Proceedings of Machine Learning Research 77:33–48, 2017 ACML 2017

A Study on Trust Region Update Rules in Newton Methods for Large-scale Linear Classification

Chih-Yang Hsia R04922021@NTU.EDU.TW
Dept. of Computer Science, National Taiwan University, Taipei, Taiwan

Ya Zhu YZ3768@NYU.EDU
Computer Science Department, New York University, New York, USA

Chih-Jen Lin CJLIN@CSIE.NTU.EDU.TW
*Dept. of Computer Science, National Taiwan University, Taipei, Taiwan**

Editors: Yung-Kyun Noh and Min-Ling Zhang

Abstract

The main task in training a linear classifier is to solve an unconstrained minimization problem. To apply an optimization method typically we iteratively find a good direction and then decide a suitable step size. Past developments of extending optimization methods for large-scale linear classification focus on finding the direction, but little attention has been paid on adjusting the step size. In this work, we explain that inappropriate step-size adjustment may lead to serious slow convergence. Among the two major methods for step-size selection, line search and trust region, we focus on investigating the trust region methods. After presenting some detailed analysis, we develop novel and effective techniques to adjust the trust-region size. Experiments indicate that our new settings significantly outperform existing implementations for large-scale linear classification.

Keywords: large-scale linear classification, Newton method, trust region, line search

1. Introduction

In linear classification, logistic regression and linear SVM are two commonly used models. We can estimate the model parameter $\boldsymbol{w}$ by solving an unconstrained optimization problem

$$\min_{\boldsymbol{w}} f(\boldsymbol{w}). \tag{1}$$

Existing unconstrained minimization methods can be conveniently applied to solve (1) though some tweaks are needed to handle large-scale data. In general, these methods generate a sequence $\{\boldsymbol{w}^k\}_{k=0}^{\infty}$ converging to the optimal solution. At the kth iteration, from the current iterate $\boldsymbol{w}^k$, a descent direction $\boldsymbol{s}^k$ is obtained. Then we decide the step size $\alpha_k > 0$ to get the next iterate $\boldsymbol{w}^{k+1}$:

$$\boldsymbol{w}^{k+1} = \boldsymbol{w}^k + \alpha_k \boldsymbol{s}^k. \tag{2}$$

The two important components of unconstrained minimization, finding a descent direction $\boldsymbol{s}^k$ and deciding the step size α_k, have been well studied in literature. For example, gradient

* This work was partially supported by MOST of Taiwan grant 104-2221-E-002-047-MY3. Part of this paper is the second author's master thesis at National Taiwan University.

descent, quasi-Newton and Newton are common techniques to find $\boldsymbol{s}^k$. For deciding α_k, line search and trust region are two most used methods.

When the training data set is huge, minimizing $f(\boldsymbol{w})$ becomes a hard optimization problem. Fortunately, it is known that $f(\boldsymbol{w})$ of linear classification possesses some special structures, so some modifications of standard optimization methods can make large-scale training possible. Existing works (e.g., **???**) mainly focus on getting the direction $\boldsymbol{s}^k$, but little attention has been paid for finding a suitable step size α_k.

Recently, we observed that occasionally a Newton method (**?**) implemented in the popular package LIBLINEAR (**?**) for linear classification has slow convergence. We suspect that the main reason is because of inappropriate step-size selection. This implementation uses a trust region setting, so in contrast to the update in (2), the step size is indirectly decided by a trust-region constraint. Specifically, $\boldsymbol{s}^k$ is obtained within a trust region with size Δ_k:

$$\|\boldsymbol{s}\| \leq \Delta_k.$$

We accept or reject $\boldsymbol{w}^k + \boldsymbol{s}^k$ depending on whether $f(\boldsymbol{w}^k + \boldsymbol{s}^k)$ leads to a sufficient reduction from $f(\boldsymbol{w}^k)$. Then Δ_k is adjusted according to the function-value reduction. Because this Newton method in LIBLINEAR is widely used, it is important to investigate and fix the slow-convergence issues.

In this work, we explain why in some situations the adjustment of Δ_k in LIBLINEAR is inappropriate. Based on our findings, we propose new rules to update Δ_k. In particular, we incorporate the information of whether the search direction $\boldsymbol{s}$ has reached the trust region boundary or not. While the change of the update rule is very simple, the slow-convergence issues are effectively solved.

This paper is organized as follows. In Section 2, we introduce Newton methods for large-scale linear classification, and detailedly discuss two strategies (line search and trust region) for deciding the step size. Some experiments are presented to illustrate the importance of finding a suitable step size. In Section 3, we investigate why slow convergence may occur and propose novel techniques for adjusting the trust-region size. Section 4 conducts extensive experiments to demonstrate the effectiveness of the proposed methods. Finally, Section 5 concludes this work. The proposed method has been incorporated into the software LIBLINEAR (version 2.11 and after). Supplementary materials are at http://www.csie.ntu.edu.tw/~cjlin/papers/newtron/.

2. Truncated Newton Methods and the Selection of Step Size

Given training data $(y_i, \boldsymbol{x}_i)$, $i = 1, ..., l$, where $y_i = \pm 1$ is the label and $\boldsymbol{x}_i \in \mathbb{R}^n$ is a feature vector, a linear classifier finds a model $\boldsymbol{w} \in \mathbb{R}^n$ by solving the following problem:

$$\min_{\boldsymbol{w}} f(\boldsymbol{w}) \equiv \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l} \xi(y_i\boldsymbol{w}^T\boldsymbol{x}_i), \tag{3}$$

where $\boldsymbol{w}^T\boldsymbol{w}/2$ is the regularization term, $\xi(y_i\boldsymbol{w}^T\boldsymbol{x}_i)$ is the loss function, and $C > 0$ is a regularization parameter. We consider logistic and L2 losses:

$$\xi_{\mathrm{LR}}(y\boldsymbol{w}^T\boldsymbol{x}) = \log(1 + \exp(-y\boldsymbol{w}^T\boldsymbol{x})), \quad \xi_{\mathrm{L2}}(y\boldsymbol{w}^T\boldsymbol{x}) = (\max(0, 1 - y\boldsymbol{w}^T\boldsymbol{x}))^2. \tag{4}$$

From the current iterate $\boldsymbol{w}^k$, Newton Methods obtain a direction $\boldsymbol{s}^k$ by minimizing the quadratic approximation of

$$f(\boldsymbol{w}^k + \boldsymbol{s}) - f(\boldsymbol{w}^k) \approx q_k(\boldsymbol{s}) \equiv \nabla f(\boldsymbol{w}^k)^T \boldsymbol{s} + \frac{1}{2}\boldsymbol{s}^T \nabla^2 f(\boldsymbol{w}^k)\boldsymbol{s}, \tag{5}$$

where the gradient and the Hessian of $f(\boldsymbol{w})$ are respectively

$$\nabla f(\boldsymbol{w}) = \boldsymbol{w} + C\sum_{i=1}^{l} \xi'(y_i \boldsymbol{w}^T \boldsymbol{x}_i) y_i \boldsymbol{x}_i, \quad \nabla^2 f(\boldsymbol{w}) = I + CX^T DX. \tag{6}$$

