Trust-Region Optimization Methods Using Limited-Memory Symmetric Rank-One Updates for Off-The-Shelf Machine Learning

by Jennifer B. Erway, Joshua Griffin, Riadh Omheni, and Roummel Marcia

TRUST-REGION OPTIMIZATION METHODS USING LIMITED-MEMORY SYMMETRIC RANK-ONE UPDATES FOR OFF-THE-SHELF MACHINE LEARNING

JENNIFER B. ERWAY, JOSHUA GRIFFIN, RIADH OMHENI, AND ROUMMEL F. MARCIA

Abstract. Machine learning (ML) problems are often posed as highly nonlinear and nonconvex unconstrained optimization problems. Methods for solving ML problems based on stochastic gradient descent generally requires fine-tuning many hyper-parameters. In this paper, we propose an alternative approach for solving ML problems based on a quasi-Newton trust-region framework that does not require extensive parameter tuning. Numerical experiments on the MNIST data set show that with a fixed computational time budget, the proposed approach achieves better results than other off-the-shelf approaches like limited-memory Broyden-Fletcher-Goldfarb-Shanno (BFGS) quasi-Newton methods and Hessian-free methods.

1. Introduction

There has been a resurgence of interest in nonconvex and nonlinear machine learning problems of the form

$$\min_{w \in \mathbb{R}^m} f(w) = \frac{1}{n} \sum_{i=1}^{n} f_i(w),$$

where $f_i$ is a function of the $i$th observation in a training data set $\{(x_i, y_i)\}$. This relatively recent openness to using nonconvex models opens up exciting research opportunities to revisit promising yet still fringe algorithms not commonly used by the machine learning community. The goal of this paper is to demonstrate how recent research ([1]) on a well-known quasi-Newton update formula opens the door to taking advantage and exploiting negative-curvature information in ways that more popular updates cannot. This capability might further illuminate why Hessian-free methods, which are computationally expensive, perform so well in practice.

In the literature, (1) is often referred to as the empirical risk. Generally speaking, these problems have several features that make traditional optimization algorithms ineffective. First, both $m$ and $n$ are very large (e.g., typically $m, n \geq 10^6$). Second, there is a special type of redundancy that is present in (1) due to similarity between data points; namely, if $S$ is a random subset of indices of $\{1, 2, \ldots, n\}$, then provided $S$ is large enough, but $|S| \ll n$, (e.g., $n = 10^9$ and $|S| = 10^5$), then

$$\frac{1}{n} \sum_{i=1}^{n} f_i(w) \approx \frac{1}{|S|} \sum_{i \in S} f_i(w).$$

Existing methods. Three prevalent algorithms in machine learning for minimizing (1) include stochastic gradient descent methods, limited-memory BFGS methods, and Hessian-free methods. The stochastic gradient descent (SGD) method [2] randomly selects an index $j$ from $\{1, 2, \ldots, n\}$ at each iteration and $w$ is updated as follows: $w = w - \eta_j \nabla f_j(w)$, where $\nabla f_j$ denotes the gradient of $f_j$ and the parameter $\eta_j$ is the learning rate. SGD exploits data set redundancy described in (2) and the iteration complexity is independent of $n$ (in contrast to classical optimization algorithms which are explicitly dependent on $n$ [3, 4, 5, 6, 7]). However, to enhance the performance of SGD in practice, developers must fine-tune many hyper-parameters–leading to many variations of SGD, (e.g., see [8, 9, 10, 11, 12, 13]). In practice, finding an effective sequence $\{\eta_j\}$ can require solving the same problem many times to find the best sequence. This dilemma has led to a resurgence of interest in auto-tune algorithms that can aid the SGD user in this search [14, 15, 16, 17, 18].
Because of the fine-tuning required for the hyper-parameters, these algorithms are not often used as off-the-shelf methods for machine learning.

An alternative to SGD are quasi-Newton methods. Generally speaking, quasi-Newton methods generate a sequence of iterates using the rule \( w_{k+1} = w_k + \alpha_k p_k \), where \( p_k = -B_k^{-1} \nabla f(w_k) \), \( B_k \) is a quasi-Newton matrix that is updated each iteration using gradient information, and \( \alpha_k \) is a step length. In machine learning, the Broyden-Fletcher-Goldfarb-Shanno (BFGS) update is exclusively used to generate the sequence of quasi-Newton matrices \( \{B_k\} \); that is,

\[
B_{k+1} = B_k - \frac{1}{s_k y_k} B_k s_k s_k^T B_k + \frac{1}{y_k s_k} y_k y_k^T,
\]

