A minibatch stochastic Quasi-Newton method adapted for nonconvex deep learning problems

Joshua D. Griffin[a], Majid Jahani[b], Martin Takáč[c], Seyedalireza Yektamaram[a], Wenwen Zhou[a]

ABSTRACT

In this study, we develop a limited memory nonconvex Quasi-Newton (QN) method, tailored to deep learning (DL) applications. Since the stochastic nature of (sampled) function information in minibatch processing can affect the performance of QN methods, three strategies are utilized to overcome this issue. These involve a novel progressive trust-region radius update (suitable for stochastic models), batched evaluation instead of the entire data set, for selecting gradient batch-size and a restart strategy when quasi-Netwon approximation accuracy deteriorates. We analyze the convergence properties of our proposed method and provide the required theoretical analysis for different components of our algorithm. The numerical results illustrate that our proposed methodology with the new adjustments outperforms the previous similar methods, and is competitive with the best tuned stochastic first-order methods, in cases where large batch-size is required. Finally, we empirically show that our method is robust to the choices of hyper-parameters, thus, requiring less tuning compared to Stochastic Gradient Descent (SGD) method.

KEYWORDS

Deep Learning; Quasi-Newton methods; Hessian-Free methods

1. Introduction

Optimization problems in machine learning applications are typically described by:

$$\min_{w\in\mathbb{R}^d} f(w) \triangleq \frac{1}{N}\sum_{i=1}^{N} f_i(w), \tag{1}$$

where d denotes the number of parameters, N the number of observations used for training, and $f_i : \mathbb{R}^d \to \mathbb{R}$ denotes some loss function measuring the error with respect to the i^{th} training observation. For loss functions in deep-learning models, $f(w)$ may be nonconvex while both d and n may be arbitrarily large.

Modern machine learning approaches consist predominantly of variants of SGD methods [23]; further, the power and efficacy of SGD in the context of machine learning is inarguable [2]. However, there is a growing gulf between writers of academic papers using tuned variants of new SGD algorithms on deep learning models to achieve the seeming impossible [25], and end-users who work under strict deadlines and may have limited time for parameter tuning [2, 24]. Thus in practice, simpler model types, such as gradient boosting trees, with fewer sensitive hyper-parameters are often preferred.

It is known for a deterministic optimization problems, second-order methods achieve faster convergence rates compared to first-order methods [9]. Requiring fewer iterations,

[a]J. Griffin, S. Yektamaram, W. Zhou: SAS Institute 100 SAS Campus Drive Cary, NC 27513 USA

[b]M. Jahani: Department of Industrial and Systems Engineering Lehigh University Bethlehem, PA 18015

[c]M. Takáč: Mohamed bin Zayed University of Artificial Intelligence (MBZUAI)

second-order methods can better take advantage of parallel computing and enjoy lower communication costs [4, 6, 12, 27]. Consequently, continual efforts are made to reintroduce second-order information into machine learning problems. Such second-order approaches [2] can be divided into three categories: (I) methods that require analytic Hessian-vector multiplications, (II) methods that use the block-diagonal matrix called K-FAC to approximate the Hessian matrix [18], (III) methods that use gradients to build second-order information called limited memory QN updates.

Studies such as [17, 19, 20] ignited early interest in the potential of second-order information to improve solution quality while reducing time spent pre-training and tuning. In this context, to handle indefinite Hessian information, [28] used a modified conjugate gradient method. Such examples, belonging to category (I), are interesting and worth further exploration; however, they require explicit second-order (directional) derivative calculations. Thus, their applications are limited to platforms that support efficient second-order automatic differentiation. Support for such higher-order functionality may be considered unnecessary overhead for a team that predominantly uses first-order algorithms.

In the category (II), K-FAC, structure of the Fisher matrix is exploited to form Hessian-inverse approximations using small dense-block factorization [18]. A limitation of this approach is that, in general, it is highly structure intensive and cannot be surfaced as a generic black-box solver plugin for arbitrary machine learning packages. In other words, for such approaches, efficiency and applicability are tightly coupled with the ML framework package.

Lastly, category (III) algorithms basically use QN updates with limited memory to shape the Hessian approximation. Using secant approximations for gradually building a curvature approximation, such methods are well established in deterministic optimization. Current study, focuses on this class of methods that leverage the power of approximated Hessian information stored in small, fixed dimensional subspaces; so called limited memory quasi-Newton methods [21]. A marked advantage of the QN approach is that, unlike the higher-order alternatives described earlier, there is potential for creating eventual support of a stand-alone plugin that may be easily used by existing first-order packages. The primary reason is simple: the inputs needed for such approaches are identical to that of the current favorite in the machine learning community, SGD. In general, SGD updates have the form

$$w_{k+1} = w_k - \alpha_k D_k g_{\mathcal{S}}(w_k), \tag{2}$$

where w_k denotes the current iterate, α_k the step-size or learning rate, D_k a diagonal approximation to the inverse Hessian, $\mathcal{I}$ the current minibatch, and $g_{\mathcal{I}}(w_k)$, given by

$$g_{\mathcal{I}}(w_k) = \tfrac{1}{|I|}\textstyle\sum_{i\in I} \nabla f_i(w_k). \tag{3}$$

The framework for the proposed approach executes nearly identical updates, requiring similar inputs, where we simply assign D_k a Tikhonov-damped quasi-Newton approximation of form:

$$D_k = (B_k + \sigma_k I)^{-1},$$

where σ_k denotes the optimal dual multiplier of a trust-region subproblem and B_k denotes the quasi-Newton Hessian approximation. Further, the learning-rate may be selected adaptively by a line-search to ensure each batch achieves a modest amount of

overall reduction. Recently, a stochastic quasi-Newton combining with approximating the Hessian by a block-diagonal matrix was proposed in [10]. It uses BFGS and LBFGS updates for quasi-Newton approximation.

While a number of different update formulas have been defined and tried in the past, two are arguably the most popular: the Symmetric Rank-1 update (SR1) and BFGS [2]; while BFGS remains arguably the more popular approach in practice, although the SR1 matrix has more attractive theoretical properties. Directly applying the Limited-memory BFGS (L-BFGS) to deep learning application requires more consideration. The main reason is the potential poor convergence of the updates, moreover, the updates can even become near singular due to non-positive definiteness of the actual Hessian. Additionally, enforcing the curvature condition for BFGS is computationally expensive when mini-batch approach is used.

