A Study on Truncated Newton Methods for Linear Classification

Leonardo Galli and Chih-Jen Lin, *Fellow, IEEE*

***Abstract*—Truncated Newton (TN) methods have been a useful technique for large-scale optimization. Instead of obtaining the full Newton direction, a truncated method approximately solves the Newton equation with an inner Conjugate Gradient (CG) procedure (TNCG for the whole method). These methods have been employed to efficiently solve linear classification problems. But even in this deeply studied field, various theoretical and numerical aspects were not completely explored. The first contribution of this work is to comprehensively study the global and local convergence when TNCG is applied to linear classification. Because of the lack of twice differentiability under some losses, many past works cannot be applied here. We prove various missing pieces of theory from scratch and clarify many proper references. The second contribution is to study the termination of the CG method. For the first time when TNCG is applied to linear classification, we show that the inner stopping condition strongly affects the convergence speed. We propose using a quadratic stopping criterion to achieve both robustness and efficiency. The third contribution is that of combining the study on inner stopping criteria with that of preconditioning. We discuss how convergence theory is affected by preconditioning and finally propose an effective preconditioned TNCG.**

***Index Terms*—Truncated Newton, Conjugate gradient, Linear classification, Truncation criteria, Preconditioning.**

I. Introduction

In this work we focus on the problem of estimating the model parameter $\boldsymbol{w}$ of a linear classifier. In particular, two widely used models are logistic regression and linear Support Vector Machines (SVM). The problem of training both these models might be written as follows

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

where $(y_i, \boldsymbol{x}_i)$, $i = 1, ..., l$ are the training data, $y_i = \pm 1$ is the label, $\boldsymbol{x}_i \in \mathbb{R}^n$ is a feature vector, $\boldsymbol{w}^T\boldsymbol{w}/2$ is the regularization term, $C > 0$ is a regularization parameter and $\xi(y_i\boldsymbol{w}^T\boldsymbol{x}_i)$ is any LC^1 (continuously differentiable with locally Lipschitz continuous gradient) convex loss function. With $\boldsymbol{w}^T\boldsymbol{w}$, f is a LC^1 strongly convex function and the minimum $\boldsymbol{w}^*$ of f exists and is unique. The following two losses respectively correspond to logistic regression (C^2, i.e. twice continuously differentiable) and l2-loss linear SVM (LC^1),

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

In this work we focus on the Truncated Newton (TN) method for solving large scale optimization problems that might be written as in (1). In this field, solving the Newton equation to obtain a direction is often very challenging because of the dimensionality of the system. A truncated method approximately solves the Newton equation with an internal iterative procedure specific for a linear system of equations. In [1] and [2] they employ a Conjugate Gradient (CG) method [3] to avoid the storage of the Hessian in solving the Newton equation. The resulting method is thus called TNCG.

Our main concern is the convergence of TNCG, both from the numerical and theoretical points of view. For this reason, we brought into question various aspects of TNCG methods for (1) that till now were taken for granted. The first contribution is to comprehensively study the global and local theoretical convergence of TNCG. In particular, when $f \notin \mathrm{C}^2$ as for ξ_{L2}, we find out that most of the proofs are not at hand and some of them are not even existing. We thus obtained global and local Q-SuperLinear (Q-SL) convergence. Moreover, we proved that the TNCG is a special case of the general common-directions framework from [4], so some nice theoretical properties follow.

The second finding was that the choice of the CG inner termination (or truncation) criterion has never been addressed for linear classification problems. In fact, we show that the convergence speed of some widely used machine learning software can be improved by very simple modifications on this stopping rule. Through conceptual and experimental illustrations we thus identify that a criterion based on checking the quadratic model of the current Newton iteration is robust and effective for large scale linear classification. Finally, with an adaptive setting in the inner stopping criteria, local Q-SL convergence is proved.

The third contribution is that of combining the study on truncation criteria with that on preconditioning. It is well known that for ill-conditioned linear systems the Preconditioned CG (PCG) can be helpful to improve the rate of convergence of the original CG. The idea is to pre-multiply the linear system by a preconditioner matrix that will improve its condition. Let us call TNPCG the complete method. In this work, we first discuss how the convergence proofs are affected by the use of preconditioners. Then, we integrate our numerical analysis on truncation criteria with that conducted in [5] on preconditioning. They found out that a mixed approach between the identity matrix and a diagonal preconditioner was able to improve convergence speed in the majority of the data sets employed. Thanks to this new extensive numerical analysis on the combination between the truncation criteria and the preconditioning, we are able to propose a highly robust and effective TNPCG method for linear classification.

This paper is organized as follows. In Section II we go

L. Galli was with the Department of Information Engineering, University of Florence, Via Santa Marta 3, Firenze, Italia e-mail: leonardo.galli@unifi.it.
C-J, Lin was with Department of Computer Science, National Taiwan University, Taipei, Taiwan, 106 e-mail: cjlin@csie.ntu.edu.tw.

Manuscript received ; revised .

over past works from the literature that have some similarities with ours. In Section III, we review TNCG for large scale linear classification. Section IV gives a detailed analysis on the theoretical aspects related to TNCG, including both global and local convergence. In Section V we first discuss the importance of having a robust inner stopping criterion. Then we investigate some criteria and prove their theoretical local convergence. In Section VI we first show how to apply preconditioning in our case, which approach has been employed, and how to obtain local and global convergence in the preconditioned case. In Section VII we conduct extensive experiments on termination criteria, while in Section VIII we combine them with the use of preconditioning. Further in Section IX, some running-time comparisons with state-of-the-art methods demonstrate the superiority of the proposed approach on ill-conditioned problems. Conclusions are given in Section X. Proofs of some theorems are in the appendix, while the rest of the proofs and of the experiments are enclosed in the supplementary available at https://www.csie.ntu.edu.tw/~cjlin/papers/tncg/. Programs used for experiments are available at the same page, while the proposed method has been incorporated into the software LIBLINEAR (version 2.40 and after).

II. Related Works

TNCG is a classical optimization method so the convergence theory has been deeply investigated. One would expect that, when TNCG is applied to linear classification, all the theorems are easily accessible. We instead found out that this is not the case. Theoretical properties (especially the fast local convergence ones) may either be partially covered in some works or be disjointedly presented in various paper fragments. The original Q-SL convergence result for TN methods was given in [6]. In this result, f is assumed to be C^2, which is true in the case of the logistic loss, but not true for the l2 loss (2). This loss is only LC^1, so the Hessian of f does not exist and we should instead refer to the generalized Hessian $\partial\nabla f$ in the sense of Clarke [7] (see Section IV-B for details). Nonetheless, in some studies on TNCG for linear classification (for instance [2]), theory has been studied for the ξ_{LR} loss, but was ignored for the ξ_{L2} loss. In the field of linear classification, many following works focused on various aspects of the TNCG (e.g. hyperparameter selection [8], globalization techniques [9], preconditioning [5]), but never on the fast convergence result for not twice continuous differentiable losses. See [10] for a detailed survey on the topic.

The first Q-SL convergence proof for TN methods in the case of a nonsmooth system of equations was given in [11]. Such a result is useful in our case since the arising Newton equation is also a nonsmooth system, but the convergence in [11] is not covering the whole theory since their theorems are only given for a non-globalized TN method (i.e. the TNCG without line search, see (8) and the discussion above it). A fast convergence result for a globalized TNCG is instead given in Theorem 5.3 of [12]. This theorem is applied to functions with globally Lipschitz gradient, but here we develop the theory to cover the more general situation of functions whose gradient is only locally Lipschitz. Besides this main distinction, [12] differs from us in the following aspects. They need to prove the regularity of the generalized Hessian, while we employ the more standard and general BD-regularity (see Lemma V of the supplementary). Next, we provide an additional intermediate result to cope with inner stopping rules with a more general shape, while they assume a precise property for their inner stopping rule (see a discussion at the end of Section V-C and the proof of Lemma 1 in the supplementary). To conclude this section we can also mention the recent paper [13] on linear SVM, even if the theory therein is not very clear. In their Theorem 1 they cite Theorem 3.2 of [14], which is the first fast convergence result for a (non-truncated) semismooth Newton method, but the truncated situation is not discussed.

III. The TNCG for Linear Classification

At each iterate $\boldsymbol{w}_k$, where k is the iteration index, a Newton method finds an update direction by minimizing the following second-order approximation

$$Q_k(\boldsymbol{s}) \approx \nabla f(\boldsymbol{w}_k)^T\boldsymbol{s} + \frac{1}{2}\boldsymbol{s}^T\nabla^2 f(\boldsymbol{w}_k)\boldsymbol{s}. \tag{3}$$

As $f \in LC^1$, $\nabla^2 f$ is the generalized Hessian; see Section IV-B. Because of the convexity of (1), this minimization is equivalent to solving the Newton equation

$$\nabla^2 f(\boldsymbol{w}_k)\boldsymbol{s} = -\nabla f(\boldsymbol{w}_k). \tag{4}$$

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

$$\begin{aligned} \boldsymbol{g} &:= \nabla f(\boldsymbol{w}) = \boldsymbol{w} + C\sum\nolimits_{i=1}^{l} \xi'(y_i\boldsymbol{w}^T\boldsymbol{x}_i)y_i\boldsymbol{x}_i \\ H &:= \nabla^2 f(\boldsymbol{w}) = I + CX^TDX, \end{aligned} \tag{5}$$

where I is the identity matrix, $X = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_l]^T$ is the data matrix and D is a diagonal matrix with $D_{ii} = \xi''(y_i\boldsymbol{w}^T\boldsymbol{x}_i)$. The linear system (4) is difficult to solve because of the possible high dimensionality of $\boldsymbol{w}_k$. Thus the CG method is applied to avoid the explicit forming of the Hessian and employing instead the following Hessian-vector products

$$H\boldsymbol{s} = (I + CX^TDX)\boldsymbol{s} = \boldsymbol{s} + CX^T(D(X\boldsymbol{s})). \tag{6}$$

Therefore, each Newton iteration (called an outer iteration from now on) involves an inner iterative procedure of some CG steps, each of which conducts a Hessian-vector product.

Unfortunately, accurately solving the Newton equation (4) may expensively require many CG steps. As we will show later, CG steps are the bottleneck of the Newton method to solve (1). To reduce the number of CG steps, TN methods are applied to solve the Newton equation approximately. This approximation is controlled by the CG inner termination criterion. We will show that this choice has a direct and great impact on the convergence speed. Assume $\boldsymbol{s}_k^1, \boldsymbol{s}_k^2, \ldots$ are inner CG iterates. CG is stopped at a step j whenever $\boldsymbol{s}_k^j$ satisfies a truncation rule as

$$ratio(\boldsymbol{s}_k^j) \le \eta_k, \tag{7}$$

where the left side is the actual condition to be checked, usually a *ratio* between two terms, and $\eta_k \in (0,1)$ is the *forcing sequence*. The resulting $\boldsymbol{s}_k^j$ is then called $\boldsymbol{s}_k$.

To ensure the convergence of TNCG, by following most existing optimization methods, a globalization procedure is needed. Usually this means to find a suitable step size ω_k so that $\omega_k \boldsymbol{s}_k$ is used to update the iterate $\boldsymbol{w}_k$. Two major globalization techniques are line search and Trust-Region (TR). While TR may be more stable, it is more sophisticated (see final discussion in Section VIII). We therefore consider the easiest line search, Armijo (8). It finds the largest $\omega_k \in \{1, \delta, \delta^2, \dots\}$ with $\delta \in (0,1)$ such that the function value is sufficiently decreased, satisfying the following condition, with $\gamma \in (0,1)$,

$$f(\boldsymbol{w}_k + \omega_k \boldsymbol{s}_k) \le f(\boldsymbol{w}_k) + \gamma \omega_k {\boldsymbol{g}_k}^T \boldsymbol{s}_k. \tag{8}$$

Our implementation of TNCG is given in Algorithm 1.

