Preconditioned Conjugate Gradient Methods in Truncated Newton Frameworks for Large-scale Linear Classification

Chih-Yang Hsia r04922021@ntu.edu.tw
Department of Computer Science
National Taiwan University
Taipei 106, Taiwan
Wei-Lin Chiang b02902056@ntu.edu.tw
Department of Computer Science
National Taiwan University
Taipei 106, Taiwan
Chih-Jen Lin cjlin@csie.ntu.edu.tw
Department of Computer Science
National Taiwan University
Taipei 106, Taiwan

Editor:

Abstract

Truncated Newton method is one of the most effective optimization methods for large-scale linear classification. The main computational task at each Newton iteration is to approximately solve a quadratic sub-problem by an iterative procedure such as the conjugate gradient (CG) method. It is known that CG has slow convergence if the sub-problem is ill-conditioned. Preconditioned CG (PCG) methods have been used to improve the convergence of the CG method, but it is difficult to find a preconditioner that performs well in most situations. Further, because Hessian-free optimization techniques are incorporated for handling large data, many existing preconditioners are not directly applicable. In this work, we detailedly study some preconditioners that have been considered in past works for linear classification. We show that these preconditioners may not help to improve the training speed in some cases. After some investigation, we propose simple and effective techniques to make the PCG method more robust in a truncated Newton framework. The idea is to avoid the situation when a preconditioner leads to a much worse condition number than when it is not applied. We provide theoretical justification. Through carefully designed experiments, we demonstrate that our method can effectively reduce the training time for large-scale problems.

1. Introduction

In linear classification, logistic regression and linear SVM are two commonly used models. The model parameters, denoted as $\boldsymbol{w} \in \mathbb{R}^n$, are obtained by solving an unconstrained optimization problem

$$\min_{\boldsymbol{w}} f(\boldsymbol{w}).$$

Truncated Newton method is one of the most effective optimization methods for large-scale linear classification. The core computational task at the kth Newton iteration is to approximately solve a sub-problem that is related to the following linear system

$$\nabla^2 f(\boldsymbol{w}_k)\boldsymbol{s}_k = -\nabla f(\boldsymbol{w}_k), \tag{1}$$

where $\nabla f(\boldsymbol{w}_k)$ and $\nabla^2 f(\boldsymbol{w}_k)$ are gradient and Hessian, respectively. For large-scale problems, the Hessian matrix is too large to be stored. Past works such as Keerthi and DeCoste (2005); Lin et al. (2008) have addressed this difficulty by showing that the special structure in linear classification allows us to solve (1) by a conjugate gradient (CG) procedure without forming the whole Hessian matrix. However, even with such a Hessian-free approach, the training of large-scale data sets is still a time-consuming process because of a possibly large number of CG steps.

It is well know that for ill-conditioned matrices, the convergence of CG methods may be slow. To reduce the number of CG steps, a well-known technique is the preconditioned conjugate gradient (PCG) method (e.g., Concus et al. (1976); Nash (1985)). The idea is to apply a preconditioner matrix to possibly improve the condition of the linear system. Unfortunately, designing a suitable preconditioner is never an easy task. Because we need extra efforts to find and use the preconditioners, the cost per CG step becomes higher. Therefore, a smaller number of CG steps may not lead to shorter running time. Further, a PCG method is not guaranteed to reduce the number of CG steps.

For linear classification, some past works such as Lin et al. (2008); Zhang and Lin (2015); Ma and Takáč (2016) have applied PCG in the truncated Newton framework. However, whether PCG is useful remains a question to be studied. One challenge is that training a linear classifier is different from solving a single linear system in the following aspects.

1. Because a sequence of CG procedures are conducted, reducing the number of CG steps at one Newton iteration may not lead to the overall improvement. For example, suppose at the first iteration PCG takes fewer steps than CG but goes to a bad point for the overall optimization process. Then subsequently they follow two different paths to the optimal point. The total number of CG steps by using PCG may not be less.
2. The effectiveness of PCG may depend on when the optimization procedure is terminated. The work Lin et al. (2008) concludes that a diagonal preconditioner is not consistently better than the ordinary CG. Later Chin et al. (2016) pointed out that the conclusion in Lin et al. (2008) is based on a strict stopping condition in solving the optimization problem. If the optimization procedure is terminated earlier, a situation suitable for machine learning applications, PCG is generally useful. Therefore, careful designs are needed in evaluating the effectiveness of PCG.
3. Because the Hessian matrix is too large to be stored and we consider a Hessian-free approach, most existing preconditioners can not be directly used. This fact makes the application of PCG for linear classification very difficult.

In this work we show that existing preconditioners may not help to improve the training speed in some cases. After some investigation, we propose a simple and effective technique to make the PCG method more robust in a truncated Newton framework. The idea is to avoid the situation when a preconditioner leads to a much worse condition number than when it is not applied. The proposed technique can be applied in single-core, multi-core, or distributed settings because the running time is always proportional to the total number of CG steps.

This work is organized as follows. In Section 2, we introduce Newton methods for large-scale linear classification and discuss how PCG methods are applied into a trust-region Newton framework. In Section 3, we discuss several existing preconditioners and their possible weakness. We then propose a strategy in Section 4 to have a more robust preconditioner in the truncated Newton framework. Theoretical properties are investigated. In Section 5, we conduct thorough experiments and comparisons to show the effectiveness of our proposed setting. Proofs and additional experimental results are available in supplementary materials. The proposed method has been included in a package for large-scale linear classification.

2. Conjugate Gradient in Newton Methods for Linear Classification

In this section we review how CG and PCG are applied in a Newton method for linear classification.