Current work focuses on the LSR1 approach. A LSR1 with the trust region approach was proposed in [8], However, the algorithm computationally is expensive and can sometimes stall. The major contributions of this paper, further include trust-region radius updates for the stochastic learning scenario, adaptive batch-size adjustment to avoid evaluating function value on the entire data set and to restart LSR1 update when the local model accuracy diverges from underlying true function. More importantly, we analyze the convergence properties of our proposed method and provide the required theoretical analysis for different components of our algorithm. Finally, we illustrate the practical performance of our method on standard neural network benchmarks and imbalanced data sets. One of the benefits of the new approach is that in scenarios where large batch-size is required or the data set is imbalanced, our method shows significantly better results compared to the best-tuned SGD.

The paper is organized as follows. In Section 2, L-SSR1-TR approach of [8] will be discussed, to lay the foundation for the new Mini-Batch stochastic LSR1 (MB-LSR1) approach given in Section 3. Section 4 presents convergence analysis of the new algorithm. Section 5 contains numerical results and comparisons on different test problems, to empirically show benefits of new method over original L-SSR1-TR, also comparisons include results for SGD. The conclusion and some possible future work are given in Section 6.

2. SR1 Trust-Region Methods for Deep Learning

A trust-region approach solves nonconvex optimization problems by iteratively modeling the objective function using quadratic models in a trusted vicinity of current iterate. This requires solving nonconvex Quadratic Programming (QP) subproblems, and dynamically adjusting the trust-region radius. That is, to solve (1), at each iteration k, the following QP subproblem is solved:

$$\begin{array}{ll} \min\limits_{p\in\mathbb{R}^d} q_k(p) = & p^T g_k + \frac{1}{2} p^T H_k p \\ \text{subject to} & \|p\| \leq \delta_k \end{array}, \tag{4}$$

where $g_k = \nabla f(w_k)$ and $H_k = \nabla^2 f(w_k)$; forming the quadratic model of loss function around iterate w_k. The ratio of the actual- and predicted- reduction for objective function $f(w)$ defined as

$$\rho_k = \frac{f(w_k + s) - f(w_k)}{q_k(s) - q_k(0)}, \tag{5}$$

helps measuring the quality of the model around radius δ_k. Thus, it makes sense to expand the region by increasing δ_k when ρ_k is sufficiently large and decrease δ_k when ρ_k is small. Trust-region approaches enjoy favorable convergence properties for nonconvex optimization problem [5].

Since evaluating $H_k = \nabla^2 f(w_k)$ at each iterate w_k could be expensive in large-scale problems, often quasi-Newton methods are used to exploit the gradient information and gradually build an approximation (B_k) of the Hessian matrix using secant updates. The SR1 update is the important indefinite update of the Broyden class, updating the B_k matrix by

$$B_{k+1} = B_k + \frac{1}{s_k^T(y_k - B_k s_k)}(y_k - B_k s_k)(y_k - B_k s_k)^T, \tag{6}$$

where s_k and y_k are defined as $y = \nabla f(w_k) - \nabla f(w_{k-1})$ and $s = w_k - w_{k-1}$, and aggregated iteratively in matrices S, Y.

At each iteration, it is assumed $s_k^T(y_k - B_k s_k) \neq 0$, i.e., all of the updates are well-defined, otherwise the update is skipped [21]. Thus, SR1 update can always be well-defined. This makes it attractive especially in deep learning where the sampled function and its gradient evaluations can involve different batches. For large-scale problems, the limited memory QN method, including LSR1, has been developed [16, 21]. A new algorithm proposed in [3], uses SR1 to update H_k and solve a trust-region subporblem (4) .

LSR1 with trust-region approaches was proposed for solving deep learning problems in [22]. Erway et al. [8], later on gave a much more sophisticated limited memory stochastic SR1 (L-SSR1-TR) with a trust-region method for deep learning problems. Despite the convergence results elaborated by [8], it has been found that the algorithm can sometimes stall and computationally expensive. This study aims to address these issues, and a new algorithm is therefore proposed in the next section.

Recently, a different trust-region LSR1 approach is proposed in [1], where (S_k, Y_k) are created differently from most of LSR1 approaches including [8]. S_k consists a set of random generated directions and Y_k uses finite difference or Hessian vector product calculations. In other words, LSR1 update is applied for a QP subproblem at given iteration k. This strategy may avoid creating a biased approximation of Hessian since S_k is randomly generated. Similarly, a block quasi-Newton method tries to create an approximated Hessian by applying quasi-Newton idea on QP subproblems as well [11]. It has been found that it is useful to use this LSR1 approach in [1] to warmstart the LSR1 approach proposed in current study.

Introducing some useful notations for stochastic optimization, would help presenting further results in next two sections. Sampled function $\hat{f}(w;\mathcal{I})$ is used when $f(w)$ is evaluated at a batch set $\mathcal{I}$:

$$\hat{f}(w;\mathcal{I}) \triangleq \frac{1}{|\mathcal{I}|}\sum\nolimits_{i\in\mathcal{I}} f_i(w). \tag{7}$$

Similar notation is used for gradient $g_k = \nabla f(w_k)$; i.e. $\hat{g}_k(\mathcal{I})$ to represent the gradient evaluated at a batch set $\mathcal{I}$. Furthermore, for the sake of simplicity, $\mathcal{I}$ in $\hat{f}(w;\mathcal{I})$ and $\hat{g}_k(\mathcal{I})$ are omitted as long as there is no confusion. That is, $\hat{f}(w)$ and $\hat{g}_k$ are used instead of $\hat{f}(w;\mathcal{I})$ and $\hat{g}_k$.

3. A Unified Stochastic SR1 Trust-Region Framework

L-SSR1-TR is very interesting for solving deep learning problems; however, there are several flaws that could affect its practical usage. For instance, it is observed that the search directions can become parallel to the previously generated directions, especially when false curvature information is captured in matrix B_k, leading to inaccurate approximation of the local quadratic model. As a result, the algorithm stalls the optimization progress. Moreover, Algorithm L-SSR1-TR has to evaluate function value on the whole data set at every K iterations, which in the new version (Algorithm 7) is not necessarily the case. In order to address the aforementioned issues, we propose a

Algorithm 1 Stochastic SR1 Trust-region scheme (MB-LSR1)