Algorithm 1: Truncated Newton Conjugate Gradient

```
  Input: $\boldsymbol{w}_1 \in \mathbb{R}^n$ starting point
1 for $k = 1, 2, \dots$ do
2     compute the direction $\boldsymbol{s}_k$ by approximately solving
      (4) with a CG method, until (7) is satisfied
3     compute a step length $\omega_k$ by an Armijo line search
      technique (8)
4     $\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + \omega_k \boldsymbol{s}_k$
```

We now discuss the complexity of the whole procedure. From (6) the cost per outer iteration is roughly

$$\mathcal{O}(nl) \times (\#\text{CG steps} + 2) + \text{cost of deciding the step size}, \tag{9}$$

where $\mathcal{O}(nl)$ is the cost associated with each evaluation of function, gradient or Hessian-vector product. If X is sparse, the term $\mathcal{O}(nl)$ above might be replaced by the number of non-zero elements (#nnz) in matrix X. In most optimization methods the cost of deciding the step size is relatively smaller, so the complexity is proportional to the total number of CG steps. In fact, it has been shown in Section 2.1 of [9] (see also Section A.3 of the supplementary) that Armijo requires $\mathcal{O}(l)$ operations for each new ω. This means that the cost of deciding the step size is not the computational bottleneck.

IV. Global and Local Convergence of TNCG

In this section we relax f to be any $f \in \mathrm{LC}^1$, rather than the particular form in (1). Assumptions needed on f will be clarified in each theorem statement.

A. Global Convergence by Treating TNCG as a Common-Directions Algorithm

At the current iteration k, a common-directions algorithm computes $\boldsymbol{s}_k$ by combining m different directions $\{\boldsymbol{d}_k^1, \dots, \boldsymbol{d}_k^m\}$ to minimize the quadratic approximation of f. In the field of empirical risk minimization, in [4] they developed a framework that provides global and local convergence results for common-directions methods that satisfy Assumption 1. Here we will show for the first time that even the TNCG method can be seen as a special common-directions algorithm.

Assumption 1. *At each iteration k a common-directions algorithm computes a direction $\boldsymbol{s}_k$ such that*

$$\min_{\boldsymbol{\alpha}_k} {\boldsymbol{g}_k}^T \boldsymbol{s}_k + \frac{1}{2} \boldsymbol{s}_k^T B_k \boldsymbol{s}_k \qquad s.t.\ \boldsymbol{s}_k = P_k \boldsymbol{\alpha}_k, \tag{10}$$

where $P_k := [\boldsymbol{d}_k^1 | \dots | \boldsymbol{d}_k^m] \in \mathbb{R}^{n \times m}$ and $\boldsymbol{g}_k \in \{\boldsymbol{d}_k^1, \dots, \boldsymbol{d}_k^m\}$. The iterate update is $\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + \omega_k \boldsymbol{s}_k$, where ω_k is the step-size computed by Armijo (8). In addition, B_k is a positive definite matrix and is bounded, i.e. there exist two constants $M_1, M_2 > 0$ such that $M_1 \ge \|B_k\| \ge M_2 \ \forall k$.

Note that their result is rather general since B_k does not need to be the Hessian, but can be any bounded positive definite matrix. To obtain the global convergence result, the gradient must be one of the common-directions. In Theorem 3.2 and Corollary 3.1 of [4] they show the following theorem.

Theorem 1. *Let $f \in LC^1$ with a positive Lipschitz constant and $\{\boldsymbol{w}_k\}$ be generated by a method that satisfies Assumption 1. Assume that the following level set*

$$\mathcal{L}_1 := \{\boldsymbol{w} \in \mathbb{R}^n : f(\boldsymbol{w}) \le f(\boldsymbol{w}_1)\} \tag{11}$$

is compact. Then the minimum of the norm of gradients of the iterates vanishes at an $\mathcal{O}(1/\epsilon)$ rate, i.e. $\min_{0 \le j \le k} \|\boldsymbol{g}_j\| = \mathcal{O}(1/\sqrt{k+1})$ and

$$\lim_{k \to \infty} \|\boldsymbol{g}_k\| = 0. \tag{12}$$

In addition, if f is strongly convex the function values linearly converge, i.e. it takes $\mathcal{O}(log(1/\epsilon))$ to get an ϵ-accurate solution satisfying $f(\boldsymbol{w}_k) - f(\boldsymbol{w}^*) \le \epsilon$.

Note that the framework from [4] assumes that m is fixed, but all their results are still valid for any bounded m (see the supplementary for a discussion). Since CG method terminates in a number of steps bounded by the dimensionality of the linear system (4), the choice of the termination criteria (7) will not affect results contained in this subsection.

Now, to show that Algorithm 1 satisfies Assumption 1 we first need to recall that the CG method is designed to minimize a strictly convex quadratic function, as in (10). In particular, this is obtained by combining a set of directions $\{\boldsymbol{d}_k^1, \dots, \boldsymbol{d}_k^m\}$ that are conjugate with respect to B_k, i.e. for any $\boldsymbol{d}_k^i, \boldsymbol{d}_k^j$ we have ${\boldsymbol{d}_k^i}^T B_k \boldsymbol{d}_k^j = 0$ (see Lemma II of the supplementary and [15]). Once $\boldsymbol{s}_k$ is obtained, Algorithm 1 is then employing an Armijo line search along $\boldsymbol{s}_k$. Now to satisfy the rest of the assumptions on f and B_k we refer to f as defined in (1).

- In [4], they assume f to have a globally Lipschitz gradient, as in the case of f defined in (1). For $f \in \mathrm{LC}^1$, this assumption is not needed if we have assumed that the level set (11) is compact. In this case, ∇f is globally Lipschitz on $\mathcal{L}_1$. In [4] they also assume that f is bounded from below. We have this from a compact $\mathcal{L}_1$.
- Since in Algorithm 1 we approximately solve (4) employing the Hessian (or the generalized one) from (5), for the two losses in (2), we have that H_k is bounded (see Lemma I of the supplementary). This also implies that the minimum and maximum eigenvalue of H_k (respectively $\lambda_{\min}(H_k)$ and

$\lambda_{\max}(H_k)$) are bounded. In particular, from (5) we have that there exist two constants, $M_1, M_2 > 0$, such that

$$\begin{aligned} M_2 = 1 \le \lambda_{\min}(H_k) &\le \lambda_{\max}(H_k) \\ &\le 1 + C\lambda_{\max}(X^T D X) = M_1. \end{aligned} \tag{13}$$

- From CG method's details we have that $\boldsymbol{d}_k^1 = -\boldsymbol{g}_k$.
- f in (1) is strongly convex by definition. Further, with the regularization term $\boldsymbol{w}^T\boldsymbol{w}$ and $\xi(\cdot) \in \mathrm{LC}^1$, f in (1) is in LC^1 with a positive Lipschitz constant.

Thus, Assumption 1 and the assumptions of Theorem 1 are satisfied, which means that Algorithm 1 is globally convergent and the sequence $\{f(\boldsymbol{w}_k)\}$ is linear convergent to $f(\boldsymbol{w}^*)$.

Global convergence of Algorithm 1 can also be proved by following a more classical way (see Proposition III and Theorem IV from the supplementary). First we need to prove that CG method obtains a direction $\boldsymbol{s}_k$ for which there exist two constants $a_1 > 0$ and $a_2 > 0$ such that

$$\boldsymbol{g}_k{}^T\boldsymbol{s}_k \le -a_1\|\boldsymbol{g}_k\|^2 \quad \text{and} \quad \|\boldsymbol{s}_k\| \le a_2\|\boldsymbol{g}_k\|. \tag{14}$$

Then, from Armijo line search properties [15] we obtain (12).

B. Q-Superlinear Local Convergence

Since ∇f is locally Lipschitz, from Rademacher's theorem it is differentiable almost everywhere [7]. We indicate by $D_{\nabla f}$ the set of points in which ∇f is differentiable. For any $\boldsymbol{w}$ we define the B-subdifferential of ∇f at $\boldsymbol{w}$ as the nonempty compact set

$$\partial_B \nabla f(\boldsymbol{w}) := \{H \in \mathbb{R}^{n\times n} : H = \lim_{\substack{\boldsymbol{w}_j \to \boldsymbol{w} \\ \boldsymbol{w}_j \in D_{\nabla f}}} \nabla^2 f(\boldsymbol{w}_j)\}.$$

Then the Clarke generalized differential $\partial\nabla f$ (or equivalently the generalized Hessian) at $\boldsymbol{w}$ is the convex hull of $\partial_B \nabla f$. We say that ∇f is semismooth at $\boldsymbol{w}$ if the limit

$$\lim_{\boldsymbol{s}_j \to \boldsymbol{s}, t_j \to 0^+} H_j \boldsymbol{s}_j \quad \text{exists} \quad \forall H_j \in \partial\nabla f(\boldsymbol{w} + t_j\boldsymbol{s}_j).$$

If ∇f is semismooth it can be proved that the above limit is equal to the directional derivative of ∇f at $\boldsymbol{w}$ in the direction $\boldsymbol{s}$. Semismooth functions are a particular class of locally Lipschitz continuous function for which generalized differentials define a legitimate Newton approximation scheme (see Chapter 7 of [16]). The class of semismooth functions is very broad, in particular ∇f is semismooth for both losses defined in (2) because they both have piece-wise smooth gradients. In addition, we say that ∇f is BD-regular at $\boldsymbol{w}$ if all elements in $\partial\nabla f(\boldsymbol{w})$ are nonsingular. Note that strong convexity implies BD-regularity.

In this subsection, we study local convergence by assuming that the *ratio* employed in the stopping rule (7) is

$$ratio = \frac{\|\boldsymbol{g}_k + H_k\boldsymbol{s}_k^j\|}{\|\boldsymbol{g}_k\|} = \frac{\|\boldsymbol{r}\|}{\|\boldsymbol{g}_k\|}, \text{where } \boldsymbol{r} := -\boldsymbol{g}_k - H_k\boldsymbol{s}_k^j \tag{15}$$

is the residual maintained though the CG procedure (indices j and k omitted) and $H_k \in \partial\nabla f(\boldsymbol{w}_k)$ is the generalized Hessian considered in Algorithm 1. This condition of checking the ratio between the residual and the right-hand side of the Newton equation is probably the most commonly used inner stopping criterion [15]. In Section V we will show all the other criteria and how to obtain the same local results.

Now to obtain Q-SL local convergence, we first need to prove that for the sequence $\{\boldsymbol{w}_k + \boldsymbol{s}_k\}$ we have the following limit (17) (see Theorem 3 of [11]).

Lemma 1. *Let ∇f be semismooth and BD-regular. Let $\{\boldsymbol{w}_k\}$ be a generic sequence convergent to a critical point $\boldsymbol{w}^*$. Further at each $\boldsymbol{w}_k$ we generate a $\boldsymbol{s}_k$ by solving the Newton linear equation* (4) *to satisfy the condition* (7) *where the $ratio$ employed is* (15)*. Then there exist $\delta > 0, M > 0$ such that for any $\boldsymbol{w}_k \in N(\boldsymbol{w}^*, \delta)$ we have*

$$\|\boldsymbol{w}_k + \boldsymbol{s}_k - \boldsymbol{w}^*\| \le o(\|\boldsymbol{w}_k - \boldsymbol{w}^*\|) + M\eta_k\|\boldsymbol{w}_k - \boldsymbol{w}^*\|. \tag{16}$$

Further if $\eta_k \to 0$ in generating $\boldsymbol{s}_k$, then we have

$$\lim_{k\to\infty} \frac{\|\boldsymbol{w}_k + \boldsymbol{s}_k - \boldsymbol{w}^*\|}{\|\boldsymbol{w}_k - \boldsymbol{w}^*\|} = 0. \tag{17}$$

Once we prove this, we can exploit a very general result for semismooth functions (see Theorem 3.2 of [17]). It ensures that any line search based algorithm will eventually accept the initial step-size $\omega_k = 1$ if the limit (17) can be proved for the direction $\boldsymbol{s}_k$.