2.1 Truncated Newton Methods

Consider training data $(y_i, \boldsymbol{x}_i)$, $i = 1, \ldots, l$, where $y_i = \pm 1$ is the label and $\boldsymbol{x}_i \in \mathbb{R}^n$ is a feature vector. In linear classification, the model vector $\boldsymbol{w} \in \mathbb{R}^n$ is obtained by solving

$$\min_{\boldsymbol{w}} f(\boldsymbol{w}), \text{where } f(\boldsymbol{w}) \equiv \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l} \xi(y_i\boldsymbol{w}^T\boldsymbol{x}_i).$$

In $f(\boldsymbol{w})$, $\boldsymbol{w}^T\boldsymbol{w}/2$ is the L2-regularization term, $\xi(y_i\boldsymbol{w}^T\boldsymbol{x}_i)$ is the loss function and $C > 0$ is a parameter to balance the two terms. Here we consider logistic and L2 losses

$$\xi_{\text{LR}}(y\boldsymbol{w}^T\boldsymbol{x}) = \log\big(1 + \exp\big(-y\boldsymbol{w}^T\boldsymbol{x}\big)\big), \tag{2}$$

$$\xi_{\text{L2}}(y\boldsymbol{w}^T\boldsymbol{x}) = (\max(0, 1 - y\boldsymbol{w}^T\boldsymbol{x}))^2. \tag{3}$$

Truncated Newton methods are one of the main approaches for large-scale linear classification. It iteratively finds a direction $\boldsymbol{s}_k$ that minimizes the quadratic approximation of

$$f(\boldsymbol{w}_k + \boldsymbol{s}) - f(\boldsymbol{w}_k) \approx q_k(\boldsymbol{s}) \equiv \nabla f(\boldsymbol{w}_k)^T\boldsymbol{s} + \frac{1}{2}\boldsymbol{s}^T\nabla^2 f(\boldsymbol{w}_k)\boldsymbol{s}, \tag{4}$$

where $\boldsymbol{w}_k$ is the current iterate, $\nabla f(\boldsymbol{w}_k)$ and $\nabla^2 f(\boldsymbol{w}_k)$ are the gradient and the Hessian, respectively. Because L2 Loss is not twice differentiable, we can consider the generalized Hessian matrix Mangasarian (2002). The direction $\boldsymbol{s}_k$ from minimizing (4) can be obtained by solving the linear system

$$\nabla^2 f(\boldsymbol{w}_k)\boldsymbol{s} = -\nabla f(\boldsymbol{w}_k). \tag{5}$$

Exactly solving (5) for large-scale problems is expensive and furthermore $\nabla^2 f(\boldsymbol{w}_k)$ may be too large to be stored. Currently, conjugate gradient (CG) methods are commonly used to approximately solve (5) for obtaining a truncated Newton direction. A CG procedure involves a sequence of Hessian-vector products. Past developments (e.g., Keerthi and DeCoste (2005); Lin et al. (2008)) have shown that the special structure of the Hessian in (6) allows us to conduct Hessian-vector products without explicitly forming the matrix. Specifically, from $f(\boldsymbol{w})$ we have

$$H_k = \nabla^2 f(\boldsymbol{w}_k) = I + CX^T DX, \tag{6}$$

where D is a diagonal matrix with $D_{ii} = \xi''(y_i \boldsymbol{w}_k^T \boldsymbol{x}_i)$, I is the identity matrix, and $X = [\boldsymbol{x}_1, \dots, \boldsymbol{x}_l]^T$ is the data matrix. The Hessian-vector product can be conducted by

$$\nabla^2 f(\boldsymbol{w})\boldsymbol{s} = (I + CX^T DX)\boldsymbol{s} = \boldsymbol{s} + CX^T(D(X\boldsymbol{s})). \tag{7}$$

The CG procedure stops after, for example, the following relative error in solving (5) is small.

$$\left\|\nabla^2 f(\boldsymbol{w}_k)\boldsymbol{s} + \nabla f(\boldsymbol{w}_k)\right\| \leq \varepsilon_{\text{CG}} \|\nabla f(\boldsymbol{w}_k)\|, \tag{8}$$

where ε_{CG} is a small positive value.[1]

To ensure the convergence, after finding a direction $\boldsymbol{s}_k$, we should adjust the step size along $\boldsymbol{s}_k$ (line search strategy) or decide if $\boldsymbol{s}_k$ should be accepted (trust region method). In this work we focus on the trust region approach because first, it is used in the package LIBLINEAR Fan et al. (2008), the platform considered in this study, and second, for logistic regression, in some situations the trust region approach is superior Hsia et al. (2017).

2.2 Trust-region Newton Method

A trust region method indirectly adjusts the step size by finding a direction $\boldsymbol{s}_k$ within a trust region. The direction is taken if it results in a sufficient function-value reduction. The size of the trust region is then adjusted. Here we mainly follow the description in Hsia et al. (2017).

Given a trust region with size Δ_k at the kth iteration, we compute the approximate Newton direction $\boldsymbol{s}_k$ by solving the following trust-region sub-problem:

$$\min_{\boldsymbol{s}} \quad q_k(\boldsymbol{s}) \quad \text{subject to} \quad \|\boldsymbol{s}\| \leq \Delta_k, \tag{9}$$

where $q_k(\boldsymbol{s})$ is defined in (4). Then, we check the ratio between the real and the predicted reduction of $f(\boldsymbol{w})$:

$$\rho_k = \frac{f(\boldsymbol{w}_k + \boldsymbol{s}_k) - f(\boldsymbol{w}_k)}{q_k(\boldsymbol{s}_k)}. \tag{10}$$

The iterate $\boldsymbol{w}$ is updated only if the ratio is large enough:

$$\boldsymbol{w}_{k+1} = \begin{cases} \boldsymbol{w}_k + \boldsymbol{s}_k, & \text{if } \rho > \eta_0, \\ \boldsymbol{w}_k, & \text{if } \rho \leq \eta_0, \end{cases} \tag{11}$$

where $\eta_0 > 0$ is a pre-defined constant. Then, we adjust Δ_k by comparing the actual and the predicted function-value reduction. More details can be found in, for example, Lin et al. (2008). We summarize a trust region Newton method in Algorithm 1.

