Gradient-only surrogate to resolve learning rates for robust and consistent training of deep neural networks

Younghwan Chae[1] · Daniel N. Wilke[1] · Dominic Kafka[1]

Abstract

Mini-batch sub-sampling (MBSS) is favored in deep neural network training to reduce the computational cost. Still, it introduces an inherent sampling error, making the selection of appropriate learning rates challenging. The sampling errors can manifest either as a bias or variances in a line search. Dynamic MBSS re-samples a mini-batch at every function evaluation. Hence, dynamic MBSS results in point-wise discontinuous loss functions with smaller bias but larger variance than static sampled loss functions. However, dynamic MBSS has the advantage of having larger data throughput during training but requires resolving the complexity regarding discontinuities. This study extends the vanilla gradient-only surrogate line search (GOS-LS), a line search method using quadratic approximation models built with only directional derivative information for dynamic MBSS loss functions. We propose a conservative gradient-only surrogate line search (GOS-LSC) with strong convergence characteristics with a defined optimality criterion. For the first time, we investigate both GOS-LS's and GOS-LSC's performance on various optimizers, including SGD, RMSProp, and Adam on ResNet-18 and EfficientNet-B0. We also compare GOS-LS and GOS-LSC against the other existing learning rate methods. We quantify both the best-performing and most robust algorithms. For the latter, we introduce a relative robust criterion that allows us to quantify the difference between an algorithm and the best performing algorithm for a given problem. The results show that training a model with the recommended learning rate for a class of search directions helps to reduce the model errors in multimodal cases. The results also show that GOS-LS ranked first in training and test results, while GOS-LSC ranked third and second in training and test results among nine other learning rate strategies.

Keywords Line search · Learning rate · Approximation model · Stochastic gradient · SNN-GPP

1 Introduction

In neural network training, choosing appropriate learning rates or schedules is non-trivial [3, 11, 27]. As the neural network architectures become larger and more complex, the cost of training increases significantly [6]. The monetary, energy, and CO_2 emission consequence of selecting a training strategy with significant performance variance, i.e. having near-optimal or poor training performance when combined with various optimizers, neural network architectures, and datasets, is noteworthy and important. Therefore, it becomes more and more important to formally quantify the robustness and consistency of a training approach instead of only quantifying its best performance. This also highlights the inherent Pareto optimal nature of selecting a training approach optimal for a specific application (specialist) or adequate over a larger domain of applications (generalist). In other words, the learning rate strategy selected needs to be robust enough that when it performs sub-optimally, the difference between its performance and the best performing approaches is limited. Incorporating this as a formal selection criterion improves the certainty with which an analyst can interpret the performance of an algorithm on a given problem without having to conduct additional exhaustive studies.

A natural consideration to resolve learning rates may be to consider line searches. Line searches are well-established

✉ Younghwan Chae
u11085160@tuks.co.za

Daniel N. Wilke
nico.wilke@up.ac.za

Dominic Kafka
u11207312@tuks.co.za

[1] University of Pretoria, Lynnwood Rd, Hatfield, Pretoria, 0002, South Africa

in mathematical programming to efficiently resolve learning rates and identify descent directions. However, they require the underlying loss function to be convex or unimodal over an identified interval. This would be the case if full-batch training of machine learning and deep learning neural networks would be attainable. However, when conducting full-batch training, computational and memory requirements are untenable for practical training. This makes full-batch training ill-suited for DNNs on modern memory-limited graphics processing unit (GPU) compute devices. As a result, the standard training procedure for machine learning and deep learning relies on mini-batch sub-sampling.

Mini-batch sub-sampling (MBSS) reduces the computational cost by using only a sub-sample of the training data at a time. This also provides a generalization effect [22] by turning a smooth continuous optimization problem into a stochastic optimization problem [25]. The stochastic or discontinuous nature of the loss function is due to the selected mini-batches' inherent sampling errors.