where \( s_k = w_{k+1} - w_k \) and \( y_k = \nabla f(w_{k+1}) - \nabla f(w_k) \). Note that \( B_0 \) is usually taken to be a positive scalar multiple of the identity. BFGS can be used to generate a sequence of positive-definite matrices \( \{B_k\} \), guaranteeing a descent direction. In large-scale optimization, a limited-memory version of this method is used (called limited-memory BFGS (L-BFGS)) to bound storage requirements and promote efficiency. In this case, only a few of the most-recently computed iteration and gradient information are stored. Unlike SGD, L-BFGS exploits information from previous iterations and requires little few hyper-parameter tuning. The main disadvantage of L-BFGS is that in a nonconvex environment, this method has the difficult task of approximating an indefinite matrix (the true Hessian) with a positive-definite matrix \( B_k \), which can result in the generation of nearly-singular matrices \( \{B_k\} \). Numerically, this creates need for heuristics such as periodically reinitializing \( B_k \) to a multiple of the identity, effectively generating a steepest-descent direction in the next iteration. This can be a significant disadvantage for machine learning problems where model quality is highly correlated with the quality of initial steps [19].

Hessian-free methods to solve (1) leverage the ability to perform Hessian-vector multiplies without storing the entire second-derivative matrix itself [19, 20, 21]. These algorithms perform approximate updates of the form

\[
w_{k+1} = w_k - \eta_k p_k \quad \text{with} \quad \nabla^2 f(w_k) p_k = -\nabla f(w_k),
\]

where \( p_k \) is an approximate Newton direction obtained using a conjugate-gradient (CG)-like algorithm. Traditional CG algorithms to solve linear systems such as (4) using only matrix vector-products to obtain an approximate Newton direction, making them attractive method for large-scale optimization. Because \( \nabla^2 f(w) \) may be indefinite, modified variants are needed to adapt for local nonconvexity. Remarkably, Martens [20] was able to show that Hessian-free methods can achieve out-of-the-box competitive results compared to manually-tuned SGD on deep learning problems. Moreover, Pearlmutter [22] showed that matrix-vector products could be computed at a computational cost on the order of a gradient evaluation. However, since multiple matrix-vector products can be required to solve (4), the iteration complexity of these algorithms is significantly greater than L-BFGS. Thus, despite its allure of being a tune-free approach, Hessian-free methods are for the most part unused and unexplored in practice.

**Proposed approach.** In this paper we propose an alternative approach for solving (1) based on quasi-Newton methods. However, unlike existing approaches that use L-BFGS updates, we propose using an update that allows for indefinite Hessian approximations. Additionally, to better accommodate indefinite Hessian approximations we consider a trust-region framework rather than a line search framework, which is used in L-BFGS methods [23].

2. METHODOLOGY

In this section, we describe quasi-Newton matrices and then demonstrate their use in a trust-region method to solve (1).

2.1. Limited-Memory Symmetric Rank-One Update. Given a continuously-differentiable function \( f(w) \) as in (1), the symmetric rank-one (SR1) is given by

\[
B_{k+1} \doteq B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{(y_k - B_k s_k)^T s_k}
\]
where \( s_j \triangleq w_{j+1} - w_j \) and \( y_j \triangleq \nabla f(w_{j+1}) - \nabla f(w_j) \), for \( j = 0, \ldots, k - 1 \). The SR1 update can be written recursively and compactly using the outer product representation

\[
B_{k+1} = B_0 + \left[ \Psi_k \right] \left[ \begin{array}{c} M_k \Psi_k^T \end{array} \right]
\]

where \( B_0 = \gamma I \) for some \( \gamma \neq 0 \). \( \Psi_k \) is an \( n \times (k+1) \) matrix and \( M_k \) is a \( (k+1) \times (k+1) \) matrix. In particular, Byrd et al. [24] show that for SR1 matrices,

\[
\Psi_k = Y_k - B_0 S_k \quad \text{and} \quad M_k = (D_k + L_k + L_k^T - S_k^T B_0 S_k)^{-1},
\]

with \( S_k \triangleq \begin{bmatrix} s_0 & s_1 & s_2 & \cdots & s_k \end{bmatrix} \in \mathbb{R}^{n \times (k+1)} \), and \( Y_k \triangleq \begin{bmatrix} y_0 & y_1 & y_2 & \cdots & y_k \end{bmatrix} \in \mathbb{R}^{n \times (k+1)} \), and \( L_k \) is the strictly lower triangular part, \( U_k \) is the strictly upper triangular part, and \( D_k \) is the diagonal part of \( S_k^T Y_k = L_k + D_k + U_k \). In our proposed approach, we use a limited-memory SR1 (L-SR1) update, where only \( r \) of the most recent pairs \( \{ s_j, y_j \} \) are stored, where the value of \( r \) is typically very small so that \( r \ll n \). Unlike the BFGS update, the SR1 update is not necessarily positive definite. Moreover, under mild conditions, SR1 updates have been shown to exhibit stronger convergence properties to the true Hessian than BFGS updates [25].