1. In experiments, we use $\epsilon_{\text{CG}} = 0.1$ by following the setting in the software LIBLINEAR Lin et al. (2008).

Algorithm 1 A CG-based trust region Newton method

Given $\boldsymbol{w}_0$.
for $k = 0, 1, 2, \ldots$ **do**
  Approximately solve trust-region sub-problem (9) by the CG method to obtain a direction $\boldsymbol{s}_k$.
  Compute ρ_k via (10).
  Update $\boldsymbol{w}_k$ to $\boldsymbol{w}^{k+1}$ according to (11).
  Update Δ_{k+1} (details not discussed).

Algorithm 2 CG for the trust region sub-problem (9)

Given $\varepsilon_{\text{CG}} < 1, \Delta_k > 0$, let $\bar{\boldsymbol{s}} = \boldsymbol{0}, \boldsymbol{r} = \boldsymbol{d} = -\nabla f(\boldsymbol{w}_k)$
$\|\boldsymbol{r}\|^2 = \boldsymbol{r}^T\boldsymbol{r}$
while True **do**
  if $\|\boldsymbol{r}\| < \varepsilon_{\text{CG}}\|\nabla f(\boldsymbol{w}_k)\|$ **then**
    return $\boldsymbol{s}_k = \bar{\boldsymbol{s}}$
  $\boldsymbol{v} \leftarrow \nabla^2 f(\boldsymbol{w}_k)\boldsymbol{d}, \alpha \leftarrow \|\boldsymbol{r}\|^2/(\boldsymbol{d}^T\boldsymbol{v}), \bar{\boldsymbol{s}} \leftarrow \bar{\boldsymbol{s}} + \alpha\boldsymbol{d}$
  if $\|\bar{\boldsymbol{s}}\| \geq \Delta_k$ **then**
    $\bar{\boldsymbol{s}} \leftarrow \bar{\boldsymbol{s}} - \alpha\boldsymbol{d}$
    compute τ such that $\|\bar{\boldsymbol{s}} + \tau\boldsymbol{d}\| = \Delta_k$
    return $\boldsymbol{s}_k = \bar{\boldsymbol{s}} + \tau\boldsymbol{d}$
  $\boldsymbol{r} \leftarrow \boldsymbol{r} - \alpha\boldsymbol{v}, \|\boldsymbol{r}\|^2_{\text{new}} \leftarrow \boldsymbol{r}^T\boldsymbol{r}$
  $\beta \leftarrow \|\boldsymbol{r}\|^2_{\text{new}}/\|\boldsymbol{r}\|^2, \boldsymbol{d} \leftarrow \boldsymbol{r} + \beta\boldsymbol{d}$
  $\|\boldsymbol{r}\|^2 \leftarrow \|\boldsymbol{r}\|^2_{\text{new}}$

Solving the sub-problem (9) is similar to solving the linear system (5) though a constraint $\|\boldsymbol{s}\| \leq \Delta_k$ must be satisfied. A classic approach in Steihaug (1983) applies CG with the initial $\boldsymbol{s} = \boldsymbol{0}$ and shows that $\|\boldsymbol{s}\|$ is monotonically increasing. Then CG stops after either (8) is satisfied or an $\boldsymbol{s}$ on the boundary of the trust region is obtained. The procedure to solve the trust-region sub-problem is given in Algorithm 2.

We explain that the computational cost per iteration of truncated Newton methods is roughly

$$\mathcal{O}(nl) \times (\text{\# CG steps}). \tag{12}$$

From (7), $\mathcal{O}(nl)$ is the cost per CG step. Besides CG, function and gradient evaluation costs about the same as one CG step. Because in general each Newton iteration requires several CG steps, we omit function/gradient evaluation in (12) and hence the total cost is proportional to the total number of CG steps. If X is a sparse matrix with #nnz non-zero entries, then the $\mathcal{O}(nl)$ term can be replaced by $\mathcal{O}(\#\text{nnz})$.

2.3 PCG for Trust-region Sub-problem

From (12), reducing the number of CG steps can speed up the Newton method. The preconditioning technique Concus et al. (1976) has been widely used to possibly improve the condition of linear systems and reduce the number of CG steps. PCG considers a preconditioner

$$M = EE^T \approx \nabla^2 f(\boldsymbol{w}_k)$$

and transforms the original linear system (5) to

$$E^{-1}\nabla^2 f(\boldsymbol{w}_k)E^{-T}\hat{\boldsymbol{s}} = -E^{-1}\nabla f(\boldsymbol{w}_k). \tag{13}$$

If the approximation is good, the condition number of $E^{-1}\nabla^2 f(\boldsymbol{w}_k)E^{-T}$ is close to 1 and less CG steps are needed. After obtaining $\hat{\boldsymbol{s}}_k$, we can get the original solution by $\boldsymbol{s}_k = E^{-T}\hat{\boldsymbol{s}}_k$. More discussion about the condition number and the convergence of CG is in supplement.

To apply PCG for the trust-region sub-problem, we follow Steihaug (1983) to modify the sub-problem (9) to

$$\begin{aligned} \min_{\hat{\boldsymbol{s}}} \quad & \frac{1}{2}\hat{\boldsymbol{s}}^T(E^{-1}\nabla^2 f(\boldsymbol{w}_k)E^{-T})\hat{\boldsymbol{s}} + (E^{-1}\nabla f(\boldsymbol{w}_k))^T\hat{\boldsymbol{s}} \\ \text{subject to} \quad & \|\hat{\boldsymbol{s}}\| \leq \Delta_k. \end{aligned} \tag{14}$$

Then the same procedure in Algorithm 2 can be applied to solve the new sub-problem. See details in Algorithm I in supplementary materials. By comparing Algorithms 2 and I, clearly the extra cost of PCG is for calculating the product between E^{-1} (or E^{-T}) and a vector; see

$$E^{-1}(\nabla^2 f(\boldsymbol{w}_k)(E^{-T}\hat{\boldsymbol{d}}))$$

in Algorithm I. In some situations the factorization of $M = EE^T$ is not practically viable, but it is know Golub and Van Loan (1996) that PCG can be performed without a factorization of M. We show the procedure in Algorithm II of supplementary materials.