For line searches, the sampling errors manifest mainly in the form of bias or variance along a descent direction, depending on whether mini-batches are sub-sampled statically or dynamically [8, 15]. Static MBSS sub-samples a new mini-batch for every descent direction, while dynamic MBSS sub-samples a new mini-batch for every function evaluation. Hence, the loss function for static MBSS is continuous along a descent direction. The consequence is that the expected value of a static MBSS loss has a small variance but a large bias compared to the expected or full-batch response. Conversely, the loss function for dynamic MBSS is point-wise discontinuous [8, 14]. The expected response of the dynamic MBSS loss has a small bias but a large variance compared to the expected or full-batch response [8, 14]. Consider Fig. 1, which contrasts a full batched sampled loss function ($\mathscr{F}$ - orange) against a static ($\bar{f}$) and dynamic ($\tilde{f}$) sampled loss functions for a fully connected feedforward neural network [21] initialized with Xavier initialization [10]. Figure 1(a) depicts 20 potential static MBSS loss functions. Each loss has zero variance but a large bias in this case. Figure 1(c) depicts a dynamic MBSS loss function. It is clear that there is a large variance in the loss response, but the expected response has a lower bias since a mini-batch does not influence it in particular.

Line searches have been implemented for both static MBSS and dynamic MBSS loss functions. The main drawback of static MBSS is that it results in significant biases in loss approximations and has been improved by applying sample variance reduction techniques [5, 9]. Meanwhile, attempts to resolve learning rates in the point-wise discontinuous loss approximations of dynamic MBSS include the probabilistic line search [21] and Gradient-Only Line Search that is Inexact (GOLS-I) [14]. The probabilistic line search resolves learning rates by minimizing an

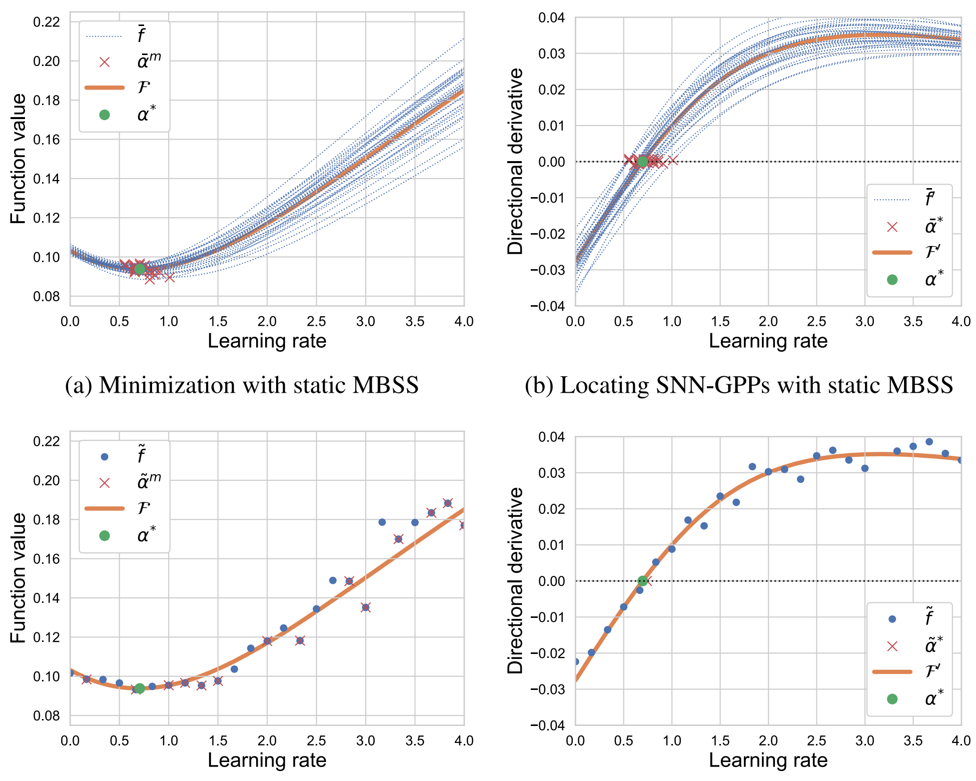


Fig. 1 Illustration of finding local minima in (a) static and (c) dynamic MBSS loss functions, as well as locating SNN-GPPs using (b) static and (d) dynamic MBSS directional derivatives

(a) Minimization with static MBSS

(b) Locating SNN-GPPs with static MBSS

(c) Minimization with dynamic MBSS

(d) Locating SNN-GPPs with dynamic MBSS

approximation constructed using both function value and directional derivative information, while GOLS-I uses only directional derivative sign change information.

