Combined First- and Second-order directions for Deep Neural Networks Training*

Ángeles Martínez[1]0000-0003-4826-1114, Marco Viola[2]0000-0002-2140-8094, and Mahsa Yousefi[3]0000-0002-2937-9654

[1] University of Trieste, Italy
amartinez@units.it
[2] University College Dublin, Ireland
marco.viola@ucd.ie
[3] University of Trieste, Italy
MAHSA.YOUSEFI@phd.units.it

Abstract. In this work, we consider a novel stochastic optimization algorithm to solve the unconstrained, nonlinear, and non-convex optimization problems arising in the training of deep neural networks. The new algorithm is based on the combination of first- and second-order information, namely, at each step the computed search direction linearly combines a variance-reduced gradient and a stochastic limited memory quasi-Newton direction. We report computational experiments showing the performance of the proposed optimizer in the training of a modern residual deep neural network for an image classification task. The numerical results show that the proposed algorithm exhibits comparable or superior performance than the state-of-the-art Adam optimizer, without the agonizing pain of tuning many hyperparameters.

Stochastic optimization Nonlinear programming
Deep Neural Networks Training.

1 Introduction

Deep learning (DL) as a leading technique of machine learning (ML) has attracted much attention and has become one of the most popular directions of research. DL approaches have been applied to solve many large-scale problems in different fields from speech recognition and natural language processing to image classification, and have gained application in many day-life sectors like entertainment, healthcare, fraud detection, drug discovery, and many others. Deep learning relies on Deep neural networks (DNNs) which are trained over large available datasets to learn to perform a given task. Let $\mathcal{N} = \{1, 2, \ldots, N\}$ be the index set of the training dataset $\{(x_i, y_i)\}_{i=1}^{N}$ with $N = |\mathcal{N}|$ sample pairs including input $x_i \in \mathbb{R}^d$ and target $y_i \in \mathbb{R}^C$. DL problems are often formulated as unconstrained optimization problems with an empirical risk function, in which

* Supported by INdAM — GNCS Project CUP_E53C22001930001.

a parametric function $h(x_i;\cdot) : \mathbb{R}^d \longrightarrow \mathbb{R}$ is found such that the prediction errors are minimized. More precisely, we obtain the following problem

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

where $w \in \mathbb{R}^n$ is the vector of trainable parameters and $f_i(w) \triangleq L(y_i, h(x_i; w))$, where $L(\cdot)$ is a relevant loss function which measures the prediction error between the target y_i and the network's output $h(x_i; w)$. It is not straightforward to apply traditional optimization algorithms to solve the large-scale problem (1) which is highly nonlinear and often non-convex. That is why much effort has recently been devoted to the development of DL optimization algorithms. Popular DL optimization methods can be divided into two *general* categories: (i) *first-order* methods using gradient information, e.g. steepest gradient descent and its variants and (ii) *second-order* methods using also curvature information, e.g. Newton or quasi-Newton methods [1]. In DL applications, usually the number of training samples N is large and the network requires a huge number of parameters n so that computing function values and gradients is expensive, and computations involving the Hessian $\nabla^2 f(w)$ are impractical. Alternatively, stochastic optimization methods using *subsampling* can be applied. At each iteration of these types of methods, an approximation of the objective function is evaluated concerning a random subset (mini-batch) of the training data whose index set is $\mathcal{N}_k \subseteq \mathcal{N}$ with sample size $N_k = |\mathcal{N}_k|$. Usually, N_k is much smaller than N for $k = 0, 1, \ldots$ so that algorithms operate in the stochastic regime. In this regime, the (stochastic) subsampled function and its (stochastic) subsampled gradient are, respectively, defined as follows

$$f_{\mathcal{N}_k}(w) = \frac{1}{N_k}\sum_{i\in\mathcal{N}_k} f_i(w), \tag{2}$$

and

$$\nabla f_{\mathcal{N}_k}(w) = \frac{1}{N_k}\sum_{i\in\mathcal{N}_k} \nabla f_i(w). \tag{3}$$

A line-search method for solving the unconstrained optimization problem (1) requires the search direction to be a descent direction. However, this condition alone does not ensure convergence, and the search direction must satisfy the angle criterion

$$\cos(-\hat{g}_k,\, p_k) = \frac{-\hat{g}_k^T p_k}{\|\hat{g}_k\|\|p_k\|} \geq \epsilon_k, \quad \epsilon_k > 0, \tag{4}$$

where $\hat{g}_k$ denotes the gradient of f at w_k, $\hat{g}_k = \nabla f(w_k)$, and the sequence $\{\epsilon_k\}$ is bounded away from 0, which means that the angle between the search direction (p_k) and the steepest descent direction $(-g_k)$ must be bounded away from the right angle. A globalization approach applicable to any Newton-type method was proposed in [16] to address the case in which the angle condition doesn't hold. The basic idea consists of linearly combining the Newton-type (e.g., Newton or

quasi-Newton) and Steepest Descent (SD) directions. The goal is to bring the iterates sufficiently close to a solution through the globally convergent Steepest Descent method so that once the iterates are in the basin of attraction of the Newton-type method, it can lead to faster convergence. In this work, we aim to extend this idea to the stochastic regime to devise an algorithm able to efficiently train DNNs.