3. Existing Preconditioners

We discuss several existing preconditioners which have been applied to linear classification.

3.1 Diagonal Preconditioner

It is well known that extracting all diagonal elements in the Hessian can form a simple preconditioner.

$$M = \text{diag}(H_k), \text{where } M_{ij} = \begin{cases} (H_k)_{ij}, & \text{if } i = j, \\ 0, & \text{otherwise.} \end{cases}$$

From (6), $\qquad (H_k)_{jj} = 1 + C\sum_i D_{ii}X_{ij}^2,$

so constructing the preconditioner costs $\mathcal{O}(nl)$. At each CG step, the cost of computing $M^{-1}\boldsymbol{r}$ is $\mathcal{O}(n)$. Thus, the extra cost by using the diagonal preconditioner is

$$\mathcal{O}(n) \times (\#\text{ CG steps}) + \mathcal{O}(nl). \tag{15}$$

In compared with (12), the cost of preconditioning at each Newton iteration is insignificant.

For linear classification, Lin et al. (2008) have applied the diagonal preconditioner, but conclude that it is not consistently better. However, Chin et al. (2016) point out that the observation was based on comparing standard CG and diagonal PCG by strictly solving the optimization problem. Because in general a less accurate optimization solution is enough for machine learning applications, Chin et al. (2016) find that diagonal preconditioning is practically useful. We will conduct thorough evaluation in Section 5.1.

3.2 Subsampled Hessian as Preconditioner

In Section 2.3, we have shown that a good preconditioner M should satisfy $M \approx \nabla^2 f(\boldsymbol{w}_k)$. In linear classification, the object function involves the sum of training losses. If we randomly select $\bar{l}$ instance-label pairs $(\bar{y}_i, \bar{\boldsymbol{x}}_i)$ and construct a subsampled Hessian Byrd et al. (2011)

$$\bar{H}_k = I + C\frac{l}{\bar{l}}\bar{X}^T\bar{D}\bar{X},$$

where $\bar{X} = [\bar{\boldsymbol{x}}_1^T, \ldots, \bar{\boldsymbol{x}}_{\bar{l}}^T] \in \mathbb{R}^{\bar{l}\times n}$ and $\bar{D} \in \mathbb{R}^{\bar{l}\times\bar{l}}$ is a diagonal matrix with $\bar{D}_{ii} = \xi''(\bar{y}_i\boldsymbol{w}_k^T\bar{\boldsymbol{x}}_i)$, then $\bar{H}_k$ is an unbiased estimator for $\nabla^2 f(\boldsymbol{w}_k)$. Therefore, $M = \bar{H}_k$ can be considered as a reasonable preconditioner.

However, $\bar{H}_k \in \mathbb{R}^{n\times n}$ has the same size as H_k so it may be too large to be stored. To get $\bar{H}_k^{-1}\boldsymbol{r}$ in PCG, Ma and Takáč (2016) apply the Woodbury formula Woodbury (1950) to obtain

$$\bar{H}_k^{-1} = I_n - \bar{X}^T G(I_{\bar{l}} + G\bar{X}\bar{X}^T G)^{-1} G\bar{X}, \tag{16}$$

where $G = (\frac{lC}{\bar{l}}D)^{\frac{1}{2}}$. Detailed derivations are in the supplementary materials. If the chosen $\bar{l}$ satisfies $\bar{l} \ll l$, then

$$(I_{\bar{l}} + G\bar{X}\bar{X}^T G)^{-1} \in \mathbb{R}^{\bar{l}\times\bar{l}}$$

can be calculated in $\mathcal{O}(n\bar{l}^2 + \bar{l}^3)$ time and is small enough to be stored. Note that $\bar{l}^3$ is for calculating the inverse. Then in PCG the product between the inverse of the preconditioner and a vector can be done by

$$\bar{H}_k^{-1}\boldsymbol{r} = \boldsymbol{r} - \bar{X}^T(G((I_{\bar{l}} + G\bar{X}\bar{X}^T G)^{-1}(G(\bar{X}\boldsymbol{r}))))$$

in the cost of $\mathcal{O}(n\bar{l}) + \mathcal{O}(\bar{l}^2)$. For large and sparse data, $\bar{l} \ll n$, so at each Newton iteration, the additional cost of using $\bar{H}_k$ as the preconditioner is

$$\mathcal{O}(n\bar{l}) \times (\#\text{ CG steps}) + \mathcal{O}(n\bar{l}^2). \tag{17}$$

A comparison with (12) shows that we can afford this cost if $\bar{l} \ll l$. Further, from (15) and (17), using a sub-sampled Hessian as the preconditioner is in general more expensive than diagonal preconditioning. We will discuss the selection of $\bar{l}$ in Section 5.2

4. Our Proposed Method

It is hard to find a preconditioner that works well on all cases. We give a simple example showing that the diagonal preconditioner may not reduce the condition number. If

$$H_k = I + \begin{bmatrix} 2 & 2 & 2 \\ 2 & 3 & 3 \\ 2 & 3 & 4 \end{bmatrix} \text{ and } M = EE^T = \text{diag}(H_k),$$

then $\qquad \kappa(E^{-1}H_kE^{-T}) = 6.781 > \kappa(H_k) = 6.723.$

In a Newton method, each iteration involves a sub-problem. A preconditioner may be useful for some sub-problems, but not for others. Thus it is difficult to ensure the shorter overall training time of using PCG. We find that if some Newton iterations need many CG steps because of inappropriate preconditioning, the savings by PCG in subsequent Newton iterations may not be able to recover the loss. Thus we check if a more robust setting can be adopted. Assume

$$M = EE^T \approx H_k$$

is any preconditioner considered for the current sub-problem. We hope to derive a new preconditioner $\bar{M} = \bar{E}\bar{E}^T$ that satisfies

$$\kappa(\bar{E}^{-1}H_k\bar{E}^{-T}) \leq \max\{\kappa(H_k), \kappa(E^{-1}H_kE^{-T})\}. \tag{18}$$

This property avoids the situation when $E^{-1}H_kE^{-T}$ has a larger condition number. Note that we assume the factorization EE^T and $\bar{E}\bar{E}^T$ only for the theoretical property (18). In practice we can just use M or $\bar{M}$ as indicated in Section 2.3.

