol{v}\| = 0 \implies \boldsymbol{v} \in \mathtt{span}(P).$$

Thus, the second result follows from the above two statements. ■

Lemma 8 *For every iteration k of Algorithm 1, we have*

$$P_k^\top P_k = I, \tag{41}$$

and

$$\|P_k^\top \nabla f(\boldsymbol{w}_k)\| = \|\nabla f(\boldsymbol{w}_k)\|. \tag{42}$$

Proof The augmenting of P in Algorithm 1 is simply the Gram-Schmidt process, and thus (41) holds. Further, from the fact that $(I - P_{k-1}P_{k-1}^\top)\nabla f(\boldsymbol{w}_k)$ is used to generate P_k, we have that $\nabla f(\boldsymbol{w}_k) \in \mathtt{span}(P_k)$, so (42) follows from Lemma 7. ■

Lemma 9 *Under Assumption 1,*

$$\sigma I \preceq P_k^\top \nabla^2 f(\boldsymbol{w}_k) P_k \preceq \rho I. \tag{43}$$

Proof Assumption 1 indicates that

$$\sigma I \preceq \nabla^2 f(\boldsymbol{w}) \preceq \rho I, \ \forall \boldsymbol{w}. \tag{44}$$

Combining (44) and (41) in Lemma 8, one has that for any $\boldsymbol{t}$,

$$\begin{aligned} \boldsymbol{t}^\top P_k^\top \nabla^2 f(\boldsymbol{w}_k) P_k \boldsymbol{t} &\geq \sigma \|P_k \boldsymbol{t}\|^2 = \sigma \|\boldsymbol{t}\|^2, \\ \boldsymbol{t}^\top P_k^\top \nabla^2 f(\boldsymbol{w}_k) P_k \boldsymbol{t} &\leq \rho \|P_k \boldsymbol{t}\|^2 = \rho \|\boldsymbol{t}\|^2, \end{aligned} \tag{45}$$

which then imply (43). ■

Lemma 10 *Under Assumption 1,*

$$\frac{1}{2\rho}\|\nabla f(\boldsymbol{w})\|^2 \leq f(\boldsymbol{w}) - f^* \leq \frac{1}{2\sigma}\|\nabla f(\boldsymbol{w})\|^2, \tag{46}$$

where f^* is the optimal objective value.

Proof The first inequality is from Nesterov (2003, Theorem 2.1.5) together with that

$$\nabla f(\boldsymbol{w}^*) = \mathbf{0},$$

where $\boldsymbol{w}^*$ is the optimal solution. The second inequality follows from Nesterov (2003, Theorem 2.1.10). ■

Proof of Theorem 3

We will show that the update direction does not deviate from the steepest descent direction too much, and the line search procedure will then ensure the sufficient decrease of the function value for linear convergence. Let

$$\boldsymbol{t} = -(P_k^\top \nabla^2 f(\boldsymbol{w}_k) P_k)^{-1} P_k^\top \nabla f(\boldsymbol{w}_k). \tag{47}$$

Note that $(P_k^\top \nabla^2 f(\boldsymbol{w}_k) P_k)$ is invertible because of Lemma 9.

We first show that the step size produced in the line search procedure in Algorithm 2 is lower-bounded. From (44) via Assumption 1, for any $\bar{\theta} > 0$, we have

$$f(\boldsymbol{w}_k + \bar{\theta} P_k \boldsymbol{t}) \leq f(\boldsymbol{w}_k) + \bar{\theta} \nabla f(\boldsymbol{w}_k)^\top P_k \boldsymbol{t} + \frac{\rho}{2} \bar{\theta}^2 \|P_k \boldsymbol{t}\|^2. \tag{48}$$

If

$$-\bar{\theta} \nabla f(\boldsymbol{w}_k)^\top P_k \boldsymbol{t} - \frac{\rho}{2} \bar{\theta}^2 \|P_k \boldsymbol{t}\|^2 \geq \frac{\lambda}{2} \bar{\theta}^2 \|P_k \boldsymbol{t}\|^2, \tag{49}$$

then with (48) the stopping condition of Algorithm 2 is satisfied. For $\bar{\theta} > 0$, (49) is equivalent to

$$\bar{\theta} \leq \frac{-\nabla f(\boldsymbol{w}_k)^\top P_k \boldsymbol{t}}{\|P_k \boldsymbol{t}\|^2} \frac{2}{\rho + \lambda}. \tag{50}$$

From (41) of Lemma 8,

$$\|P_k \boldsymbol{t}\| = \|\boldsymbol{t}\|. \tag{51}$$

Thus, by the definition of $\boldsymbol{t}$ in (47) and Lemma 9,

$$\frac{-\nabla f(\boldsymbol{w}_k)^\top P_k \boldsymbol{t}}{\|P_k \boldsymbol{t}\|^2} = \frac{\boldsymbol{t}^\top (P_k^\top \nabla^2 f(\boldsymbol{w}_k) P_k) \boldsymbol{t}}{\|\boldsymbol{t}\|^2} \geq \sigma, \tag{52}$$

whenever $\nabla f(\boldsymbol{w}_k) \neq \mathbf{0}$. Then we have that (49), or equivalently (50), holds whenever

$$\bar{\theta} \leq \frac{2\sigma}{\rho + \lambda}.$$

This ensures the finite termination of Algorithm 2 and guarantees that the generated step size θ_k satisfies

$$\theta_k \geq \min(1, \frac{2\beta\sigma}{\rho + \lambda}) \geq \frac{\beta\sigma}{\rho + \lambda}, \tag{53}$$

where the second inequality is from $\beta < 1$, $\lambda > 0$, and $\sigma \leq \rho$. Therefore, the line-search procedure terminates after at most $\lceil \log_\beta(\beta\sigma/(\rho + \lambda)) \rceil$ steps.

Next we prove the linear convergence. By (51), the definition of $\boldsymbol{t}$ in (47), Lemma 9, and (42) of Lemmas 8, we have

$$\begin{aligned}
&\|P_k\boldsymbol{t}\|^2 = \|\boldsymbol{t}\|^2 \\
&= \nabla f(\boldsymbol{w}_k)^\top P_k (P_k^\top \nabla^2 f(\boldsymbol{w}_k) P_k)^{-2} P_k^\top \nabla f(\boldsymbol{w}_k) \\
&\geq \frac{1}{\rho^2}\|P_k^\top \nabla f(\boldsymbol{w}_k)\| = \frac{1}{\rho^2}\|\nabla f(\boldsymbol{w}_k)\|^2.
\end{aligned}$$

With this inequality and Lemma 10, the line search stopping condition implies

$$\begin{aligned}
f(\boldsymbol{w}_{k+1}) - f^* &\leq f(\boldsymbol{w}_k) - f^* - \frac{\lambda\theta_k^2}{2}\|P_k\boldsymbol{t}\|^2 \\
&\leq f(\boldsymbol{w}_k) - f^* - \frac{\lambda\theta_k^2}{2\rho^2}\|\nabla f(\boldsymbol{w}_k)\|^2 \leq (1 - \frac{\lambda\theta_k^2\sigma}{\rho^2})(f(\boldsymbol{w}_k) - f^*). \qquad (54)
\end{aligned}$$

Plugging in the lower bound (53) of the step-size θ_k to (54), we have the linear convergence

$$f(\boldsymbol{w}_{k+1}) - f^* \leq (1 - \frac{\lambda\beta^2\sigma^3}{(\rho+\lambda)^2\rho^2})(f(\boldsymbol{w}_k) - f^*).$$

This non-asymptotic linear convergence holds for any λ. Specifically, if we take $\lambda = \rho$, then we have the iteration complexity

$$O(\frac{\rho^3}{\sigma^3}\log(\frac{1}{\epsilon})),$$

Further, if Algorithm 2 always stops at step size $\theta_k = 1$, the rate (54) becomes

$$f(\boldsymbol{w}_{k+1}) - f^* \leq (1 - \frac{\lambda\sigma}{\rho^2})(f(\boldsymbol{w}_k) - f^*).$$

In this case, picking $\lambda = \rho$ recovers the iteration complexity of gradient descent methods (Nesterov, 2003)

$$O\left(\frac{\rho}{\sigma}\log\left(\frac{1}{\epsilon}\right)\right).$$

Thus, all the results are established. ■

Proof of Theorem 4

If $\nabla f(\boldsymbol{w}_0) = \mathbf{0}$, then the initial point is the optimal solution, and there is nothing to prove. Therefore, because $\nabla f(\boldsymbol{w}_0)/\|\nabla f(\boldsymbol{w}_0)\| \neq \mathbf{0}$ is included in P_0, we start our discussion with $\text{rank}(P_k) \geq 1$. The procedure of adding columns to P_k in Algorithm 1 ensures that $\text{rank}(P_k)$ is a monotonically increasing and bounded sequence with

$$1 \leq \text{rank}(P_k) \leq n, \ \forall k \in \mathbb{N}.$$

Thus, there exists a k_0 such that

$$\text{rank}(P_k) = \text{rank}(P_{k_0}), \ \forall k \geq k_0.$$

Together with $\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + \theta_k P_k \boldsymbol{t}_k$ in Algorithm 1, this implies

$$\boldsymbol{w}_k - \boldsymbol{w}_0 \in \texttt{span}(P_{k_0}),\ \forall k \geq 0.$$

From Theorem 3, we know that $f(\boldsymbol{w}_k)$ converges to $f(\boldsymbol{w}^*)$. This property and the strong convexity of f imply that $\boldsymbol{w}_k$ also converges to $\boldsymbol{w}^*$ globally because

$$f(\boldsymbol{w}_k) - f(\boldsymbol{w}^*) \geq \frac{\sigma}{2}\|\boldsymbol{w}_k - \boldsymbol{w}^*\|^2.$$

Therefore, we have $\boldsymbol{w}^* - \boldsymbol{w}_0 \in \texttt{span}(P_{k_0})$. This means that the minimization procedure takes update directions wholly in the space $\texttt{span}(P_{k_0})$, and each step in the algorithm is equivalent to applying the Newton method with line search in the subspace. Consider the following equivalent reformulation

$$\boldsymbol{w}_k = \boldsymbol{w}_0 + P_{k_0}\hat{\boldsymbol{t}}_k, \boldsymbol{w}^* = \boldsymbol{w}_0 + P_{k_0}\hat{\boldsymbol{t}}^*, \forall k \geq k_0.$$

Our method after the k_0-th iteration is equivalent to applying the Newton method over the variables $\hat{\boldsymbol{t}}$. That is, the following optimization problem is solved

$$\min_{\hat{\boldsymbol{t}}} \quad f(\boldsymbol{w}_0 + P_{k_0}\hat{\boldsymbol{t}}). \tag{55}$$

From Lemma 9, (55) possesses a unique optimal solution $\hat{\boldsymbol{t}}^*$. See also (47), which has the form of the Newton direction of (55). Note that for $g(\boldsymbol{t}) \equiv f(\boldsymbol{w}_0 + P_{k_0}\boldsymbol{t})$, Assumption 2 is still satisfied because with $\|P_{k_0}\| = 1$,

$$\begin{aligned}&\|\nabla^2 g(\boldsymbol{t}_1) - \nabla^2 g(\boldsymbol{t}_2)\| = \|P_{k_0}^\top(\nabla^2 f(\boldsymbol{w}_0 + P_{k_0}\boldsymbol{t}_1) - \nabla^2 f(\boldsymbol{w}_0 + P_{k_0}\boldsymbol{t}_2))P_{k_0}\| \\ \leq\ &\|P_{k_0}\|^2 M\|P_{k_0}(\boldsymbol{t}_1 - \boldsymbol{t}_2)\| \leq M\|\boldsymbol{t}_1 - \boldsymbol{t}_2\|.\end{aligned}$$

Further, the line search process is scale-invariant because the columns in P_{k_0} are orthogonal due to the Schmidt process. Thus, the quadratic convergence of $\hat{\boldsymbol{t}}_k$ follows from that of the Newton method for problems satisfying Assumption 2 (Nocedal and Wright, 2006, Theorem 3.5).

$$\|\hat{\boldsymbol{t}}_{k+1} - \hat{\boldsymbol{t}}^*\| = O(\|\hat{\boldsymbol{t}}_k - \hat{\boldsymbol{t}}^*\|^2).$$

Because columns in P_{k_0} are orthogonal to each other, we have that

$$\|P_{k_0}(\hat{\boldsymbol{t}}_{k+1} - \hat{\boldsymbol{t}}^*)\| = \|(\hat{\boldsymbol{t}}_{k+1} - \hat{\boldsymbol{t}}^*)\| = O(\|\hat{\boldsymbol{t}}_k - \hat{\boldsymbol{t}}^*\|^2) = O(\|P_{k_0}(\hat{\boldsymbol{t}}_k - \hat{\boldsymbol{t}}^*)\|^2).$$

Therefore, quadratic convergence for the original variables $\boldsymbol{w}_k$ holds. ■

Appendix B. Proof of Theorem 5

We first prove that, if every iteration of Algorithm 4 reaches the optimal solution of (21), then the optimal convergence rate is guaranteed. The main idea of this proof follows from the convergence rate proof of Nesterov's accelerated gradient (Nesterov, 1983) in Bubeck (2015, Theorem 3.17).