The rest of the paper is organized as follows. In Section 2 we describe the stochastic trust-region quasi-Newton algorithm used to compute the second-order information. Section 3 introduces the proposed hybrid algorithm which combines first- and second-order directions. Section 4 describes the computational experiments we have performed to assess the performance of the proposed algorithm in the training of a deep convolutional neural network for image classification tasks. We also include a comparison with Adam optimizer [10] with optimal parameters obtained through grid-searching. Adam is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments and it's considered the state-of-the-art optimizer for deep learning applications. Section 5 ends the paper with some concluding remarks.

2 Stochastic TR quasi-Newton Algorithms

In a trust-region framework, the original nonconvex optimization problem (1) is reduced to a sequence of quadratic optimization problems. In the stochastic setting, at each iteration of the trust-region algorithm, a random mini-batch J_k of the training set is selected, with index set $\mathcal{N}_k \subseteq \mathcal{N}$ and size $N_k = |\mathcal{N}_k|$, and the subsampled objective function and corresponding gradients are evaluated as defined in (2) and (3) with respect to the selected mini-batch. Using this information, the search direction p_k is obtained at each iteration as follows

$$p_k = \arg\min_{p \in \mathbb{R}^n} Q_k(p) \triangleq \frac{1}{2} p^T B_k p + g_k^T p \quad \text{s.t.} \quad \|p\|_2 \leq \delta_k, \quad k = 0, 1, \ldots, \tag{5}$$

for some $\delta_k > 0$, where B_k is an approximation of the Hessian and g_k is the subsampled gradient (3) which is evaluated at w_k i.e.,

$$g_k = \nabla f_{\mathcal{N}_k}(w_k).$$

In this fashion, a sequence $\{w_k\}$ is produced whether the trial point $w_t = w_k + p_k$ is accepted or rejected. Moreover, the positive constant δ_k in (5) is the radius of the region in which the agreement between the model and the (subsampled) function can be trusted. The adjustment of δ_k, as well as the decision of accepting or rejecting the trial point w_t at each iteration, relies on the value of the so-called actual to predicted reduction ratio

$$\rho_k = \frac{f_{\mathcal{N}_k}(w_t) - f_{\mathcal{N}_k}(w_k)}{Q_k(p_k) - Q_k(0)}. \tag{6}$$

Precisely, it is safe to expand $\delta_k \in (\delta_0, \delta_{max})$ with δ_0, $\delta_{max} > 0$ when $\rho_k \approx 1$ meaning that there is a *very good* agreement between the model and the

Algorithm 1 Trust-Region radius adjustment

Inputs: Current iteration k, p_k, δ_k, ρ_k, $0 < \tau_2 < 0.5 < \tau_3 < 1$, $0 < \eta_2 \leq 0.5$, $0.5 < \eta_3 < 1 < \eta_4$
if $\rho_k > \tau_3$ **then**
  if $\|p_k\| \leq \eta_3 \delta_k$ **then**
    $\delta_{k+1} = \delta_k$
  else
    $\delta_{k+1} = \min(\eta_4 \delta_k, \delta_{max})$
  end if
else if $\tau_2 \leq \rho_k \leq \tau_3$ **then**
  $\delta_{k+1} = \delta_k$
else
  $\delta_{k+1} = \eta_2 \delta_k$
end if

(subsampled) objective function. Instead, the current δ_k is not changed if there is a *good* agreement, or it is shrunk when there is only a *poor* agreement between the quadratic model and the (subsampled) objective function. Typically, this adjustment is done as described in Algorithm 1. Moreover, since the denominator in (6) is nonpositive, we have $w_{k+1} \triangleq w_t$ if ρ_k is positive ($\rho_k > \eta$); otherwise, $w_{k+1} \triangleq w_k$.