Theorem 2. *Let f be strongly convex and ∇f be semismooth. Let $\mathcal{L}_1$ defined in* (11) *be compact. Let $\{\boldsymbol{w}_k\}$ be generated by Algorithm 1 with $\gamma \in (0, \frac{1}{2})$ and $\eta_k \to 0$. Then:*
(1) there exists $\hat{k}$ such that (8) *is satisfied with $\omega_k = 1 \forall k \ge \hat{k}$.*
(2) $\{\boldsymbol{w}_k\}$ is Q-SL convergent to a critical point $\boldsymbol{w}^*$.

Proofs of theorems in this section are in Section I.3 of the supplementary. This is the place where it is possible to notice the differences (e.g. local Lipschitz gradient) between our theory and that of [12].

V. Truncation Criteria in TNCG

Truncation (or CG inner stopping) criteria as (7) are explored here. In this paper, for the first time on linear classification problems, a study on truncation criteria is performed.

A. The Importance of Truncation Criteria

By a simple example of checking the settings in some popular software, we point out that without a careful choice of the inner stopping criterion, the convergence of the TNCG method can be very slow. In Figure 1 we show the total number of CG steps needed to solve the training problem (1) on the dataset `kdd2010b`. Note that as showed in (9) the running time is always proportional to the total number of CG steps. On the y axis we report the relative reduction of the function value, computed by $\frac{f(\boldsymbol{w}_k) - f(\boldsymbol{w}^*)}{f(\boldsymbol{w}^*)}$, where $\boldsymbol{w}^*$ is the optimal solution of (1). The hyper-parameter C used is equal to $100C_{\text{Best}}$, where C_{Best} is the value that yields the best accuracy on the validation set (see Section VII for details). In Figure 1 we compare

- Scikit: This is the TNCG method implemented in the package Scikit-learn [18], version 0.22. The CG truncation criterion is a l1-norm variant of (15) with

$$ratio = \frac{\|\boldsymbol{g}_k + H_k\boldsymbol{s}_k^j\|_1}{\|\boldsymbol{g}_k\|_1} \tag{18}$$

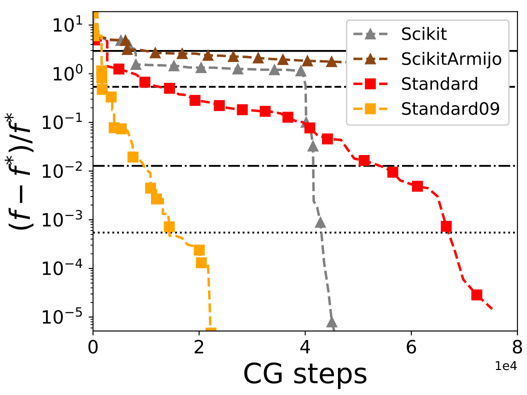


Fig. 1: A comparison of truncation criteria used in some popular software on the convergence of TNCG methods for logistic regression. The dataset `kdd2010b` is considered with $C = 100 \times C_{\text{Best}}$. The y-axis is the relative reduction of function value in log-scale. The x-axis is the cumulative number of CG steps. We give a mark for every five Newton iterations. See the explanation of horizontal lines in (19).

where $H_k \in \partial\nabla f(\boldsymbol{w}_k)$ and $\eta_k = \min\{0.5; \|\boldsymbol{g}_k\|_1^{0.5}\}$. They require Armijo and Wolfe conditions.

- ScikitArmijo: Same implementation as Scikit above, but using only the Armijo condition.
- Standard: This is the TNCG method implemented in the package LIBLINEAR [19]. They consider the TR technique, but we replace it with an Armijo line search. The criterion used there is (15) with $\eta_k = 0.1$.
- Standard09: Even for the simple Standard setting above, it is unclear what the threshold η_k should be. We arbitrary change it from 0.1 to 0.9 and see if the TNCG method performs similarly or not.

The four horizontal lines in Figure 1 indicate places where the following stopping condition is met respectively with $\epsilon = \{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$[1]

$$\|\boldsymbol{g}_k\| \leq \epsilon \frac{\min\{\#\text{pos}, \#\text{neg}\}}{l} \cdot \|\boldsymbol{g}_1\|, \tag{19}$$

where #pos, #neg are the numbers of positive- and negative-labeled instances respectively, and $\epsilon > 0$ is the thresholding constant that controls the precision of the training procedure. Note that (19) with $\epsilon = 10^{-2}$ is the outer stopping condition employed in LIBLINEAR. Thus the behavior before 10^{-1} and after 10^{-4} is not crucial, since the training would be stopped too early or too late. From Figure 1 we can observe:

- By changing only the inner stopping criterion, the overall convergence can be dramatically different. We see that Scikit and Standard require more than four times many CG steps as Standard09 to reach the third horizontal line, while ScikitArmijo does not even reach it.
- From Standard to Standard09, we see that even just the simple change of a constant in the truncation criteria might yield remarkable improvements in the convergence speed.
- In Figure 1, every mark indicates 5 outer Newton iterations. Besides Standard09, all others waste many CG steps in each early iteration. Apparently, the inner stopping criteria may be too strict in the early stage of the optimization process.

We conclude that without a careful choice of the inner stopping criterion, the TNCG method may converge slowly.

[1]In Figure 1 the fourth line is hidden by the y-axis lower limit.

B. Details on Truncation Criteria

We investigate truncation criteria by combining various *ratios* and *forcing sequences*. We check three different *ratio*:

- residual: this is the *ratio* used in (15), which evaluates the norm of the residual of equation (4) w.r.t. the norm of the gradient. The setting of comparing with the norm of the gradient $\|\boldsymbol{g}_k\|$ addresses a well-known issue of Newton methods: in early iterations the quadratic model is not a good local approximation of the function, so the Newton direction might not be consistently better than a simpler gradient-based direction. For this reason, over-solving (4) to better approximate the Newton direction might result in a waste of resources. Because $\|\boldsymbol{g}_k\|$ is not close to zero at early iterations, the relative setting in (15) avoids a too large ratio. Then the CG procedure can stop without accurately solving the Newton equation.
- residual$_{l1}$: the choice employed in Scikit-learn; see (18).
- quadratic: in [20] it was first introduced

$$ratio = \frac{(Q_j - Q_{j-1})}{Q_j / j}, \tag{20}$$

where $Q_j := Q(\boldsymbol{s}_k^j)$ and $Q_{j-1} := Q(\boldsymbol{s}_k^{j-1})$, and $Q(\cdot)$ is defined in (3). In (20) the reduction in the quadratic model at the current CG step $(Q_j - Q_{j-1})$ is compared with the average reduction per CG step (Q_j/j). The rationale of using (20) rather than (15) is that Q_j reflects what we are minimizing in applying a CG method. Note in addition that computing the quadratic ratio does not require any expensive extra computations, since from (15) it can be computed by

$$Q(\boldsymbol{s}_k^j) = -\frac{1}{2}\boldsymbol{s}_k^{j\,T}(\boldsymbol{r} - \boldsymbol{g}_k) = \frac{1}{2}\boldsymbol{s}_k^{j\,T} H_k \boldsymbol{s}_k^j + \boldsymbol{s}_k^{j\,T}\boldsymbol{g}_k \tag{21}$$

that just requires one subtraction and one inner product between vectors. Thus, its cost is roughly $\mathcal{O}(n)$, while the bottleneck in each CG step is still $\mathcal{O}(nl)$ required by each Hessian-vector product.

Regarding η_k, many different families of *forcing sequences* have been proposed [6], [21], [22], [23], [24]. In this work we mainly investigate the following three rules:

- constant: $\eta_k = c_0 \qquad \text{with } c_0 \in (0,1).$ (22)

  This is the η_k employed in LIBLINEAR in [9] with $c_0 = 0.1$. Selecting a suitable constant c_0 is never easy as indicated from the example in Section V-A.
- adaptive: to improve the constant setting (22), the following adaptive one was proposed in [6],

$$\eta_k = \min\{c_1; c_2\|\boldsymbol{g}_k\|^{c_3}\}, \tag{23}$$

where $c_1 \in (0,1)$, $c_2 > 0$, $c_3 \in (0,1]$. The rationale behind this adaptive setting is to distinguish global and local phases. As discussed above, at the beginning of the TN method, the iterate might be quite far from the solution, thus c_1 would be the minimum in (23). This value should stop the CG

ratio \ η_k	(23)	(24)	(22)
(15)	Q-SL	Q-SL	F-L
(18)	Q-SL	Q-SL	F-L
(20)	Q-SL	Q-SL	F-L

TABLE I: Local convergence results of different truncation rules. Q-SL= Q-SuperLinear. F-L = Linear convergence of the Function values.

procedure early enough to avoid over-solving (4), since it is also likely that in this stage the quadratic model might not be a good approximation of the function. On the other side, when iterates are approaching the solution, the minimum in (23) will select $c_2\|\boldsymbol{g}_k\|^{c_3}$ and lead the CG procedure to solve (4) till a good approximation of the Newton direction.

- adaptive$_{l1}$: $\eta_k = \min\{c_1; c_2\|\boldsymbol{g}_k\|_1^{c_3}\}$, (24)

  where $c_1 \in (0,1),\ c_2 > 0,\ c_3 \in (0,1]$. This is the η_k employed in Scikit-learn with $c_1 = 0.5, c_2 = 1, c_3 = 0.5$. It differs from (23) only in the use of the l1-norm.

Beyond the above rules, more complex *forcing sequences* like those proposed in [21], [23], [24] are available for general optimization problems. However, from some preliminary results we conduct, they are not performing consistently better than the adaptive families (23) and (24). The reason may be that the adaptive setting in (23), as it was designed, has distinguished well the global phase (i.e. early iterations) from the local phase (i.e. final iterations). Because (1) is convex, a good distinction between the two phases is often enough to achieve an efficient TNCG implementation. Moreover some of the proposed choices from the literature are very highly parametrized and/or might often lead to over-solve (4).

C. Theoretical Properties

In Table I we present the local convergence results of inner stopping criteria discussed in Section V-B. The first 2×2 square is pretty easy to fill, since (23) and (24) imply $\eta_k \to 0$ and Theorem 2 then leads to the Q-SL convergence. For criteria involving l1-norm we need

$$\begin{aligned}\|H_k\boldsymbol{s}_k^j + \boldsymbol{g}_k\|_2 &\leq \|H_k\boldsymbol{s}_k^j + \boldsymbol{g}_k\|_1 \\ &\leq \eta_k\|\boldsymbol{g}_k\|_1 \leq \sqrt{n}\eta_k\|\boldsymbol{g}_k\|_2,\end{aligned} \tag{25}$$

where $H_k \in \partial\nabla f(\boldsymbol{w}_k)$.

If the quadratic ratio (20) is considered, filling the first two entries of the last row in Table I is instead not straightforward. From some private communication with the author of [25], the result may have been given there, but that technical report is no longer available. Many steps of the proof may be found in [26], but not some of the most crucial ones. With the help of Dr. Nash we are able to prove Theorem 3, whose details are in the Appendix.

Theorem 3. *If we employ* (20) *as the truncation criteria of the CG procedure to solve* $H_k\ \boldsymbol{s} = \boldsymbol{g}_k$*, with* H_k *a symmetric positive definite matrix, then we get* $\|H_k\boldsymbol{s}_k^j + \boldsymbol{g}_k\| \leq N_k\sqrt{\eta_k}\|\boldsymbol{g}_k\|$*, where* $N_k = \sqrt{\frac{K_k\|H_k\|\cdot\|H_k^{-1}\|}{1-K_k}}$, $K_k = \left(\frac{\lambda_{\max}(H_k)-\lambda_{\min}(H_k)}{\lambda_{\max}(H_k)+\lambda_{\min}(H_k)}\right)^2$.

Note that in our case $H_k \in \partial\nabla f(\boldsymbol{w}_k)$ and since f is strongly convex H_k is symmetric positive definite. Moreover, from (13) and the fact that H_k is bounded we get that also $N_k,\ \forall k$ is bounded. Thus if $\eta_k \to 0$ by (23) or (24), then $N_k\sqrt{\eta_k} \to 0$ and the Q-SL convergence follows from Theorem 2.