The next issue is how to achieve the inequality (18). We offer two approaches.

4.1 Running CG and PCG in Parallel

If in an ideal situation we can run CG and PCG in parallel, then we can conjecture that the one needing less steps has a smaller condition number. Therefore, the following rule can choose between CG and PCG at each Newton iteration.

$$\bar{M} = \begin{cases} I, & \text{if CG uses less steps,} \\ M, & \text{if PCG uses less steps.} \end{cases}$$

In general we consider preconditioners that do not incur much extra cost, so this setting is similar to checking which one finishes first. Our proposed strategy can be easily implemented in a multi-core or a distributed environment.

4.2 Weighted Average of the Preconditioner and the Identity Matrix

If running CG and PCG in parallel is not possible, we prove that $\bar{M}$ can be found by a weighted average of M and an identity matrix (details are in supplementary materials).

Theorem 1 *For any M and any symmetric positive definite H_k, the following matrix satisfies* (18) *for all* $0 \leq \alpha \leq 1$.

$$\bar{M} = \alpha M + (1-\alpha)I.$$

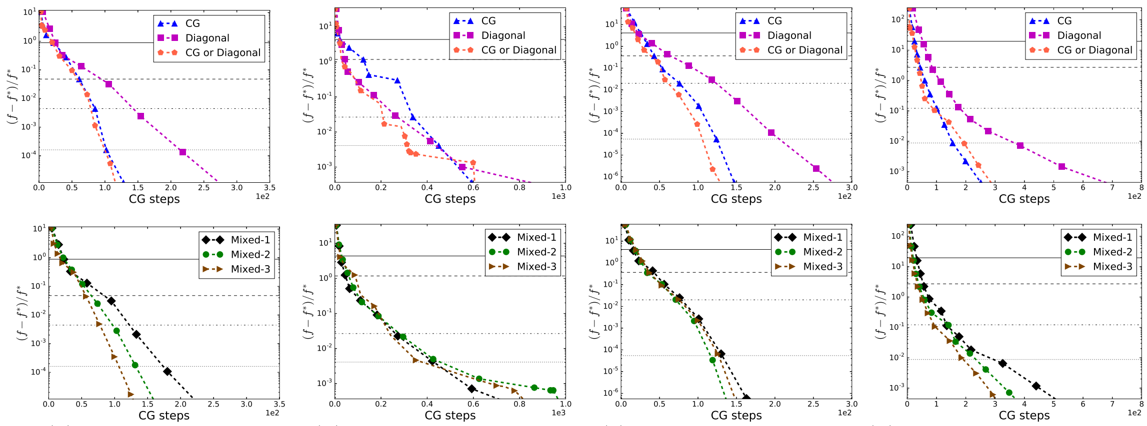
Note that $\alpha = 0$ and 1 respectively indicate that CG (no preconditioning) and PCG (with preconditioner M) are used. If M is the diagonal preconditioner in Section 3.1, then

$$\bar{M} = \alpha \times \mathrm{diag}(H_k) + (1-\alpha) \times I. \tag{19}$$

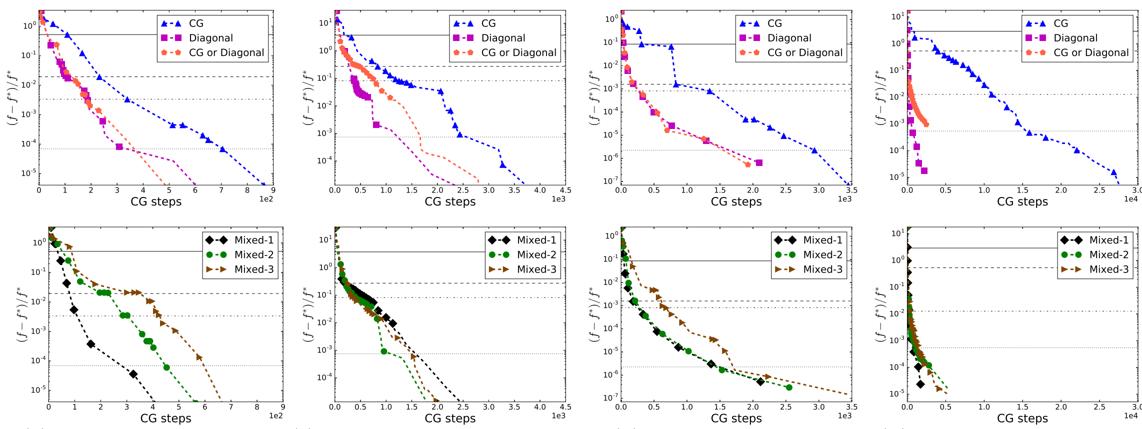
In Section 5.1 we will investigate the effectiveness of using (19) and the selection of α.

5. Experiments

In Section 5.1, we compare the diagonal preconditioner and the proposed methods. For the preconditioner of using the subsampled Hessian, we check the performance in Section 5.2. Finally, an overall comparison is in Section 5.3.