We consider the case in which the Hessian approximation B_k is obtained employing a limited-memory quasi-Newton update from the Broyden class [14]. In this case, matrix B_k is iteratively constructed so that the *secant condition*

$$B_{k+1} s_k = y_k \tag{7}$$

holds with curvature pair (s_k, y_k) where

$$s_k = p_k, \qquad y_k = g_t - g_k, \tag{8}$$

and

$$g_t = \nabla f_{\mathcal{N}_k}(w_t).$$

More precisely, we employ the SR1 (Symmetric Rank-one) formula which generates good approximations to the true Hessian matrix, often better than those provided by the well-known BFGS update [14]. The general SR1 updating formula verifying the secant condition (7) is given by

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{(y_k - B_k s_k)^T s_k}, \quad k = 0, 1, \ldots, \tag{9}$$

where the difference between the symmetric approximations B_k and B_{k+1} is a rank-one matrix. To prevent the condition number of the Hessian approximation B_{k+1} from exploding, a simple safeguard that works well in practice is simply skipping the update if the denominator is small [14]; i.e., $B_{k+1} = B_k$. In this study, B_k is updated only if the following rule, given $\tau = 10^{-8}$, holds

$$|s^T (y_k - B_k s_k)| \geq \tau \|s_k\| \|y_k - B_k s_k\|, \qquad \tau \in (0, 1). \tag{10}$$

If B_k in (9) is positive definite, B_{k+1} may have not the same property. The SR1 update generates a sequence of matrices that may be indefinite regardless of the sign of $y_k^T s_k$ for each k. We note that the value of the quadratic model in (5) evaluated at the descent direction p_k is always smaller if this direction is also a direction of negative curvature. Therefore, the ability to generate indefinite approximations can be regarded as one of the chief advantages of SR1 updates in non-convex settings like DL applications.

The main computational burden of most TR methods is in solving the trust-region subproblem (5) to compute the search direction. In [9], an efficient algorithm named OBS was proposed for solving (5) where the Hessian approximation B_k is obtained by quasi-Newton updates and is not necessarily positive definite. The OBS algorithm exploits the compact form of the limited-memory SR1 updated matrix B_k:

$$B_k = B_0 + \Psi_k M_k \Psi_k^T, \quad k = 1, 2, , \ldots, \tag{11}$$

where

$$\Psi_k = Y_k - B_0 S_k, \qquad M_k = (D_k + L_k + L_k^T - S_k^T B_0 S_k)^{-1}, \tag{12}$$

and $B_0 = \gamma_k I$. We used OBS and selected the value of γ_k following the initialization strategy proposed in [4].

3 A Stochastic Hybrid Method

In this section, we describe a new stochastic TR algorithm making use of a combination strategy to address the case in which the second-order direction determined by the stochastic quasi-Newton TR method described in the previous section does not satisfy the angle condition. The new algorithm also incorporates a line-search step to avoid resolving the TR subproblem due to occasional step rejections. More in detail, the new algorithm uses LSR1 Hessian approximation to produce the corresponding second-order LSR1 direction p_k which is employed as long as

$$\cos\left(-v_k,\, p_k\right) = \frac{-v_k^T p_k}{\|g_k\| \|p_k\|} \geq \epsilon_k, \quad \epsilon_k > 0, \tag{13}$$

where v_k is a stochastic approximation of the SD direction. At any iteration in which the condition (13) is not satisfied, the algorithm computes a new search direction by the combination of the first- and second-order directions. The combination strategy we propose is as follows

$$p_k^C = \beta_k p_k - (1 - \beta_k) \xi_k v_k, \tag{14}$$

where $0 \leq \beta_k \leq 1$ and $\xi_k > 0$. In (14), ξ_k scales the first-order direction v_k and β_k guarantees that the combined direction p_k^C in (14) satisfies (13), i.e., $\cos\left(-v_k,\, p_k^C\right) \geq \epsilon_k$. Our method computes the parameter β_k as it is obtained according to Theorem 1 in [16] where β_k is a lower bound of the smallest root in $(0, 1)$ of a certain second-order polynomial; i.e.,

$$\beta_k = \frac{r_k}{r_k + \pi_k}, \tag{15}$$

where

$$r_k = \xi_k(1-\epsilon_k), \qquad \pi_k = \frac{v_k^T p_k}{\|v_k\|^2} + \epsilon_k \frac{\|p_k\|}{\|v_k\|}.$$

Selecting the scaling parameter $\boldsymbol{\xi_k}$ A suitable scaling of v_k is a key issue in making this approach effective, there are different strategies to choose this parameter, see [16] and references therein. In our algorithm, we adaptively set this parameter as follows

$$\xi_k = \frac{\delta_k}{\|v_k\|}, \tag{16}$$

to ensure that the combined direction p_k^C is pushed to be still inside the trust region.