For filling the last column of Table I we note that Q-L convergence is not proved in Section IV-B. First, when (17) is not satisfied (e.g., when η_k is a generic constant as in (22)), Theorem 3.2 of [17] cannot be applied. This is actually a crucial step for the proof of Theorem 2 (see Section I.3 of the supplementary). Second, the intermediate result (16) does not ensure Q-L convergence for any $\eta < 1$, but it only proves that there exists a $\eta < 1$ for which this is obtained. In fact, in [11] they provide a counterexample in which a Q-L sequence does not converge when (15) is applied with a generic $\eta_k < 1$. To the best of our knowledge, if no additional modifications on Algorithm 1 are added and η_k is a generic constant (less than 1), Q-L convergence of TNCG for linear classification is still an open question. On the other hand, from the newly discovered connection between TNCG and the framework in [4] we get from Theorem 1 the local linear convergence of the function value (F-L).

VI. Preconditioning in TNCG

Algorithm 2: PCG for solving (26). Assume M has not been factorized to EE^T.

```
Input: $\epsilon > 0, \eta_k > 0,\ M \in \mathbb{R}^{n\times n}$.
1  Let $\boldsymbol{s} = \boldsymbol{0}, \boldsymbol{r} = -\boldsymbol{g}_k, \boldsymbol{d} = \boldsymbol{z} = M^{-1}\boldsymbol{r}, \gamma = \boldsymbol{r}^T\boldsymbol{z}$
2  for $j = 1, 2, \dots$ do
3      $\boldsymbol{v} \leftarrow H_k\boldsymbol{d}$
4      $\alpha \leftarrow \boldsymbol{r}^T\boldsymbol{z}/(\boldsymbol{d}^T\boldsymbol{v})$
5      $\boldsymbol{s} \leftarrow \boldsymbol{s} + \alpha\boldsymbol{d}$
6      $\boldsymbol{r} \leftarrow \boldsymbol{r} - \alpha\boldsymbol{v}$
7      $\boldsymbol{z} \leftarrow M^{-1}\boldsymbol{r}$
8      if (7) holds with ratio = (27) then
9          return $\boldsymbol{s}_k = \boldsymbol{s}$
10     $\gamma^{\text{new}} \leftarrow \boldsymbol{r}^T\boldsymbol{z}$
11     $\beta \leftarrow \gamma^{\text{new}}/\gamma$
12     $\boldsymbol{d} \leftarrow \boldsymbol{z} + \beta\boldsymbol{d}$
13     $\gamma \leftarrow \gamma^{\text{new}}$
```

To reduce the total amount of inner CG steps needed to solve the system (4) we can apply a preconditioner matrix and solve instead the equivalent system

$$E^{-1}H_kE^{-T}\hat{\boldsymbol{s}}_k = -E^{-1}\boldsymbol{g}_k, \tag{26}$$

where E is a symmetric nonsingular matrix. Once the preconditioned solution $\hat{\boldsymbol{s}}_k$ is obtained, we can get the original $\boldsymbol{s}_k$ by employing $\boldsymbol{s}_k = E^{-T}\hat{\boldsymbol{s}}_k$. The idea behind the preconditioning techniques [27] is that of considering a matrix $M = EE^T \approx H_k$ to obtain a condition number of $E^{-1}H_kE^{-T}$ to be as closer as possible to 1 and/or to cluster its eigenvalues [28]. When the approximation is good, this technique would lead to a new system (26) that is easier to be solved than the original non-preconditioned one (4).

To solve (26) one could either use the original CG procedure (see Algorithm III of the supplementary) on it or, as shown in [29], avoid the factorization of $M = EE^T$ and use instead Algorithm 2, which has iterates $\boldsymbol{s}_k^j$ instead of $\hat{\boldsymbol{s}}_k^j$. Note that Algorithm 2 is still solving the preconditioned system (26) even if it operates mostly with non-preconditioned variables. In [5] it is has been shown that when the factorization is actually available, Algorithm 2 is still preferable, because, even if the two algorithms have basically the same cost, Algorithm 2 may be numerically more stable. For sake of simplicity in Algorithm 2 and in this section, we will not use the iteration counter j and k will only be reported on H_k and $\boldsymbol{g}_k$. Now, from the properties of the CG procedures (see Section III of the supplementary) we have that $\hat{\boldsymbol{r}} := E^{-1}\boldsymbol{r}$, $\hat{\boldsymbol{g}}_k := E^{-1}\boldsymbol{g}_k$ and $\boldsymbol{z} := M^{-1}\boldsymbol{r}$. Thus

$$\|\hat{\boldsymbol{r}}\|_2 = \|\boldsymbol{r}\|_{M^{-1}} = \sqrt{\boldsymbol{r}^T M^{-1}\boldsymbol{r}} = \sqrt{\boldsymbol{r}^T\boldsymbol{z}},$$
$$\|\hat{\boldsymbol{g}}_k\|_2 = \|\boldsymbol{g}_k\|_{M^{-1}} = \sqrt{\boldsymbol{g}_k^T M^{-1}\boldsymbol{g}_k},$$

where given a vector $\boldsymbol{a}$ and a matrix A we have $\|\boldsymbol{a}\|_A := \sqrt{\boldsymbol{a}^T A\boldsymbol{a}}$. Then the classical *ratio* (15) employed in the stopping criterion of Algorithm 2 becomes

$$\frac{\|\hat{\boldsymbol{r}}\|}{\|\hat{\boldsymbol{g}}_k\|} = \frac{\|\boldsymbol{r}\|_{M^{-1}}}{\|\boldsymbol{g}_k\|_{M^{-1}}} = \frac{\sqrt{\boldsymbol{r}^T M^{-1}\boldsymbol{r}}}{\sqrt{\boldsymbol{g}_k^T M^{-1}\boldsymbol{g}_k}} = \frac{\sqrt{\boldsymbol{r}^T\boldsymbol{z}}}{\sqrt{\boldsymbol{g}_k^T M^{-1}\boldsymbol{g}_k}}. \quad (27)$$

Note that also the other *ratios* introduced in Section V-B can be employed in the preconditioned case. In particular, the quadratic ratio (20) can be computed in the same way by (21) for both the preconditioned and non-preconditioned cases. In fact, since Algorithm 2 is working with non-preconditioned variables like $\boldsymbol{s}, \boldsymbol{r}$ and $\boldsymbol{g}_k$, we still have $\boldsymbol{r} = -H_k\boldsymbol{s} - \boldsymbol{g}_k$.

A. The Preconditioning by [5]

In this work we will apply the preconditioning proposed in [5]. We first recall that a diagonal preconditioner can be obtained by extracting all the diagonal elements in the Hessian

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

Once the above diagonal preconditioner is built, the original idea from [5] was that of solving in parallel the preconditioned system and the original one at each Newton iteration, and use the direction $\boldsymbol{s}$ of the procedure that terminated first. As they extensively studied both theoretically and numerically, a robust single thread alternative was instead that of combining the above diagonal preconditioner $\bar{M}$ with the identity matrix

$$M = \alpha \times \bar{M} + (1-\alpha) \times I, \quad (28)$$

where $\alpha \in (0, 1)$ is the scalar that weights the combination.

We will now show that the cost of using a diagonal preconditioner at each Newton iteration is irrelevant if compared with (9). From (5) we get that

$$(H_k)_{jj} = 1 + C\sum\nolimits_i D_{ii}X_{ij}^2, \quad (29)$$

which means that constructing the above diagonal preconditioner costs $\mathcal{O}(nl)$. From Algorithm 2 we can see that at each CG step we additionally need to compute $M^{-1}\boldsymbol{r}$, which cost $\mathcal{O}(n)$. Thus, the extra cost of using the diagonal preconditioner is simply $\mathcal{O}(n) \times (\text{\# CG steps}) + \mathcal{O}(nl)$.

B. Global and Local Convergence in the Preconditioned Case

In this section we assume that the minimum and the maximum eigenvalue of M^{-1} (respectively $\lambda_{\min}(M^{-1})$ and $\lambda_{\max}(M^{-1})$) are bounded. This means that also $\lambda_{\min}(M)$ and $\lambda_{\max}(M)$ are bounded. In addition we remind that given a vector $\boldsymbol{a}$ and a matrix A we have

$$\lambda_{\min}(A)\|\boldsymbol{a}\|_2 \le \|\boldsymbol{a}\|_A \le \lambda_{\max}(A)\|\boldsymbol{a}\|_2. \quad (30)$$

Note that the factorization $M = EE^T$ is only required for the convergence analysis in this section, but never in practice.

We first point out that the global convergence result is still valid even if in Algorithm 1 instead of solving the original Newton equation (4) we solve the preconditioned one (26). In fact, the two systems are equivalent, e.g. $\hat{\boldsymbol{s}}$ is the solution of (26) if and only if $\boldsymbol{s} = E^{-1}\hat{\boldsymbol{s}}$ is the solution of (4). In particular, applying Algorithm 2 is equivalent to employing Algorithm III of the supplementary and then obtaining $\boldsymbol{s}$ from $\boldsymbol{s} = E^{-1}\hat{\boldsymbol{s}}$ (see Section III of the supplementary or [29]). This means that we can apply Proposition III of the supplementary on the preconditioned system (26) and obtain

$$\hat{\boldsymbol{g}}_k^T\hat{\boldsymbol{s}} \le -a_1\|\hat{\boldsymbol{g}}_k\|^2 \quad \text{and} \quad \|\hat{\boldsymbol{s}}\| \le a_2\|\hat{\boldsymbol{g}}_k\|. \quad (31)$$

From (31), (30), and with definitions of $\hat{\boldsymbol{g}}_k$ and $\hat{\boldsymbol{s}}$, we get both

$$\begin{aligned} \boldsymbol{g}_k^T\boldsymbol{s} = \hat{\boldsymbol{g}}_k^T\hat{\boldsymbol{s}} &\le -a_1\|\hat{\boldsymbol{g}}_k\|^2 = -a_1\boldsymbol{g}_k^T M^{-1}\boldsymbol{g}_k \\ &= -a_1\|\boldsymbol{g}_k\|_{M^{-1}}^2 \le -a_1\lambda_{\min}(M^{-1})\|\boldsymbol{g}_k\|_2^2, \end{aligned} \quad (32)$$

$$\begin{aligned} \lambda_{\min}(M)\|\boldsymbol{s}\|_2 \le \|\boldsymbol{s}\|_M = \|\hat{\boldsymbol{s}}\|_2 &\le a_2\|\hat{\boldsymbol{g}}_k\|_2 \\ &= a_2\|\boldsymbol{g}_k\|_{M^{-1}} \le a_2\lambda_{\max}(M^{-1})\|\boldsymbol{g}_k\|_2. \end{aligned} \quad (33)$$

Now, from the fact that $\lambda_{\min}(M), \lambda_{\max}(M), \lambda_{\min}(M^{-1})$ and $\lambda_{\min}(M^{-1})$ are bounded and together with (32) and (33) we get that there exist two new positive constants $\hat{a}_1, \hat{a}_2$ such that

$$\boldsymbol{g}_k^T\boldsymbol{s} \le -\hat{a}_1\|\boldsymbol{g}_k\|^2 \quad \text{and} \quad \|\boldsymbol{s}\| \le \hat{a}_2\|\boldsymbol{g}_k\|.$$

This means that we can apply Theorem IV of the supplementary to prove global convergence.