GOLS-I locates optima by searching for stochastic non-negative gradient projection points (SNN-GPP). These manifest as positive directional derivatives with a non-zero probability around a ball encapsulating SNN-GPPs. Any point around the SNN-GPP ball is taken and the directional derivative from the SNN-GPP is computed. An SNN-GPP along a descent direction merely manifests as a sign change from negative to positive. Importantly, a sign change from negative to positive is necessary and sufficient to identify an SNN-GPP as a minimizer inferred solely from derivative information for this univariate case. This is empirically demonstrated in Fig. 1. Note that we multiply the unitless learning rate with descending direction vectors to update the weights by addition.

Learning rates resolved by locating minimizers or SNN-GPPs are equivalent for static MBSS loss functions as depicted by the minimizer solution, $\bar{\alpha}^m$, and SNN-GPP solution, $\bar{\alpha}^*$, in Fig. 1(a) and (b), respectively. However, minimizers are identified over the entire domain for dynamic sampled loss functions, as shown in Fig. 1(c). In turn, SNN-GPPs are concentrated around the full batch solution, as shown in Fig. 1(d). SNN-GPPs indicate a lower bias to the expected full-batch minimizer than minimizers of SNN-GPPs resolved from the static MBSS loss.

A recent empirical study investigated the effectiveness of dynamic mini-batch sampled function values and directional derivatives to construct quadratic approximation models to resolve learning rates [8]. Chae and Wilke [8] demonstrated that using only function value information resulted in learning rates with a larger variance due to the larger variance when predicting function values when conducting dynamic MBSS. Using only derivative information resulted in learning rates with smaller variance as the derivative information predicts more consistently when considering dynamic MBSS. Hence, gradient-only quadratic approximations result in stable and consistent learning rate predictions. Chae and Wilke [8] constructed derivative-only approximations using only two directional derivative evaluations referred to as gradient-only surrogates (GOS). One evaluated at the origin, and the other at an "initial guess" learning rate along the descent direction. The directional derivative at the origin is strictly less than zero for descent directions. If the directional derivative at the initial guess is also less than zero, the initial guess learning rate is immediately accepted. In turn, if the directional derivative at the initial guess is positive, linear interpolation between the two points is performed to approximate the location of a directional derivative sign change. The proposed approach served as an initial investigation and proof of a "vanilla" line-search concept for only stochastic gradient descent (SGD). However, the "vanilla" gradient-only surrogate line search (GOS-LS), proposed and investigated by [8], has no strong convergence characteristics, lacks a robust bracketing strategy, and has not been demonstrated for descent direction strategies other than SGD. Hence, the primary contributions of this study include:

1. Proposing GOS-LSC: It extends the shortcomings of the vanilla GOS-LS [8] by proposing a line search with strong convergence characteristics. This is achieved by introducing a robust bracketing strategy to improve linear interpolation accuracy, the conservative GOS-LS (GOS-LSC). The bracketing strategy is based on a modified strong Wolfe condition [26] to isolate SNN-GPP. We essentially propose a conservative algorithm with strong convergence characteristics, which may sacrifice the performance for convergence.
2. Experiments on various descent directions: Both GOS-LS and GOS-LSC are demonstrated as a suitable line search strategy for descent direction approaches other than SGD, including RMSProp and Adam on ResNet-18 [13] and EfficientNet-B0 [28] with CIFAR-10 [16].
3. Experiments for comparing different learning rate strategies: GOS-LS and GOS-LSC are compared to fixed learning rates, cosine annealing, and GOLS-I on a shallow neural network architecture, N-II [21], with MNIST [17]. GOS-LS and GOS-LSC exhibit competitive results compared to other line search methods.

To compare the performance of different strategies, we introduced a relative robustness measure (RRM) to quantify the differences between an algorithm and the best-performing algorithm for a given problem. In contrast to the traditional performance measure that only considers the best performance, the RRM considers all accounts, including poor performance. Hence, the criterion favors the strategy that performs well overall across different problems and optimizers rather than a problem-specific strategy.

While the experiment results for varying the descent directions showed that our proposed algorithm GOS-LSC-4, which is GOS-LSC with specific hyperparameter settings close to that of vanilla GOS-LS, outperforms GOS-LS in robustness, the performance comparison experiment results on a shallower architecture, N-II, showed that GOS-LSC-4 ranked third and second for overall training and test relative robustness among ten strategies, led by GOS-LS. Although GOS-LS is not robust in convergence, it can be more aggressive in training due to no curvature condition restricting its learning rates.