Input: Iterate w_0, memory m, restart memory $\hat{m} \leq m$, initial batch-size n, initial radius δ_0,
restart tolerance τ, progress check frequency K, progress threshold parameters γ_1 and γ_2, progress check batch-size sequence $\hat{n}^k$, progressive radius parameter $\zeta \in [0, 1]$, momentum μ and learning-rate $\alpha_s \in [0, 1]$ used for hybrid approach.
Set: $T = 0$, $\hat{\rho} = 0$, $s = y = v = 0$, $S = Y = []$.
for $k = 0, 1, 2, ...$ **do**
  $[s, y, v, \rho, R_k] = \textbf{getStep}(s, v, \delta, \eta, \mu, \alpha_s)$
  $w_{k+1} = w_k + s$
  if $(\textbf{ExitCheck}(\hat{g}_k) == 1)$ **then** return
  $[\delta, \hat{\rho}, T] = \textbf{getRadius}(\delta, s, \rho, \zeta, \hat{\rho}, T)$
  $[S, Y] = \textbf{updateSY}(S, Y, s, y, \rho, \tau, \hat{m})$
  $[n, \hat{f}_K] = \textbf{getBatch}(n, \hat{f}_K, K, \gamma_1, \gamma_2)$
end for

new method MB-LSR1, summarized in Algorithm 1. To better help the readers follow MB-LSR1, a brief description of the algorithm is given first. Contrary to Algorithm L-SSR1-TR, there is no function evaluation on whole data set unless the batch-size reaches N. However, the new algorithm, utilizes the same SR1 update strategy and also solution approach to the trust-region subproblems as Algorithm L-SSR1-TR does. Batch gradient information and the intrinsic stochastic noise, deteriorates the accuracy of quadratic model, around current iterate; therefore, a new restart strategy for the SR1 update is used, to remedy cases of small ratio ρ (5) due to inaccurate model. Progress checks with less strict conditions monitor the reduction in loss function to be sufficient, so that the algorithm could move on to the next batch, without increasing the batch-size. Additionally, a new progressive trust-region radius update for defining next QP trust-region subproblems is used to account for randomness due to batching.

Algorithm 2 Termination Check

function ExitCheck(g_k)
$ifStop \leftarrow 0$
if $\|\hat{g}_k\| \leq \epsilon$ **then**
  Evaluate new g_k over whole data set
  if new $\|g_k\| \leq \epsilon$ **then**
    $ifStop \leftarrow 1$
  end if
end if
return $[ifStop]$

Algorithm 3 Pair selection algorithm

function updateSY$(S, Y, s, y, \rho, \tau, \hat{m})$
if $\rho < \tau$ **and** $\hat{m} \geq 0$ **then**
  if $\hat{m} = m$ **then**
    Warm start according to [1]
    Generate m pairs and store in S and Y
  else if $\hat{m} = 0$ **then**
    $S = []; Y = []$; % Restart
  end if
else
  % Classic S, Y update
  $S = [S, s]; Y = [Y, y]$;
  Drop initial column of S and Y if needed
end if
return $[S, Y]$

Algorithm 4 Search direction computation algorithm

function getStep$(s, v, \delta, \eta, \mu, \alpha_s)$
Compute batch gradient $\hat{g}_k = \nabla \hat{f}(w_k)$
Compute p^* by applying the algorithm proposed in [8] using latest S, Y pairs
$v = \mu v - \eta \alpha_s \hat{g}_k + (1 - \eta)s$
$v = \min(1, \delta / \|v\|)v$
$p = (1 - \eta)p^* + \mu v$
$p = \min(1, \delta / \|p\|)p$
if $p^T g_k > 0$ **then**
  set $p = -p$
end if
if $\min\left(|\hat{g}_k^T p|, \frac{|\hat{g}_k^T p|}{\|p\|_2}\right) < \xi \|g_k\|_2$ **then**
  restart LSR1 with initial matrix being I
  return and goto Step 5 of Algorithm 1
end if
Get step-size α from line-search on current batch satisfying
$\hat{f}(w_k + \alpha p) \leq \hat{f}(w_k) + c_1 \alpha \nabla \hat{f}(w_k)$
$s = \alpha p$;
$R_k = \hat{f}(w_k + p^*) - \hat{f}(w_k)$
$\rho = (R_k / \mathcal{Q}_k(p^*))$
$y = \nabla \hat{f}(w_k + s) - \nabla \hat{f}(w_k)$
return $[s, y, v, \rho, R]$

The new hyper-parameters[1] not included in the original L-SSR1-TR [8] are

(1) the restart memory option $\hat{m}$,
(2) the restart tolerance τ,
(3) the second progress threshold γ_2,
(4) and progressive radius parameter ζ,
(5) progress check batch-size sequence $\hat{n}_b^k$,
(6) SGD hybrid parameter η

[1]It is noteworthy to mention that the performance of our proposed method, MB-LSR1, does not change significantly if the hyper-parameters are not perfectly hand-tuned. In particular, in section 5 of the paper, we perform some sensitivity analyses on the hyper-parameters of MB-LSR1, and we show that the proposed MB-LSR1 method is robust with respect to different choices of hyper-parameters and it has a thin spectrum of changes.

Algorithm 1 is a general framework which can also encompass L-SSR1-TR [8] as a special case, simply by setting $\hat{m} = -1$, $\gamma_2 = \eta = \zeta = 0$, and n_b^k set equal to N (the number of all observations). Thus, the interested reader may focus on these five parameters to see the primary contributions made by this work.

In particular, the restart tolerance τ is only referenced in the case $\hat{m} \geq 0$. If $\hat{m} = 0$, then we perform a vanilla restart of the current stored quasi-Newton pairs as shown in Algorithm 3.

Algorithm 5 Example update radius function

function getRadius$(\delta, s, \rho, \zeta, \hat{\rho}, T)$
% Update non-monotone ratio threshold $\hat{\rho}$
$\hat{\rho} = \zeta T \hat{\rho} + \rho$
$T = \zeta T + 1$
$\hat{\rho} = \hat{\rho}/T$
if $\hat{\rho} < 0.1$ **then**
    $\delta = \min(\delta, \|s\|_2)$
else if $\hat{\rho} \geq 0.5$ **and** $\|s\| \geq \delta$ **then**
    $\delta = 2\delta$
end if
return $[\delta, \hat{\rho}, T]$

Algorithm 6 Example batch-size correction function

function getBatch$(n_b, \hat{f}_K, K, \gamma_1, \gamma_2)$
if $\mod(k, K) = 0$ **then**
    Randomly draw new progress check sample of size $\hat{n} = \min(1.1 n_b, 5000)$
    Define $\hat{f}$ using the new sample
    if $\hat{f}(w_k) - \hat{f}_K \geq -\gamma_1 + \gamma_2 \sum_{j=k-K}^{k} R_j$ **then**
        % Progress worse than simple line-search
        $n_b = \min(2n_b, N)$
        if $n_b == N$ **then**
            $\zeta \leftarrow 0$
        end if
    end if
    % Update new target bound $\hat{f}_K$
    $\hat{f}_K = \hat{f}(w_k)$
end if
Randomly draw batch $\mathcal{I}_{k+1}$ with size n_b defining $\hat{f}$.
return $[n_b, \hat{f}_K]$