Selecting the first-order direction $\boldsymbol{v_k}$ As already mentioned, the first-order direction v_k in (14) plays the role of the SD direction, even if it is only an approximation of the true SD direction in the stochastic setting due to the error introduced by subsampling. There is a *variance reduction* family of algorithms that exploit the information obtained from periodical computation of the full gradient strategically to try to eliminate the noise in the subsampled gradient; they save a full gradient evaluation at a reference point, known as *snapshot gradient*, and modify it using stochastic gradient estimates to form the variance reduced gradient in the subsequent iterations. Therefore, a natural choice of v_k can be a first-order variance reduced (VR) direction. Three representative variance-reducing techniques are mini-batch SVRG [9], SAGA [3], and SARAH [12] algorithms, which compute the variance-reduced gradient at iteration k are listed below (for SAGA we consider a reduced memory version):

1. *Reduced memory* variant of SAGA:

$$v_k = \nabla f_{\mathcal{N}_k}(w_k) - \frac{1}{|\mathcal{N}_k|}\sum_{i\in\mathcal{N}_k} J^k_{:,i} + \frac{1}{|\mathcal{N}|}\sum_{i\in\mathcal{N}} J^k_{:,i}, \tag{17}$$

   where

$$J^{k+1}_{:,i} = \begin{cases} J^k_{:,i}, & \text{if} \quad i \notin \mathcal{N}_k, \\ \frac{1}{|\mathcal{N}_k|}\sum_{i\in\mathcal{N}_k} J^k_{:,i}, & \text{if} \quad i \in \mathcal{N}_k, \end{cases}$$

2. SAGA:

$$v_k = \nabla f_{\mathcal{N}_k}(w_k) - \frac{1}{|\mathcal{N}_k|}\sum_{i\in\mathcal{N}_k} J^k_{:,i} + \frac{1}{|\mathcal{N}|}\sum_{i\in\mathcal{N}} J^k_{:,i}, \tag{18}$$

   where

$$J^{k+1}_{:,i} = \begin{cases} J^k_{:,i}, & \text{if} \quad i \notin \mathcal{N}_k, \\ \frac{1}{|\mathcal{N}_k|}\sum_{i\in\mathcal{N}_k} \nabla f_i(w_k), & \text{if} \quad i \in \mathcal{N}_k, \end{cases}$$

   and $J^0_{:,i} = \nabla f_i(w_0)$.

3. SVRG:
$$v_k = \nabla f_{\mathcal{N}_k}(w_k) - \nabla f_{\mathcal{N}_k}(w_s) + \nabla f(w_s), \tag{19}$$
where s is a previous iteration ($s < k$).
4. SARAH:
$$v_k = \nabla f_{\mathcal{N}_k}(w_k) - \nabla f_{\mathcal{N}_k}(w_{k-1}) + v_{k-1}, \quad v_s = \nabla f(w_s). \tag{20}$$

Algorithm 2 sCLSR1-TR

Inputs: $k = 0,\ w_0,\ S_0 = Y_0 = [\,],\ \gamma_0, \delta_0 > 0,\ T = 1, \epsilon,\ \nu \in (0,1)$, bs, `eps`: machine epsilon
for *epoch* $= 1, 2, \ldots,$ **do**
  Shuffle N samples for randomly creating N_b mini-batches
  for *iter* $= 1, 2, \cdots, N_b$ **do**
    Compute $f_{\mathcal{N}_k}(w_k)$ and $g_k \triangleq \nabla f_{\mathcal{N}_k}(w_k)$ w.r.t. a given $\mathcal{N}_k$ of size $N_k = bs$
    if Some stopping conditions hold **then**
      Stop training
    end if
    if $k = 0$ or $S_k = [\,]$ **then**
      Set $B_k = \gamma_0 I$, and compute $p_k = \frac{-\delta_k g_k}{\|g_k\|}$
    else
      Compute $B_k = \gamma_k I + \Psi_k M_k^{-1} \Psi_k^T$, and find p_k using the OBS solver
    end if
    Find v_k as a first-order direction
    if $\cos(-v_k,\, p_k) \geq \epsilon_k$ **then**
      $p_k^C = p_k$
      $\epsilon_{k+1} = \epsilon_k$
    else
      Compute $p_k^C = \beta_k p_k - (1 - \beta_k)\xi_k v_k$ such that $\cos(-v_k,\, p_k^C) \geq \epsilon_k$
      $\epsilon_{k+1} = \max\{10\,\texttt{eps}, \nu\epsilon_k\}$
    end if
    Find a step-length α_k by backtracking and Armijo rule; i.e.
$$f_{\mathcal{N}_k}(w_k + \alpha_k p_k^C) \leq f_{\mathcal{N}_k}(w_k) + c_1 \alpha_k v_k^T p_k^C$$
    Set $w_{k+1} = w_k + \alpha_k p_k^C$
    Evaluate $f_{\mathcal{N}_k}(w_{k+1})$ and $g_{k+1} \triangleq \nabla f_{\mathcal{N}_k}(w_{k+1})$
    Compute $s_k = \alpha_k p_k^C$, and $y_k = g_{k+1} - g_k$
    Find γ_{k+1} and construct a new *well-defined* B_{k+1} as in (11)
    Compute ρ_k via (6)
    $\hat{\rho}_k = T\hat{\rho}_k + \rho_k$
    $T = T + 1$
    $\hat{\rho}_k = \frac{\hat{\rho}_k}{T}$
    Update δ_k by Algorithm 1 with $\hat{\rho}_k$ and s_k
  end for
  $k = k + 1$
end for