(a) `url`, $C = C_{\text{Best}}$ (b) `url`, $C = 100C_{\text{Best}}$ (c) `news20`, $C = C_{\text{Best}}$ (d) `news20`, $C = 100C_{\text{Best}}$



(e) `yahookr`, $C = C_{\text{Best}}$ (f) `yahookr`, $C = 100C_{\text{Best}}$ (g) `kddb`, $C = C_{\text{Best}}$ (h) `kddb`, $C = 100C_{\text{Best}}$

Figure 1: Each sub-figure is separated to two parts because of too many curves. The upper part shows the convergence of the trust region Newton method without a preconditioner, with a diagonal preconditioner and by using techniques in Section 4.1. The lower one shows the convergence of using the preconditioner in (19) with different α values. LR loss is used. We show relative difference to the optimal function value (log scaled) versus the total CG steps. Horizontal lines show that LIBLINEAR's stopping condition with tolerances $10^{-1}, 10^{-2}$(default)$, 10^{-3}$ and 10^{-4} is reached; Such information indicates when the training algorithm should stop; see more explanation in Section VII of supplementary materials.

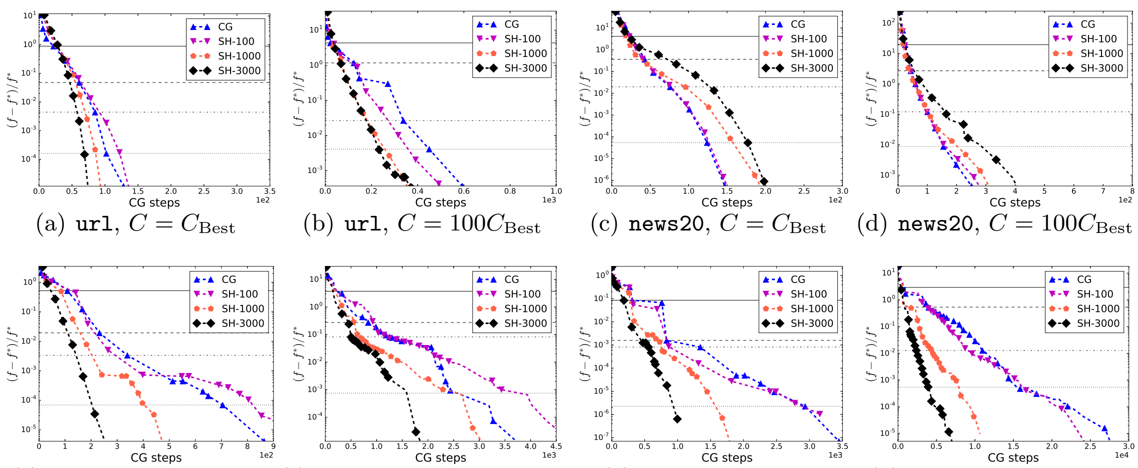
In experiments, if the same type of preconditioners are compared, we check the convergence of the truncated Newton method by showing the cumulative number of CG steps versus the relative difference to the optimal function value

$$\frac{f(\boldsymbol{w}_k) - f(\boldsymbol{w}^*)}{f(\boldsymbol{w}^*)}, \tag{20}$$

where $\boldsymbol{w}^*$ is an approximate optimal solution by running the Newton method with many iterations. From (12), this setting gives the same information as that of checking the running time. However, for comparing different types of preconditioners in Section 5.3, we

Table 1: Data statistics. C_{Best} is the regularization parameter selected by cross validation.

Data sets	#instances	#features	$\log_2(C_{\text{Best}})$	
			LR	L2
`news20`	19,996	1,355,191	9	3
`yahoojp`	176,203	832,026	3	-1
`yahookr`	460,554	3,052,939	6	1
`url`	2,396,130	3,231,962	-7	-10
`kdda`	8,407,752	20,216,831	-3	-5
`kddb`	19,264,097	29,890,095	-1	-4
`criteo`	45,840,617	1,000,000	-15	-12
`kdd12`	149,639,105	54,686,452	-4	-11



(a) `url`, $C = C_{\text{Best}}$ (b) `url`, $C = 100C_{\text{Best}}$ (c) `news20`, $C = C_{\text{Best}}$ (d) `news20`, $C = 100C_{\text{Best}}$

(e) `yahookr`, $C = C_{\text{Best}}$ (f) `yahookr`, $C = 100C_{\text{Best}}$ (g) `kddb`, $C = C_{\text{Best}}$ (h) `kddb`, $C = 100C_{\text{Best}}$

Figure 2: Convergence of using different $\bar{l}$ values in the sub-sampled Hessian preconditioner. LR loss is used. Other settings are the same as those in Figure 1.

give timing results. For the regularization parameter C, we consider

$$C = C_{\text{Best}} \times \{0.01, 0.1, 1, 10, 100\}, \tag{21}$$

where C_{Best} for each data set is the value leading to the best cross validation accuracy.

We consider binary classification data sets listed in Table 1. All data sets except `yahoojp` and `yahookr` can be downloaded from LIBSVM Data Sets (2007). We extend the trust region Newton implementation in LIBLINEAR (version 2.11) to incorporate various preconditioners. We present results of using the logistic loss, while leave results of the l2 loss in the supplementary materials.

5.1 Diagonal Preconditioner and the Proposed Method in Section 4

We compare the following settings.

- `CG`: LIBLINEAR version 2.11, a trust-region method without preconditioning.
- `Diagonal`: the diagonal preconditioner in Section 3.1.

- `CG or Diagonal`: the technique in Section 4.1 to run CG and diagonal PCG in parallel.[2]

In Figure 1 (upper part of each sub-figure), we present results of using $C = C_{\text{Best}} \times \{1, 100\}$, while leave full results under all C values in (21) in supplementary materials.