However, if $\hat{m} = m$, then we perform the warm restart by random sampling around the current iterate as described in [1]. The second progress threshold parameter γ_2 allows us to relax the strict-decrease condition enforced in L-SSR1-TR, to prevent batch-size n quickly reaching to N and cover the entire data set if a sufficient decrease in the objective is obtained. Further, ζ in Algorithm 5 can be used to relax the trust-region ratio progress check by averaging past ratios; this helps to soften the effect of the ratio ρ_k as a stochastic term. Ideally it's preferred that $\lim_{k\to\infty} \rho_k = 1$; however, fluctuations in ρ_k are expected. It should be noted that with respect to the value of the trust-region parameter the overall progress is truly of interest. To soften the computational cost of the full-batch evaluation check made every J iterations in L-SSR1-TR, we permit a subsequence to be used that may start small initially to reflect the observation that the

initial batch-size $\mathcal{I}_0$ is itself initially small.

Lastly, the hybrid parameter η is introduced to further generalize the algorithm framework as well as leveraging the power of SGD direction. We propose a modification of update directions according to steps 4 and 6 of Algorithm 4. This setting involves a convex combination of the trust-region and SGD directions parameterized by η. The update would reflect pure trust-region solution s and pure SGD direction $-\alpha_k g$ in two extreme cases of $\eta = 0$ and $\eta = 1$ respectively.

4. Convergence analysis

As the emphasis of this paper is on making practical enhancement for the minibatch version of L-SSR1-TR, we make similar assumptions. The following assumption is arguably ubiquitous fundamental in machine learning to justify the use of training and testing sets as surrogates for the true expected value of the model function.

Assumption 1. *There exists a bounding function $\gamma : \mathbb{R} \to \mathbb{R}$ such that $|\hat{f}(w;\mathcal{I}) - f(w)| \leq \gamma(n)$ where $\gamma(n) \to 0$ as $n \to \infty$ where $n = |\mathcal{I}|$.*

One drawback of L-SSR1-TR is the full evaluation of loss $f(w)$ at every K iteration. The cost of such a check may be prohibitive for very large data sets and undefined for cases such as online learning. An enhancement in this paper is thus to permit batched evaluation check as shown in Algorithm 6. Our only requirement is that the batch for progress check is randomly selected and not correlated with the batch used in Algorithm 4. Thus, while the total added computational burden of L-SSR1-TR over a classical stochastic approach is $N\lfloor(k)\rfloor/K$, for our approach it may remain independent of the data set size. Nevertheless, we can prove the same asymptotic convergence similar to Theorem 2 of [8].

Theorem 4.1. *If $\mu = 0, \eta = 0$, and $0 \leq \gamma_1, \gamma_2 \leq 1$ and $\gamma_1 + \gamma_2 > 0$, then with probability one either*

$$\lim_{k\to\infty} \inf \|\nabla f(w_k)\| = 0 \text{ or } \lim_{k\to\infty} \inf f(w_k) = -\infty. \tag{8}$$

Furthermore, when $\gamma_1 = 0$, for the case of online learning, the batch-size is bounded.

Proof. The proof can be divided into two parts

(1) We first prove theorem when γ_1 Suppose the batch-size n is increased a finite number of times in Algorithm 6. Then there exist an integer L such that for all iterations $k > L$ where $mod(k, K) = 0$,

$$\hat{f}(w_k) - \hat{f}_K < -\gamma_1 + \gamma_2 \sum_{j=k-K}^{k} R_j \leq -\gamma_1.$$

Let us now consider the sub-sequence of iterates $\{w_{k_j}\}_{j=1}^{\infty}$ where $k_j > L$ and mod $(k_j, K) = 0$ for all j. This implies that

$$\hat{f}(w_{k_j}) < \hat{f}(w_0) - j\gamma_1 \quad \text{for all } j \geq 1.$$

Thus $\lim_{j\to\infty} \hat{f}(w_{k_j}) \to -\infty$. Because the test batch used in the evaluation

of $\hat{f}(w)$ is randomly selected for each j, with probability one we must have $\lim_{j\to\infty} f(w_{k_j}) \to -\infty$ as well. If the batch-size is increased infinitely often then by Assumption 1 the method converges to a deterministic approach with inequality (9) and as in [8] we inherit the convergence properties of line-search methods completing the proof.

(2) We now prove the theorem when $\gamma_1 = 0$. Because the batch is not changed during the line-search, we have from standard line-search theory that

$$\hat{R}_k = \hat{f}(w_k + \alpha p) - \hat{f}(w_k) \leq \alpha\beta\hat{g}_k^T p$$

where β denotes the Armijo sufficient decrease constant $\in (0, 1)$.

First assume that the batch-size remains bounded for all iterations. Let us now consider the sub-sequence of iterates $\{w_{k_j}\}_{j=1}^{\infty}$ where $k_j > L$ and $\mod(k_j, K) = 0$ for all j. Then we have

$$\hat{f}(w_{k_j}) \leq \hat{f}(w_0) + \gamma_2 \sum_{i=0}^{k_j} R_k \leq \hat{f}(w_0) + \gamma_2 \sum_{i=0}^{k_j} \alpha_i\beta\hat{g}_i^T p.$$

Also, step 11 of Algorithm 4 ensures

$$\min\left(|\hat{g}_k^T p|, \frac{|\hat{g}_k^T p|}{\|p\|_2}\right) \geq \xi\|g_k\| \tag{9}$$

holds for some $\xi > 0$. By classic line-search theory and (9), there exists an $\hat{\alpha} > 0$ such that $\alpha_i > \hat{\alpha}$. Again by the (9) we have that either

$$\lim_{k\to\infty} \inf \|\nabla\hat{f}(w_k)\| = 0 \text{ or } \lim_{k\to\infty} \inf \hat{f}(w_k) = -\infty. \tag{10}$$

Because the test batch used in the evaluation of $\hat{f}(w)$ is randomly selected for each j, the result of the theorem follows.

For the case of online learning, if the batch-size is not bounded, we show that a contradiction will be created. With the test batch, we will then have:

$$\hat{f}(w_k) - \hat{f}_K \geq \gamma_2 \sum_{j=k-K}^{k} R_j \tag{11}$$

But with the train batch, we will then have:

$$\hat{f}(w_k) - \hat{f}_K = \sum_{j=k-K}^{k} \hat{R}_j \tag{12}$$

Taking limit on both (11) and (12), by Assumption 1 we have that

$$\sum_{j=k-K}^{k} R_j > \gamma_2 \sum_{j=k-K}^{k} R_j$$

leading to a contradiction and completing the proof.