Let's assume the index set $\mathcal{N}$ is partitioned into a fixed-number N_b of random mini-batches of the same size. The reduced memory variant of mini-batch SAGA, like the standard mini-batch SAGA, requires computing the full gradient at the beginning of the algorithm. Unlike the standard SAGA algorithm, however, it needs the storage of N_b previously computed stochastic gradients so that columns of the storage matrix $J \in \mathbb{R}^{n \times N_b}$ can be individually updated as the method progresses. In contrast, the mini-batch SVRG does not require extra storage but it does require computing the full gradient periodically; see e.g. [15]. In fact, at each outer iteration, SVRG computes one full gradient $\nabla f(w_s)$, where $s < k$ is a previous iterate, then takes t steps along random directions to obtain stochastic corrections of this full gradient. SARAH combines some ideas from SVRG and SAGA and differs from SVRG in terms of the inner loop step. The SARAH VR direction (20) involves a gradient snapshot which is also updated at each inner iteration and thus exhibits better convergence properties than SVRG. See [6, 15, 13], respectively, for more details on the VR directions listed above.

We selected a reduced memory SAGA gradient as the first-order search direction v_k. With this approach, the hybrid algorithm needs one full gradient evaluation and a storage matrix with N_b columns of length n. Moreover, we can get rid of finding the optimum value of t as well as computing a full gradient periodically every t iterations. We should notice that, in practice, SVRG can be quite sensitive to the choice of t [2, 9].

We consider the resulting hybrid TR algorithm obtained using the combination strategy described above with the LSR1 and reduced memory SAGA gradient directions. We called this procedure sCLSR1-TR and described it in Algorithm 2. Given the search direction computed at iteration k, the proposed algorithm finds a suitable step length through a line-search step and then updates w_k. The line-search step in line 22 of sCLSR1-TR avoids resolving again the TR subproblem due to occasional rejections of iterates. We also relax the progress of ρ_k by averaging past ratios [7]; in fact, we soften its effect. This averaging value, i.e., $\hat{\rho}_k$ in line 27 of sCLSR1-TR, is employed to adjust the TR radius δ_k.

3.1 Regularization

Regularization is a broad term that describes attempts to avoid overfitting. Overfitting occurs when a trained network performs accurately on the given data, but cannot generalize well to new data. Since large weights may lead to neurons that are sensitive to their inputs and hence less reliable when new data is presented, one can modify the objective function in order to encourage small weights by using L_2 regularization as follows

$$\min_{w \in \mathbb{R}^n} f(w) \triangleq \frac{1}{N}\Big(\sum_{i \in \mathcal{N}} f_i(w) + \frac{\lambda}{2}\|w\|_2^2\Big), \tag{21}$$

where $\lambda > 0$ is the regularization factor.

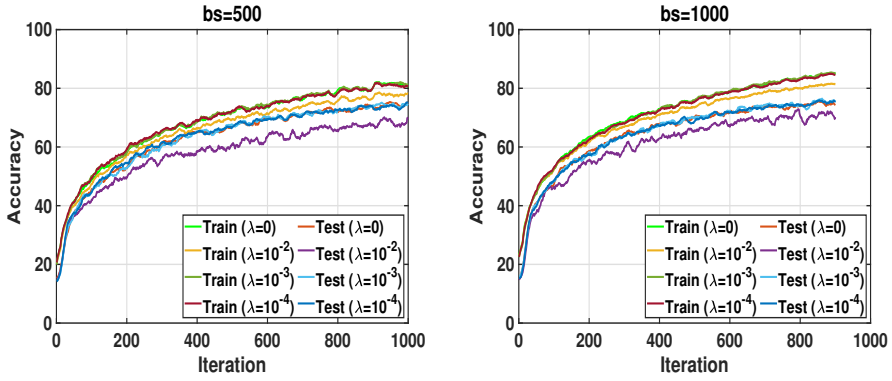

Fig. 1: Effect of regularization on the accuracy provided by sCLSR1-TR after 10 (left) and 18 (right) epochs.

4 Computational experiments

To illustrate the performance of the proposed algorithm we coded it in MATLAB and applied it to the solution of the regularized DL optimization problem (21) arising from the training of `ResNet20` [8] to classify the `CIFAR10` dataset [4][11].