In (6), I is the identity matrix, D is a diagonal matrix with

$$D_{ii} = \xi''(y_i \boldsymbol{w}^T \boldsymbol{x}_i), \text{and } X = [\boldsymbol{x}_1, ..., \boldsymbol{x}_l]^T \text{ is the data matrix.} \tag{7}$$

Note that L2 loss is not twice differentiable, but we can consider the generalized Hessian (**?**). From (6), $\nabla^2 f(\boldsymbol{w}^k)$ is positive definite, so we can obtain $\boldsymbol{s}^k$ by solving the following linear system.

$$\nabla^2 f(\boldsymbol{w}^k)\boldsymbol{s} = -\nabla f(\boldsymbol{w}^k). \tag{8}$$

Exactly solving (8) is often expensive, so truncated Newton methods that approximately solve (8) have been commonly used. Typically an inner iterative procedure such as the conjugate gradient (CG) method (**?**) is applied. The CG procedure involves a sequence of Hessian-vector products, but for a large number of features, $\nabla^2 f(\boldsymbol{w}^k) \in \mathbb{R}^{n\times n}$ is too large to be stored. Past developments (e.g., **??**) have shown that the special structure in (6) allows us to conduct Hessian-vector products without explicitly forming the Hessian:

$$\nabla^2 f(\boldsymbol{w})\boldsymbol{s} = (I + CX^T DX)\boldsymbol{s} = \boldsymbol{s} + CX^T(D(X\boldsymbol{s})). \tag{9}$$

The CG procedure returns a direction after an approximate solution of (8) is obtained. For example, LIBLINEAR considers the following condition to terminate the CG procedure.

$$\| - \nabla^2 f(\boldsymbol{w}^k)\boldsymbol{s} - \nabla f(\boldsymbol{w}^k)\| \le 0.1\|\nabla f(\boldsymbol{w}^k)\|. \tag{10}$$

For most optimization approaches including Newton methods, after finding a direction we must decide a suitable step size in order to ensure the convergence. Before describing two main strategies to choose the step size, we briefly discuss the complexity of a truncated Newton method. The cost per iteration is roughly

$$O(ln) \times (\#\text{ CG iterations} + 2) + \text{ cost of deciding the step size,} \tag{11}$$

where the $O(ln)$ term comes from the cost of each function, gradient evaluation, or Hessian-vector product. If X is a sparse matrix, then the $O(ln)$ term is replaced by $O(\#\text{nnz})$, where #nnz is the number of non-zero entries in X. For simplicity in the rest of the discussion we use $O(ln)$ all the time. We will show that the cost of deciding the step size is relatively small, but it can strongly affect the number of iterations and hence the total running time.

2.1. Line Search Methods

Line search methods aim to find a step size α_k along the ray $\{\boldsymbol{w}^k+\alpha\boldsymbol{s}^k \mid \alpha > 0\}$ to minimize $f(\boldsymbol{w}^k + \alpha\boldsymbol{s}^k)$. Ideally we may find

$$\alpha_k = \arg\min_{\alpha>0} f(\boldsymbol{w}^k + \alpha\boldsymbol{s}^k), \tag{12}$$

but the exact minimization of $f(\boldsymbol{w}^k+\alpha\boldsymbol{s}^k)$ may not be easy. A cheaper setting is to inexactly find an α_k that can produce a substantial reduction of $f(\boldsymbol{w})$.

Among the inexact line search methods, backtrack line search is a popular one because of its simplicity and efficiency. This approach finds the largest $\alpha \in \{1, \beta, \beta^2, ...\}$ with $0 < \beta < 1$ so that the new function value is sufficiently decreased:

$$f(\boldsymbol{w}^k + \alpha\boldsymbol{s}^k) \le f(\boldsymbol{w}^k) + \tau\alpha\nabla f(\boldsymbol{w}^k)^T\boldsymbol{s}^k, \text{ where } 0 < \tau < 1. \tag{13}$$

Note that $\nabla f(\boldsymbol{w}^k)^T\boldsymbol{s}^k < 0$ because the CG procedure obtians a descent direction. A concern is that many function evaluations $f(\boldsymbol{w}^k + \alpha\boldsymbol{s}^k)$ under various α values are conducted. For linear classification, a trick to save the cost (e.g., **?**) is by considering

$$X(\boldsymbol{w}^k + \alpha\boldsymbol{s}^k) = X\boldsymbol{w}^k + \alpha X\boldsymbol{s}^k. \tag{14}$$

Note that $X(\boldsymbol{w}^k+\alpha\boldsymbol{s}^k)$ is the main operation in calculating the function value $f(\boldsymbol{w}^k+\alpha\boldsymbol{s}^k)$; see (3)-(4). Each matrix-vector product in (14) expensively costs $O(ln)$, but if $X\boldsymbol{w}^k$ and $X\boldsymbol{s}^k$ are available, at each α, (14) takes only $O(l)$. To get $X\boldsymbol{w}^k$ and $X\boldsymbol{s}^k$, we first maintain $X\boldsymbol{w}^k$ throughout iterations: after (13) is satisfied, the current $X(\boldsymbol{w}^k + \alpha\boldsymbol{s}^k)$ can be passed as $X\boldsymbol{w}^{k+1}$ to the next Newton iteration. Therefore, $X\boldsymbol{s}^k$ is the only $O(ln)$ operation to be taken. Because it can be considered as the major cost for getting $X\boldsymbol{w}^{k+1}$ for the next function evaluation, by excluding it the cost of backtrack line search is merely

$$O(l) \times (\#\text{line-search steps}). \tag{15}$$

From (15) and (11), the cost of deciding the size is generally a small portion of the algorithm.

2.2. Trust Region Methods

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.

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

$$\min_{\boldsymbol{s}} \quad q_k(\boldsymbol{s}) \quad \text{subject to} \quad \|\boldsymbol{s}\| \le \Delta_k, \tag{16}$$

where $q_k(\boldsymbol{s})$ is defined in (5). Then, we update $\boldsymbol{w}^k$ by checking the ratio between the real and the predicted reduction of $f(\boldsymbol{w})$:

$$\rho_k = \frac{f(\boldsymbol{w}^k + \boldsymbol{s}^k) - f(\boldsymbol{w}^k)}{q_k(\boldsymbol{s}^k)}. \tag{17}$$

Algorithm 1: A framework of CG-based trust region Newton methods

Given $\boldsymbol{w}^0$.

For $k = 0, 1, 2, ...$

1. If $\nabla f(\boldsymbol{w}^k) = \mathbf{0}$, stop.
2. Approximately solve trust-region sub-problem (16) by the CG method to obtain a direction $\boldsymbol{s}^k$.
3. Compute ρ_k via (17).
4. Update $\boldsymbol{w}^k$ to $\boldsymbol{w}^{k+1}$ according to (18).
5. Update Δ_{k+1} according to (19).