From Figure 1, using the diagonal preconditioner significantly reduces the total number of CG steps for `yahookr` and `kddb`. However, for `news20` and `url`, it is not useful at a later stage. This result shows that the diagonal preconditioner may not be robust.

For the approach `CG or Diagonal`, it is never the worst among the three compared methods. In fact, it is either the best or close to the best. Therefore, the technique in Section 4.1 effectively improves the robustness of the diagonal preconditioner. For the other setting in Section 4 by combining a preconditioner and the identity matrix, we check the performance by considering the following settings.

- `Mixed-1`: the preconditioner in (19) with $\alpha = 10^{-1}$
- `Mixed-2`: the preconditioner in (19) with $\alpha = 10^{-2}$
- `Mixed-3`: the preconditioner in (19) with $\alpha = 10^{-3}$

Results in Figure 1 (lower part of each sub-figure) show that the proposed setting leads to a preconditioner more robust than the diagonal preconditioner.

For the selection of α, Figures 1(e) and 1(g) show that $\alpha = 10^{-3}$ may cause the resulting preconditioner to be close to the setting without preconditioning. Otherwise, the performance is not too sensitive to the change of α.

5.2 Subsampled Hessian as the Preconditioner

We use different sample size $\bar{l}$ in the subsampled Hessian preconditioner described in Section 3.2 and compare the following settings.

- `CG`: LIBLINEAR 2.11; no preconditioning.
- `SH-100`: method in Section 3.2 with $\bar{l} = 100$.
- `SH-1000`: method in Section 3.2 with $\bar{l} = 1,000$.
- `SH-3000`: method in Section 3.2 with $\bar{l} = 3,000$.

From (17), the cost of using subsampled Hessian is not negligible, so we will present timing results in Section 5.3.

From Figure 2, a larger $\bar{l}$ generally leads to better final convergence. However, in some situations (e.g., Figures 2(c) and 2(d)), a larger $\bar{l}$ is not useful. From a detailed investigation in supplementary materials, we even find that the performance is sensitive to the random seeds in constructing the sub-sampled Hessian. Thus in some cases the current $\bar{l}$ is not large enough to give a good approximation of $\nabla^2 f(\boldsymbol{w}_k)$. Our observation is slightly different from that in Ma and Takáč (2016), which uses only $\bar{l} = 100$. It is unclear why different results are reported, but this situation seems to indicate that selecting a suitable $\bar{l}$ is not an easy task.

5.3 Comparison of Different Preconditioners

We compare the following settings. Because these preconditioners incur different extra cost at each CG step, besides the total CG steps, we present the training time.

2. Because of checking the total number of CG steps, we can easily use a single machine to simulate the parallel setting.

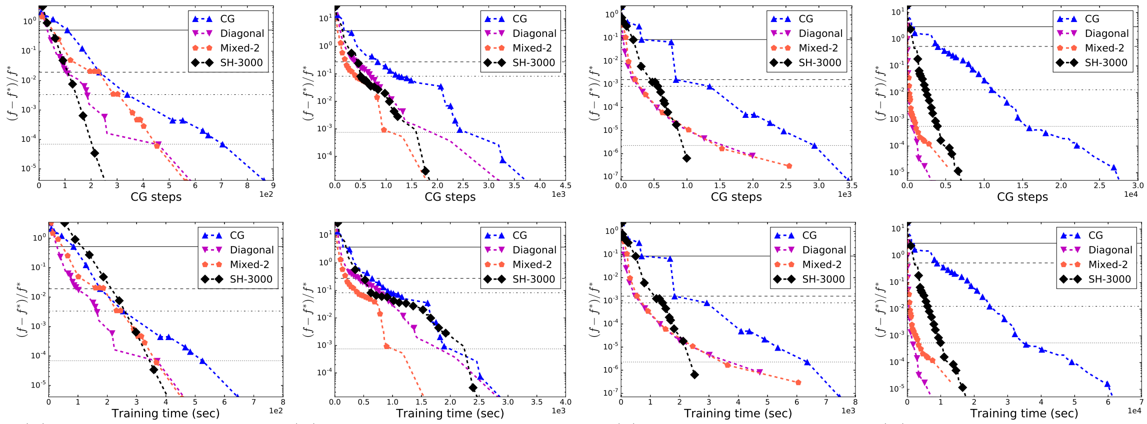
- `CG`: LIBLINEAR 2.11; no preconditioning.
- `Diagonal`: the diagonal preconditioner in Section 3.1.
- `Mixed-2`: the preconditioner in (19) with $\alpha = 10^{-2}$
- `SH-3000`: method in Section 3.2 with $\bar{l} = 3,000$.

Results in Figure 3 lead to the following observations.

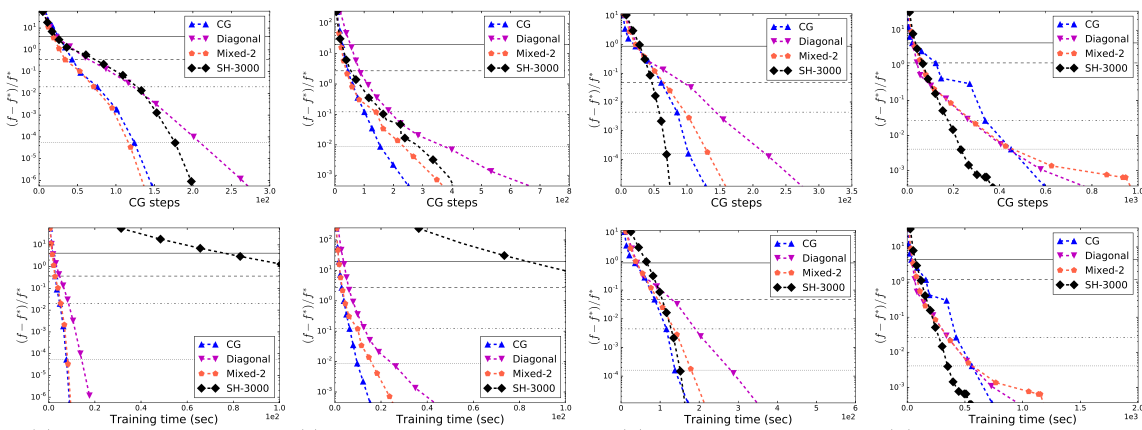
1. In some situations, the cost of using a preconditioner is not negligible. For example, in training set `news20`, Figures 3(e) and 3(f) show that `SH-3000` is competitive in terms of CG steps, but is the slowest in terms of time.
2. Overall `Mixed-2` is the best approach. It is more robust than `Diagonal`, and is often much faster than `CG` and `SH-3000`. For example, under LIBLINEAR's default condition (second horizontal line), `Mixed-2` is 3 and 5 times faster than `CG` respectively for `kddb` and `kdd12`.