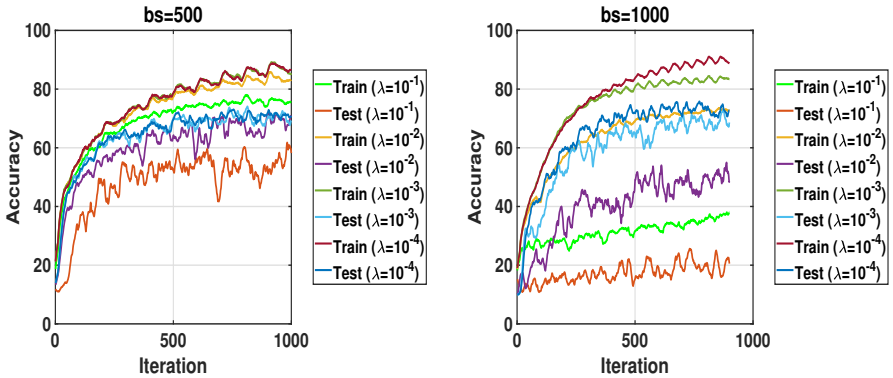

Fig. 2: Effect of regularization on the accuracy provided by Adam after 10 (left) and 18 (right) epochs.

The `CIFAR10` dataset consists of 60×10^3 RGB images of 32×32 pixels, belonging to ten different categories. The images dataset is divided into training and testing sets including $N = 50 \times 10^3$ and $\hat{N} = 10 \times 10^3$ samples, respectively.

[4] `https://www.mathworks.com/help/deeplearning/ug/data-sets-for-deep-learning.html`

The pixels of each image are in the range $[0, 255]$ and are divided by 255 so that the pixel intensity range is bounded in $[0, 1]$ (zero-one rescaling). We applied also z-score normalization through the input layer, i.e., data standardization to have zero mean and a standard deviation of 1.

We compared the performance of the new algorithm with the state-of-the-art Adam optimizer [10] massively employed in deep learning applications. All experiments were conducted with the MATLAB DL toolbox on a Ubuntu 20.04.4 LTS (64-bit) Linux server VMware with 20GB memory using a VGPU NVIDIA A100D-20C. We allow the code to run for a fixed budget of time, namely 90 minutes. To define the initialized network and training loop of our algorithm, we have used `dlarray` and `dlnetwork` MATLAB objects; see [18] for more information. The network's parameters $w \in \mathbb{R}^n$, i.e. weights and biases of the convolutional layers, were initialized using the Glorot (Xavier) initializer [5] and zeros, respectively. We used the same initial parameter $w_0 \in \mathbb{R}^n$ by specifying the same seed to the MATLAB random number generator in all experiments, for a fair comparison between the two algorithms. The scale and offset variables of the batch normalization layers were initialized to ones and zeros, respectively.

We considered two batch sizes $|\mathcal{N}_k| = bs$, $bs = 500$ and $bs = 1000$, yielding 10 and 18 epochs, respectively. We tested four different values of the L2 regularization parameter for sCLSR1-TR, namely $\lambda \in \{10^{-2}, 10^{-3}, 10^{-4}, 0\}$ and four values for grid-searching Adam, that is, $\lambda \in \{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$. Grid-searching for tuning Adam included also three different values for the learning rate parameter $\alpha \in \{10^{-1}, 10^{-2}, 10^{-3}\}$. The limited memory parameter for LSR1 Hessian approximation was set to $l = 30$. For a detailed study on the effect of the limited memory parameter in the final accuracy reached by stochastic quasi-Newton trust-region methods, we refer the reader to [17].

In all experiments, the deep neural networks were trained for a fixed amount of time. Training could finish ahead of that time if 100% accuracy had been reached. The accuracy is the ratio of the number of correct predictions to the number of total predictions. Our study reports the accuracy in percentage and overall loss values for both train and test datasets. Following prior published works in the optimization community (see e.g., [4]) we use the whole testing set as the validation set, that is, at the end of each iteration of the training phase (after the network parameters have been updated) the prediction capability of the recently updated network is evaluated using all the samples of the test dataset. The computed value is the measured testing accuracy corresponding to iteration k. Consequently, we report accuracy and loss across epochs for both training samples and unseen samples of the validation set (=test set) during the training phase. Accuracy plots showing the evolution of the achieved accuracy at each iteration are obtained reporting c-point mean values of the measured accuracies, where each mean is computed over a sliding window of length c across neighboring elements of the saved accuracy vector. We set $c = 20$.

The results of the experiments are reported in Figures 1–4.

We can observe from Figure 1 and Figure 2 that, differently from Adam which is highly sensitive to the values of its hyper-parameters (including the L2

regularization parameter λ and the learning rate α), our method provides similar accuracy for all tested values of λ except for the largest one, i.e. $\lambda = 10^{-2}$. This indicates that sCLSR1-TR can be employed for training purposes without any need to apply L2 regularization. Alternatively, if regularization is desired, it is recommended to utilize small values of the parameter λ.

As displayed in Figure 3, the proposed hybrid algorithm without regularization and within a fixed number of epochs achieves similar or superior performance compared to tuned Adam. We recall that an epoch is a complete pass through the whole data sample set.