Let $\boldsymbol{w}_k$ be the iterate obtained from Algorithm 4 at the k-th iteration. Define

$$\phi_{k+1}(\boldsymbol{w}) \equiv (1-\alpha)\,\phi_k(\boldsymbol{w}) + \alpha\left(f(\boldsymbol{w}_k) + \nabla f(\boldsymbol{w}_k)^\top(\boldsymbol{w}-\boldsymbol{w}_k) + \frac{\sigma}{2}\|\boldsymbol{w}-\boldsymbol{w}_k\|^2\right), k \geq 0, \tag{56}$$

$$\text{with } \alpha \equiv \sqrt{\frac{\sigma}{\rho}} \in (0,1], \quad \text{and} \quad \phi_0(\boldsymbol{w}) \equiv \frac{\sigma}{2}\|\boldsymbol{w}-\boldsymbol{w}_0\|^2 + f(\boldsymbol{w}_0). \tag{57}$$

We will first show that, by induction,

$$\nabla^2\phi_k(\boldsymbol{w}) = \sigma I, \ \forall k. \tag{58}$$

The claim holds trivially at $k=0$ from (57). Now suppose the claim (58) holds for some $k \geq 0$. For the $(k+1)$-th iteration, by (56) we have

$$\nabla^2\phi_{k+1}(\boldsymbol{w}) = (1-\alpha)\nabla^2\phi_k(\boldsymbol{w}) + \alpha\sigma I = \sigma I.$$

Thus, the induction on (58) is established.

Now we examine the recurrence relation in ϕ_k. Because $\phi_0(\boldsymbol{w})$ and the recurrence (56) are all strongly convex quadratic, we have $\phi_k(\boldsymbol{w})$ is also a strongly convex quadratic by induction. Let $\boldsymbol{v}_k$ be a constant that minimizes ϕ_k, which implies $\nabla\phi_k(\boldsymbol{v}_k) = \boldsymbol{0}$. By (58) and the Taylor expansion, we obtain

$$\begin{aligned}\phi_k(\boldsymbol{w}) &= \phi_k(\boldsymbol{v}_k) + \nabla\phi_k(\boldsymbol{v}_k)^\top(\boldsymbol{w}-\boldsymbol{v}_k) + \frac{\sigma}{2}\|\boldsymbol{w}-\boldsymbol{v}_k\|^2 \\ &= \phi_k(\boldsymbol{v}_k) + \frac{\sigma}{2}\|\boldsymbol{w}-\boldsymbol{v}_k\|^2, \ \forall \boldsymbol{w}. \end{aligned} \tag{59}$$

Further, let $\boldsymbol{v}_{k+1}$ be the minimizer of ϕ_{k+1}. Using the derivative of (56) together with (59), we have

$$\begin{aligned}&\nabla\phi_{k+1}(\boldsymbol{v}_{k+1}) \\ &= (1-\alpha)\nabla\phi_k(\boldsymbol{v}_{k+1}) + \alpha(\nabla f(\boldsymbol{w}_k) + \sigma(\boldsymbol{v}_{k+1}-\boldsymbol{w}_k)) \\ &= (1-\alpha)\sigma(\boldsymbol{v}_{k+1}-\boldsymbol{v}_k) + \alpha(\nabla f(\boldsymbol{w}_k) + \sigma(\boldsymbol{v}_{k+1}-\boldsymbol{w}_k)) = \boldsymbol{0}.\end{aligned}$$

Thus, we obtain a recurrent definition of $\boldsymbol{v}_{k+1}$:

$$\begin{aligned}\boldsymbol{v}_0 &= \boldsymbol{w}_0, \\ \boldsymbol{v}_{k+1} &= (1-\alpha)\boldsymbol{v}_k + \alpha\boldsymbol{w}_k - \frac{\alpha}{\sigma}\nabla f(\boldsymbol{w}_k).\end{aligned} \tag{60}$$

From (60), we have

$$\boldsymbol{v}_{k+1} - \boldsymbol{v}_k = -\alpha(\boldsymbol{v}_k - \boldsymbol{w}_k) - \frac{\alpha}{\sigma}\nabla f(\boldsymbol{w}_k), \tag{61}$$

$$\boldsymbol{v}_{k+1} - \boldsymbol{w}_k = (1-\alpha)(\boldsymbol{v}_k - \boldsymbol{w}_k) - \frac{\alpha}{\sigma}\nabla f(\boldsymbol{w}_k). \tag{62}$$

Next, we obtain a recurrent relation of $\phi_{k+1}(\boldsymbol{v}_{k+1})$.

$$\begin{aligned}\phi_{k+1}(\boldsymbol{v}_{k+1}) = \; & (1-\alpha)\left(\phi_k(\boldsymbol{v}_k) + \frac{\sigma}{2}\|\boldsymbol{v}_{k+1}-\boldsymbol{v}_k\|^2\right) \\ & + \alpha\left(f(\boldsymbol{w}_k) + \nabla f(\boldsymbol{w}_k)^\top(\boldsymbol{v}_{k+1}-\boldsymbol{w}_k) + \frac{\sigma}{2}\|\boldsymbol{v}_{k+1}-\boldsymbol{w}_k\|^2\right) \qquad (63) \\ = \; & (1-\alpha)\phi_k(\boldsymbol{v}_k) + \alpha f(\boldsymbol{w}_k) \\ & + \alpha(1-\alpha)\left(\nabla f(\boldsymbol{w}_k)^\top(\boldsymbol{v}_k-\boldsymbol{w}_k) + \frac{\sigma}{2}\|\boldsymbol{v}_k-\boldsymbol{w}_k\|^2\right) - \frac{1}{2\rho}\|\nabla f(\boldsymbol{w}_k)\|^2, \qquad (64)\end{aligned}$$

where (63) is by (56) and (59), and (64) is by (61), (62), and applying the definition $\alpha = \sqrt{\sigma/\rho}$ to the coefficient of $\|\nabla f(\boldsymbol{w}_k)\|^2$. Note that $\{\phi_k\}$ is called the estimation sequence in Nesterov (1983).

Now we show by induction that, for $\boldsymbol{w}_k$ minimizing (21) exactly,

$$\min_{\boldsymbol{w}\in\mathbb{R}^n} \quad \phi_k(\boldsymbol{w}) = \phi_k(\boldsymbol{v}_k) \geq f(\boldsymbol{w}_k),\ \forall k \geq 0. \qquad (65)$$

First, (65) holds trivially at $k=0$ from the definition (57). Assume (65) holds at the k-th iteration. At the $(k+1)$-th iteration, (64) and (65) imply

$$\phi_{k+1}(\boldsymbol{v}_{k+1}) \geq f(\boldsymbol{w}_k) + \alpha(1-\alpha)\left(\nabla f(\boldsymbol{w}_k)^\top(\boldsymbol{v}_k-\boldsymbol{w}_k) + \frac{\sigma}{2}\|\boldsymbol{v}_k-\boldsymbol{w}_k\|^2\right) - \frac{1}{2\rho}\|\nabla f(\boldsymbol{w}_k)\|^2. \qquad (66)$$

By the definition of P_k and induction on (60), both $\boldsymbol{v}_k$ and $\boldsymbol{w}_k$ belong to $\boldsymbol{w}_0 + \texttt{span}(P_{k-1})$. Thus, the optimality of $\boldsymbol{w}_k$ on

$$\min_{\boldsymbol{t}} \quad f(\boldsymbol{w}_{k-1} + P_{k-1}\boldsymbol{t})$$

implies that

$$P_{k-1}^\top \nabla f(\boldsymbol{w}_k) = \boldsymbol{0}. \qquad (67)$$

Thus

$$\nabla f(\boldsymbol{w}_k)^\top(\boldsymbol{v}_k - \boldsymbol{w}_k) = 0. \qquad (68)$$

This result, (66), and $\alpha \in (0,1]$ from (57) then lead to

$$\phi_{k+1}(\boldsymbol{v}_{k+1}) \geq f(\boldsymbol{w}_k) - \frac{1}{2\rho}\|\nabla f(\boldsymbol{w}_k)\|^2. \qquad (69)$$

Further,

$$\begin{aligned} & f(\boldsymbol{w}_k) - \frac{1}{2\rho}\|\nabla f(\boldsymbol{w}_k)\|^2 = f(\boldsymbol{w}_k) - \frac{1}{\rho}\|\nabla f(\boldsymbol{w}_k)\|^2 + \frac{\rho}{2}\frac{1}{\rho^2}\|\nabla f(\boldsymbol{w}_k)\|^2 \\ \geq \; & f(\boldsymbol{w}_k - \frac{1}{\rho}\nabla f(\boldsymbol{w}_k)) \qquad (70) \\ \geq \; & f(\boldsymbol{w}_{k+1}). \qquad (71)\end{aligned}$$

The inequality (70) comes from the Taylor expansion of $f(\boldsymbol{w}_k - \nabla f(\boldsymbol{w}_k)/\rho)$ and (44):

$$\begin{aligned} f(\boldsymbol{w}_k - \frac{1}{\rho}\nabla f(\boldsymbol{w}_k)) &= f(\boldsymbol{w}_k) - \frac{1}{\rho}\|\nabla f(\boldsymbol{w}_k)\|^2 + \frac{1}{2\rho^2}\nabla f(\boldsymbol{w}_k)^\top \nabla^2 f(\tilde{\boldsymbol{w}})\nabla f(\boldsymbol{w}_k) \\ &\leq f(\boldsymbol{w}_k) - \frac{1}{\rho}\|\nabla f(\boldsymbol{w}_k)\|^2 + \frac{1}{2\rho}\|\nabla f(\boldsymbol{w}_k)\|^2, \end{aligned}$$

where $\tilde{\boldsymbol{w}}$ is between $\boldsymbol{w}_k$ and $\boldsymbol{w}_k - \frac{1}{\rho}\nabla f(\boldsymbol{w}_k)$. For (71), it holds because $\nabla f(\boldsymbol{w}_k) \in \texttt{span}(P_k)$ and $\boldsymbol{w}_{k+1}$ is a minimizer of f over $\boldsymbol{w}_0 + \texttt{span}(P_k)$. Combining (69) and (71), we have (65) holds at iteration $k+1$. Therefore, (65) holds for every $k \geq 0$ by induction.

Now, we arrive at the final steps. From (65), (56), and the strong convexity of f in Assumption 1, we have

$$\begin{aligned}
& f(\boldsymbol{w}_{k+1}) - f(\boldsymbol{w}^*) \\
\leq & \min_{\boldsymbol{w}} \quad \phi_{k+1}(\boldsymbol{w}) - f(\boldsymbol{w}^*) \\
\leq & \ \phi_{k+1}(\boldsymbol{w}^*) - f(\boldsymbol{w}^*) \qquad (72) \\
= & \ (1-\alpha)\phi_k(\boldsymbol{w}^*) + \alpha(f(\boldsymbol{w}_k) + \nabla f(\boldsymbol{w}_k)^\top(\boldsymbol{w}^* - \boldsymbol{w}_k) + \frac{\sigma}{2}\|\boldsymbol{w}^* - \boldsymbol{w}_k\|^2) - f(\boldsymbol{w}^*) \\
\leq & \ (1-\alpha)\phi_k(\boldsymbol{w}^*) + \alpha f(\boldsymbol{w}^*) - f(\boldsymbol{w}^*) \\
= & \ (1-\alpha)(\phi_k(\boldsymbol{w}^*) - f(\boldsymbol{w}^*)) \qquad (73) \\
\leq & \ (1-\alpha)^{k+1}(\phi_0(\boldsymbol{w}^*) - f(\boldsymbol{w}^*)). \qquad (74)
\end{aligned}$$

Note that (74) comes from repeating the steps between (72) and (73). By taking logarithm at both sides, we have proven that Algorithm 4 achieves the optimal rate when (21) is solved exactly.