2.2. Trust-Region Methods. Trust-region methods are one of two important classes of methods for unconstrained optimization (see, e.g., [23, 26]). Basic trust-region methods generate a sequence of iterates \( \{ w_k \} \) given by \( w_{k+1} = w_k + p_k \), where \( p_k \) is an approximate solution to the trust-region subproblem given by

\[
p_k = \arg \min_{p \in \mathbb{R}^n} Q_k(p) \triangleq g_k^T p + \frac{1}{2} p^T B_k p \quad \text{s.t.} \quad \|p\| \leq \delta_k,
\]

where \( g_k \triangleq \nabla f(w_k) \), \( B_k \approx \nabla^2 f(w_k) \), and \( \delta_k \) is a given positive constant known as the trust-region radius. At the end of each trust-region iteration, \( \delta_k \) is used to update \( \delta_{k+1} \); depending on how well the quadratic model predicted actual decreases in the function \( f(w) \) from \( w_k \) to \( w_{k+1} \), the trust-region radius is possibly increased or decreased for the next iteration. Unlike in line search methods, the Hessian approximation \( B_k \) need not be positive definite in trust-region methods. In our proposed approach, we use the L-SR1 matrix, which may be indefinite.

Methods for solving the trust-region subproblem (6) are generally based on the optimality conditions for a global solution to the trust-region subproblem (see [27, 28]), given in the following theorem:

**Theorem:** Let \( \delta \) be a given positive constant. A vector \( p^* \) is a global solution of the trust-region problem (6) if and only if \( \|p^*\|_2 \leq \delta \) and there exists a unique \( \sigma^* \geq 0 \) such that \( B_k + \sigma^* I \) is positive semidefinite with

\[
(B_k + \sigma^* I)p^* = -g_k, \quad \text{and} \quad \sigma^*(\delta_k - \|p^*\|_2) = 0.
\]

Moreover, if \( B_k + \sigma^* I \) is positive definite, then the global minimizer is unique.

The optimality condition in (7) indicates that the solution \( p^* \) must lie in interior of the trust region or \( p^* \) must lie on the boundary and that it is the exact minimizer of a nearby convex quadratic problem. In recent work by the authors [1], an efficient solver for the L-SR1 trust-region subproblem (6) was proposed. We describe this approach next.

2.3. Solving the Trust-Region Subproblem. Solving the trust-region subproblem (6) is generally the computational bottleneck of trust-region methods. To efficiently solve the subproblems, we exploit the structure of the L-SR1 matrix to obtain global solutions to high accuracy.

To begin, we transform the optimality equations (7) using the spectral decomposition of \( B_k \). As shown in [1], the spectral decomposition of \( B_k \) can be obtained efficiently using the compact formulation of \( B_k \). If \( B_k = B_0 + \Psi M \Psi^T \) and \( \Psi = QR \) is the “thin” QR factorization of \( \Psi \), then \( B_k = \gamma I + QMRM^T Q^T \), where \( B_0 = \gamma I \) and \( \gamma > 0 \). Since \( RMR^T \) is a small \( (k+1) \times (k+1) \) matrix, its spectral decomposition \( V \Lambda V^T \) can be quickly computed. Then, letting \( P = \left[ QV \quad (QV)^T \right] \in \mathbb{R}^{n \times n} \) such that \( P^T P = PP^T = I \), the spectral decomposition of \( B_k \) is given by

\[
B_k = P \Lambda P^T, \quad \text{where} \quad \Lambda = \begin{bmatrix} \Lambda_1 & 0 \\ 0 & \Lambda_2 \end{bmatrix} = \begin{bmatrix} \hat{\Lambda} + \gamma I & 0 \\ 0 & \gamma I \end{bmatrix},
\]
where $\Lambda_1 \in \mathbb{R}^{(k+1)\times(k+1)}$, and $\Lambda_2 = \gamma I_{n-(k+1)}$. Using the spectral decomposition of $B_k$, the optimality equations (7) become

$$
(\Lambda + \sigma^* I)v^* = -P_T g \quad \text{and} \quad \sigma^*(\delta_k - \|v^*\|_2) = 0,
$$

for some scalar $\sigma^* \geq 0$ and $v^* = P_T p^*$, where $p^*$ is the global solution to (6). The Lagrange multiplier $\sigma^*$ can be obtained by substituting

$$
\|v^*\|_2^2 = \| (\Lambda + \sigma^* I)^{-1} P_T g \|_2^2 = \sum_{i=0}^{k} \frac{(P_T g)^2_i}{(\lambda_i + \sigma^*)^2}
$$

into the second equation in (8). Once $\sigma^*$ is obtained, $v^*$ can be computed from (8) and as well as the solution $p^* = P v^*$ to the original trust-region subproblem (6).