Next we discuss local convergence results. Note that even if Algorithm 2 is employing non-preconditioned variables like $\boldsymbol{s}$ and $\boldsymbol{r}$, the *ratio* (27) is focusing on the norm of the residual of the preconditioned system $\hat{\boldsymbol{r}}$ and on the norm of $\hat{\boldsymbol{g}}_k$. For this reason, to ensure that local Q-SL convergence results are not harmed by the application of the preconditioning, we must connect the *ratio* (27) to the one in (15). From (30) we have

$$\begin{aligned} \lambda_{\min}(M^{-1})\|\boldsymbol{r}\|_2 \le \|\boldsymbol{r}\|_{M^{-1}} \le \lambda_{\max}(M^{-1})\|\boldsymbol{r}\|_2 \\ \lambda_{\min}(M^{-1})\|\boldsymbol{g}_k\|_2 \le \|\boldsymbol{g}_k\|_{M^{-1}} \le \lambda_{\max}(M^{-1})\|\boldsymbol{g}_k\|_2, \end{aligned} \quad (34)$$

from which we get that

$$\begin{aligned} \|\boldsymbol{r}\|_2 &\le \frac{1}{\lambda_{\min}(M^{-1})}\|\boldsymbol{r}\|_{M^{-1}} \le \frac{\eta_k}{\lambda_{\min}(M^{-1})}\|\boldsymbol{g}_k\|_{M^{-1}} \\ &\le \eta_k\frac{\lambda_{\max}(M^{-1})}{\lambda_{\min}(M^{-1})}\|\boldsymbol{g}_k\|_2. \end{aligned} \quad (35)$$

Thus, from (35), boundness of $\lambda_{\min}(M^{-1})$ and $\lambda_{\min}(M^{-1})$ and the norm equivalence results (34) and (25), one could either use (15), (18) or (27) in combination with (23) or (24) and still obtain Q-SL convergence.

Now, to prove that Q-SL convergence can be ensured also if the quadratic $ratio$ (20) is employed, we need to notice that

$$\begin{aligned}\hat{Q}_k(\hat{\boldsymbol{s}}) &:= (E^{-1}\boldsymbol{g}_k)^T\hat{\boldsymbol{s}} + \frac{1}{2}\hat{\boldsymbol{s}}^T(E^{-1}H_kE^{-T})\hat{\boldsymbol{s}} \\ &= \boldsymbol{g}_k{}^T\boldsymbol{s} + \frac{1}{2}\boldsymbol{s}^T H_k\boldsymbol{s} = Q_k(\boldsymbol{s}).\end{aligned}$$

This means that we can repeat the same proof of Theorem 3 for the system (26) and obtain

$$\|\boldsymbol{r}\|_{M^{-1}} \leq \sqrt{\eta_k}\hat{N}_k\|\boldsymbol{g}_k\|_{M^{-1}}, \tag{36}$$

where $\hat{N}_k = \sqrt{\frac{\hat{K}_k\|\hat{H}_k\|\cdot\|\hat{H}_k^{-1}\|}{1-\hat{K}_k}}$, $\hat{K}_k = \left(\frac{\hat{\lambda}_{\max}-\hat{\lambda}_{\min}}{\hat{\lambda}_{\max}+\hat{\lambda}_{\min}}\right)^2$, and $\hat{\lambda}_{\max}$ and $\hat{\lambda}_{\min}$ are respectively the highest and lowest eigenvalues of $\hat{H}_k = E^{-1}H_kE^{-T}$. Note that thanks to the fact that $\lambda_{\min}(M^{-1})$ and $\lambda_{\max}(M^{-1})$ are bounded, we also get that $\hat{\lambda}_{\min}$ and $\hat{\lambda}_{\max}$ are bounded. Together with (36) and (35) this ensures that if we use (20) in combination with (23) or (24) we can still obtain Q-SL convergence. Finally, in our case we can ensure that $\lambda_{\min}(M^{-1})$ and $\lambda_{\max}(M^{-1})$ are bounded thanks to the preconditioner defined in (28) and (29).

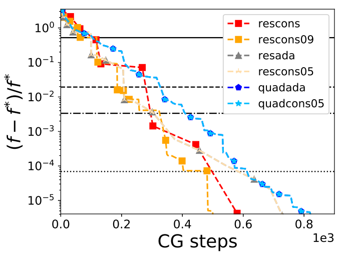
VII. Numerical Analysis on Truncation Criteria

We conduct experiments to analyze different inner stopping conditions. We focus on applying logistic regression on the three sets `kdd2010a`, `kdd2010b` and `yahookr`, while we leave data statistics, detailed settings and complete results including those of l2-loss linear SVM in Section IV of the supplementary. For a fair evaluation, all different settings are implemented based on the LIBLINEAR package. To simulate the practical use we conduct a five-fold cross-validation to select the regularization parameter C_{Best} that achieves the best validation accuracy. See the selected values in Table I of the supplementary. Then in experiments we consider $C = C_{\text{Best}} \times \{1, 100\}$. All other settings are the same as those used in Section V-A.

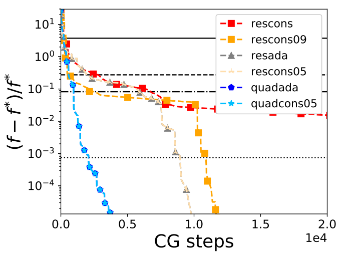
A. Selection of the Forcing Sequence in Truncation Criteria

In this section, we will first focus on the effect of using various *forcing sequences* on both the residual and the quadratic $ratio$. We propose a comparison between various constant thresholds in the *forcing sequence* (22) and the adaptive one (23). In fact, in Figure 1 we showed that even changing one single constant in the truncation rule might have a great impact on the speed of convergence. Moreover, we want to evaluate the effect of employing the adaptive *forcing sequence* instead of the constant one. Unless differently specified, the parameter setting used for both (23) and (24) is $(c_1 = 0.5, c_2 = 1, c_3 = 0.5)$, since this is the one used in Scikit-learn and suggested also in [15][2]. Note that we will use the standard l2-norm, but in the next section we will analyze the effect of changing the norm. In Sections IV.1-IV.3 of the supplementary is reported the isolated analysis on

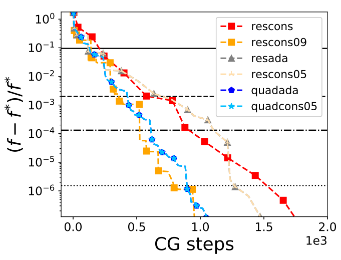
[2]Experiments with many other settings of c_1, c_2 and c_3 have been carried out, but there is no evidence of configurations that work consistently better than the standard one $(c_1 = 0.5,\ c_2 = 1,\ c_3 = 0.5)$.



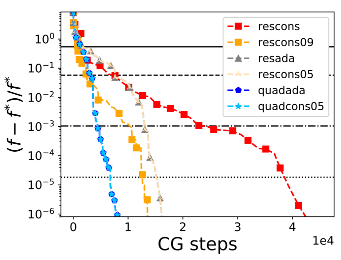
(a) `yahookr`, C_{Best}



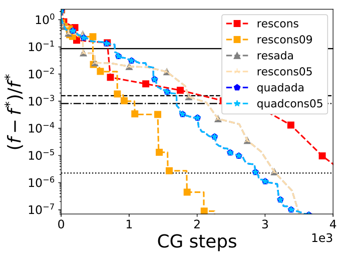
(b) `yahookr`, $100C_{\text{Best}}$



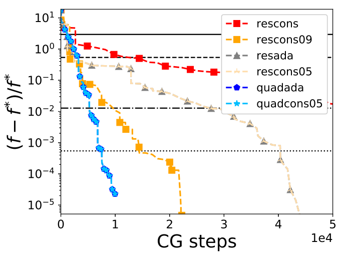
(c) `kdd2010a`, C_{Best}



(d) `kdd2010a`, $100C_{\text{Best}}$



(e) `kdd2010b`, C_{Best}



(f) `kdd2010b`, $100C_{\text{Best}}$

Fig. 2: A comparison of various forcing sequences in the inner stopping condition. We show the convergence of a truncated Newton method for logistic regression. See Figure 1 for an explanation of information in each sub-figure.

each different aspect (e.g. constant forcing sequence against adaptive one).

In Figure 2 we compare

- rescons: *ratio*= (15), η_k= (22), $c_0 = 0.1$ (in Figure 1 it was called Standard);
- rescons05: *ratio*= (15), η_k= (22), $c_0 = 0.5$;
- rescons09: *ratio*= (15), η_k= (22), $c_0 = 0.9$ (in Figure 1 it was called Standard09);
- resada: *ratio*= (15), η_k= (23);
- quadcons05: *ratio*= (20), η_k= (22), $c_0 = 0.5$;
- quadada: *ratio*= (20), η_k= (23).

From Figure 2 we can make the following observations:

- The setting rescons09 is faster than rescons in the large majority of the cases (the same can be observed also for the quadratic $ratio$; see Section IV.2 of the supplementary). This result seems to indicate that over-solving the Newton equation (4) by a small constant threshold has often a negative effect on the speed of convergence.
- The setting rescons09 is generally faster than rescons05 and rescons. However, in the case of `yahookr` with $C = 100C_{\text{Best}}$, rescons09 is slower than rescons05 apparently due to under-solving the Newton equation (4). Further, for l2-loss SVM we observe that rescons09 is inferior on some

datasets; see Section IV.12 from the supplementary. All these confirm again the sensitivity of the constant threshold c_0. On the other hand, if the quadratic *ratio* is considered, quadcons09 is not improving results w.r.t. the setting quadcons05 (see Section IV.2 of the supplementary). This is probably caused by the fact that the quadratic *ratio* was originally designed to fight over-solving, being often smaller than the residual one. From the more detailed comparison in Section IV.2 of the supplementary, the quadratic *ratio* seems to be less influenced by changes in the constant threshold c_0.

- The settings resada and quadada are respectively identical to rescons05 and quadcons05 in the large majority of the cases (see Section IV.3 of the supplementary). This means that the initial threshold $c_1 = 0.5$ is (almost) always smaller than $c_2\|\nabla f(\boldsymbol{w}_k)\|^{c_3}$.
- The setting quadada is overall performing better than resada (see the supplementary). The difference is remarkable in the case $C = 100C_{\text{Best}}$. In fact if we check the third horizontal line, quadada is always four times faster than resada. This gives another evidence of the fact that the quadratic *ratio* is well suited for fighting over-solving.
- Next we compare the two best settings rescons09 and quadada. By checking the third horizontal line we see that in the configuration $C = 100C_{\text{Best}}$, on both `yahookr` and `kdd2010a`, quadada is more than two times faster than rescons09. The opposite situation is instead happening when $C = C_{\text{Best}}$, on `yahookr` and `kdd2010b`, although rescons09 is no more than two times faster.

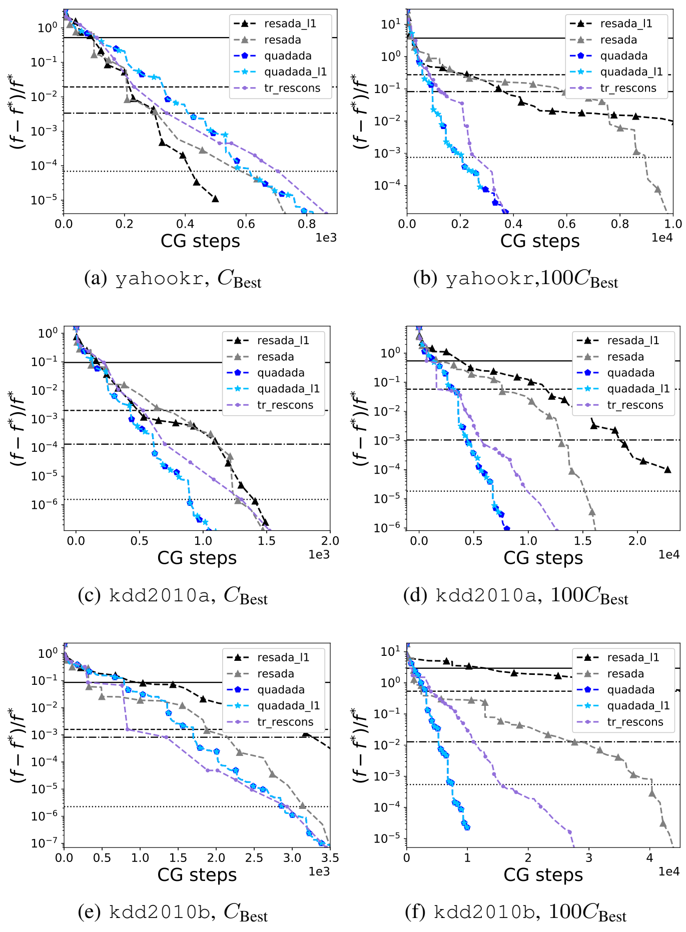
While our experiments with/without the adaptive forcing sequence are in most of the case identical, we decided to maintain the use for the following reasons. First, an adaptive forcing sequence is a safeguard for avoiding possible under-solving issues in the later stage of the optimization procedure. Second, from the theoretical properties in Section IV-B, local Q-SL convergence is ensured.