Only if the ratio is large enough, will we update $\boldsymbol{w}$:

$$\boldsymbol{w}^{k+1} = \begin{cases} \boldsymbol{w}^k + \boldsymbol{s}^k, & \text{if } \rho > \eta_0, \\ \boldsymbol{w}^k, & \text{if } \rho \le \eta_0, \end{cases} \tag{18}$$

where $\eta_0 > 0$ is a pre-defined constant. Then, we adjust Δ_k by comparing the actual and the predicted function-value reduction. A common framework (**?**) is:

$$\Delta_{k+1} \in \begin{cases} [\gamma_1 \min\{\|\boldsymbol{s}^k\|, \Delta_k\}, \gamma_2 \Delta_k], & \text{if } \rho \le \eta_1, \\ [\gamma_1 \Delta_k, \gamma_3 \Delta_k], & \text{if } \rho \in (\eta_1, \eta_2), \\ [\Delta_k, \gamma_3 \Delta_k], & \text{if } \rho \ge \eta_2, \end{cases} \tag{19}$$

where $0 < \eta_1 < \eta_2 \le 1$ and $0 < \gamma_1 < \gamma_2 < 1 < \gamma_3$. If $\rho \ge \eta_2$, then we consider the current Newton step is successful and enlarge the region for the next iteration. In contrast, if $\rho \le \eta_1$, then we shrink the trust region by considering the current step as an unsuccessful one. We summarize a trust region Newton method in Algorithm 1.

To approximately solve the sub-problem (16), a classic approach (**?**) has been used in **?** for LIBLINEAR. The CG procedure starts with $\boldsymbol{s} = \mathbf{0}$ and satisfies that $\|\boldsymbol{s}\|$ is monotonically increasing. The CG procedure stops after either (10) is satisfied or an $\boldsymbol{s}$ on the boundary is obtained after CG iterates exceed the trust region.

For more details and the asymptotic convergence of the trust-region framework considered here, see Section 2 in **?**.

Various ways can be considered for implementing the update rule in (19). The one implemented in LIBLINEAR is

$$\Delta_{k+1} = \begin{cases} \min((\max(\alpha_k^*, \gamma_1))\|\boldsymbol{s}^k\|, \gamma_2 \Delta_k), & \text{if } \rho < \eta_0, & \text{(20a)} \\ \max(\gamma_1 \Delta_k, \min(\alpha_k^* \|\boldsymbol{s}^k\|, \gamma_2 \Delta_k)), & \text{if } \rho \in [\eta_0, \eta_1], & \text{(20b)} \\ \max(\gamma_1 \Delta_k, \min(\alpha_k^* \|\boldsymbol{s}^k\|, \gamma_3 \Delta_k)), & \text{if } \rho \in (\eta_1, \eta_2), & \text{(20c)} \\ \max(\Delta_k, \min(\alpha_k^* \|\boldsymbol{s}^k\|, \gamma_3 \Delta_k)), & \text{if } \rho \ge \eta_2. & \text{(20d)} \end{cases}$$

We see that the first condition in (19) is separated to two conditions here using the parameter η_0 in (18). Then clearly (20) falls into the framework of (19). In (20), $\alpha_k^* \|\boldsymbol{s}^k\|$ is introduced as an estimate of Δ_k (**?**), where

$$\alpha_k^* = \frac{-\nabla f(\boldsymbol{w}^k)^T \boldsymbol{s}^k}{2(f(\boldsymbol{w}^k + \boldsymbol{s}^k) - f(\boldsymbol{w}^k) - \nabla f(\boldsymbol{w}^k)^T \boldsymbol{s}^k)} \tag{21}$$

is the minimum of $\phi(\alpha)$, a quadratic interpolation of $f(\boldsymbol{w}^k + \alpha \boldsymbol{s}^k)$ such that

$$\phi(0) = f(\boldsymbol{w}^k), \quad \phi'(0) = \nabla f(\boldsymbol{w}^k)^T \boldsymbol{s}^k, \quad \phi(1) = f(\boldsymbol{w}^k + \boldsymbol{s}^k). \tag{22}$$

Because $f(\boldsymbol{w})$ is strictly convex, the denominator in (21) is always positive and thus α_k^* is well defined. Then in (20), we choose $\alpha_k^* \|\boldsymbol{s}^k\|$ or the closest endpoint in the corresponding interval as Δ_{k+1}. Other existing works that have incorporated $\alpha \|\boldsymbol{s}^k\|$ in the update rule include, for example, (**?**, Section 17.1).

2.3. Demonstration of Slow-convergence Situations

As mentioned in Section 1, this study is motivated from the occasional slow convergence of the trust region Newton method in LIBLINEAR. Here we demonstrate some real cases.

2.3.1. Settings of Evaluation

We carefully design our evaluation of optimization methods for linear classification. From (11), the major computation of the truncated Newton procedure is in the CG iterations. Thus for each method we check the cumulative number of CG iterations versus the relative reduction of the function value defined as

$$\frac{f(\boldsymbol{w}^k) - f(\boldsymbol{w}^*)}{f(\boldsymbol{w}^*)}, \tag{23}$$

where $\boldsymbol{w}^*$ is the optimal solution approximately obtained by running Newton methods with many iterations. Regarding the regularization parameter C, many past works simply pick a fixed value such as $C = 1$ or $1,000$ (e.g., **?**). However, such a setting may be different from the practical use of linear classification, where a suitable C value is selected from cross validation on a sequence of candidate values. Therefore, our strategy is to first identify the C_{best} value, which leads to the highest CV accuracy. Then training speed under values around C_{best} such as

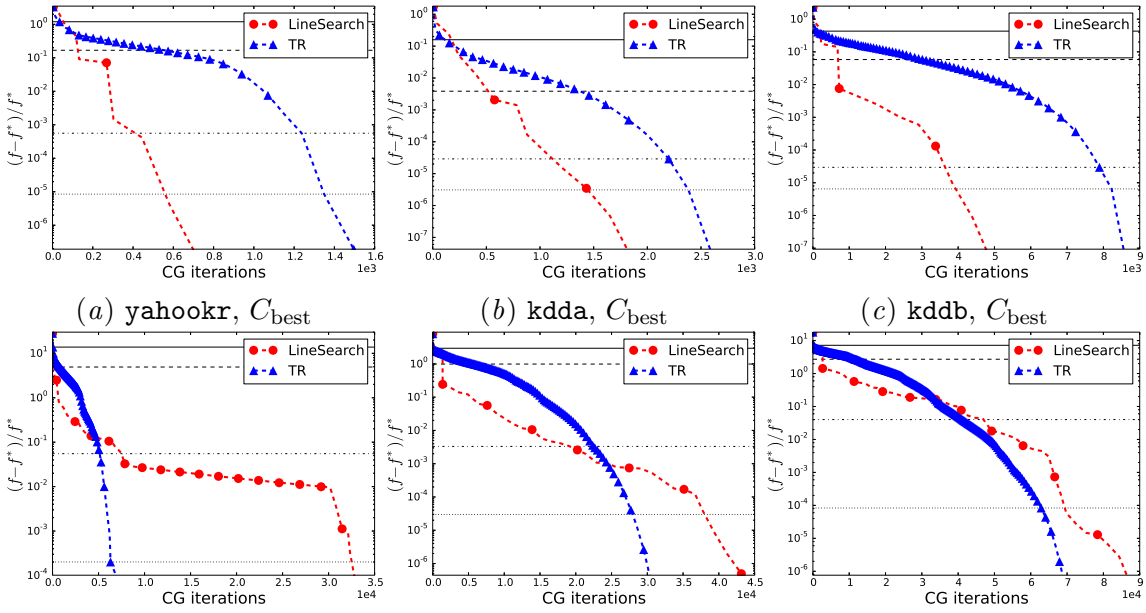
$$\{0.01, 0.1, 1, 10, 100\} \times C_{\text{best}}$$

is checked because these C values are used in practice.