Next we discuss the situation when (21) is approximately solved. In the above proof we can see that having a $\boldsymbol{w}_{k+1}$ that satisfies (65) is sufficient for proving the linear convergence. Because (65) comes from (66)-(69) and (71),[11] we can consider running inner iterations in Algorithm 4 until the following inner stopping condition is satisfied.

$$\begin{cases} \nabla f(\boldsymbol{w}_{k+1})^\top(\boldsymbol{v}_{k+1} - \boldsymbol{w}_{k+1}) + \frac{\sigma}{2}\|\boldsymbol{v}_{k+1} - \boldsymbol{w}_{k+1}\|^2 \geq 0, & \text{(75a)} \\ f(\boldsymbol{w}_{k+1}) \leq f(\boldsymbol{w}_k - \frac{1}{\rho}\nabla f(\boldsymbol{w}_k)). & \text{(75b)} \end{cases}$$

Denote $\boldsymbol{t}_k^*$ as the exact solution of (21) and let

$$\boldsymbol{w}_{k+1}^* \equiv \boldsymbol{w}_k + P_k \boldsymbol{t}_k^*.$$

From the discussion in (68) and (71), clearly $\boldsymbol{w}_{k+1}^*$ satisfies (75). The remaining task is to prove that (75) can be achieved in a finite number of iterations. If $\boldsymbol{w}_{k+1}^*$ strictly satisfies inequalities in (75), then by the continuity of f and ∇f, there is a neighborhood of $\boldsymbol{w}_{k+1}^*$ in which every point satisfies (75). Because inner iterations lead to the convergence to $\boldsymbol{w}_{k+1}^*$, the inner loop must terminate finitely. On the other hand, we can easily rule out situations where $\boldsymbol{w}_{k+1}^*$ satisfies one of the conditions in (75) as an equality. When (75a) becomes an equality under $\boldsymbol{w}_{k+1}^*$, (68) implies that

$$\boldsymbol{w}_{k+1}^* = \boldsymbol{v}_{k+1}.$$

When (75b) becomes an equality, the strong convexity of $f(\boldsymbol{w}_k + P_k\boldsymbol{t})$ on $\boldsymbol{t}$, which follows from Lemma 9, implies that the optimum $\boldsymbol{w}_{k+1}^*$ of $\min_{\boldsymbol{t}} f(\boldsymbol{w}_k + P_k\boldsymbol{t})$ is unique. This property, the fact that $\nabla f(\boldsymbol{w}_k)$ lies in the column space of P_k, and the equality in (75b) imply that

$$\boldsymbol{w}_{k+1}^* = \boldsymbol{w}_k - \frac{1}{\rho}\nabla f(\boldsymbol{w}_k).$$

11. Note that we ensure (75) holds for every k, so (75a) with k rather than $k+1$ is what we use for (66).

Therefore, before starting the inner loop of Algorithm 4, we can first check if $\boldsymbol{v}_{k+1}$ or $\boldsymbol{w}_k - (1/\rho)\nabla f(\boldsymbol{w}_k)$ satisfies (75). If one of them satisfies both inequalities, then we can take this vector as $\boldsymbol{w}_{k+1}$ without going into the loop. Otherwise, neither $\boldsymbol{v}_{k+1}$ nor $\boldsymbol{w}_k - (1/\rho)\nabla f(\boldsymbol{w}_k)$ is $\boldsymbol{w}^*_{k+1}$, so $\boldsymbol{w}^*_{k+1}$ must strictly satisfy inequalities in (75).

Therefore, after a finite number of inner iterations we can always find an iterate $\boldsymbol{w}_{k+1}$ satisfying (75). This condition ensures (69) and (71) hold. Therefore, (65) is established.

Appendix C. Proof of Theorem 6

At the k-th iteration, if (21) is solved exactly, the optimality condition (67) implies that any column in P_k is orthogonal to $\nabla f(\boldsymbol{w}_{k+1})$. Therefore, if $\nabla f(\boldsymbol{w}_{k+1}) \neq \mathbf{0}$, the new gradient will augment P_{k+1} by one column. By induction, if $\nabla f(\boldsymbol{w}_{k'}) \neq \mathbf{0}$, $\forall k' = 0, \ldots, k-1$, then P_k has k orthogonal columns. Because each column of P_k is in $\mathbb{R}^n$, there can be at most n orthogonal directions in P_k. Therefore, when $k \geq n$, $\texttt{span}(P_k) = R^n$ and the solution of (21) is equivalent to the solution of (1). Thus, the optimum is reached within n iterations.

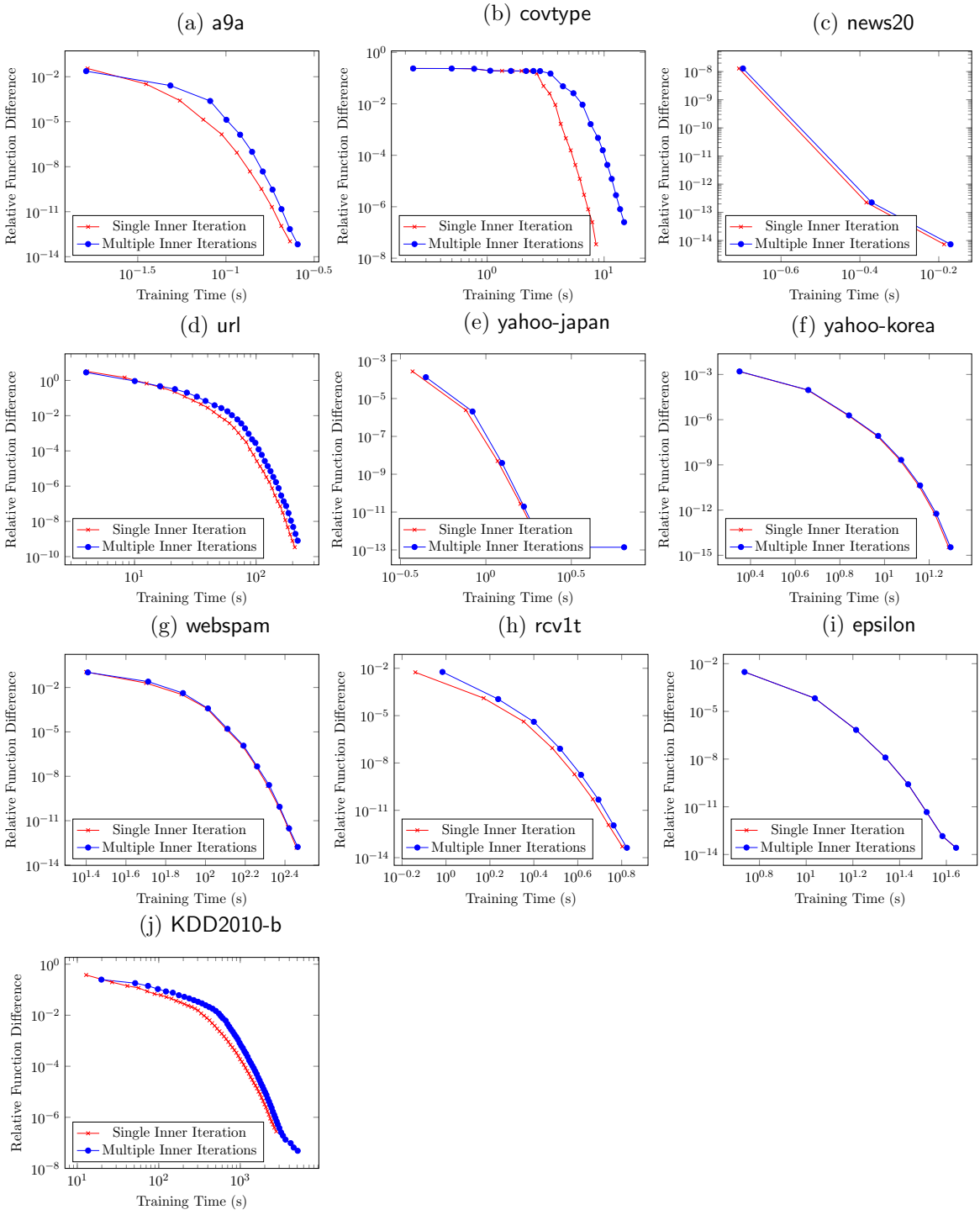

Figure 1: Comparison between single inner iteration and multiple inner iterations variants of the common-directions method. We present training time (in log scale) of logistic regression with $C = 10^{-3}$.

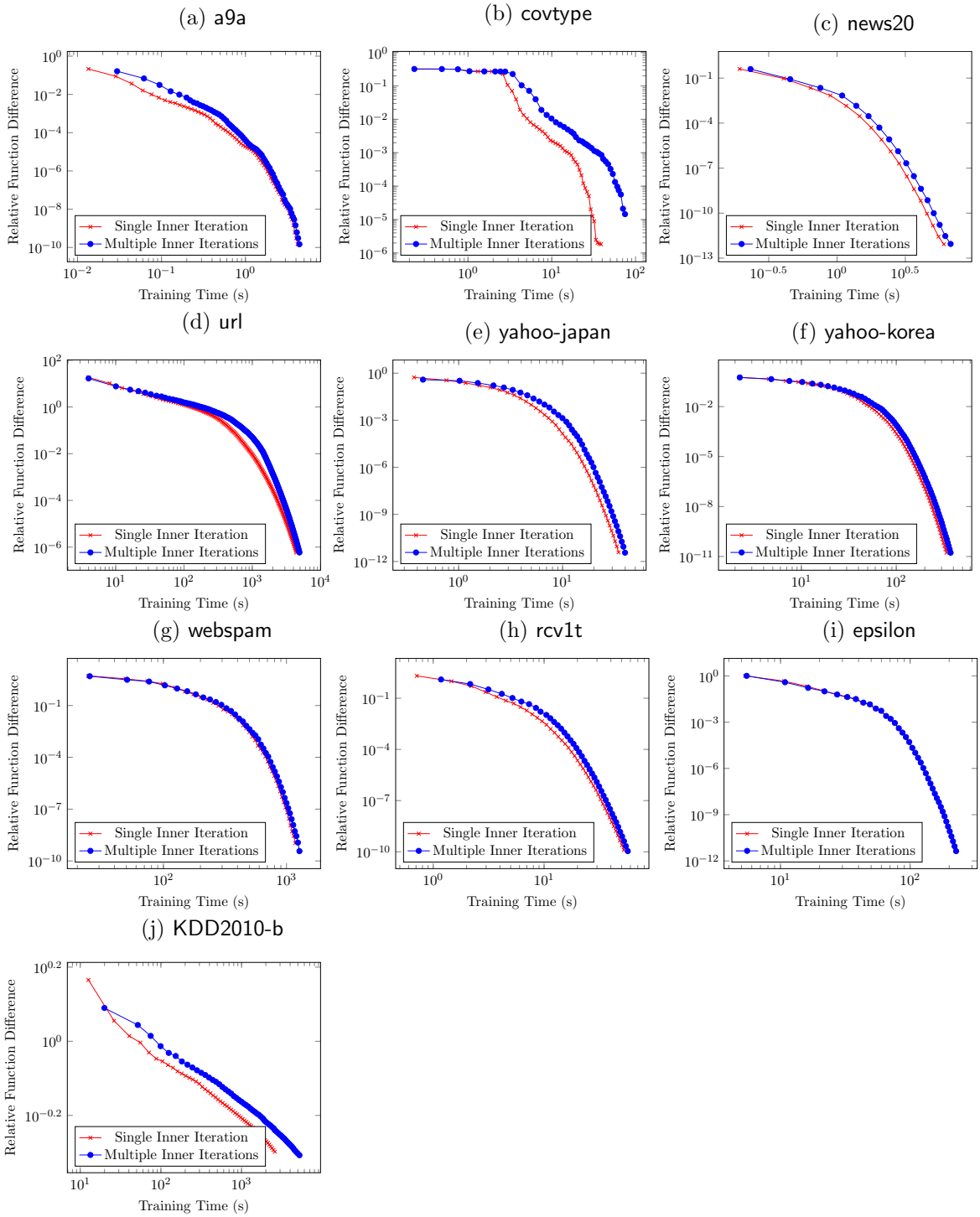

Figure 2: Comparison between single inner iteration and multiple inner iterations variants of the common-directions method. We present training time (in log scale) of logistic regression with $C = 10^1$.