6. Conclusions

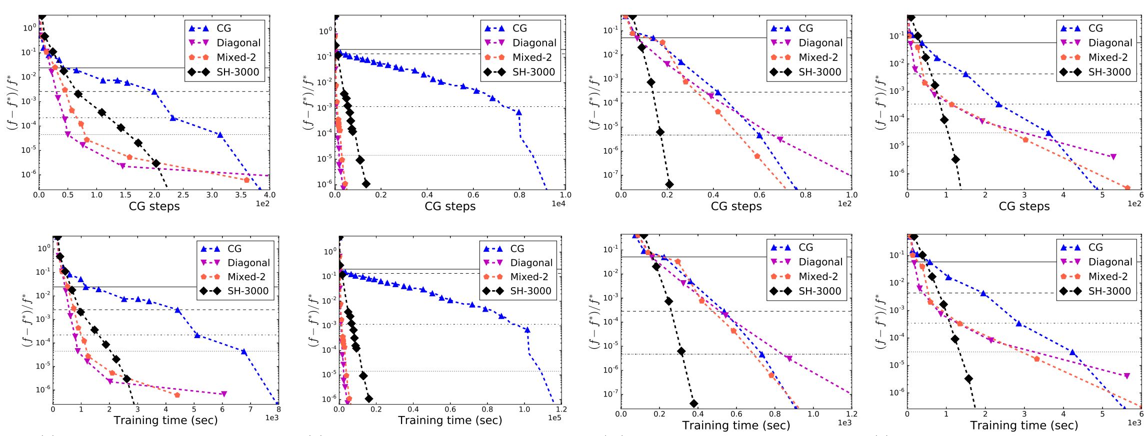
In this work we show that applying preconditioners in Newton methods for linear classification is not an easy task. Improvements made at one Newton iteration may not lead to better overall convergence. We propose using a reliable but maybe less aggressive preconditioner at each iteration. The idea is to ensure that at each Newton iteration the preconditioned CG procedure is at least as good as that without preconditioning. Experiments confirm that the proposed method leads to faster overall convergence.



(a) `yahookr`, $C = C_{\text{Best}}$ (b) `yahookr`, $C = 100C_{\text{Best}}$ (c) `kddb`, $C = C_{\text{Best}}$ (d) `kddb`, $C = 100C_{\text{Best}}$



(e) `news20`, $C = C_{\text{Best}}$ (f) `news20`, $C = 100C_{\text{Best}}$ (g) `url`, $C = C_{\text{Best}}$ (h) `url`, $C = 100C_{\text{Best}}$



(i) `kdd12`, $C = C_{\text{Best}}$ (j) `kdd12`, $C = 100C_{\text{Best}}$ (k) `criteo`, $C = C_{\text{Best}}$ (l) `criteo`, $C = 100C_{\text{Best}}$

Figure 3: Convergence of using different preconditioners. For the same data set and under the same C value, the x-axis of the upper figure is the cumulative number of CG steps and the x-axis of the lower one is the running time. We align curves of the approach `CG` in upper and lower figures for an easy comparison. Other settings are the same as those in Figure 1.

References

Richard H. Byrd, Gillian M. Chin, Will Neveitt, and Jorge Nocedal. On the use of stochastic Hessian information in optimization methods for machine learning. *SIAM Journal on Optimization*, 21(3):977–995, 2011.

Wei-Sheng Chin, Bo-Wen Yuan, Meng-Yuan Yang, and Chih-Jen Lin. A note on diagonal preconditioner in large-scale logistic regression. Technical report, Department of Computer Science and Information Engineering, National Taiwan University, 2016. URL `https://www.csie.ntu.edu.tw/~cjlin/papers/logistic/pcgnote.pdf`.

Paul Concus, Gene H Golub, and Dianne P OĹeary. A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations. Technical report, 1976.

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`.

G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, third edition, 1996.

Chih-Yang Hsia, Ya Zhu, and Chih-Jen Lin. A study on trust region update rules in Newton methods for large-scale linear classification. In *Proceedings of the Asian Conference on Machine Learning (ACML)*, 2017. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/newtron/newtron.pdf`.

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.

LIBSVM Data Sets, 2007. `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets`.

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`.

Chenxin Ma and Martin Takáč. Distributed inexact damped Newton method: data partitioning and load-balancing. *arXiv preprint arXiv:1603.05191*, 2016.

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

Stephen G Nash. Preconditioning of truncated-Newton methods. *SIAM Journal on Scientific and Statistical Computing*, 6(3):599–616, 1985.

Trond Steihaug. The conjugate gradient method and trust regions in large scale optimization. *SIAM Journal on Numerical Analysis*, 20:626–637, 1983.

Max A Woodbury. Inverting modified matrices. Technical Report 106, Statistical Research Group, Princeton University, 1950.

Yuchen Zhang and Xiao Lin. DiSCO: Distributed optimization for self-concordant empirical loss. In *Proceedings of the Thirty Second International Conference on Machine Learning (ICML)*, pages 362–370, 2015.