Our another setting to ensure a suitable evaluation is by taking the stopping condition into consideration. It is not useful to check (23) when $\boldsymbol{w}$ is very close to $\boldsymbol{w}^*$ because the test accuracy may have long been stabilized. That is, we do not care the behavior of an optimization algorithm if it should have been stopped earlier. Therefore, we consider LIBLINEAR's stopping condition

$$\|\nabla f(\boldsymbol{w}^k)\| \le \epsilon \cdot \frac{\min(\#\text{pos}, \#\text{neg})}{l} \cdot \|\nabla f(\boldsymbol{w}^0)\|, \tag{24}$$

where #pos, #neg are the numbers of positive- and negative-labeled instances respectively, and l is the total number of instances. When the default condition with $\epsilon = 10^{-2}$ is reached, roughly an accurate solution has been obtained. That is, in testing a new instance, the prediction by the obtained model is generally the same as that by the optimal solution. In every figure we draw horizontal lines indicating that (24) has been satisfied with $\epsilon =$



(a) `yahookr`, C_{best} (b) `kdda`, C_{best} (c) `kddb`, C_{best}

(d) `yahookr`, $100C_{\text{best}}$ (e) `kdda`, $100C_{\text{best}}$ (f) `kddb`, $100C_{\text{best}}$

Figure 1: Convergence of truncated Newton methods for logistic regression using line search and trust region to decide the step size. The y-axis is the relative reduction of function value in log-scale; see (23). The x-axis is the cumulative number of CG iterations. We give a mark in each curve for every five Newton iterations. C_{best} for each data is listed in supplementary materials. The horizontal lines indicate that the stopping condition (24) has been satisfied with $\epsilon = \{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$.

$\{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$.[1] The behavior of an optimization algorithm before $\epsilon = 10^{-1}$ or $\epsilon = 10^{-4}$ is less important because the training process either stops too early or too late. More details about the data sets and our experimental setting are in Section 4.1.

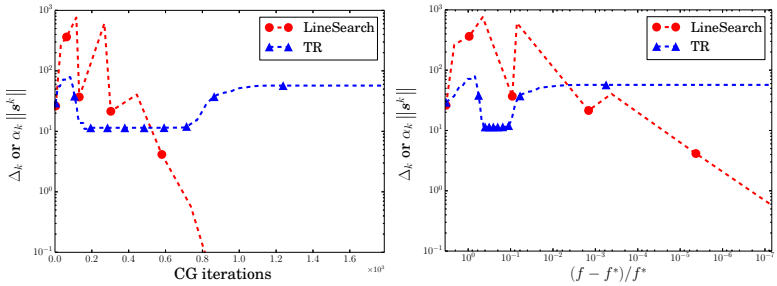
2.3.2. Situations of Slow Convergence

We show in Figure 1 that for some problems the trust region Newton method in LIBLINEAR that applies (20) to update Δ_k may converge slowly. Besides we make a comparison between line search and trust region methods. We make the following observations.

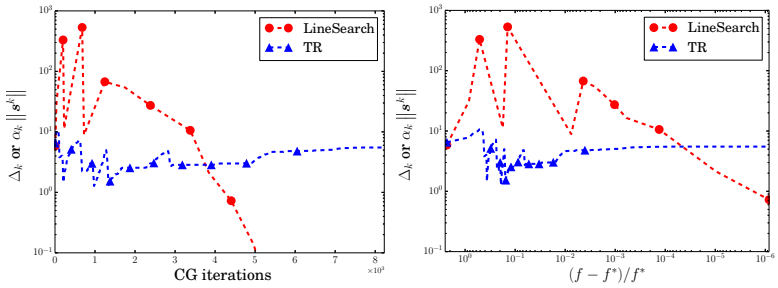
- Under the same truncated Newton framework, the convergence speed can be very different only because of different strategies (line search or trust region) in deciding the step sizes.
- When $C = C_{\text{best}}$, line search leads to much faster convergence than trust region, but the opposite result is observed for $C = 100C_{\text{best}}$.

The above experiment shows the importance of perfecting the step-size selection in truncated Newton methods for linear classification.

1. This setting depends on the optimization methods, because each method generates a sequence $\{\boldsymbol{w}^k\}$. Here we always consider the sequence obtained by the Newton implementation in LIBLINEAR 2.1.



(a) `yahookr`, LR loss, $C = C_{\text{best}}$



(b) `kddb`, LR loss, $C = C_{\text{best}}$

Figure 2: An investigation on the changes of Δ_k (for trust region methods) and $\alpha_k\|\boldsymbol{s}^k\|$ (for line search methods). The left column shows Δ_k or $\alpha_k\|\boldsymbol{s}^k\|$ versus the cumulative number of CG iterations, while the right column shows Δ_k or $\alpha_k\|\boldsymbol{s}^k\|$ versus the relative difference to the optimal function value.

3. Analysis and New Update Rules

In this section, after investigating the scenario of slow convergences, we propose novel and effective rules to update the trust region.

3.1. Investigation of Why Inappropriate Step Sizes are Obtained

We begin with analyzing the update rule (20) by considering some scenarios in Figure 1 for which it does not perform well. In Figure 2, we present the relationship between Δ_k and the cumulative number of CG iterations, and between Δ_k and the relative difference to the optimal function value. For line search, we replace Δ_k with $\alpha_k\|\boldsymbol{s}^k\|$. The purpose is to check how the step size changes in the optimization process. We are mainly interested in Δ_k and $\alpha_k\|\boldsymbol{s}^k\|$ in early iterations because values in final iterations are not comparable: for line search methods, $\alpha_k\|\boldsymbol{s}^k\| \to 0$ as $k \to \infty$, but Δ_k in trust region methods does not possess such a property.

Results in Figure 2 indicate that in early iterations, Δ_k of the trust region method is much smaller than $\alpha_k\|\boldsymbol{s}^k\|$ of line search. The small Δ_k causes that the CG procedure terminates because of hitting the trust-region boundary rather than satisfying the stopping condition (10) of solving the Newton linear system. To illustrate this point, in Table 1 we present details of early iterations in training the data set `yahookr` (logistic loss, $C = C_{\text{best}}$). Clearly, for the trust region setting, Δ_k does not increase much in early iterations and the CG procedure often hits the trust-region boundary. The decrease of the function value in early iterations is worse than that of using line search. The analysis seems to indicate

Table 1: Details of training the data set `yahookr` (logistic loss, $C = C_{\text{best}}$). The value $\alpha_{\min}$ is defined in (25). The column of $\|\boldsymbol{s}^k\| = \Delta_k$ indicates if the CG procedure stops because of hitting the trust-region boundary or not. The last row shows the function value of each approach after the listed iterations.