Finally, because the configuration of $C = 100C_{\text{Best}}$ leads to a more difficult optimization problem with the total amount of CG steps 10 times more than that of $C = C_{\text{Best}}$, it seems that quadada is overall a more effective choice.

B. *Comparison Between l2-Norm and l1-Norm and with TR*

In Figure 3 we now address the effect of switching between l2-norm and l1-norm on resada. The setting quadada should be very little influenced by this modification since the *ratio* remains the same and for the forcing sequence we have $\|\boldsymbol{v}\|_2 \leq \|\boldsymbol{v}\|_1 \ \forall \boldsymbol{v}$ and, as showed above, $c_1 = 0.5$ is already (almost) always smaller than $c_2\|\nabla f(\boldsymbol{w}_k)\|_2^{c_3}$. Moreover we will also check the TR implementation of LIBLINEAR. In Figure 3 we compare

- resada_l1: *ratio*= (18), η_k= (24). This setting is exacly the same as the approach ScikitArmijo in Section V-A. It is a simplification of the setting used in Scikit-learn without considering the Wolfe condition;
- resada: *ratio*= (15), η_k= (23);
- quadada_l1: *ratio*= (20), η_k= (24);
- quadada: *ratio*= (20), η_k= (23);
- tr_rescons: LIBLINEAR employs a TR globalization method (instead of the line search); see [9].



(a) `yahookr`, C_{Best} (b) `yahookr`,100C_{Best}
(c) `kdd2010a`, C_{Best} (d) `kdd2010a`, 100C_{Best}
(e) `kdd2010b`, C_{Best} (f) `kdd2010b`, 100C_{Best}

Fig. 3: A comparison between l2-norm and l1-norm and with the trust-region approach. We show the convergence of a truncated Newton method for logistic regression. See Figure 1 for an explanation of information in each sub-figure.

From Figure 3 we can make the following observations:

- The setting resada is overall performing better than resada_l1 (see the supplementary). This finding naturally suggests to question why in the software Scikit-learn [18] it is used the l1-norm instead of the l2-norm. On the other side, as we have seen in Section VII-A, resada is slower than quadada in most cases.
- As expected, quadada_l1 and quadada have exactly the same speed of convergence in the large majority of the cases. In contrast, we observe that the speed of resada_l1 and resada are sometimes significantly different.
- The setting quadada is generally slightly faster than tr_rescons, even if their speed of convergence is similar. In fact they both exploit the information obtained from the quadratic model. The setting tr_rescons uses the information to adjust the size of the trust region, while quadada uses it to terminate the CG procedure. What we have achieved here is that with a suitable inner stopping criterion, the simpler line search setting becomes comparable or even faster than the more complicated TR approach. On the other hand, resada and resada_l1 do not use the information from the quadratic model, and their convergence is slower.

We conclude that the truncation criteria that exploit a quadratic model are faster than those that do not use such information.

C. *Analysis via Conditions for Global Convergence*

In Section IV.5 of the supplementary we conduct a detailed analysis that combines theoretical and numerical aspects. Here we sum up the contribution by reporting the main discovery. When the regularization parameter C is large, the two conditions (14) (whose combination is called *angle condition*) are more difficult to be numerically satisfied by the resulting TN direction $\boldsymbol{s}_k$. Figures ix and x of the supplementary confirm that Algorithm 1 is facing some slow convergence issues when $\boldsymbol{s}_k$ has some more difficulties satisfying the *angle condition*.

VIII. Numerical Analysis on Preconditioning and Truncation Criteria

In this section we combine the study on truncation criteria with that of preconditioning. We focus on applying logistic regression on the four sets `yahookr`, `kdd2010a`, `kdd2010b` and `news20` while complete results, including those of l2-loss linear SVM, can be found in Sections IV.6, IV.7 and IV.13 of the supplementary. The experimental settings are the same as those in Section VII.

As explained at the end of Section VII-A, for some approaches in the experiment we decided to maintain the use of the adaptive *forcing sequence*, even if also in the preconditioned case (see Section IV.8 of the supplementary) this setting is (almost) always overlapping with the constant one with $c_0 = 0.5$.

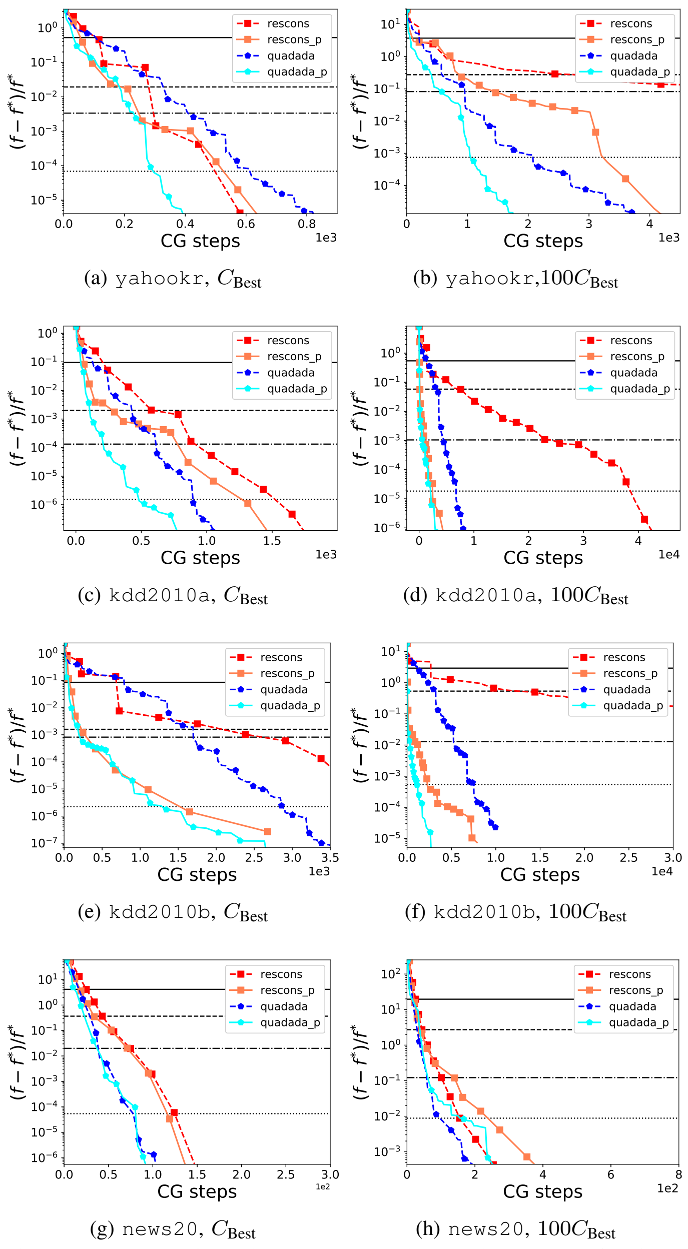
A. *Effect of Preconditioning on Residual and Quadratic Ratios*

Our first comparison is to check both quadada and rescons by applying the preconditioner from [5]. Recall that slow convergence occurred for rescons in Figure 1 (there called Standard) so subsequently we developed quadada. It is essential to check the situation after preconditioning. Note that rescons is the truncation rule employed in LIBLINEAR, even if in LIBLINEAR they employ a TR technique instead of a line search. In Figure 4 we compare

- rescons: *ratio*= (15), η_k= (22) $c_0 = 0.1$;
- rescons_p: *ratio*= (15), η_k= (22) $c_0 = 0.1$, preconditioned;
- quadada: *ratio*= (20), η_k= (23);
- quadada_p: *ratio*= (20), η_k= (23), preconditioned.

From Figure 4 we can make the following observations:

- The settings rescons_p and quadada_p are respectively faster than rescons and quadada in the large majority of the cases (see the supplementary). This result confirms that the preconditioner suggested in [5] is improving the speed of convergence for most of the ill-conditioned linear systems.
- On `news20` with $C = 100C_{\text{Best}}$ the non-preconditioned versions are slightly faster than the preconditioned ones. In fact, as detailed in [5], designing a preconditioner matrix that improves convergence in every case is very difficult. Nonetheless, the difference is not remarkable and this is one of the very few cases in which this situation is encountered.
- The setting quadada_p is generally faster than rescons_p (see the supplementary). This result seems to show that even in the preconditioned case the quadratic termination rule is obtaining better performances.



(a) `yahookr`, C_{Best} (b) `yahookr`,$100C_{\text{Best}}$

(c) `kdd2010a`, C_{Best} (d) `kdd2010a`, $100C_{\text{Best}}$

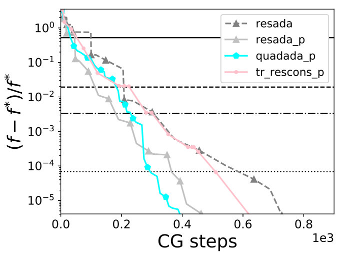
(e) `kdd2010b`, C_{Best} (f) `kdd2010b`, $100C_{\text{Best}}$

(g) `news20`, C_{Best} (h) `news20`, $100C_{\text{Best}}$

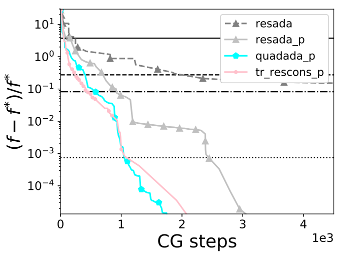
Fig. 4: A comparison between adaptive and constant forcing sequences in the preconditioned case. We show the convergence of a truncated Newton method for ξ_{LR} loss. See Figure 1 for an explanation of information in each sub-figure.

B. *Comparison with Trust Region and Other Rules*

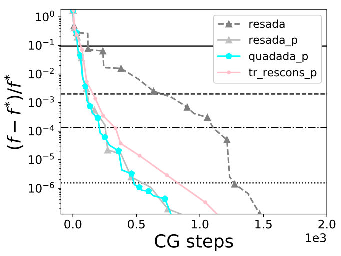
We now apply the preconditioning on resada, the l2-norm version of the truncation rule implemented in the package Scikit-learn. Alternatively, as resada is almost identical to rescons with $c_0 = 0.5$ (see Section IV.8 of the supplementary), we essentially extend the comparison in Section VIII-A by using a different constant forcing sequence. Moreover we will also check the TR implementation of LIBLINEAR. In Figure 5 we compare



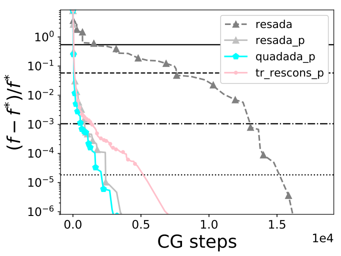
(a) yahookr, C_{Best}



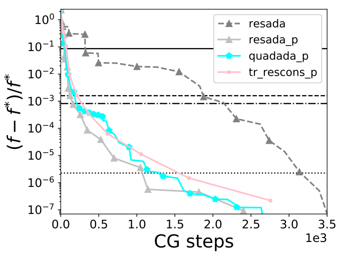
(b) yahookr,$100C_{\text{Best}}$



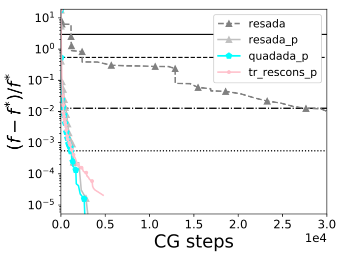
(c) kdd2010a, C_{Best}



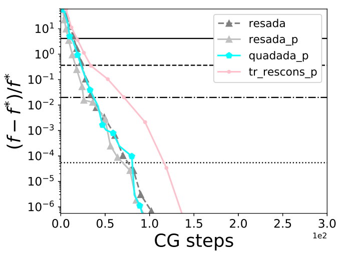
(d) kdd2010a, $100C_{\text{Best}}$



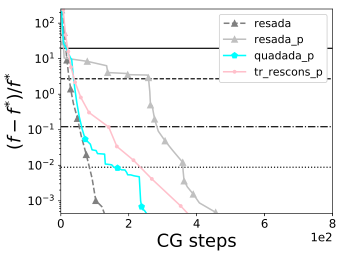
(e) kdd2010b, C_{Best}



(f) kdd2010b, $100C_{\text{Best}}$



(g) news20, C_{Best}



(h) news20, $100C_{\text{Best}}$

Fig. 5: A comparison between adaptive rules and the trust region approach in the preconditioned case. We show the convergence of a truncated Newton method for ξ_{LR} loss. See Figure 1 for an explanation of information in each sub-figure.