□

Empirically, we noticed that the smallest negative eigenvalue of B_k could become very negative, perhaps due to the batch-size being too small and reduced accuracy of the corresponding secant estimate. Because we employ a trust-region approach, we prove a theorem similar to [8] that shows the search direction's dependency on the gradient g_k is dependent on the eigenvalues of B_k remaining bounded below.

Theorem 4.2. *Let λ_1 be the smallest eigenvalue of B_k. Suppose $\lambda_1 \to -\infty$ as $k \to \infty$, and δ_k in subproblem (4) is bounded away from 0 and g_k is bounded for all k. For Algorithm 4, we then have:*

- *The direction of the optimal trust-region solution, p_k^* converges to the eigen-space of B_k corresponding to unbounded eigenvalues.*
- *Suppose W_{neg} containing all the eigen-vectors w_i and its corresponding $\lambda_i < C$ and rank$(W_{neg}) < m$, we now have*
  - *If $\eta = 0$ then the momentum term v is likewise constrained in limit. Furthermore, suppose that the restart option is disabled, then we have $span(S_k) \to span(W_{neg})$ as $k \to \infty$. Therefore, rank$(S_k) < m$ and $p_k^* \in span(W_{neg})$, when k is large.*
  - *If $\eta > 0$, when k is large, and $\hat{g}_k \notin span(W_{neg})$, then we have that $\hat{g_k}$ is not parallel to p^*. Furthermore, similar to S_k, let G_k store the last k gradients, then*

$$span(S_k) = span(W_{neg}, G_k). \tag{13}$$

Proof. For simplicity we drop the suffix k notation and consider B and g. Let $B = W\Lambda_k W^T$ denote the eigen decomposition of B where the diagonals of Λ_k satisfy $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$. Let J denotes the maximum integer such that λ_J remains bounded. That is, there exists a constant C such that $\lambda_i \geq C$ for all $i \geq J$ as $k \to \infty$. Then because of the trust-region theory[21], we have that the global trust-region solution has form

$$p^* = -\sum_{\lambda_j < C}^{n} \frac{\hat{g}^T w_j}{\lambda_j + \sigma^*} w_j - \sum_{\lambda_j \geq C}^{n} \frac{\hat{g}^T w_j}{\lambda_j + \sigma^*} w_j,$$

where $\sigma^* \geq -\lambda_1$ denotes the corresponding dual multiplier for the trust-region constraint. Because $\lambda_1 \to -\infty$ we must have that $\sigma^* \to \infty$. Thus $\left\| \sum_{\lambda_j \geq C}^{n} \frac{\hat{g}^T w_j}{\lambda_j + \sigma^*} w_j \right\| \to 0$. because g_k is bounded. So p^* converges to the space of $\{w_j\}_{\lambda_J < C}$.

Now, when $\eta = 0$, Because v is a geometric average, asymptotically we must have that v converges to the same subspace as p^*. Furthermore, p^* converges to the space of $\{w_j\}_{\lambda_J < C}$ which is assumed to have rank less than m. Because $\eta = 0$ the momentum term is always a contraction to p^* and hence any contribution to the rank of S_k decays to zero.Therefore, $span(S_k) \to span(W_{neg})$ as $k \to \infty$, and rank$(S_k) < m$.

Now, when $\eta > 0$, from the above analysis about $\eta = 0$, if we let $p_1 = (1-\eta)p^* + \mu v_1$, and $v_1 = \mu v + (1-\eta)s$, we have that p_1 is not parallel to $\hat{g}$ because $\hat{g}_k \notin \text{span}(W_{\text{neg}})$. $\hat{g_k}$ is therefore not parallel to p^* . Furthermore, (13) follows based on the similar argument. □

The above theorem states that the direction of the trust-region solution, p_k^* will converge to the eigen-space of B_k corresponding to the unbounded eigenvalues. Therefore,

in order to prevent false curvature from dominating the search directions generated in Algorithm 4, we can add a component along the current gradient g_k to the search direction similar to dog-leg approaches, and above theorem states the new search space is enlarged to $\text{span}(W_{\text{neg}}, G_k)$. Therefore, when $\eta > 0$, there is a very small chance that S_k does not have full rank.

Additionally, when divergence of negative curvature of B_k is detected, we believe that it is better to restart SR1 update. Therefore in Algorithm 1 we either restart or warm-start the current (S, Y) stored to define B_k. This will hopefully give a better Hessian matrix approximation to avoid that B_k diverges.

It is hoped that the algorithm will use a first order approach when the second order model approximation is bad, and a second order approach when the approximation is good[2]. The next theorem is thus motivated and it use of an "average" trust-region ratio progress check as opposed to the classical trust-region update strategy used in [8].

Theorem 4.3. *When $\delta_k \to 0$, Algorithm 4 generates a direction similar to classical SGD approach.*

Proof. The proof follows directly from classical theory for the trust-region sub-problem solution. It is is well known that as $\delta_k \to 0$ then the solution p^* to problem 4 converges to a multiple of g_k [21]. □

Thus, if $\hat{\rho}$ (Algorithm 5), is a reliable estimate for true progress of $f(w)$, we will naturally revert back to SGD whenever second-order information does more harm than good. It is simple to see that the classical approach used in [8] cannot guarantee this behavior since the classical definition for the ratio $\hat{\rho}_k = (\hat{f}_k(w_k) - \hat{f}_k(w_{k-1}))/Q_k$ may have little bearing on the true performance metric $\rho_k = (f(w_k) - f(w_{k-1}))/Q_k$ until the batch-size is very large. This follows simply because the error $|f(w) - \hat{f}(w)|$ can be large for a small batch-size, as $\gamma(n_b)$ depends on n_b. However, by keeping a running average of both the current $\hat{R}_k = \hat{f}_k(w_k; I_{k-1}) - \hat{f}_k(w_{k-1}; I_{k-1})$ where I_k denotes the batch set used at iteration k, and the corresponding predicted reductions Q_k, we can recover this property asymptotically.