	Line Search			Trust Region				
iter	#CG	$\alpha_k\|\boldsymbol{s}^k\|$	α_k	#CG	Δ_k	$\|\boldsymbol{s}^k\| = \Delta_k$	α_k^*	$\alpha_{\min}$
1	4	26	1	4	29	Y	1.121	1.445
2	28	266	1	4	37	Y	1.273	1.518
3	31	364	1	5	48	Y	1.279	1.465
4	53	759	1	6	55	Y	1.153	1.245
5	16	37	1	7	55	Y	0.965	0.940
6	136	609	1	9	63	Y	1.141	1.204
7	35	21	1	9	72	Y	1.139	1.214
8	141	41	1	12	72	Y	0.978	0.972
9	136	4	1	5	72	N	1.053	1.091
10	157	6e-01	1	9	79	Y	1.098	1.178
11	126	3e-02	1	13	79	Y	0.824	0.868
12	150	6e-03	1	14	55	Y	0.699	0.618
13	138	3e-04	1	9	38	Y	0.694	0.641
14	159	5e-05	1	11	27	Y	0.696	0.645
f	4.727e+06			7.218e+06				

that the update rule in (20) is too conservative to enlarge Δ_k. However, if we aggressively increase Δ_k, some CG steps may be wasted because of the following situations:

1. Because Δ_k is large, the CG procedure takes many steps without hitting the boundary.
2. However, the resulting direction $\boldsymbol{s}^k$ does not lead to the sufficient decrease of the function value. That is, $\rho_k \leq \eta_0$ in (18), so we reduce Δ_k to be Δ_{k+1} and repeat the same CG procedure in a smaller region.

Similar situations may occur for the line search setting. In Table 1, many CG steps are taken in each of the early iterations. The efforts pay off because the resulting directions lead to the sufficient decrease of function values with α_k close to 1. However, when C is enlarged to $100C_{\text{best}}$, in Figure 1d, slow convergence occurs in the middle of the optimization procedure. Each iteration requires many CG steps but the step size obtained by backtrack line search is much smaller than one. Thus $\boldsymbol{w}^k$ is slowly updated. This situation is where trust region is superior to line search because if we anticipate a small step size, the number of CG iterations can be restricted by the trust region constraint.

To gain more understanding about the update rule (20), we investigate its use of $\alpha_k^*\|\boldsymbol{s}^k\|$ to estimate Δ_k. In Table 1, we present α_k^* and

$$\alpha_{\min} \equiv \arg\min_\alpha f(\boldsymbol{w}^k + \alpha \boldsymbol{s}^k). \tag{25}$$

Results show that α_k^* is smaller than $\alpha_{\min}$ in early iterations. If $\alpha_{\min}$ is used instead of α_k^*, Δ_k is increased more quickly and a faster reduction of the function value may be obtained. Therefore, α_k^* obtained by a quadratic interpolation in (22) may not be accurate enough

to approximate $\alpha_{\min}$ in (25). Based on the observation here, later we will propose more accurate estimates of $\alpha_{\min}$ as our α_k^*.

3.2. Our New Update Rules

Based on the investigation in Section 3.1, we propose and investigate the following changes for the trust-region update rule.
1. We will more quickly enlarge Δ_k in early iterations.
2. We will devise effective methods to accurately solve $\min_\alpha f(\boldsymbol{w}^k + \alpha \boldsymbol{s}^k)$.

We explain the first change here while leave details of the second in Section 3.3.

In the framework (19), we have

$$\Delta_{k+1} \in [\Delta_k, \gamma_3 \Delta_k], \quad \text{if } \rho \geq \eta_2.$$

That is, if the predicted reduction is close enough to the actual reduction, we may enlarge Δ. The realization in (20) has

$$\Delta_{k+1} = \max(\Delta_k, \min(\alpha_k^* \|\boldsymbol{s}^k\|, \gamma_3 \Delta_k)), \quad \text{if } \rho \geq \eta_2. \tag{26}$$

We mentioned in Section 3.1 that in early iterations, $\|\boldsymbol{s}^k\| = \Delta_k$ often occurs. If α_k^* is not large (say ≈ 1), then the setting in (26) does not increase Δ_k much for Δ_{k+1}. However,

$$\rho \geq \eta_2 \quad \text{and} \quad \|\boldsymbol{s}^k\| = \Delta_k \tag{27}$$

suggest that the quadratic approximation (5) is reasonably good and the solution of the linear system (8) is outside the region $\|\boldsymbol{s}\| \leq \Delta_k$. Therefore, it is suitable to enlarge Δ if (27) holds. To this end, we split (27) to two cases and have the following update rule:

$$\Delta_{k+1} = \begin{cases} \text{same rules in (20a)-(20c)}, & \text{if } \rho < \eta_0, \rho \in [\eta_0, \eta_1], \text{ or } \rho \in (\eta_1, \eta_2), \\ \max(\Delta_k, \min(\alpha_k^* \|\boldsymbol{s}^k\|, \gamma_3 \Delta_k)), & \text{if } \rho \geq \eta_2 \text{ and } \|\boldsymbol{s}^k\| < \Delta_k, \\ \gamma_3 \Delta_k, & \text{if } \rho \geq \eta_2 \text{ and } \|\boldsymbol{s}^k\| = \Delta_k. \end{cases} \tag{28}$$

Some optimization works on trust region methods (e.g., **?**, Chapter 4) have specifically handled the case of (27), though the importance is seldom studied. We will show that for some problems, the convergence speed can be dramatically improved.

3.3. Accurate Minimization of $f(\boldsymbol{w}^k + \alpha \boldsymbol{s}^k)$

The analysis in Section 3.1 indicates that an accurate estimate of $\arg\min_\alpha f(\boldsymbol{w}^k + \alpha \boldsymbol{s}^k)$ may serve as a better α_k^* in the update rule. Whereas, for general optimization problems an accurate estimate is often expensive. This situation has caused that for line search methods, backtrack rather than exact search is used in practice. However, an important finding in this work is that this one-variable minimization is cheap for linear classification. Define

$$g(\alpha) = f(\boldsymbol{w}^k + \alpha \boldsymbol{s}^k). \tag{29}$$

Algorithm 2: A bisection method to minimize $g(\alpha) = f(\boldsymbol{w}^k + \alpha \boldsymbol{s}^k)$

Initialize $\alpha_l = 0$, $\alpha_r = 2$, max_steps $= 10$ and $\epsilon_b = 10^{-3}$.
While $g'(\alpha_r) < 0$ **do** $\alpha_l = \alpha_r; \alpha_r = \alpha_l + 2$ **end**
For $i = 1, \ldots,$ max_steps
  $\alpha_m = (\alpha_l + \alpha_r)/2$
  If $g'(\alpha_m) < 0$ **then** $\alpha_l = \alpha_m$ **else** $\alpha_r = \alpha_m$
  If $|g'(\alpha_m)| < \epsilon_b |g'(0)|$ **then break**
return α_l

We can see that

$$g(\alpha) = \frac{1}{2}(\boldsymbol{w}^k)^T \boldsymbol{w}^k + \alpha (\boldsymbol{w}^k)^T \boldsymbol{s}^k + \frac{1}{2}\alpha^2 (\boldsymbol{s}^k)^T \boldsymbol{s}^k + C \sum_{i=1}^{l} \xi\left(y_i (\boldsymbol{w}^k + \alpha \boldsymbol{s}^k)^T \boldsymbol{x}_i\right), \tag{30}$$

$$g'(\alpha) = (\boldsymbol{w}^k)^T \boldsymbol{s}^k + \alpha (\boldsymbol{s}^k)^T \boldsymbol{s}^k + C \sum_{i=1}^{l} \xi'\left(y_i (\boldsymbol{w}^k + \alpha \boldsymbol{s}^k)^T \boldsymbol{x}_i\right) y_i \boldsymbol{x}_i^T \boldsymbol{s}^k, \tag{31}$$

$$g''(\alpha) = (\boldsymbol{s}^k)^T \boldsymbol{s}^k + C \sum_{i=1}^{l} \xi''\left(y_i (\boldsymbol{w}^k + \alpha \boldsymbol{s}^k)^T \boldsymbol{x}_i\right) (y_i \boldsymbol{x}_i^T \boldsymbol{s}^k)^2. \tag{32}$$