2 Background

In general, line searches can be employed to train deep neural networks to identify minimizers, first-order optimality candidate solutions (directional derivatives equal to 0), and SNN-GPPs. For convex functions, all three are equivalent. Several line searches which implement static MBSS have been presented, which take advantage of continuous loss functions that often assume convexity [2, 4, 5, 7, 9, 23, 29].

However, as illustrated in Fig. 1, for dynamic MBSS loss functions, SNN-GPPs identify sensible solutions compared to full batch solutions. Minimizers are hampered by local minima resulting in large variance, while first-order optimality candidate solutions may not exist for point-wise discontinuous loss functions. The present section summarizes several state-of-the-art sub-sampling and line search schemes applied to dynamic MBSS loss functions in machine learning literature. Firstly, we formalize dynamic MBSS and SNN-GPPs in Sections 2.1 and 2.2, respectively.

2.1 Dynamic mini-batch sub-sampling

Given weights $\boldsymbol{x}$, the function value computed with dynamic MBSS is expressed as

$$\tilde{L}(\boldsymbol{x}) = \frac{1}{|\mathscr{B}_{n,i}|} \sum_{b \in \mathscr{B}_{n,i}} \ell(\boldsymbol{x}; \boldsymbol{t}_b), \tag{1}$$

where $\ell(\boldsymbol{x}; \boldsymbol{t}_b)$ is computed using training samples in the sampled mini-batch, $\boldsymbol{t}_b$, with an approximate gradient given by

$$\tilde{\boldsymbol{g}}(\boldsymbol{x}) = \frac{1}{|\mathscr{B}_{n,i}|} \sum_{b \in \mathscr{B}_{n,i}} \nabla \boldsymbol{\ell}(\boldsymbol{x}; \boldsymbol{t}_b), \tag{2}$$

where i denotes the i-th function evaluation of the n-th iteration of a given algorithm. The loss function as a function of the learning rate, α, along a given descent direction, $\boldsymbol{d}_n$, starting from $\boldsymbol{x}_n$ is given by:

$$\tilde{f}_n(\alpha) = \tilde{L}(\boldsymbol{x}(\alpha)) = \tilde{L}(\boldsymbol{x}_n + \alpha \boldsymbol{d}_n), \tag{3}$$

with the directional derivative, $\tilde{f}_n'$, given by

$$\tilde{f}_n'(\alpha) = \boldsymbol{d}_n^{\intercal} \tilde{\boldsymbol{g}}(\boldsymbol{x}_n + \alpha \boldsymbol{d}_n). \tag{4}$$

Dynamic MBSS loss functions are point-wise discontinuous functions with point-wise discontinuous gradient fields.

2.2 Gradient-only optimality criterion

Multiple local minima would be found when locating minimizers for discontinuous functions such as a dynamic MBSS loss function. Instead, we may opt to locate Non-Negative Gradient Projection Points (NN-GPPs) for which its gradient-only optimality criterion was specifically designed for deterministic discontinuous function [26], given by

$$\boldsymbol{d}_n^{\intercal} \nabla \mathscr{L}(\boldsymbol{x}_{nngpp} + \alpha_n \boldsymbol{d}_n) \geq 0, \quad \forall \|\boldsymbol{d}_n \in \mathbb{R}^p\|_2 = 1, \quad \forall \alpha \in (0, \alpha_{\max}], \tag{5}$$

for the 1-D case, along a given search direction, $\boldsymbol{d}_n$. NN-GPP is representative of a local optimum because no descent directions are allowed away from it. This is only possible at a critical point or a local minimum in a smooth and continuous function.

The NN-GPP definition is limited to deterministic discontinuous functions. Therefore, to accommodate stochastic discontinuous functions, the NN-GPP definition was generalized and extended to the Stochastic NN-GPP (SNN-GPP), given by

$$\begin{aligned} \boldsymbol{d}_n^{\intercal} \tilde{\boldsymbol{g}}(\boldsymbol{x}_{snngpp} + \alpha_n \boldsymbol{d}_n) &\geq 0, \ \forall \|\boldsymbol{d}_n \in \mathbb{R}^p\|_2 \\ &= 1, \ \forall \alpha \in (0, \alpha_{\max}], \ \ p(\boldsymbol{x}_{snngpp}) > 0, \end{aligned} \tag{6}$$

with probability, $p(\boldsymbol{x}_{snngpp})$, greater than 0 [14].