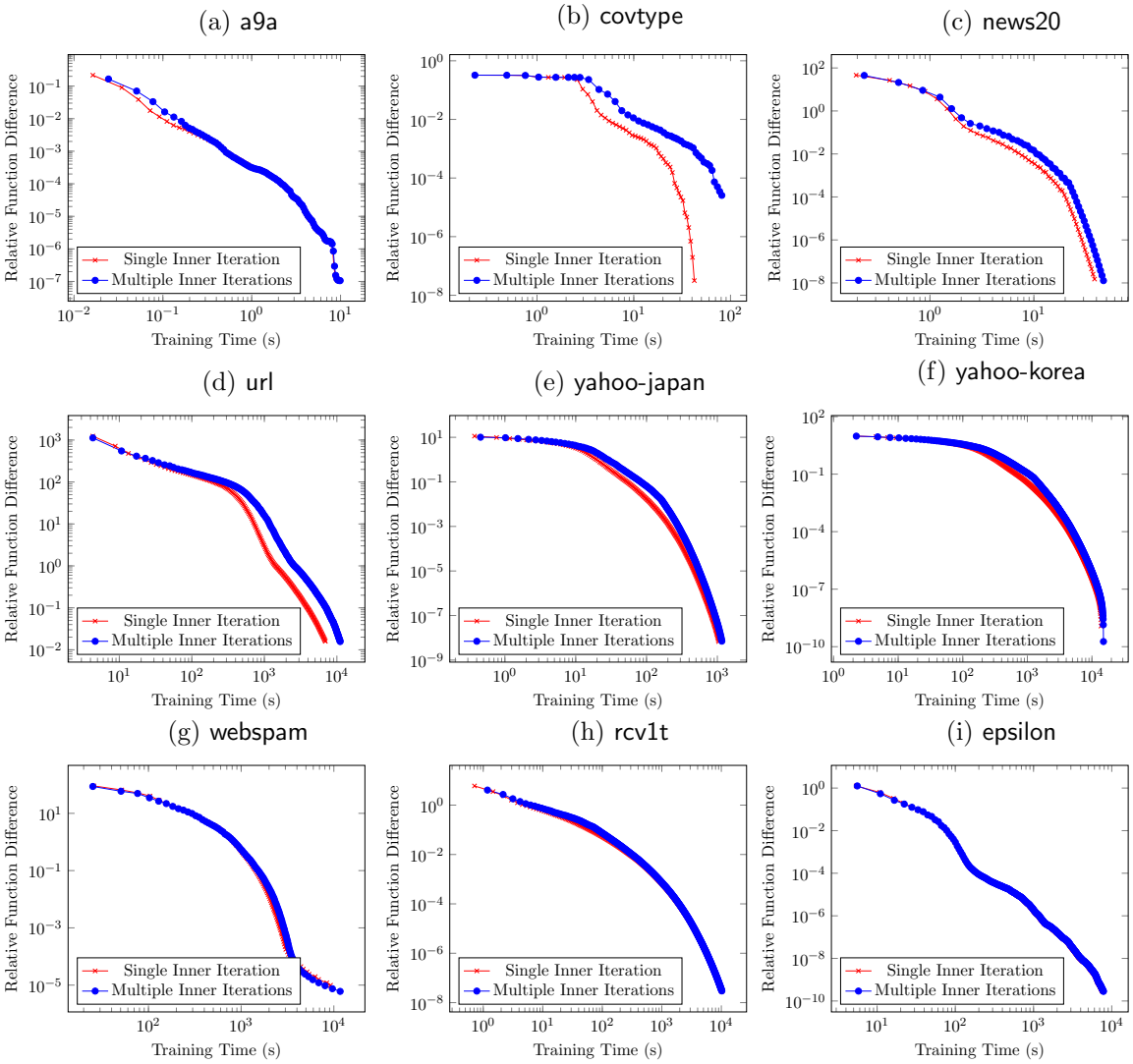

Figure 3: Comparison between single inner iteration and multiple inner iterations variants of the common-directions method. We present training time (in log scale) of logistic regression with $C = 10^3$.

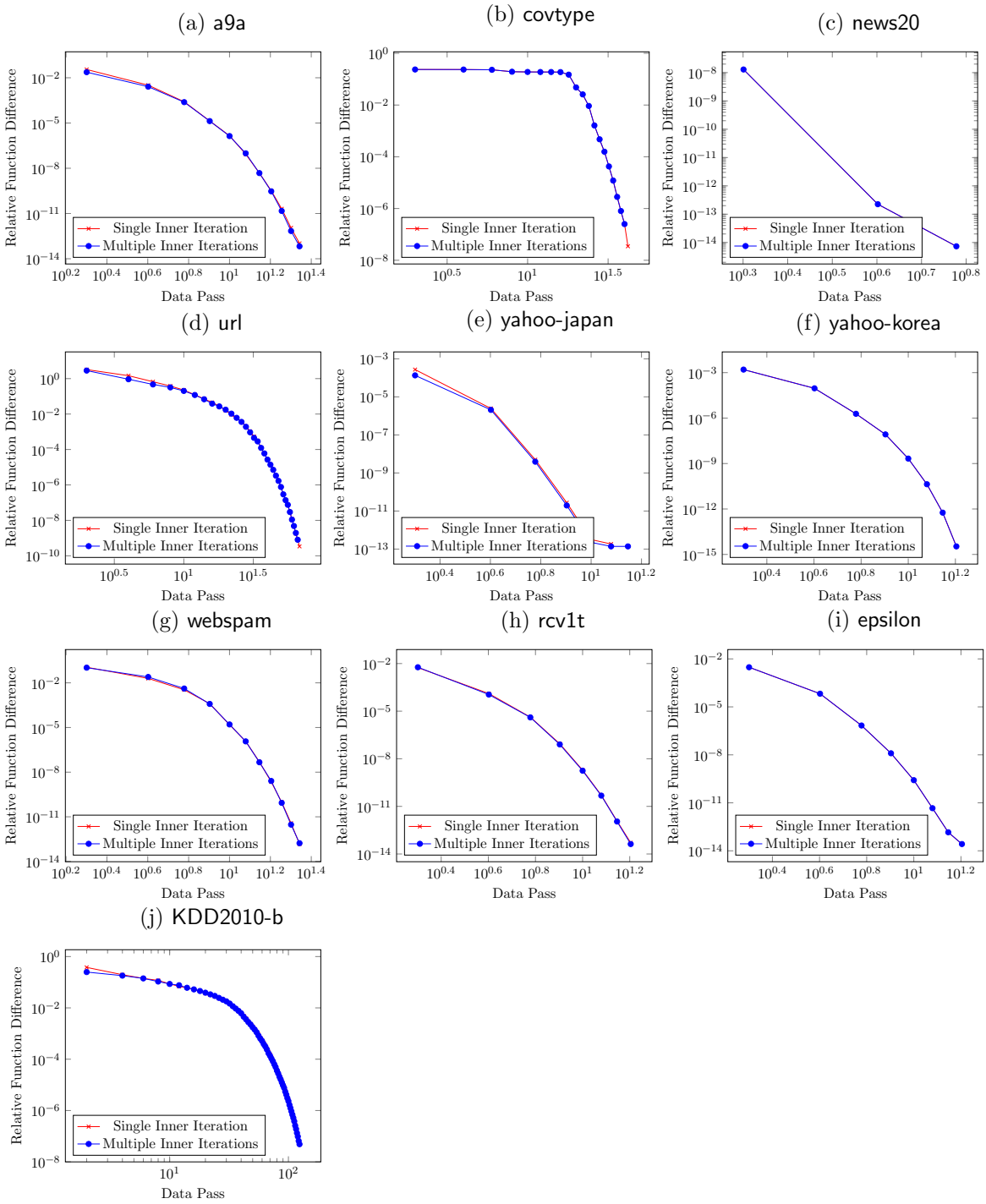

Figure 4: Comparison between single inner iteration and multiple inner iterations variants of the common-directions method. We present data passes (in log scale) of logistic regression with $C = 10^{-3}$.

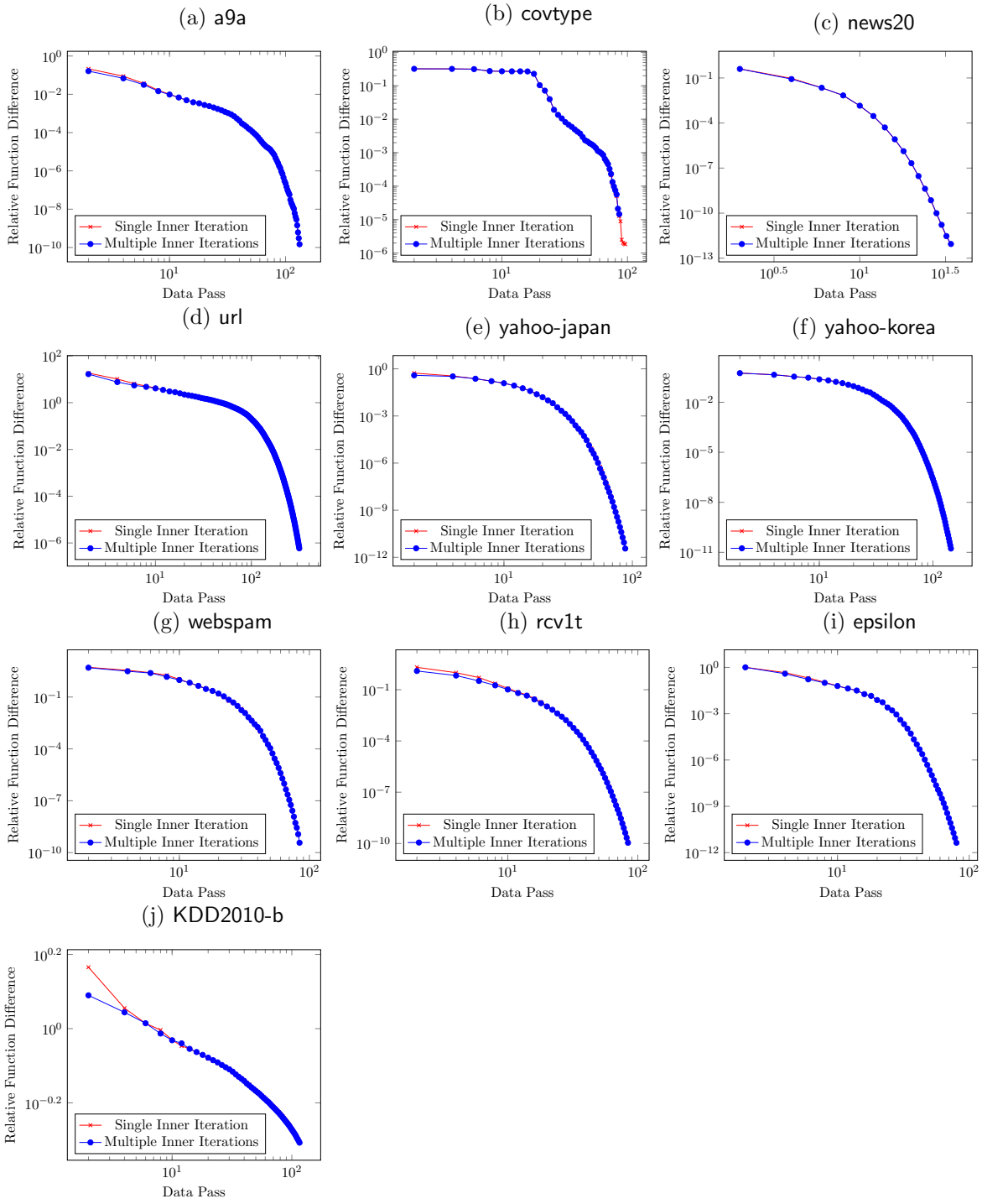

Figure 5: Comparison between single inner iteration and multiple inner iterations variants of the common-directions method. We present data passes (in log scale) of logistic regression with $C = 10^1$.