Theorem 4.4. *Suppose $\zeta = 1$, let $\rho_k^{(num)} = \frac{1}{k}\sum_{i=1}^{k} \hat{R}_k$ and $\rho_k^{(den)} = \frac{1}{k}\sum_{i=1}^{k} Q_k$, with $\hat{\rho}_k = \rho_k^{(num)}/\rho_k^{(den)}$. Then $\hat{\rho}_k$ converges to the true average improvement for $f(w)$ divided by the average of the previous predicted reductions.*

Proof. By definition $\hat{R}_k = \hat{f}(w_k; I_{k-1}) - \hat{f}(w_{k-1}; I_{k-1})$. We know $\hat{f}(w; I) = f(w) + \xi$, where $\mathbb{E}(\xi) = 0$. Then we have

$$\begin{aligned} \rho_k^{(num)} = \quad & \frac{1}{k}\sum_{i=1}^{k}(f(w_k) - f(w_{k-1})) \\ & + \frac{1}{k}\sum_{i=1}^{k}\xi_k - \frac{1}{k}\sum_{i=1}^{k}\xi_{k-1}. \end{aligned}$$

But

$$\lim_{k\to\infty} \frac{1}{k}\sum_{i=1}^{k}\xi_k = \lim_{k\to\infty} \frac{1}{k}\sum_{i=1}^{k}\xi_{k-1} = 0.$$

[2]For this study, we consider a fixed η; we leave it for future studies to consider an adaptive η_k.

And thus

$$\rho_k^{(num)} \to \frac{1}{k}\sum_{i=1}^{k}(f(w_k) - f(w_{k-1})).$$

□

Thus, the latter theorem shows that we can expect this ratio to provide a better measure of true progress made over a series of minibatch iterations. We may then expect that if progress is indeed poor, radius δ_k will reliably shrink to ensure that the algorithm behaves more like SGD when second-order correction is not currently helping progress. Note that ζ may be tuned to more aggressively decay the effect of earlier iterations, however, we found good performance by simply setting $\zeta = 1$.

5. Numerical Experiments

In this section, we present numerical investigations of the proposed method on standard machine learning problems. We first show the results on the best tuned SGD[3] on the set of imbalanced data sets, and then compare our method on a popular benchmarking neural network training tasks. We will show that our proposed method outperforms the previous similar methods, and is competitive with the best tuned stochastic first-order methods, in cases where large batch-size is required or the data set is imbalanced. It will also be shown that our method is robust to the choices of hyper-parameters, thus, requiring less tuning compared to Stochastic Gradient Descent (SGD) method. Finally, the numerical results show that the new developed method improves the results from L-SSR1-TR.

5.1. Problem with Imbalanced Data Set

A set of 27 different problems, mostly used in benchmarking the imbalanced data set scenarios, were chosen from [7]. Each of these problems demonstrate a different case with different imbalanced ratio of classes. For each problem, seven different batch-sizes n_b are considered for training a simple neural network with a single fully-connected layer with 50 neurons and tanh nonlinearity; that is, $n_b = \{16, 32, 64, 128, \frac{N}{10}, \frac{N}{8}, \frac{N}{4}\}$, maintaining an increasing order in most problems. Figure 1, depicts the accuracy obtained for the trained model using MB-LSR1 minus SGD (entry on positive side means MB-LSR1 results are better and on negative side would be in favor of SGD). It has been shown that MB-LSR1 is competitive with the best tuned SGD in cases where the data set is imbalanced.

5.2. When Large Batch-size is used

In this section, we evaluate the empirical performance of our proposed MB-LSR1 method, L-SSR1-TR [8] and stochastic first order methods including SGD, SVRG (Stochastic Variance Reduced Gradient [13]), RMSprop(Adaptive Stochastic First Order Method [26]) on the image classification tasks of MNIST [15] and CIFAR10 [14].

[3]Note that “SGD with momentum" is intended when referred to as “SGD" during the Numerical Experiments Section.

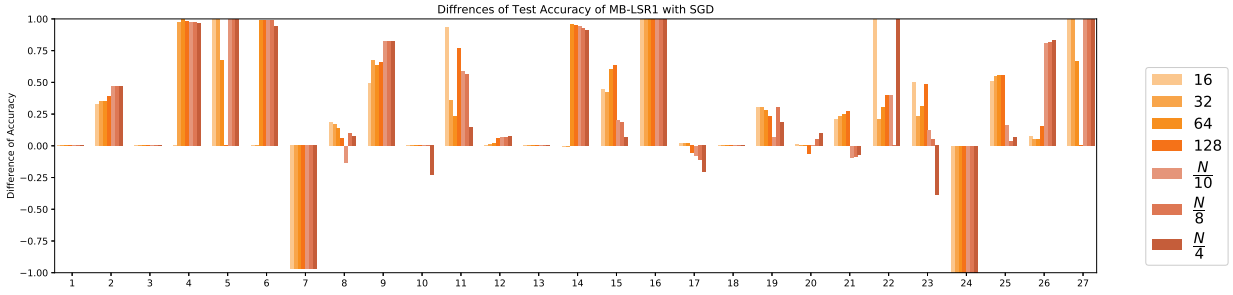

Figure 1. Difference of Test Accuracy obtained by MB-LSR1 Minus that of SGD for Imbalanced data Set

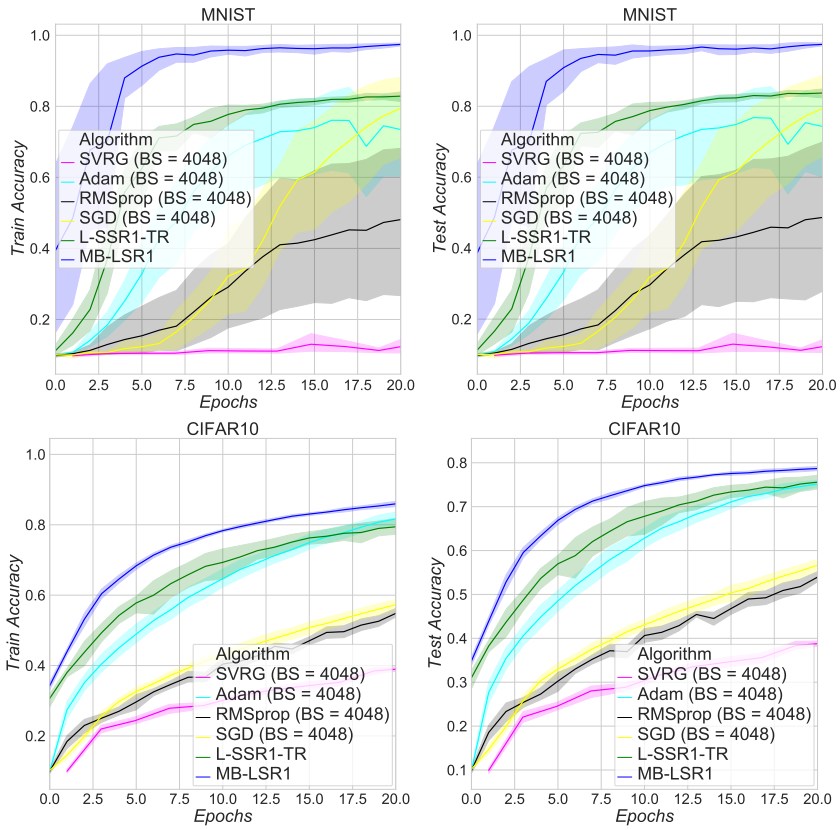

Figure 2. Comparison of train and test accuracy for SGD, Adam, SVRG, RMSProp (large batch sizes), L-SSR1-TR and MB-LSR1 on MNIST and CIFAR10 problems.