A direct calculation of (30)-(32) expensively costs $O(ln)$. However, if

$$(\boldsymbol{w}^k)^T \boldsymbol{w}^k, \quad (\boldsymbol{w}^k)^T \boldsymbol{s}^k, \quad (\boldsymbol{s}^k)^T \boldsymbol{s}^k, \quad X\boldsymbol{w}^k, \quad X\boldsymbol{s}^k \tag{33}$$

are available, then the cost is significantly reduced to $O(l)$. The idea is similar to how we reduce the cost for line search in Section 2.1. That is, by $O(ln)$ cost to calculate all values in (33), for any given α, (30)-(32) can be cheaply calculated in $O(l)$ time. Then many unconstrained optimization methods can be applied to minimize (29) if they do not evaluate (30)-(32) too many times. The cost of minimizing (29) becomes much smaller than

$$O(ln) \times \#\text{CG iterations}$$

for finding the Newton direction; see the discussion on the complexity of truncated Newton methods in (11). In Section **??** of supplementary materials we investigate various methods to minimize (29). Our conclusion is that instead of using general optimization technique a simple bisection method is effective. We discuss the bisection implementation in detail in Section 3.3.1, while leave others in supplementary materials.

3.3.1. Bisection Method

The bisection method is a well known approach to find a root of a one-variable function. It starts with two points having positive and negative function values and continues to shrink the interval by half. The following theorem shows that the bisection method can be used to solve $g'(\alpha) = 0$ for minimizing (29). The proof is in supplementary materials.

Theorem 1 *Consider logistic or L2 loss. The function $g(\alpha)$ defined in (30) satisfies*
1. $g'(\alpha) = 0$ has a unique root at an $\alpha^* > 0$. This root is also the unique minimum of (29).
2. $g'(\alpha), \alpha \geq 0$ is a strictly increasing function. Further, $g'(0) < 0$.

An implementation of the bisection method is in Algorithm 2. To find left and right initial points, we take the properties $g'(0) < 0$ and $g'(\alpha) > 0$, $\forall \alpha \geq \alpha^*$. We start with $\alpha_l = 0$ and $\alpha_r = 2$, and check if $g'(\alpha_r) > 0$. If $g'(\alpha_r) < 0$, we continually increase α_l and α_r by 2 until $g'(\alpha_r) > 0$. Note that the initial $\alpha_r = 2$ is reasonable because for the backtrack line search the start point is one, which is the middle point of the interval $[0, 2]$. After obtaining initial left and right points, we shrink the interval by half at each iteration and maintain $g'(\alpha_l) < 0$ and $g'(\alpha_r) > 0$. At each iteration the cost is $O(l)$ for calculating $g'(\alpha_m)$, where $\alpha_m = (\alpha_l + \alpha_r)/2$. With a given tolerance ϵ_b, the algorithm stops after $|g'(\alpha_m)| < \epsilon_b |g'(0)|$ or reaching a specified maximal number of steps. We then return α_l rather than α_r because $g'(\alpha_l) < 0$ ensures the decrease of the function value. That is, if $\alpha_l > 0$, there exists $\bar{\alpha} \in [0, \alpha_l]$ such that

$$f(\boldsymbol{w}^k + \alpha_l \boldsymbol{s}^k) = g(\alpha_l) = g(0) + g'(\bar{\alpha})\alpha_l < g(0) = f(\boldsymbol{w}^k),$$

where we use the increasing property of $g'(\alpha)$, $\forall \alpha \geq 0$ to have $g'(\bar{\alpha}) \leq g'(\alpha_l) < 0$.

Because the interval is cut to half each time, the number of steps is small in practice. In supplementary materials, we experimentally confirm that the cost of using the bisection method to solve (25) is much cheaper than that of CG iterations for finding the direction.

3.3.2. Discussion

The earlier work (**?**) devises an $O(l \log l)$ algorithm to exactly minimize (29) when the L2 loss is used. They take the special structure of the L2 loss into account. In contrast, by techniques in (30)-(33), under a comparable cost of

$$O(l) \times (\#\text{ of } g(\alpha), g'(\alpha), \text{ or } g''(\alpha) \text{ evaluations}),$$

our method can handle any convex and differentiable loss function in the form of $\xi(y\boldsymbol{w}^T\boldsymbol{x})$.

4. Experiments

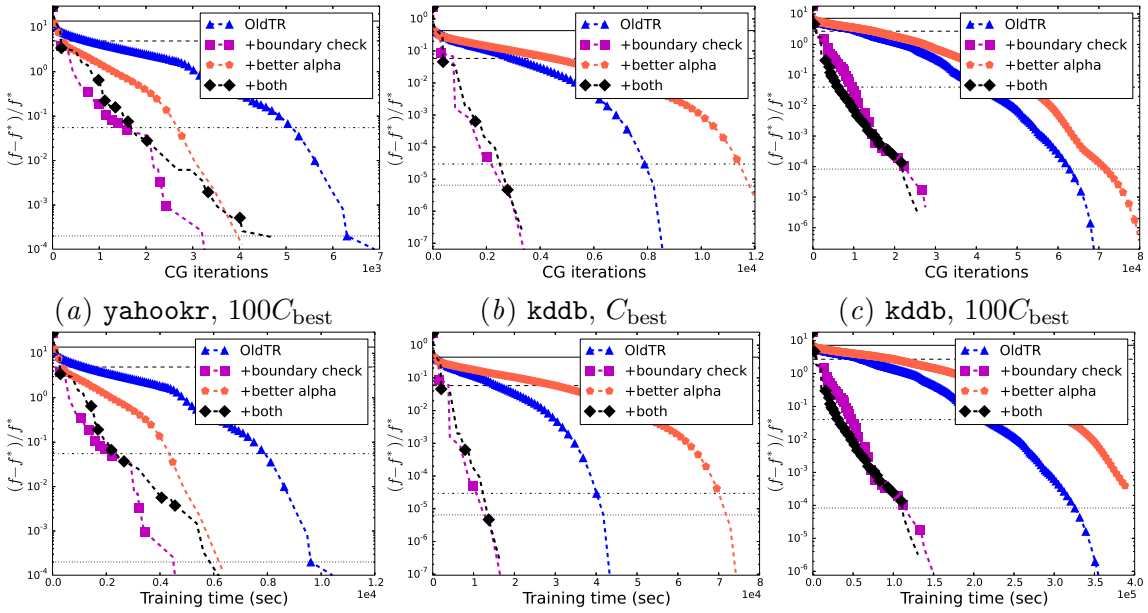
In this section, we begin with investigating the effectiveness of techniques proposed in Section 3. Then we demonstrate that for logistic regression the new update rule outperforms the rule (20) used in LIBLINEAR and the line search method. Because of space limit, only part of experimental results are presented. More experiments as well as results for L2-loss SVM are given in supplementary materials.