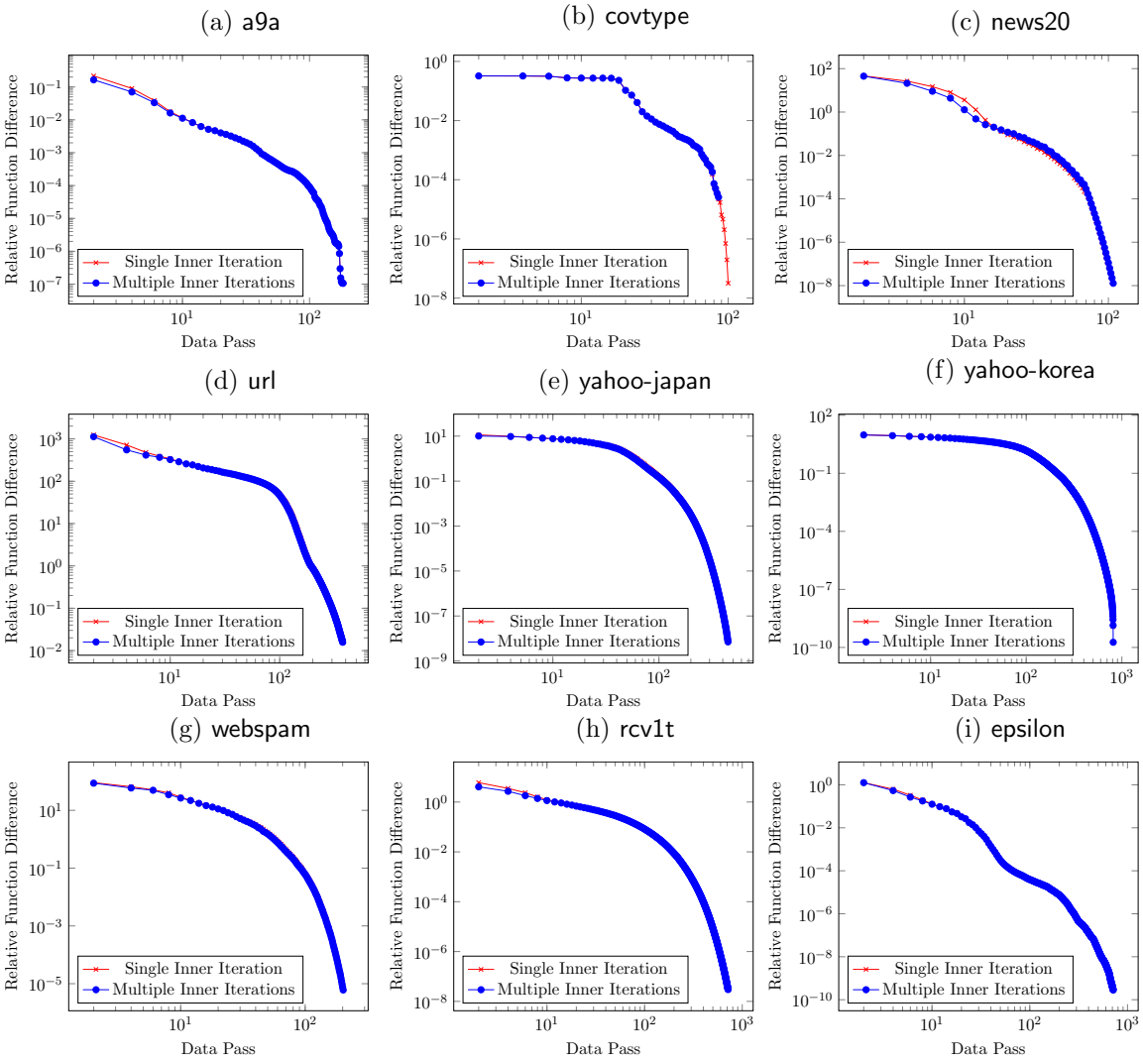

Figure 6: Comparison between single inner iteration and multiple inner iterations variants of the common-directions method. We present data passes (in log scale) of logistic regression with $C = 10^3$.

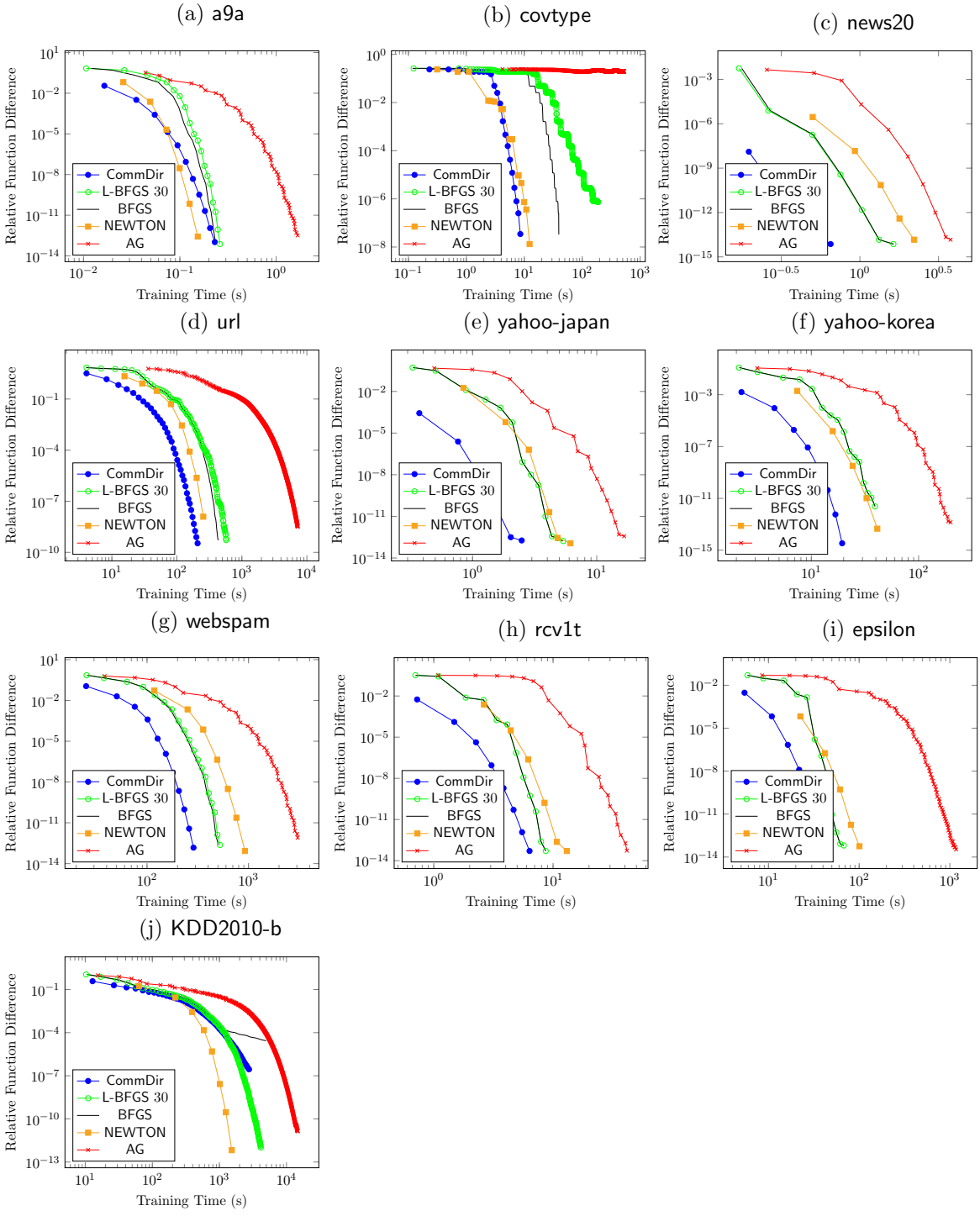

Figure 7: Training time of logistic regression with $C = 10^{-3}$.

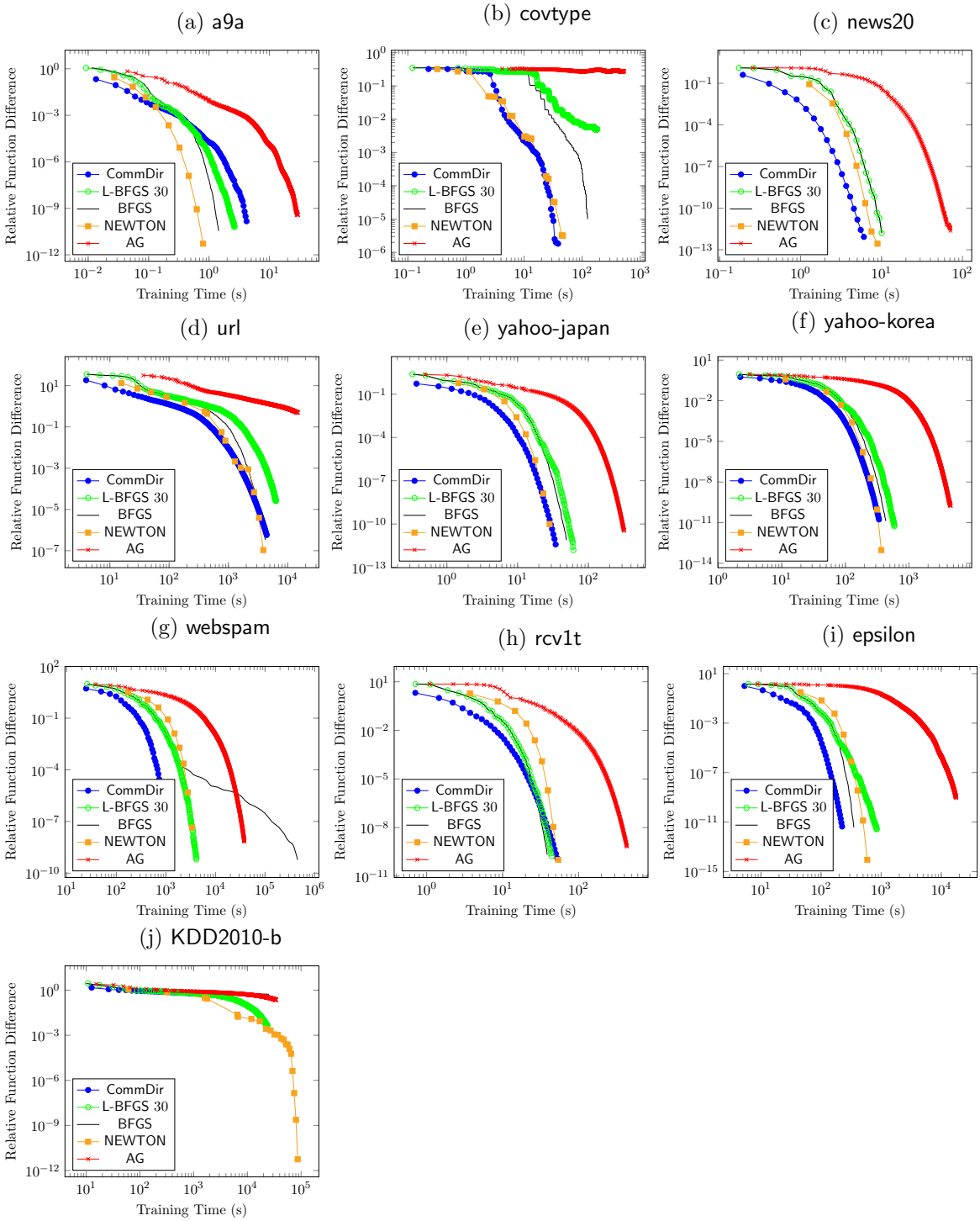

Figure 8: Training time of logistic regression with $C = 1$.