The proposed L-SR1 Trust-Region Method (L-SR1-TR) is described in Algorithm 1. For details on the subproblem solver and all related computations, see [1, Algorithm 1].

Define parameters: $m, 0 < \tau_1 < 0.5, 0 < \varepsilon$;
Initialize $w_0 \in \mathbb{R}^n$ and compute $g_0 = \nabla f(w_0)$;
Let $k = 0$;
while not converged
  if $\|g_k\|_2 \leq \varepsilon$ then done
    Find $p_k$ that solves (6) using [1, Algorithm 1];
    Compute $\rho_k = (f(w_k + p_k) - f(w_k))/Q_k(p_k)$;
    Compute $g_{k+1}$ and update $B_{k+1}$;
    if $p_k \geq \tau_1$ then
      $w_{k+1} = w_k + p_k$;
    else
      $w_{k+1} = w_k$;
    end if
    Compute trust-region radius $\delta_{k+1}$;
    $k \leftarrow k + 1$;
  end while

**Algorithm 1**: Limited-Memory SR1 Trust-Region (L-SR1-TR) Method

3. Numerical Results

In this section, we present numerical results comparing the performance of several methods, including the proposed L-SR1-TR approach, on the MNIST data set [29]. Here, we use a neural network model with various hidden layers. We optimize the model parameters $w$ so that chosen model function $p(x, w)$ predicts the observed target vector $y$ as accurately as possible over the set of training data with respect to the chosen loss and activation functions. The model $p(x, w)$ is formed by a predetermined sequence of affine transformations nested by a nonlinear activation function applied component-wise. Specifically, we set $a_o = x$ and define $a_j = \sigma(W_j a_{j-1} + b_j)$ for $j = 1, \ldots, \ell$ where $\ell$ denotes the number of layers and $\sigma$ is the chosen activation function. In our experiments $\sigma$ was defined to be the logistic function for the hidden layers while softmax paired with cross-entropy was used for the final output layer to form the resulting loss function element $f_i$ in (1) (see [30, Chap. 11] for details.)

Two errors are used to train a network: training error and test error. The training error is used to define the optimization problem (1). Most approaches that use training data tend to find models that overfit the data, i.e., the models find relationships specific to the training data that are not true in general. In other words, overfitting prevents machine learning algorithms from correctly generalizing. To help prevent overfitting, an independent data set, called the test set is used to validate the accuracy of the model to gage its usefulness in making future predictions. Training errors and test errors are computed using the total Loss function $f(w)$ in (1).

For the results in this section, we compared the training and validation errors of three methods: (i) a Hessian-free utilizing the Generalized Gauss-Newton method described in [20], (ii) an L-BFGS method based on [31], and (iii) the proposed L-SR1 trust-region method (L-SR1-TR). (Because existing SGD methods are already finely tuned on this data set, they were not considered.) We tested the three methods on the full MNIST data set with full training and testing observations. We compared these methods on a variety of network configuration; all tests were performed in MATLAB. The limited-memory methods were run with $r = 20$. These experiments were designed
Figure 1. Plots of the Loss function versus iterations ((a)-(c)) and time ((d)-(f)) on the MNIST data set using 3 different hidden layers (\(\{500, 350\}\), \(\{350, 250, 150\}\), and \(\{400, 250, 150, 100, 30\}\)) with in input layer of size 784 and output layer of size 10. While the Hessian-free method has the lowest test and training loss with respect to iteration, the proposed L-SR1-TR approach outperforms both L-BFGS and Hessian-free methods with respect to wall-time.

4. Conclusion

In this paper, we presented an alternative approach for solving machine learning problems that is based on the L-SR1 update that allows for indefinite Hessian approximations. This approach is particularly suitable for non-convex problems where exploiting directions of negative curvature is crucial. Numerical experiments demonstrate that the proposed approach outperforms the more commonly used quasi-Newton approach (L-BFGS) both in terms of computational efficiency and test and training loss.

Acknowledgments. J. Erway’s research was funded by NSF Grants CMMI-1334042 and IIS-1741264. R. Marcia’s research was funded by NSF Grants CMMI-1333326 and IIS-1741490.

References