Regarding the training time, each iteration of the proposed hybrid algorithm requires more time on a GPU compared to tuned Adam, due to its higher iteration complexity (two subsampled function and gradient evaluations instead of one as in Adam). Nevertheless, given a fixed budget of time, e.g. 90 minutes in our experiments, the results in Figure 4 show that sCLSR1-TR attains the same level of testing accuracy as tuned Adam when a batch-size of $bs = 1000$ samples is used (left plot). The testing accuracy provided by sCLSR1-TR is higher than the one provided by Adam for the smaller batch size (right plot).

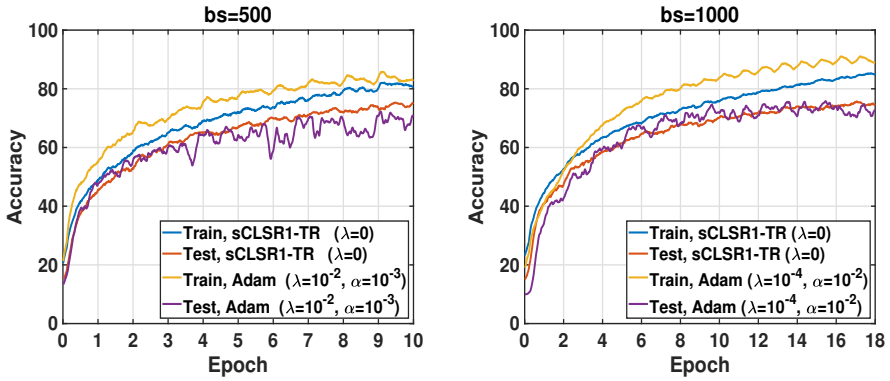

Fig. 3: Accuracy vs epochs for sCLSR1-TR and *tuned* Adam.

5 Conclusions

In this paper, we have developed a novel second-order stochastic optimization algorithm based on the combination of first- and second-order information. The algorithm combines a trust-region limited memory SR1 second-order direction with a variance-reduced stochastic gradient and is tailored to efficiently address the solution of unconstrained, nonlinear, and non-convex optimization problems arising in the training of deep neural networks. We have reported computational experiments showing that the proposed algorithm exhibits comparable

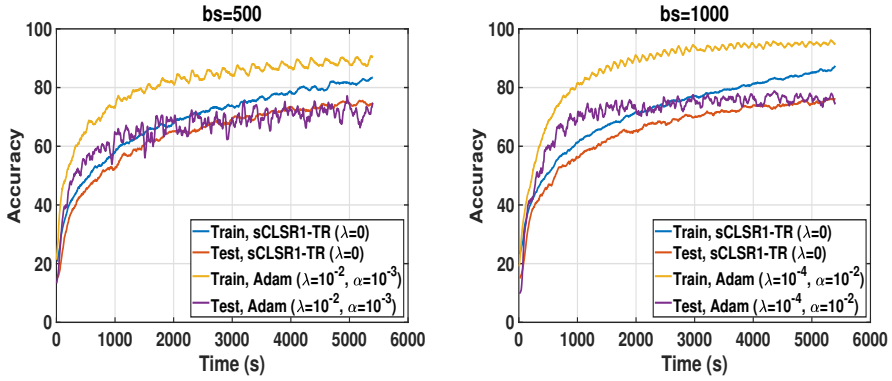

Fig. 4: Accuracy vs time for sCLSR1-TR and *tuned* Adam after 90 minutes training.

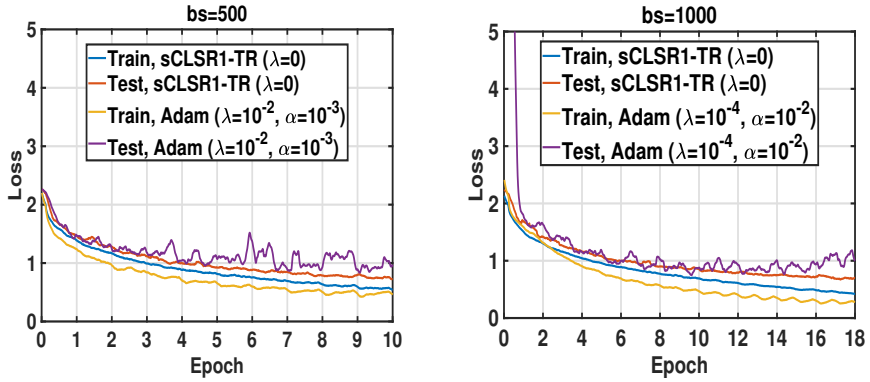

Fig. 5: Loss vs epoch for sCLSR1-TR and *tuned* Adam.