- resada: *ratio*= (15), η_k= (23);
- resada_p: *ratio*= (15), η_k= (23), preconditioned;
- quadada_p: *ratio*= (20), η_k= (23), preconditioned;
- tr_rescons_p: *ratio*= (15), η_k= (22), $c_0 = 0.1$, preconditioned; LIBLINEAR employs a TR globalization method (instead of the line search); see [5]. This is the latest LIBLINEAR verision 2.30.

From Figures 4 and Figure 5 we can observe:

- Exactly as in Figure 4, also in Figure 5 the preconditioned version (resada_p) is faster than the non-preconditioned one (resada) in the large majority of the cases (see the supplementary).
- In Figure 5 by checking news20 with $C = 100C_{\text{Best}}$ we can see that resada is remarkably faster than resada_p. Instead, by comparing quadada and quadada_p in Figure 4, the convergence speed is very similar until the third horizontal line, and even after that quadada_p is not much slower than quadada.
- The settings quadada_p and resada_p often have a very similar convergence speed (see the supplementary). Nonetheless, there are still some differences, especially when $C = 100C_{\text{Best}}$. In fact, if we check the second and the third horizontal line on news20, quadada_p is around three time faster than resada_p. On yahookr, we can instead check the fourth horizontal line to see that quadada_p is more than twice faster than resada_p.
- The setting quadada_p is generally faster than tr_rescons_p. In fact, on news20 we can already see a detachment between the two curves starting from the second horizontal line, while on kdd2010a it starts from the third line.

From Figure 4 and Figure 5 we conclude that the slow convergence issues pointed out in Figure 1 for both LIBLINEAR and Scikit-learn can either be solved by employing the quadratic termination rule (as resulting from the analysis of Section VII) or by applying the preconditioner designed in [5]. Moreover, the combination of the two modifications is more efficient and robust than each of the two alone. Finally, the line search version quadada is also generally faster than tr_rescons_p implemented in the latest LIBLINEAR 2.30.

Apart from the numerical pieces of evidence just showed, we also prefer quadada_p instead of tr_rescons_p because the line search is conceptually simpler than the TR. In fact, while the TR is both deciding the step size and the direction by solving a constrained quadratic problem, all the line search based algorithms first determine the direction and then the step size. Such a separation of concern is a desirable feature, because once the direction is obtained, the line search simply needs to address a one-dimension problem to choose a scalar value. This means that if we implement a backtracking line search (e.g. Armijo as in quadada), whenever a failure is encountered (i.e. the sufficient decrease condition is not satisfied), we only need to try a new step size, while the direction remains untouched. On the contrary in the case of TR algorithms, whenever a failure is encountered (i.e. there is not a good agreement between the actual and the predicted reduction), the TR radius needs to be updated and, thus, the resulting direction might be different from the previous one. In addition, the TR update is generally more complicated than the step size update. Finally, even if the procedure for computing the direction is the same in both versions of the TNCG algorithm (i.e. an internal CG procedure like Algorithm 2), in the case of the TR there is also the need of addressing the situation in which the norm of the direction is reaching the TR boundary.

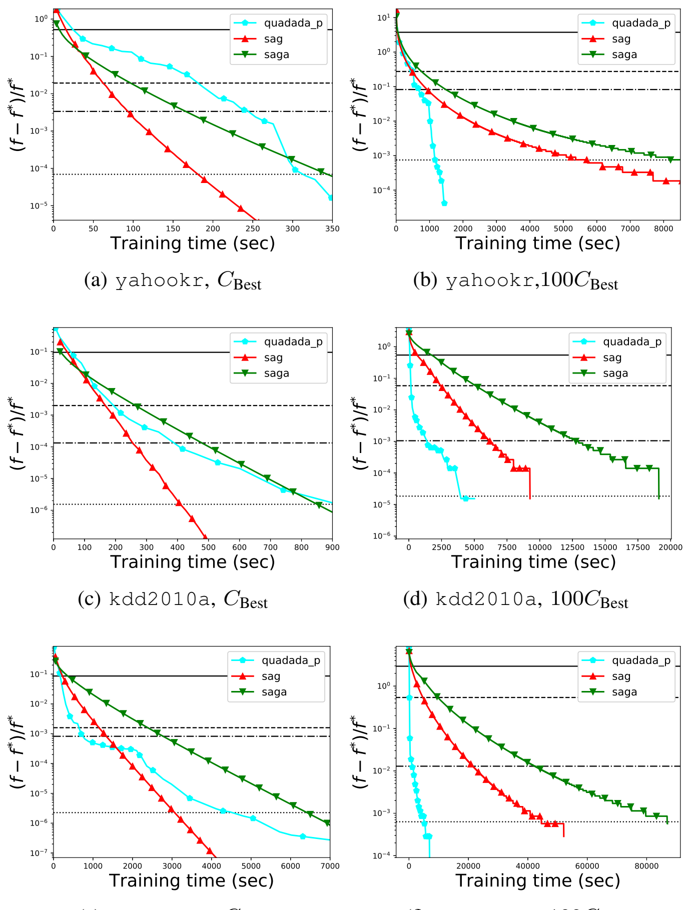
As a final remark we want to point out that in the preconditioned case the choice of the truncation rule does not cause the same dramatic difference in the speed of convergence as in the non-preconditioned case. Nonetheless, as showed in Figures 4 and 5 (see also the supplementary) quadada_p is generally more robust and effective than the other truncation rules. Moreover, the quadratic *ratio* has an additional advantage on the residual-based *ratios*: it does not rely on norms. As showed in Section VI, this makes it norm- and preconditioner-independent.

IX. Comparison with Other State-of-the-Art Methods

Finally in this section we show a comparison between the proposed version of the TNCG and two other state-of-the-art methods: SAG [30] and SAGA [31]. In particular, SAG and SAGA are both first-order approaches that implement a variant of the stochastic gradient method. Performances in this case have been measured in terms of the computational time (instead of the CG steps). The rest of the experimental settings is the same as that in Section VII, while details about the external software used for SAG and SAGA and about the timing measurements can be found in Section IV.14 of the supplementary. In Figure 6 we compare quadada_p (*ratio*= (20), η_k= (23), preconditioned), SAG and SAGA. From Figure 6 we can observe:

- The time required by each SAG or SAGA iteration is much more regular if compared with that required by each CG procedure. In fact, while the difficulty of computing a TN direction changes along with the optimization procedure, the time needed in each SAG or SAGA iteration is almost always the same from the beginning till the end.
- When $C = C_{\text{Best}}$, by looking at the general picture (see the supplementary) quadada_p is not generally slower than SAG and SAGA since it obtains the best time on various datasets (`news20`, `w8a`, `covtype`, `rcv1`, `real-sim` and `kddb` till the third horizontal line) and the worst on others (`url`, `yahookr`, `webspam` and `criteo`). On these latest datasets, quadada_p is sometimes more than twice slower than SAG (e.g. the second horizontal line on `yahookr`) and it thus requires a few minutes more than the best.
- When $C = 100C_{\text{Best}}$, quadada_p is overall performing much faster than the others. In fact, in the majority of the datasets quadada_p is twice or even three times faster than the best of the two methods. This is for instance happening at the second horizontal line on `kddb`, where quadada_p is saving more than one hour of computation w.r.t. SAG. Moreover, in the majority of the cases the advantage of quadada_p is getting more and more remarkable as the solution gets finer. This is an expected behavior for first-order methods since they lack the second-order information needed to maintain fast performances on ill-conditioned problems.

We conclude that the TNCG in the setting quadada_p is generally as fast as the two state-of-the-art methods SAG and SAGA. Moreover, in the case of ill-conditioned problems, the proposed method is outperforming first-order approaches.



(a) `yahookr`, C_{Best} (b) `yahookr`,$100C_{\text{Best}}$

(c) `kdd2010a`, C_{Best} (d) `kdd2010a`, $100C_{\text{Best}}$

(e) `kdd2010b`, C_{Best} (f) `kdd2010b`, $100C_{\text{Best}}$

Fig. 6: A comparison between the proposed implementation of the TNCG and the two first-order methods SAG and SAGA. The x-axis is the cumulative time in seconds, while the rest of the information are the same as those of Figure 1.

X. Conclusions

In this paper we focused on both theoretical and numerical convergence of TNCG for linear classification. We first proved global and local convergence, finding out that literature was surprisingly wanting or unclear, especially in the case of not twice differentiable losses. We filled some gaps, enlightened some open questions and discovered a new connection between TNCG and common-directions algorithms. From the algorithmic and numerical point of view, for the first time in the field of linear classification, we show the importance of selecting an appropriate truncation rule. We found out that various machine learning software are affected by slow convergence issues and we identify the circumstances that are causing them. We propose a truncation rule based on the quadratic model that is able to avoid these circumstances and demonstrate its effectiveness and robustness for linear classification.

Finally we combined the study on truncation rules with that on preconditioning. We first discussed how to obtain the same global and local convergence results also in the preconditioned case. Second, we integrated our investigation on truncation rules with the use of preconditioning. Thus we proposed a new preconditioned TNCG algorithm that is able to improve the state of the art in the field of linear classification.

Appendix A
Proof of Theorem 3

Before giving the proof of Theorem 3 we need to remember that thanks to CG method properties, we have that Q_j is monotonically decreasing and

$$Q_1 = -\frac{1}{2}\frac{\boldsymbol{g}_k^T\boldsymbol{g}_k}{\boldsymbol{g}_k^T H_k \boldsymbol{g}_k} < 0.$$

Thus, we have that $(Q_j - Q_{j-1}) < 0$ and $Q_j < 0$ for every $j \geq 1$. For this reason (20) is equivalent to

$$Q_{j-1} - Q_j \leq \eta_k \cdot \frac{-Q_j}{j}. \tag{37}$$

We now recall two technical lemmas. In particular, the following result can be found in Lemma 4.3.1 of [25].

Lemma 2. *If H is symmetric and positive-definite then*

$$\boldsymbol{y}^T H^2 \boldsymbol{y} \leq \|H\| \boldsymbol{y}^T H \boldsymbol{y}. \tag{38}$$

Proof. We have

$$\begin{aligned}\boldsymbol{y}^T H^2 \boldsymbol{y} &= (H^{\frac{1}{2}}\boldsymbol{y})^T H (H^{\frac{1}{2}}\boldsymbol{y})\\ &\leq \lambda_{\max}(H) \cdot (H^{\frac{1}{2}}\boldsymbol{y})^T (H^{\frac{1}{2}}\boldsymbol{y})\\ &= \|H\| \boldsymbol{y}^T H \boldsymbol{y}.\end{aligned}$$ □

The following result can be found in Theorem 4.3.3 of [25].