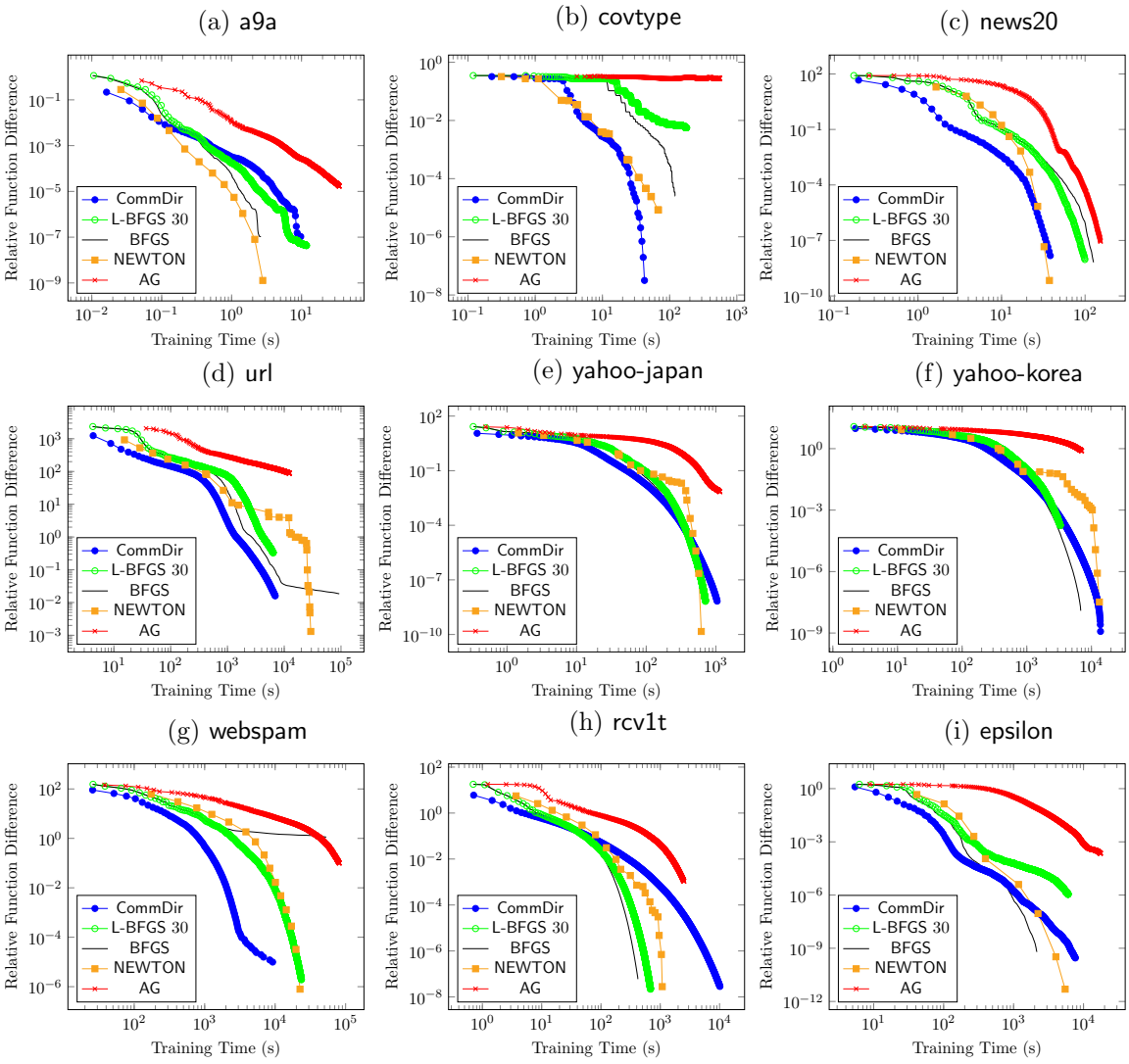

Figure 9: Training time of logistic regression with $C = 10^3$.

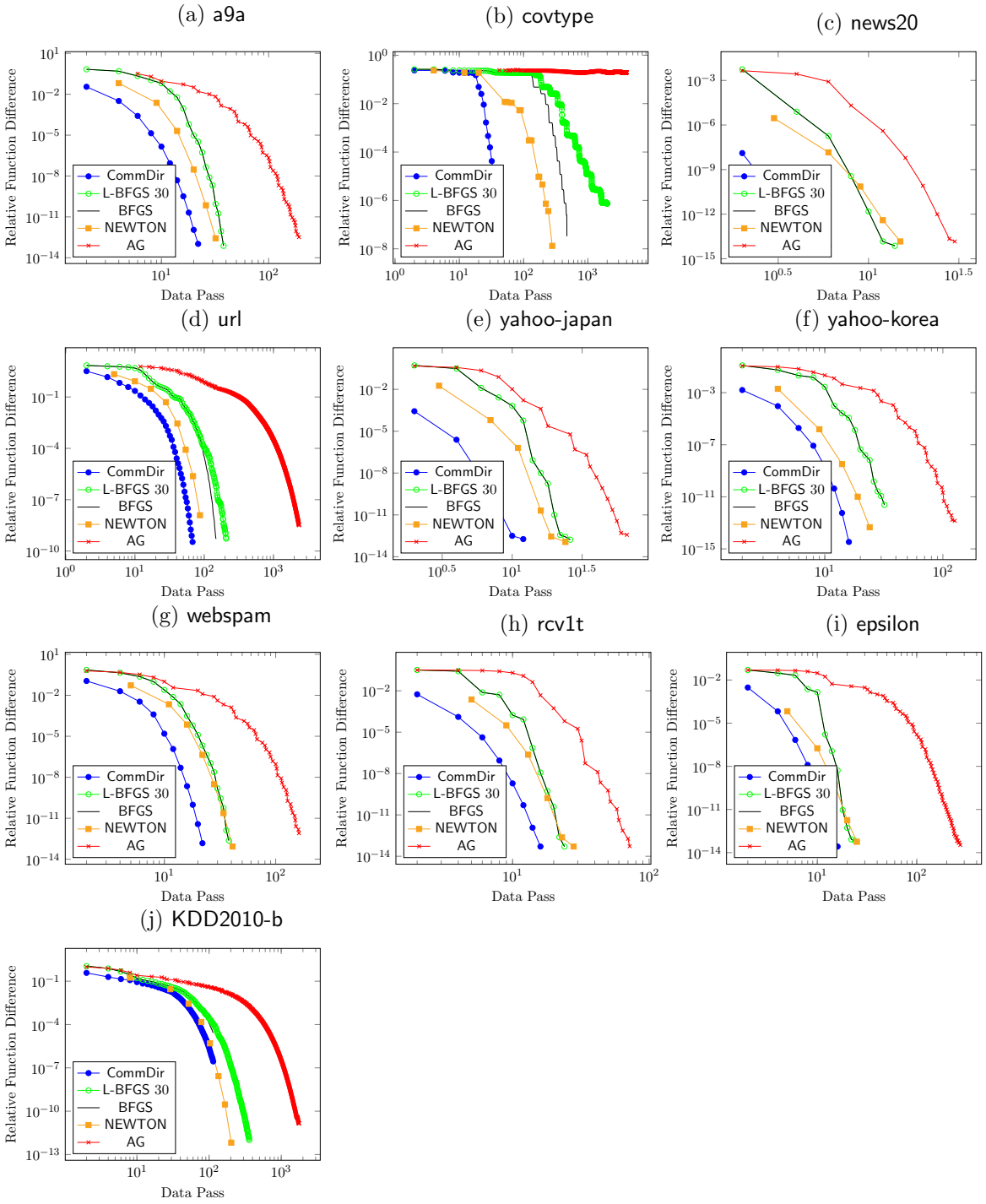

Figure 10: Number of data passes of logistic regression with $C = 10^{-3}$.

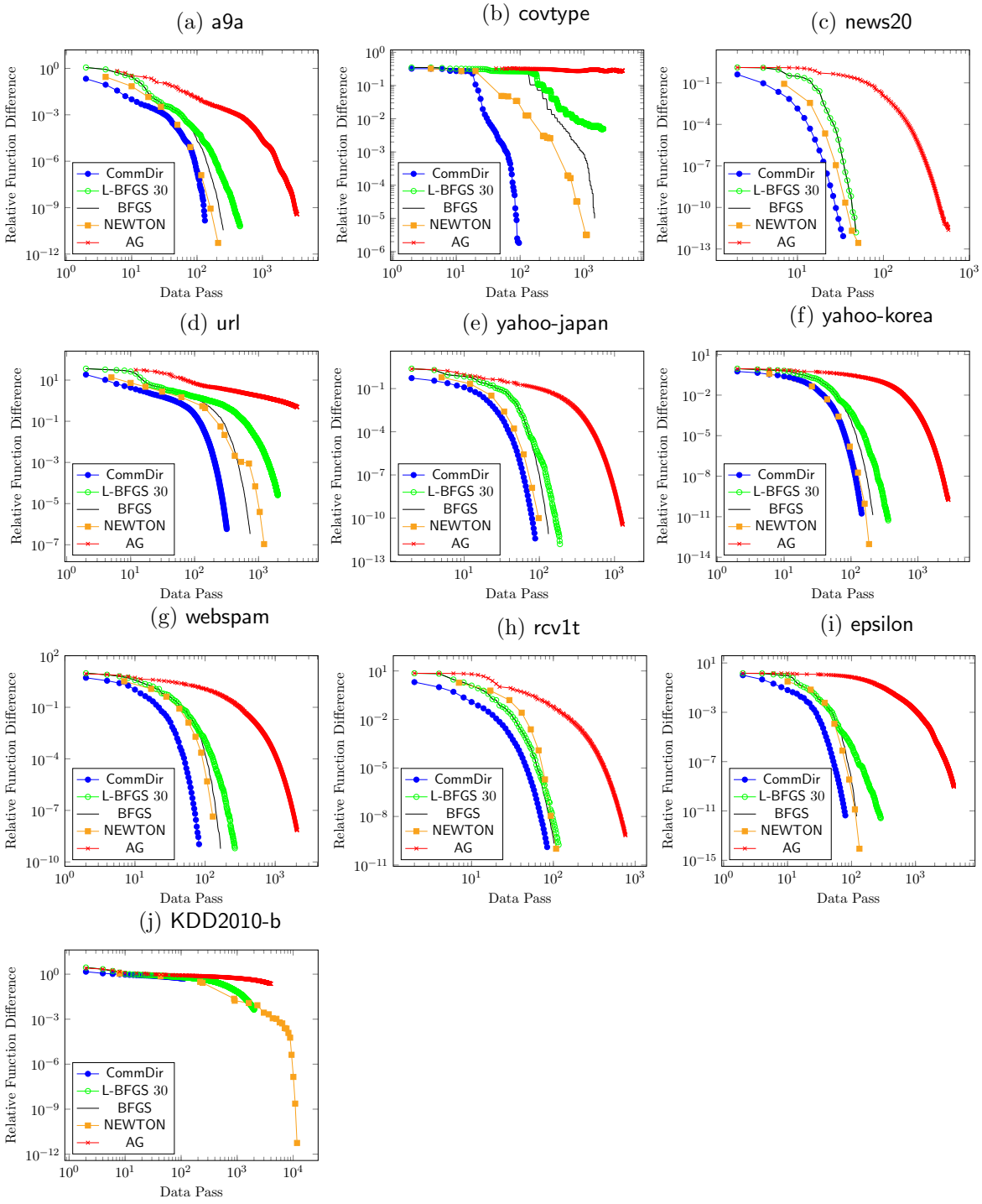

Figure 11: Number of data passes of logistic regression with $C = 1$.

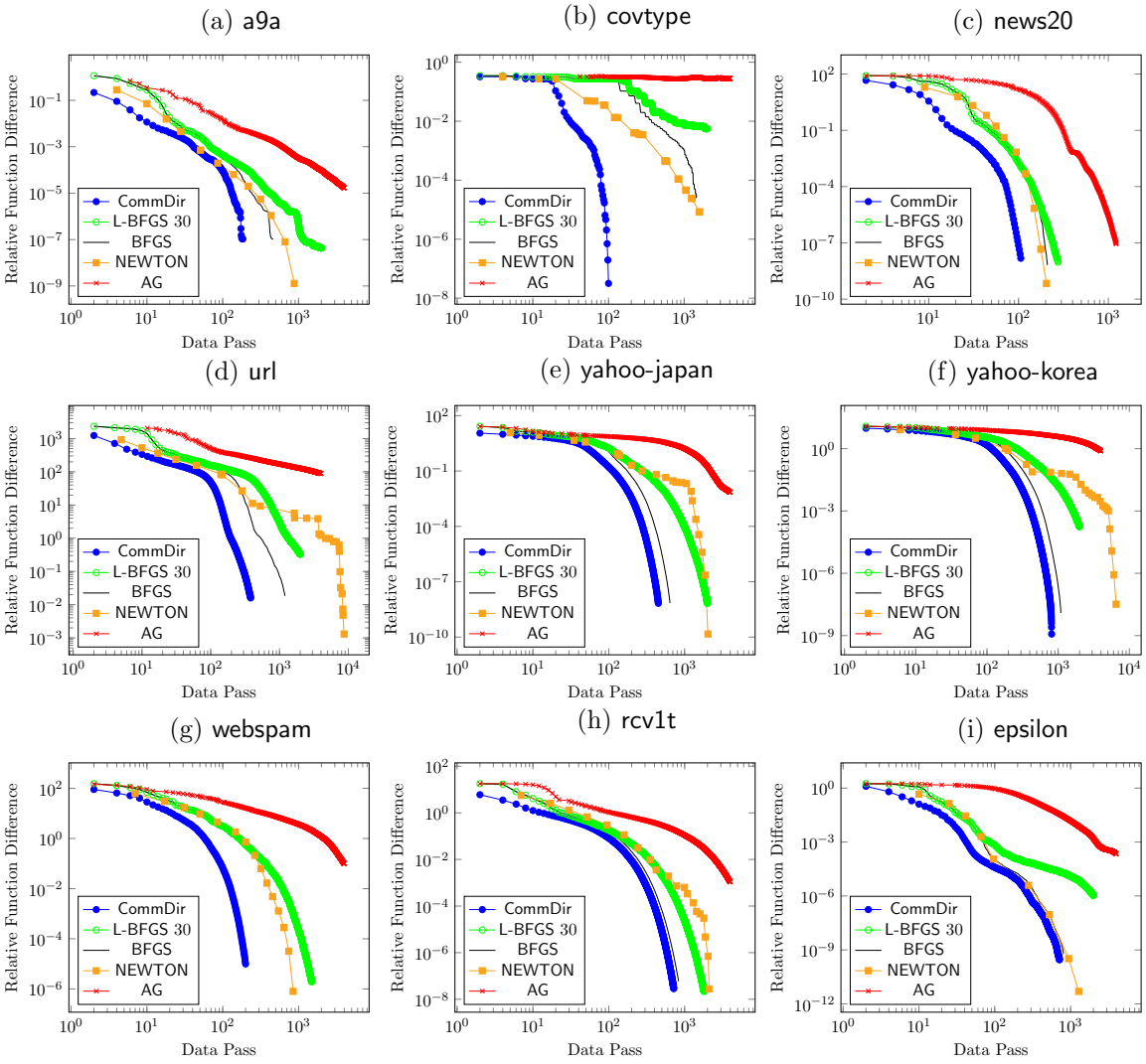

Figure 12: Number of data passes of logistic regression with $C = 10^3$.