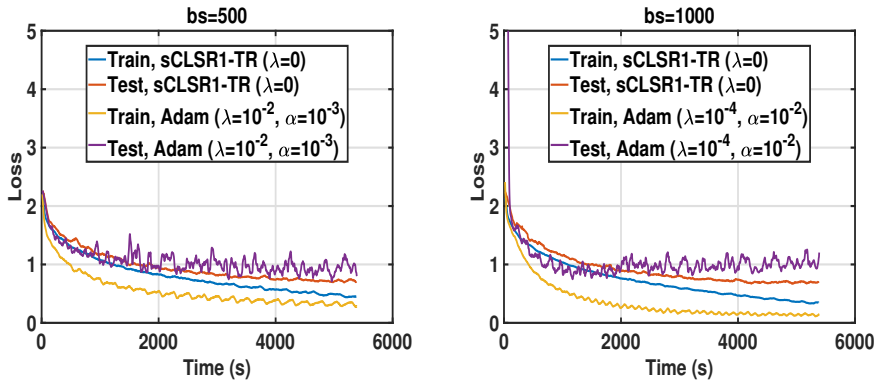

Fig. 6: Loss vs time for sCLSR1-TR and *tuned* Adam within 90 minutes training.

or superior performance to the state-of-the-art Adam optimizer, while requiring significantly less tuning effort.

Acknowledgments

The authors gratefully acknowledge the support of INdAM — GNCS Project CUP_E53C22001930001. This study was carried out within the PNRR research activities of the consortium iNEST (Interconnected North-Est Innovation Ecosystem) funded by the European Union Next-GenerationEU (Piano Nazionale di Ripresa e Resilienza (PNRR) – Missione 4 Componente 2, Investimento 1.5 – D.D. 1058 23/06/2022, ECS_00000043). This manuscript reflects only the Authors' views and opinions, neither the European Union nor the European Commission can be considered responsible for them

The authors want to express this tribute to the memory of our dear colleague and friend Daniela di Serafino. We lost her along the path from the initial idea and experiments to the submission of this manuscript. We all will miss her and her infinite passion for research.

References

1. Bottou, L., Curtis, F.E., Nocedal, J.: Optimization methods for large-scale machine learning. SIAM Review **60**(2), 223–311 (2018). https://doi.org/10.1137/16M1080173
2. Curtis, F.E., Scheinberg, K.: Optimization methods for supervised machine learning: From linear models to deep learning. In: Leading Developments from INFORMS Communities, pp. 89–114. INFORMS (2017)
3. Defazio, A., Bach, F., Lacoste-Julien, S.: SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. In: Advances in neural information processing systems. pp. 1646–1654 (2014)
4. Erway, J.B., Griffin, J., Marcia, R.F., Omheni, R.: Trust-region algorithms for training responses: machine learning methods using indefinite Hessian approximations. Optimization Methods and Software **35**(3), 460–487 (2020)
5. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the thirteenth international conference on artificial intelligence and statistics. pp. 249–256. JMLR Workshop and Conference Proceedings (2010)
6. Gower, R.M., Richtárik, P., Bach, F.: Stochastic quasi-gradient methods: Variance reduction via jacobian sketching. Mathematical Programming **188**, 135–192 (2021)
7. Griffin, J.D., Jahani, M., Takac, M., Yektamaram, S., Zhou, W.: A minibatch stochastic quasi-newton method adapted for nonconvex deep learning problems. Optimization Online preprint https://optimization-online.org/?p=18601 (2022)
8. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
9. Johnson, R., Zhang, T.: Accelerating stochastic gradient descent using predictive variance reduction. Advances in neural information processing systems **26**, 315–323 (2013)

10. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: 3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings (2015)
11. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images. Available at: https://www.cs.toronto.edu/~kriz/cifar.html (2009)
12. Nguyen, L.M., Liu, J., Scheinberg, K., Takáč, M.: SARAH: A novel method for machine learning problems using stochastic recursive gradient. In: International Conference on Machine Learning. pp. 2613–2621. PMLR (2017)
13. Nguyen, L.M., Liu, J., Scheinberg, K., Takáč, M.: Stochastic recursive gradient algorithm for nonconvex optimization. arXiv preprint arXiv:1705.07261 (2017)
14. Nocedal, J., Wright, S.: Numerical optimization. Springer Series in Operations Research and Financial Engineering, Springer Science & Business Media (2006)
15. Reddi, S.J., Hefny, A., Sra, S., Poczos, B., Smola, A.: Stochastic variance reduction for nonconvex optimization. In: International conference on machine learning. pp. 314–323. PMLR (2016)
16. di Serafino, D., Toraldo, G., Viola, M.: Using gradient directions to get global convergence of newton-type methods. Applied Mathematics and Computation **409**, 125612 (2021)
17. Yousefi, M., Martínez, A.: Deep neural networks training by stochastic quasi-newton trust-region methods. Algorithms **16**(10) (2023), https://doi.org/10.3390/a16100490
18. Yousefi, M., Martínez Calomardo, Á.: A matlab-based tutorial on implementing custom loops for training a deep neural network (2022), http://doi.org/10.13140/RG.2.2.33008.94720