4.1. Data Sets and Experimental Settings

We consider binary classification data sets shown in Table **??** of the supplementary materials. Except `yahoojp` and `yahookr`, all other sets are available from https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets. We modify LIBLINEAR to implement methods discussed in this paper. We let $\boldsymbol{w}^0 = \boldsymbol{0}$ as the initial solution. For the line search methods, at each Newton iteration, the CG procedure stops if the direction $\boldsymbol{s}$ satisfies (10). If trust region methods are used, the CG procedure stops if either (10) holds or $\|\boldsymbol{s}\| \geq \Delta_k$.

For the condition (13) in backtrack line search, we set $\tau = 0.01, \beta = 1/2$. For trust region methods, we consider the same parameter values in LIBLINEAR:

$$\eta_0 = 0, \quad \eta_1 = 0.25, \quad \eta_2 = 0.75, \quad \gamma_1 = 0.25, \quad \gamma_2 = 0.5, \quad \gamma_3 = 4.$$



(a) `yahookr`, $100C_{\text{best}}$ (b) `kddb`, C_{best} (c) `kddb`, $100C_{\text{best}}$

(d) `yahookr`, $100C_{\text{best}}$ (e) `kddb`, C_{best} (f) `kddb`, $100C_{\text{best}}$

Figure 3: Convergence of using different trust region update rules. Upper: x-axis is the cumulative number of CG iterations. Lower: x-axis is the training time. See Figure 1 for explanation of information in each sub-figure.

4.2. Effectiveness of the Proposed Techniques in Section 3

We compare the following trust-region settings proposed in Section 3.

- OldTR: the setting in LIBLINEAR 2.1. The rule (20) is used, and α_k^* is obtained by (21).
- +boundary check: the rule (20) is modified to (28).
- +better alpha: α_k^* is by the bisection method in Section 3.3.1 rather than by (21).
- +both: the above two changes are both applied.

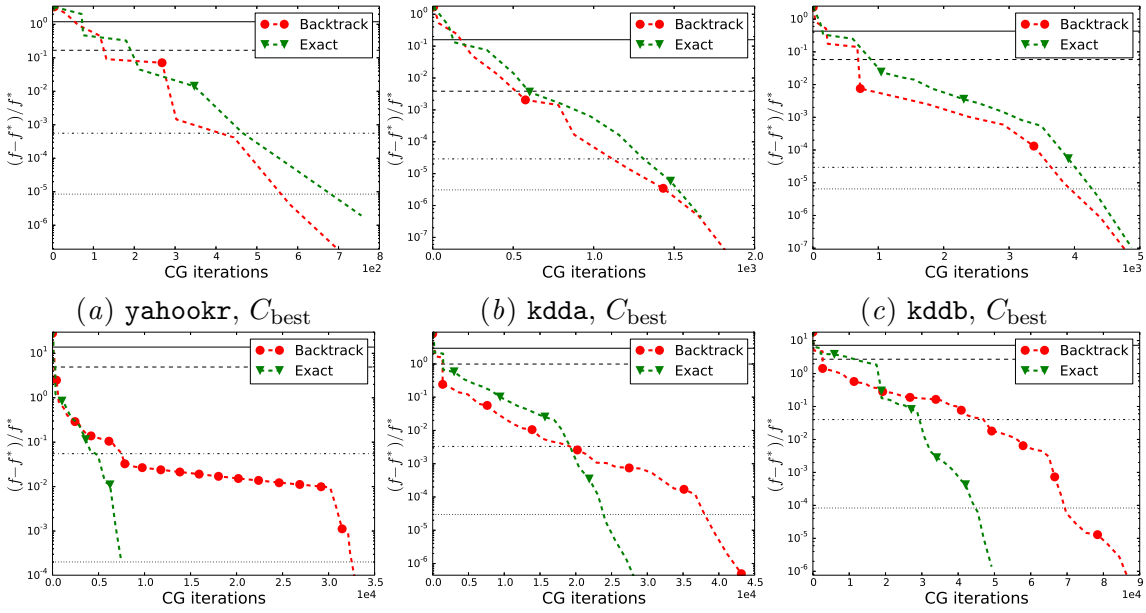
We present two comparisons in Figure 3:

1. function-value reduction versus the cumulative number of CG iterations.
2. function-value reduction versus the running time.

The use of both the cumulative number of CG iterations and the running time helps to check if they are strongly correlated. From Figures 3, we make the following observations.

1. Figures of showing the cumulative numbers of CG iterations are almost the same as those of showing the running time. Therefore, experiments confirm our analysis in Section 3 that even if we accurately minimize $f(\boldsymbol{w}^k + \alpha \boldsymbol{s}^k)$ by the bisection method, CG steps are still the dominate operations. Subsequently we will only present figures of function-value reduction versus CG iterations.
2. The approach "+boundary check" of enlarging Δ_k when (27) holds is very effective. It is very remarkable that a small change leads to huge improvements.
3. By comparing "OldTR" and "+better alpha" we see that an accurate minimization of $f(\boldsymbol{w}^k + \alpha \boldsymbol{s}^k)$ may not always improve the convergence speed.

Our results are surprising because the accurate minimization of $f(\boldsymbol{w}^k + \alpha \boldsymbol{s}^k)$ seems to be less useful than the simple change in (28) by specially handling the situation when $\|\boldsymbol{s}^k\| = \Delta_k$. We can see that "+boundary check" is as competitive as "+both." To examine



(a) `yahookr`, C_{best} (b) `kdda`, C_{best} (c) `kddb`, C_{best}

(d) `yahookr`, $100C_{\text{best}}$ (e) `kdda`, $100C_{\text{best}}$ (f) `kddb`, $100C_{\text{best}}$

Figure 4: Comparison of two line search methods on logistic regression with $C = \{1, 100\} \times C_{\text{best}}$. See Figure 1 for explanation of information in each sub-figure.

if incorporating $\alpha_k^* \|\boldsymbol{s}^k\|$ in the trust-region update rule is really needed, in Section 4.3 we conduct some further investigation on the step size α.

Overall we can see that the proposed techniques in Section 3 effectively improve upon the current setting in LIBLINEAR.