The details regarding the network structures can be found in Table 1 below.

Table 1. Deep Neural Networks used in the experiments.

Network	d	**Type**	**# of layers**
NetDNN	59.9k	conv+FC	4
ResNet20	272k	conv+BN+FC	9

conv := convolutional layers
FC := Fully Connected layers
BN := Batch Normalization

In order to have a fair comparison, we tuned each single algorithm separately. Details of the hyper-parameters explored for each algorithms during tuning phase is given in Table 2. Once tuned, the best hyper-parameter setting for each method was ran with 10 different random seeds. Figure 2 shows the results for the aforementioned algorithms for MNIST and CIFAR10 data sets based on 10 runs, with NetDNN and ResNet20 architectures respectively.

It has been shown that MB-LSR1 outperforms the best tuned stochastic first order methods (with large batches[4]) for the benchmarks on neural network tasks; Note that Figure 2 also shows that MB-LSR1 improves Erway et al. [8] results noticeably. It is clear to see from Figure 2, MB-LSR1 generalizes very well for unseen data sets, which is the main goal of learning.

5.3. Sensitivity Analysis for different Algorithms

In this section, Figure 3 has been used to show numerically that MB-LSR1 is robust w.r.t. different setting of hyper-parameters compared to SGD (with different settings of hyper-parameters including various batch sizes), as SGD requires meticulous tuning when CIFAR10 is used. The similar results have been found with MNIST. But due to limited space, only the results with CIFAR10 are given in this paper.

6. Conclusions and Future Work

In this paper, we have developed a trust-region stochastic LSR1 method to address the issues related to the noise features from using minibatch approach for deep learning problems. The major contributions of this study are three remedies for L-SSR1-TR to control for stochasticity and better adaptation to minibatch processing in large scale optimization. These involve a novel progressive trust-region radius update (suitable for stochastic models), batch evaluation instead of entire dataset evaluation to select gradient batch size and restart strategy for LSR1 when model accuracy deteriorates. We analyzed the convergence properties of our proposed method and provided the required theoretical analysis for different components of our algorithm. The numerical results show that the new developed method improve the results from L-SSR1-TR. It also can give better results compared to SGD in cases where large-batch size are needed or the data set is imbalanced. Moreover, our proposed MB-LSR1 is robust to the changes of hyper-parameters.

[4]As a clarification for the stochastic first order methods, batch-size is fixed and the rest of the hyper-parameters are tuned accordingly.

Table 2. Hyper-parameter Tuning Details

First-Order Variant	Learning-Rates	Batch-Sizes	Momentum
SGD	$\{0.01, 0.001, 0.0001\}$	$\{64, 128, 256\}$	$\{0.9, 0.95, 0.99\}$
SGD (Large Batch)	$\{0.01, 0.001, 0.0001\}$	$\{4048, 8192\}$;	$\{0.9, 0.95, 0.99\}$
Adam	$\{0.3, 0.1, 0.01, 0.001, 0.0001\}$	$\{16, 64, 128, 256\}$	-
Adam (Large Batch)	$\{0.3, 0.1, 0.01, 0.001, 0.0001\}$	$\{4048, 8192\}$	-
SVRG	$\{0.3, 0.1, 0.01, 0.001, 0.0001\}$	$\{16, 64, 128, 256\}$	-
SVRG (Large Batch)	$\{0.3, 0.1, 0.01, 0.001, 0.0001\}$	$\{4048, 8192\}$	-
RMSprop	$\{0.3, 0.1, 0.01, 0.001, 0.0001\}$	$\{16, 64, 128, 256\}$	-
RMSprop (Large Batch)	$\{0.3, 0.1, 0.01, 0.001, 0.0001\}$	$\{4048, 8192\}$	-
SR1-Variant	**Memory Set Sizes**	**Batch-Sizes**	**Initial Radius δ_0**
L-SSR1-TR	$\{16, 32, 64\}$	$\{128, 256, 1024\}$	$\{10^{-4}, 10^{-5}\}$
MB-LSR1	$\{16, 32, 64\}$	$\{128, 256, 1024\}$	$\{10^{-4}, 10^{-5}\}$