Lemma 3. *Let $\boldsymbol{s}^*$ be the point that minimizes $Q(\boldsymbol{s})$. Then, for any $\boldsymbol{s}$ we have*

$$\|H_k\boldsymbol{s} + \boldsymbol{g}_k\|^2 \leq 2\|H_k\| \cdot (Q(\boldsymbol{s}) - Q(\boldsymbol{s}^*)) \tag{39}$$

Proof. From Lemma 2 and $\boldsymbol{g}_k = -H_k\boldsymbol{s}^*$ we have

$$\begin{aligned}\|H_k\boldsymbol{s} + \boldsymbol{g}_k\|^2 &= (H_k\boldsymbol{s} + \boldsymbol{g}_k)^T (H_k\boldsymbol{s} + \boldsymbol{g}_k)\\ &= (H_k\boldsymbol{s} - H_k\boldsymbol{s}^*)^T (H_k\boldsymbol{s} - H_k\boldsymbol{s}^*)\\ &= (\boldsymbol{s} - \boldsymbol{s}^*)^T H_k^2 (\boldsymbol{s} - \boldsymbol{s}^*)\\ &\leq \|H_k\| (\boldsymbol{s}^T H_k \boldsymbol{s} - 2{\boldsymbol{s}^*}^T H_k \boldsymbol{s} + {\boldsymbol{s}^*}^T H_k \boldsymbol{s}^*)\\ &= \|H_k\| \Big((\boldsymbol{s}^T H_k \boldsymbol{s} + 2\boldsymbol{s}^T\boldsymbol{g}_k)\\ &\quad - ({\boldsymbol{s}^*}^T H_k \boldsymbol{s}^* + 2{\boldsymbol{s}^*}^T\boldsymbol{g}_k) \Big)\\ &= 2\|H_k\| (Q(\boldsymbol{s}) - Q(\boldsymbol{s}^*)).\end{aligned}$$ □

Proof of Theorem 3. From (6.18) of [3] we get that

$$K_k(Q(\boldsymbol{s}_k^j) - Q(\boldsymbol{s}_k^*)) \geq Q(\boldsymbol{s}_k^{j+1}) - Q(\boldsymbol{s}_k^*),$$

where $\boldsymbol{s}_k^*$ is the point minimizing $Q(\boldsymbol{s})$. Thus, moving $K_kQ(\boldsymbol{s}_k^*)$ on the right side and subtracting $K_kQ(\boldsymbol{s}_k^{j+1})$ from both sides of the above inequality we get

$$\begin{aligned}&K_kQ(\boldsymbol{s}_k^j) - K_kQ(\boldsymbol{s}_k^{j+1}) \geq\\ &- K_kQ(\boldsymbol{s}_k^{j+1}) + K_kQ(\boldsymbol{s}_k^*) + Q(\boldsymbol{s}_k^{j+1}) - Q(\boldsymbol{s}_k^*) =\\ &(1 - K_k)\left(Q(\boldsymbol{s}_k^{j+1}) - Q(\boldsymbol{s}_k^*)\right),\end{aligned}$$

which means that

$$Q(\boldsymbol{s}_k^{j+1}) - Q(\boldsymbol{s}_k^*) \leq \frac{K_k}{1 - K_k}\left(Q(\boldsymbol{s}_k^j) - Q(\boldsymbol{s}_k^{j+1})\right). \tag{40}$$

Now since $Q(\boldsymbol{s}_k^j)$ is monotonically decreasing as j increases we get that

$$-Q(\boldsymbol{s}_k^*) \geq -Q(\boldsymbol{s}_k^j) \quad \forall j. \tag{41}$$

At the solution $\boldsymbol{s}_k^*$, from $\boldsymbol{s}_k^* = -H_k^{-1}\boldsymbol{g}_k$, we have

$$Q(\boldsymbol{s}_k^*) = {\boldsymbol{s}_k^*}^T\boldsymbol{g}_k + \frac{1}{2}{\boldsymbol{s}_k^*}^T H_k \boldsymbol{s}_k^* = -\frac{1}{2}{\boldsymbol{g}_k}^T H_k^{-1}\boldsymbol{g}_k.$$

Thus, together with (41), we get

$$-Q(\boldsymbol{s}_k^j) \leq -Q(\boldsymbol{s}_k^*) \leq \frac{1}{2}\|H_k^{-1}\| \cdot \|\boldsymbol{g}_k\|^2. \tag{42}$$

Finally from Lemma 3, (40), (37), (42) we get

$$\begin{aligned}\|H_k\boldsymbol{s}_k^j + \boldsymbol{g}_k\|^2 &\leq 2\|H_k\| \cdot (Q(\boldsymbol{s}_k^j) - Q(\boldsymbol{s}_k^*))\\ &\leq \frac{2K_k\|H_k\|}{1 - K_k}\left(Q(\boldsymbol{s}_k^{j-1}) - Q(\boldsymbol{s}_k^j)\right)\\ &\leq \eta_k \cdot \frac{2K_k\|H_k\|}{1 - K_k} \cdot \frac{-Q(\boldsymbol{s}_k^j)}{j}\\ &\leq \eta_k \cdot \frac{K_k\|H_k\| \cdot \|H_k^{-1}\|}{1 - K_k} \cdot \|\boldsymbol{g}_k\|^2\\ &= \eta_k \cdot N_k^2 \cdot \|\boldsymbol{g}_k\|^2,\end{aligned}$$

where $N_k^2 = \frac{K_k\|H_k\| \cdot \|H_k^{-1}\|}{1 - K_k}$ and the last inequality follows from the fact that $j \geq 1$. □

Acknowledgment

This work was partially supported by MOST of Taiwan grant 107-2221-E-002-167-MY3.

References

[1] S. S. Keerthi and D. DeCoste, "A modified finite Newton method for fast solution of large scale linear SVMs," *Journal of Machine Learning Research*, vol. 6, pp. 341–361, 2005.

[2] C.-J. Lin, R. C. Weng, and S. S. Keerthi, "Trust region Newton method for large-scale logistic regression," *Journal of Machine Learning Research*, vol. 9, pp. 627–650, 2008.

[3] M. R. Hestenes and E. Stiefel, "Methods of conjugate gradients for solving linear systems," *Journal of Research of the National Bureau of Standards*, vol. 49, no. 1, pp. 409–436, 1952.

[4] C.-P. Lee, P.-W. Wang, W. Chen, and C.-J. 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.

[5] C.-Y. Hsia, W.-L. Chiang, and C.-J. Lin, "Preconditioned conjugate gradient methods in truncated Newton frameworks for large-scale linear classification," in *Proceedings of the Asian Conference on Machine Learning (ACML)*, 2018.

[6] R. S. Dembo, S. C. Eisenstat, and T. Steihaug, "Inexact Newton methods," *SIAM Journal on Numerical Analysis*, vol. 19, pp. 400–408, 1982.

[7] F. H. Clarke, *Optimization and Nonsmooth Analysis*. New York: Wiley, 1983.

[8] B.-Y. Chu, C.-H. Ho, C.-H. Tsai, C.-Y. Lin, and C.-J. Lin, "Warm start for parameter selection of linear classifiers," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2015.

[9] C.-Y. Hsia, Y. Zhu, and C.-J. 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.

[10] G.-X. Yuan, C.-H. Ho, and C.-J. Lin, "Recent advances of large-scale linear classification," *Proceedings of the IEEE*, vol. 100, no. 9, pp. 2584–2603, 2012.

[11] J. M. Martínez and L. Qi, "Inexact newton methods for solving nonsmooth equations," *Journal of Computational and Applied Mathematics*, vol. 60, no. 1-2, pp. 127–145, 1995.

[12] H. Qi and D. Sun, "A quadratically convergent Newton method for computing the nearest correlation matrix," *SIAM Journal on Matrix Analysis and Applications*, vol. 28, no. 2, pp. 360–385, 2006.

[13] J. Yin and Q. Li, "A semismooth Newton method for support vector classification and regression," *Computational Optimization and Applications*, vol. 73, no. 2, pp. 477–508, 2019.

[14] L. Qi and J. Sun, "A nonsmooth version of Newton's method," *Mathematical programming*, vol. 58, no. 1-3, pp. 353–367, 1993.

[15] J. Nocedal and S. J. Wright, *Numerical Optimization*. New York, NY: Springer-Verlag, 1999.

[16] F. Facchinei and J.-S. Pang, *Finite-dimensional variational inequalities and complementarity problems*. Springer, 2003.

[17] F. Facchinei, "Minimization of SC1 functions and the Maratos effect," *Operations Research Letters*, vol. 17, no. 3, pp. 131–138, 1995.

[18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[19] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "LIBLINEAR: a library for large linear classification," *Journal of Machine Learning Research*, vol. 9, pp. 1871–1874, 2008.

[20] S. G. Nash and A. Sofer, "Assessing a search direction within a truncated-Newton method," *Operations Research Letters*, vol. 9, no. 4, pp. 219–221, 1990.

[21] S. C. Eisenstat and H. F. Walker, "Choosing the forcing terms in an inexact Newton method," *SIAM Journal on Scientific Computing*, vol. 17, no. 1, pp. 16–32, 1996.

[22] S. G. Nash, "A survey of truncated-Newton methods," *Journal of Computational and Applied Mathematics*, vol. 124, no. 1–2, pp. 45–59, 2000.

[23] H.-B. An, Z.-Y. Mo, and X.-P. Liu, "A choice of forcing terms in inexact Newton method," *Journal of Computational and Applied Mathematics*, vol. 200, no. 1, pp. 47–60, 2007.

[24] L. Botti, "A choice of forcing terms in inexact Newton iterations with application to pseudo-transient continuation for incompressible fluid flow computations," *Applied Mathematics and Computation*, vol. 266, pp. 713–737, 2015.

[25] S. G. Nash, "Truncated-Newton methods," Department of Computer Science, Stanford University, Tech. Rep. STAN-CS-82-906, 1982.

[26] ——, "Preconditioning of truncated-Newton methods," Report 371. Mathematical Sciences Dept., The Johns Hopkins University, Tech. Rep., 1982.

[27] P. Concus, G. H. Golub, and D. P. OĹeary, "A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations," Tech. Rep. STAN-CS-76-533, 1976.

[28] I. Griva, S. G. Nash, and A. Sofer, *Linear and nonlinear optimization*, 2nd ed. SIAM, 2009.

[29] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 2nd ed. The Johns Hopkins University Press, 1989.

[30] M. Schmidt, N. Le Roux, and F. Bach, "Minimizing finite sums with the stochastic average gradient," *Mathematical Programming*, vol. 162, no. 1-2, pp. 83–112, 2017.

[31] A. Defazio, F. Bach, and S. Lacoste-Julien, "SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives," in *Advances in Neural Information Processing Systems*, 2014, pp. 1646–1654.

Leonardo Galli is currently a postdoc at the Department of Mathematics, Rheinisch-Westfälische Technische Hochschule (RWTH) Aachen. He obtained his B.S., M.S. and Ph.D. degrees from University of Florence respectively in 2013, 2016 and 2020. His major research areas include optimization methods, machine learning and operational research. In particular, together with his tutors Marco Sciandrone and Fabio Schoen, he deeply explored the fields of nonmonotone techniques and prescriptive analytics. During his studies, he had various international collaborations. In 2015, he spent a research period at University of Würzburg where he worked on generalized Nash equilibrium problems together with professor Christian Kanzow. Starting from 2018, he collaborated with professor Chih-Jen Lin on truncated Newton methods for linear classification. In particular, they aimed at updating LIBLINEAR, one of the most widely used and cited linear classification packages. Thanks to this project, he could visit professor Lin for a research period at University of California Los Angeles (UCLA) in 2019 and at National Taiwan University in 2020. The project has ended in 2020, when a faster release of the LIBLINEAR software was released. More information about him and his work can be found at https://webgol.dinfo.unifi.it/leonardo-galli/.

Chih-Jen Lin is currently a distinguished professor at the Department of Computer Science and Information Engineering, National Taiwan University. He obtained his B.S. degree from National Taiwan University in 1993 and Ph.D. degree from University of Michigan in 1998. His major research areas include machine learning, data mining, and numerical optimization. He is best known for his work on support vector machines (SVM) for data classification. His software LIBSVM is one of the most widely used and cited SVM packages. He has received many awards for his research works including ACM KDD 2010, ACM RecSys 2013, ACML 2018 best paper awards. He is an IEEE fellow, a AAAI fellow, and an ACM fellow for his contribution to SVM algorithms and software design. More information about him and his software tools can be found at http://www.csie.ntu.edu.tw/~cjlin.