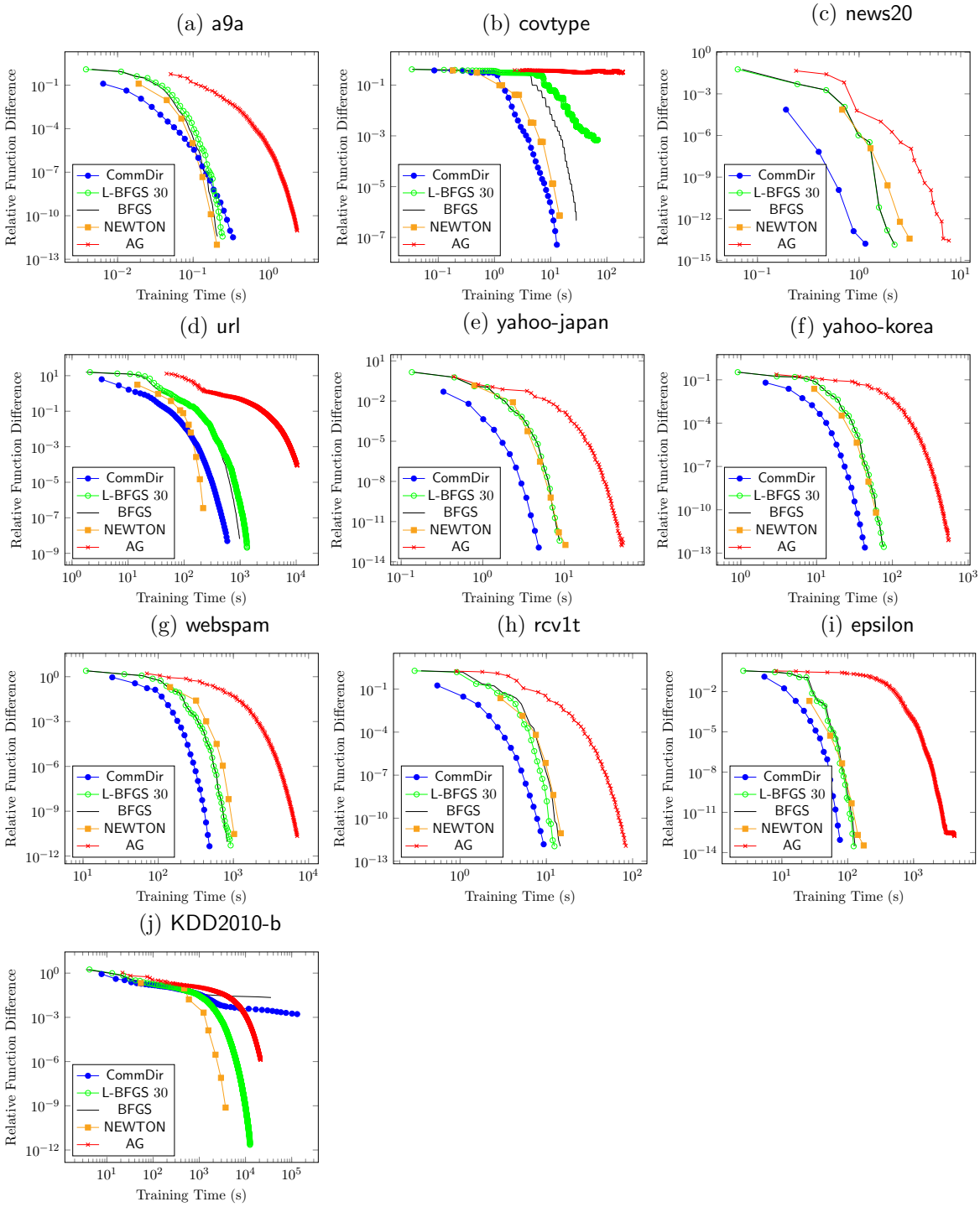

Figure 13: Training time of L2-loss SVM with $C = 10^{-3}$.

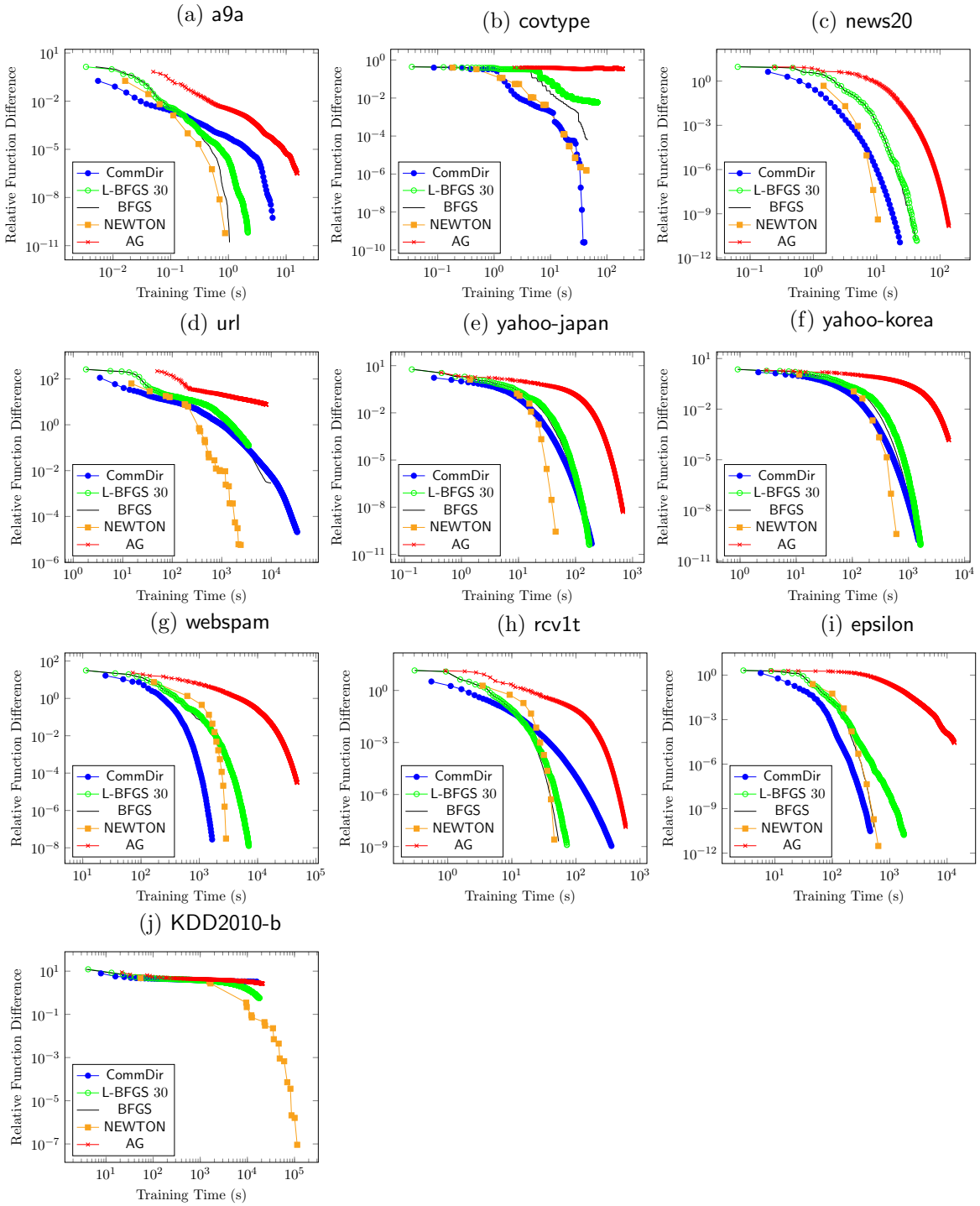

Figure 14: Training time of L2-loss SVM with $C = 1$.

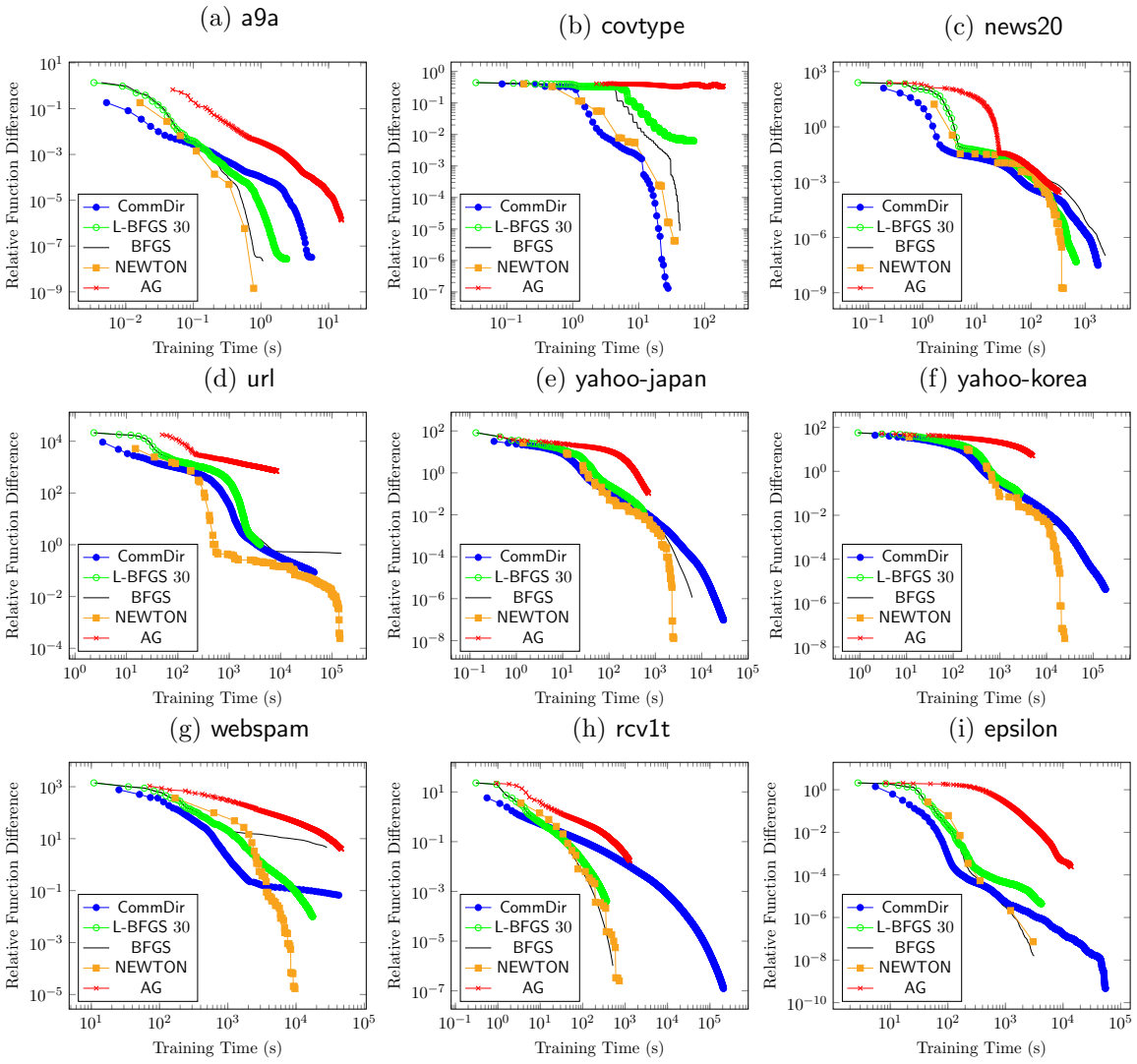

Figure 15: Training time of L2-loss SVM with $C = 10^3$.