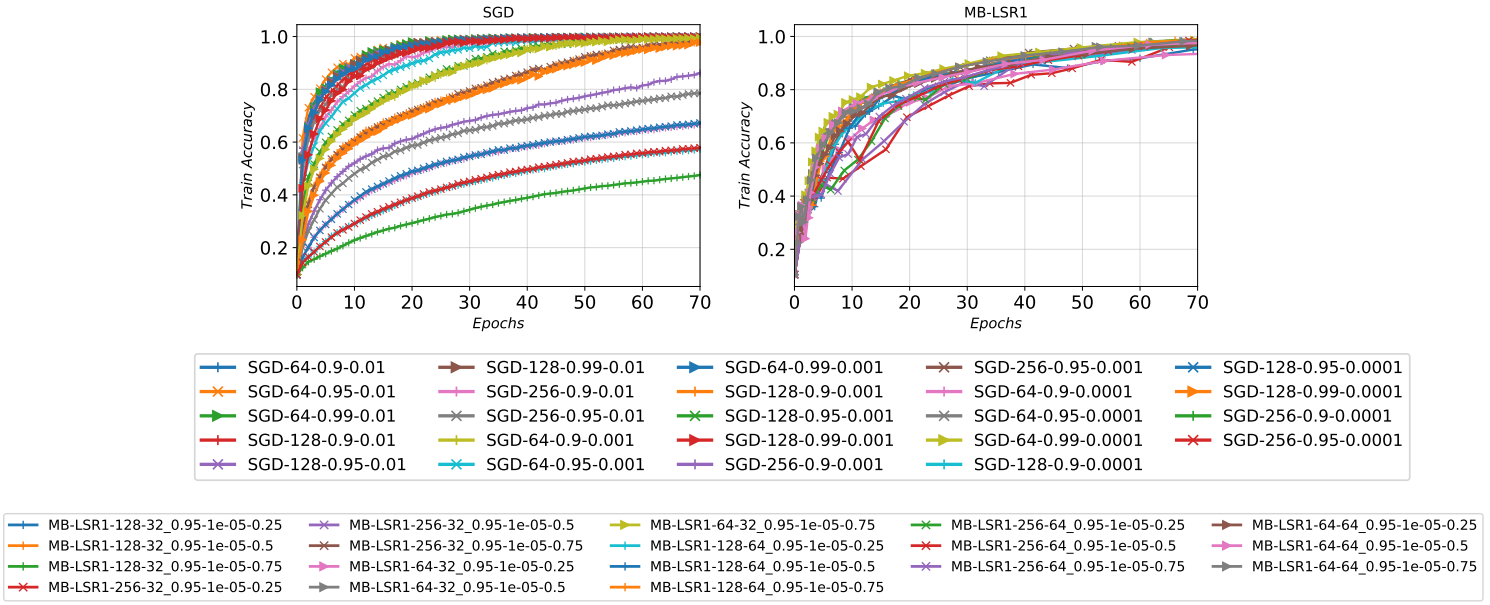

Figure 3. Sensitivity analysis of SGD and MB-LSR1 for CIFAR10 problem on ResNet20.

References

[1] A. Berahas, M. Jahani, and M. Takáč, *Quasi-newton methods for deep learning: Forget the past, just sample*, arXiv preprint arXiv:1901.09997 (2019).

[2] L. Bottou, F. Curtis, and J. Nocedal, *Optimization methods for large-scale machine learning*, SIAM Review 60 (2016), pp. 223–311.

[3] J. Brust, J. Erway, and R. Marcia, *On solving l-sr1 trust-region subproblems*, Computational Optimization and Applications 66 (2017), pp. 245–266.

[4] W. Chen, Z. Wang, and J. Zhou, *Large-scale l-bfgs using mapreduce*, Advances in Neural Information Processing Systems 27 (2014), pp. 1332–1340.

[5] A. Conn, N. Gould, and P. Toint, *Trust-Region Methods*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2000.

[6] R. Crane and F. Roosta, *Dingo: Distributed newton-type method for gradient-norm optimization*, arXiv preprint arXiv:1901.05134 (2019).

[7] Z. Ding, *Diversified ensemble classifiers for highly imbalanced data learning and their application in bioinformatics*, Ph.D. diss., Georgia State University, 2011.

[8] J. Erway, J. Griffin, R. Marcia, and R. Omheni, *Trust-region algorithms for training responses: machine learning methods using indefinite hessian approximations*, Optimization Methods and Software (2019), pp. 1–28.

[9] R. Fletcher, *Practical methods of optimization*, Wiley-Interscience [John Wiley & Sons], New York, 2001.

[10] D. Goldfarb, Y. Ren, and A. Bahamou, *Practical quasi-newton methods for training deep neural networks*, arXiv preprint arXiv:2006.08877 (2021).

[11] R. Gower, D. Goldfarb, and P. Richtarik, *Stochastic Block BFGS: Squeezing More Curvature out of Data*, in *Proceedings of The 33rd International Conference on Machine Learning*, Proceedings of Machine Learning Research, Vol. 48, 20–22 Jun, Available at `https://proceedings.mlr.press/v48/gower16.html`, PMLR, New York, New York, USA, 2016, pp. 1869–1878.

[12] M. Jahani, X. He, C. Ma, A. Mokhtari, D. Mudigere, A. Ribeiro, and M. Takáč, *Efficient distributed hessian free algorithm for large-scale empirical risk minimization via accumulating sample strategy*, arXiv preprint arXiv:1810.11507 (2018).

[13] R. Johnson and T. Zhang, *Accelerating stochastic gradient descent using predictive variance reduction*, Advances in neural information processing systems 26 (2013), pp. 315–323.

[14] A. Krizhevsky, *Learning multiple layers of features from tiny images*, Master's thesis, University of Toronto, 2009.

[15] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, Proceedings of the IEEE 86 (1998), pp. 2278–2324.

[16] X. Lu, *A study of the limited memory SR1 method in practice*, University of Colorado at Boulder, 1996.

[17] J. Martens, *Deep Learning via Hessian-Free Optimization*, in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, Haifa, Israel, Omnipress, Madison, WI, USA, 2010, p. 735–742.

[18] J. Martens and R. Grosse, *Optimizing neural networks with kronecker-factored approximate curvature*, arXiv preprint arXiv:1503.05671 (2016).

[19] J. Martens and I. Sutskever, *Learning Recurrent Neural Networks with Hessian-Free Optimization*, in *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, Bellevue, Washington, USA, Omnipress, Madison, WI, USA, 2011, p. 1033–1040.

[20] J. Martens and I. Sutskever, *Training deep and recurrent networks with hessian-free optimization*, in *Neural networks: Tricks of the trade*, Springer, 2012, pp. 479–535.

[21] J. Nocedal and S. Wright, *Numerical Optimization*, 2nd ed., Springer, New York, 2006.

[22] V. Ramamurthy and N. Duffy, *L-sr1: A second order optimization method for deep learning*, https://openreview.net/pdf?id=By1snw5gl (2017).

[23] H. Robbins and S. Monro, *A stochastic approximation method*, The Annals of Mathematical

Statistics 22 (1951), pp. 400–407.

[24] N. Schraudolph, J. Yu, and S. Günter, *A Stochastic Quasi-Newton Method for Online Convex Optimization*, in *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics*, Proceedings of Machine Learning Research, Vol. 2, 21–24 Mar, PMLR, San Juan, Puerto Rico, 2007, pp. 436–443.

[25] D. Silver, A. Huang, C.J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, *Mastering the game of go with deep neural networks and tree search*, Nature 529 (2016), p. 484–489.

[26] T. Tieleman and G. Hinton, *Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude*, COURSERA: Neural networks for machine learning 4 (2012), pp. 26–31.

[27] Y. Zhang and L. Xiao, *Communication-efficient distributed optimization of self-concordant empirical loss*, arXiv preprint arXiv:1501.00263 (2015).

[28] W. Zhou, I. Akrotirianakis, S. Yektamaram, and J. Griffin, *A matrix-free line-search algorithm for nonconvex optimization*, Optimization Methods and Software (2017), pp. 1–24.