4.3. More Investigation on the Step Size α

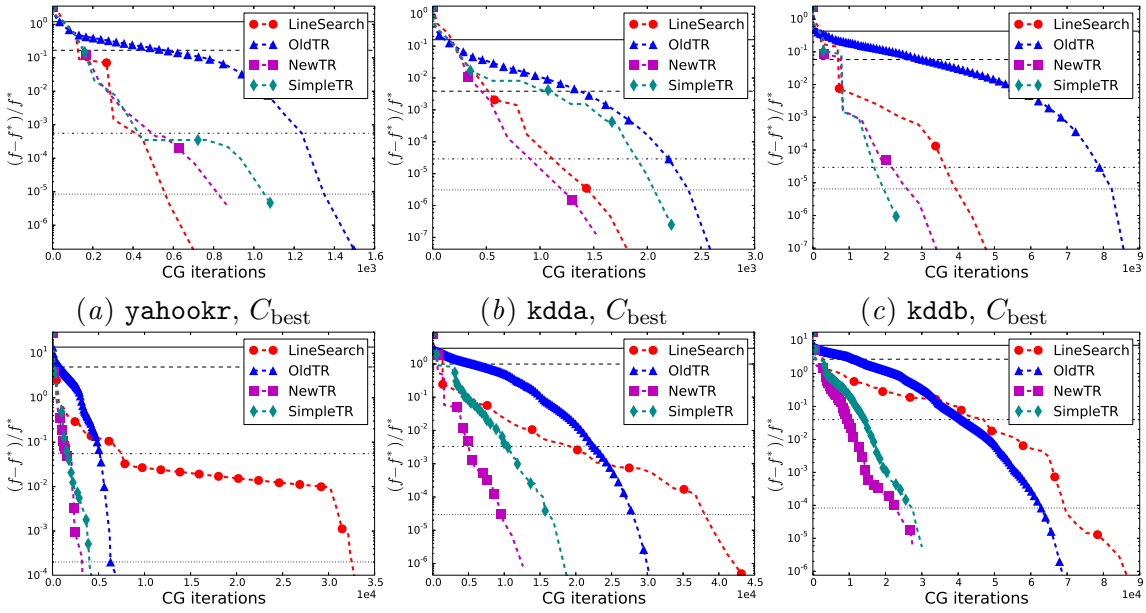
We mentioned in Section 4.2 that in Figure 3, in some situations a more accurate minimization of $f(\boldsymbol{w}^k + \alpha \boldsymbol{s}^k)$ leads to slower convergence than the crude estimate by (22). This result motivates us to further study the role of α in the optimization process. We begin with comparing the following two settings to decide the step size in the line search method.

- Backtrack: the backtrack line search procedure discussed in Section 2.1.
- Exact: exact line search, where $f(\boldsymbol{w}^k + \alpha \boldsymbol{s}^k)$ is minimized by a bisection method.

We show results of using $C = \{1, 100\} \times C_{\text{best}}$ in Figures 4 for logistic regression. When $C = C_{\text{best}}$, backtrack is slightly better than exact line search. However, when $C = 100C_{\text{best}}$, the opposite result is observed and the difference is significant.

From more results in supplementary materials, we conclude that an accurate minimization of $f(\boldsymbol{w}^k + \alpha \boldsymbol{s}^k)$ may not be always beneficial. Our experiment is very interesting because most past works were unable to afford the exact line search and make a comparison. An explanation of the result is that we choose α under a fixed direction $\boldsymbol{s}^k$, but in the whole optimization process α_k and $\boldsymbol{s}^k$ do affect each other. Therefore, a more sophisticated scheme on the direction/step size selection seems to be useful. For example, trust region methods are designed to take the direction generation into account by the constraint $\|\boldsymbol{s}\| \le \Delta_k$.

In Section 4.2, we conclude that the change to enlarge Δ_k when $\|\boldsymbol{s}^k\| = \Delta_k$ is more helpful than accurately minimizing $f(\boldsymbol{w}^k + \alpha \boldsymbol{s}^k)$. Thus it is important to investigate if the



(a) `yahookr`, C_{best} (b) `kdda`, C_{best} (c) `kddb`, C_{best}

(d) `yahookr`, $100C_{\text{best}}$ (e) `kdda`, $100C_{\text{best}}$ (f) `kddb`, $100C_{\text{best}}$

Figure 5: Comparison of line search and different trust region methods on logistic regression with $C = \{1, 100\} \times C_{\text{best}}$. See Figure 1 for explanation of information in each sub-figure.

use of $\alpha_k^* \|\boldsymbol{s}^k\|$ in trust-region update rules is really needed. To this end, we consider a rule by completely removing $\alpha_k^* \|\boldsymbol{s}^k\|$ in (28). However, an issue is that we then have

$$\Delta_{k+1} = \gamma_3 \Delta_k, \text{ if } \rho \geq \eta_1. \tag{34}$$

This rule is not suitable because first Δ_k is always enlarged even if ρ is relatively small (e.g., $\rho = \eta_1$), and second, we no longer have the setting in (28) to check if $\|\boldsymbol{s}^k\| = \Delta_k$ or not for deciding Δ_{k+1}. Thus, we split (34) to three cases and get the following update rule:

$$\Delta_{k+1} = \begin{cases} \gamma_1 \Delta_k, & \text{if } \rho < \eta_0, \\ \gamma_2 \Delta_k, & \text{if } \rho \in [\eta_0, \eta_1], \\ \Delta_k, & \text{if } \rho \in (\eta_1, \eta_2), \\ \Delta_k, & \text{if } \rho \geq \eta_2 \text{ and } \|\boldsymbol{s}^k\| < \Delta_k, \\ \gamma_3 \Delta_k, & \text{if } \rho \geq \eta_2 \text{ and } \|\boldsymbol{s}^k\| = \Delta_k. \end{cases} \tag{35}$$

In Figure 5, for logistic regression we compare several trust region and line search settings.

- OldTR: the setting in LIBLINEAR of using (20).
- NewTR: the new rule (28) is used. It is the same as "+boundary check" in Section 4.2.
- SimpleTR: the rule (35) is used.
- LineSearch: the backtrack line search procedure discussed in Section 2.1.

We present only the relation between the function value and the cumulative number of CG iterations because Section 4.2 has concluded that the running-time results are very similar. We set $C = \{1, 100\} \times C_{\text{best}}$ while leave results of other C values in supplementary materials.

Results show that SimpleTR is generally competitive. For some problems it is as good as NewTR. We learned from Section 4.2 that specifically handling the situation of $\|\boldsymbol{s}^k\| = \Delta_k$

is very helpful. This setting is included in SimpleTR so its good performance is expected. However, for some problems such as `yahookr` with $C = C_{\text{best}}$, SimpleTR has slow convergence in the middle of the optimization procedure. We find that SimpleTR reduces Δ_k in several iterations without improving the function value. We have explained in Section 3.1 that if at some point Δ_k is too large, several iterations may be wasted for reducing Δ_k. In this situation, the use of $\alpha_k^* \|\boldsymbol{s}^k\|$ can more quickly shrink Δ_k in fewer iterations. Therefore, incorporating $\alpha_k^* \|\boldsymbol{s}^k\|$ in the update rule is still useful although from Section 4.2 a very accurate estimation of α_k^* may not be needed.

For the comparison between trust region and line search, although in Section 2.3 we observed that line search is sometimes better than OldTR, from Figures 5(a)-(c) the improvements made in NewTR have caused that trust region methods reach the same convergence speed as line search's when $C = C_{\text{best}}$. Line search becomes slower when C is large as we mentioned in Section 2.3. We can see from Figures 5(d)-(f) that the new trust region update rule leads to much faster convergence than line search when $C = 100C_{\text{best}}$.

Based on our extensive experiments, we conclude that NewTR is a suitable setting in trust-region Newton methods for linear classification.

5. Discussion and Conclusions

We discuss some related works in Section **??** of supplementary materials. In summary, we point out some slow convergence issues of existing trust region update rules in Newton methods for linear classification. Through some deep analysis, we propose new update rules that dramatically improve the convergence speed of the optimization procedure.