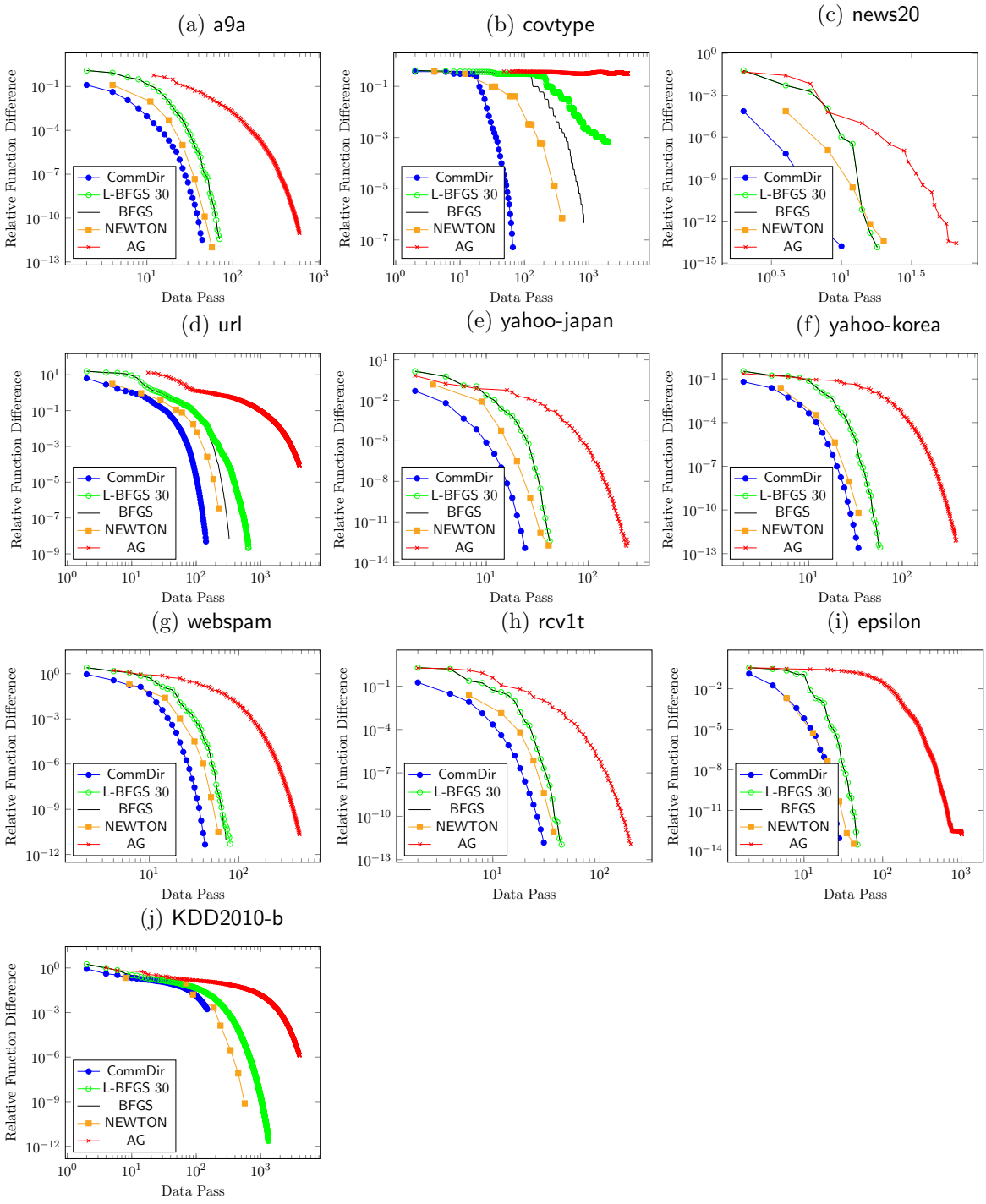

Figure 16: Number of data passes of L2-loss SVM with $C = 10^{-3}$.

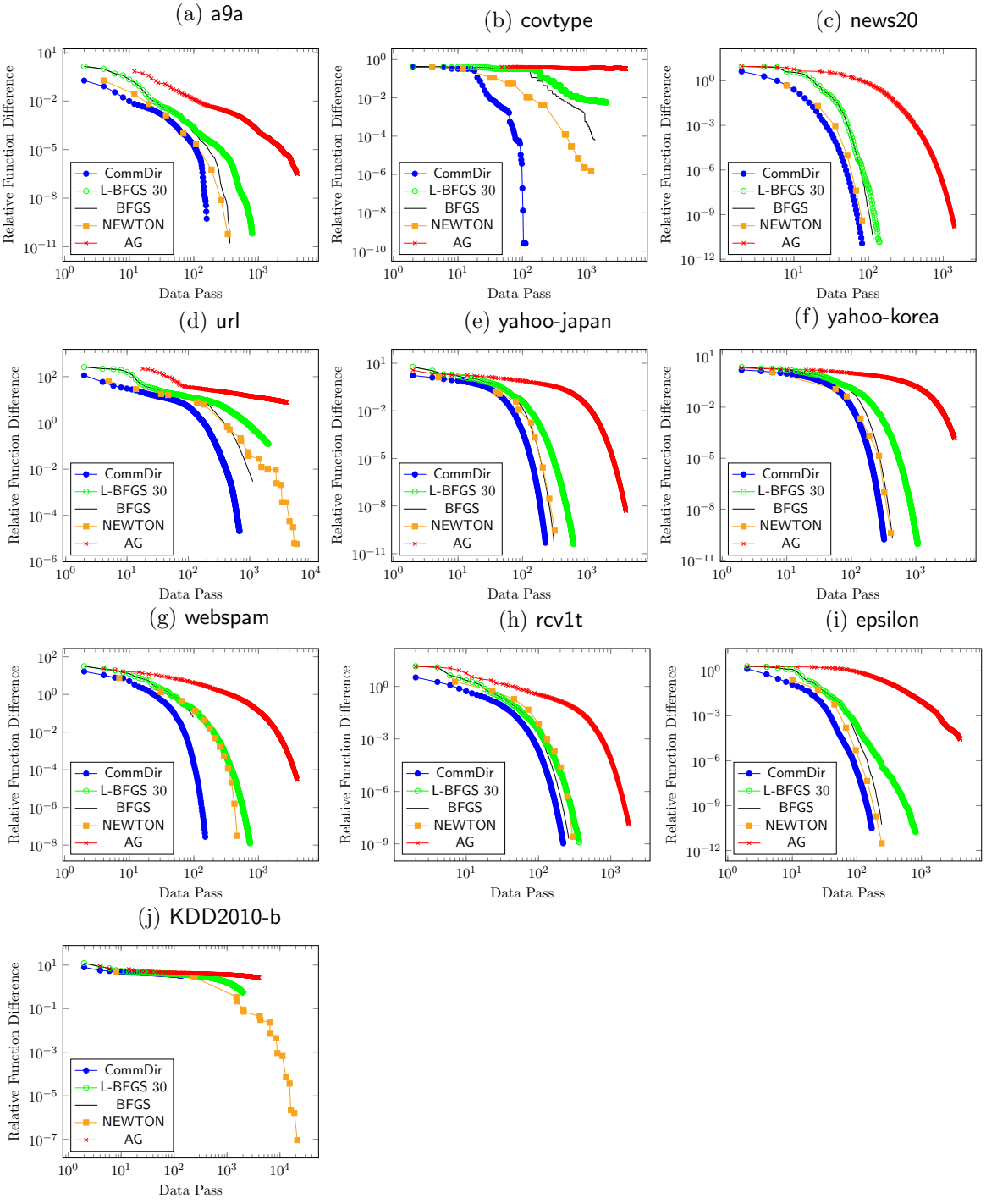

Figure 17: Number of data passes of L2-loss SVM with $C = 1$.

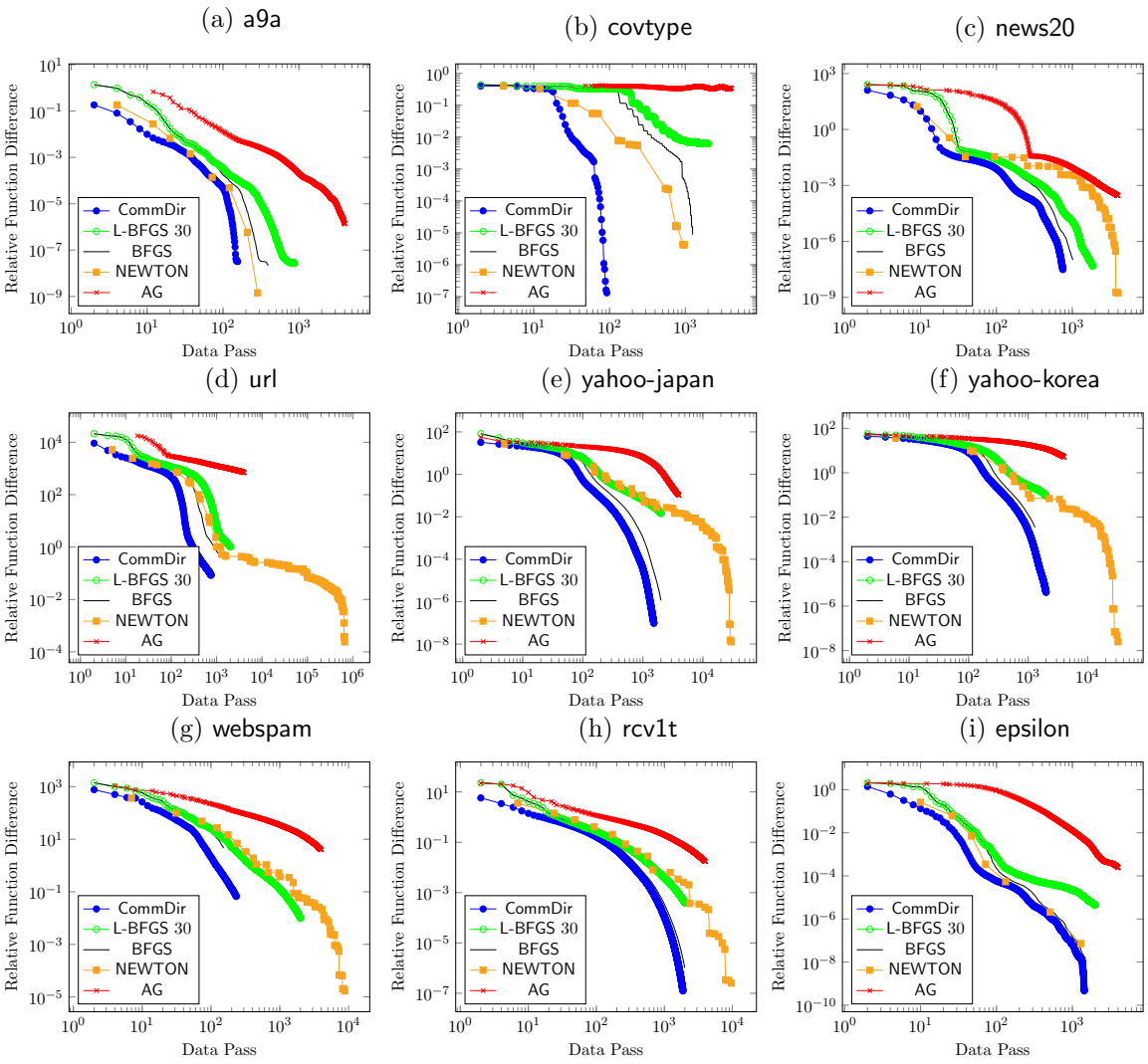

Figure 18: Number of data passes of L2-loss SVM with $C = 10^3$.