The difference between NN-GPP and SNN-GPP is that NN-GPP is a point where the signs of directional derivatives change in the deterministic setting. However, in the stochastic setting, a directional derivative sign change location may vary, depending on the instance of the sampled stochastic loss. Transferred to dynamic MBSS losses, this means that for each distinct mini-batch, $\mathscr{B}$, selected, we have a distinct location of a sign change. However, these remain bounded in a ball, B_ϵ, [14] of a given neighborhood. The size of B_ϵ is, among other factors, dependent on the variance in the stochastic loss function, which in dynamic MBSS losses is dependent on the mini-batch size. Hence, the larger the difference between individual samples, $\mathscr{B}_{n,i}$, the larger the size of the ball, B_ϵ. Notably, the SNN-GPP definition also generalizes to the NN-GPP, critical point, and local minimum, as these are all SNN-GPPs with probability 1.

2.3 Line searches for dynamic MBSS loss functions

To the best of the author's knowledge, only three line search techniques have been proposed to resolve learning rates for dynamic MBSS loss functions, namely:

1. A probabilistic line search using Bayesian optimization with Gaussian surrogate models, built using both function value and directional derivative information [21].
2. The Gradient-only line search, Inexact (GOLS-I), locates SNN-GPPs along the search directions, using only directional derivative information [14].
3. Proof of concept quadratic approximations [8].

All three line search methods showed competitive training performance for dynamic MBSS losses and outperformed various constant learning rates.

Notably, [8] the quality of function values and directional derivatives in the context of approximation-based line searches is investigated empirically. The quality of information used to produce approximations in dynamic MBSS losses was studied by constructing five types of quadratic approximation models using different information sampled at two locations (e.g. only function value, only directional derivative, both function value and directional derivative models) [8]. The results showed that using directional derivative information at the origin (starting point) of a line search is critical for constructing quality approximations, decreasing the variances in optimal learning rates. The two best-performing models were

1. Derivative-only quadratic model: A quadratic approximation is built using two directional derivatives, values measured at the origin and another point along the descent direction.
2. Mixed quadratic model: A quadratic approximation is built using the directional derivative measured at the origin and function values at both origin and another point.

Note that both quadratic approximation models proposed by [8] demonstrate "vanilla" algorithms without guaranteed convergence. Although the mixed-model has been investigated by [21] and [23] before, the derivative-only model has not been extended to a fully automated line search technique, which is the aim of this paper. Next, we discuss the heuristics and the corresponding shortcomings of the vanilla derivative-only approximation using the derivative-only model proposed in [8], extending in this study.

2.4 Gradient-only surrogate line search (GOS-LS)

Given a multivariate dynamic MBSS loss function, $\tilde{L}(\boldsymbol{x}_n)$, along a given descent direction, $\boldsymbol{d}_n$, $\tilde{f}(\alpha)$, we want to resolve the learning rate, α. The quadratic approximation model, $\hat{f}(\alpha)$, of $\tilde{f}(\alpha)$ is given by

$$\hat{f}(\alpha) := k_1\alpha^2 + k_2\alpha + k_3 \approx \tilde{f}(\alpha), \tag{7}$$

where k_1, k_2, and k_3 are the constants to be computed. Similarly, the first-order derivative of the quadratic approximation, $\hat{f}'(\alpha)$, is a linear approximation given by

$$\hat{f}'(\alpha) := 2k_1\alpha + k_2 \approx \tilde{f}'(\alpha). \tag{8}$$

Note that $\hat{f}'(\alpha)$ is the derivative-only approximation proposed by [8], and is implemented throughout this paper. The approximation uses only directional derivative information. The constants at n-th iteration, $k_{1,n}$ and $k_{2,n}$ can be solved using a linear system of equations, constructed from two instances of (8), given by

$$\begin{bmatrix} 2\alpha_{0,n} & 1 \\ 2\alpha_{1,n} & 1 \end{bmatrix} \begin{bmatrix} k_{1,n} \\ k_{2,n} \end{bmatrix} = \begin{bmatrix} \tilde{f}'_{0,n} \\ \tilde{f}'_{1,n} \end{bmatrix}, \tag{9}$$

where $\tilde{f}'_{0,n}$ and $\tilde{f}'_{1,n}$ denote the dynamic MBSS directional derivatives measured at $\alpha_{0,n}$ and $\alpha_{1,n}$ respectively, where $\alpha_{0,n} = 0$ is the current starting point and $\alpha_{1,n}$ is the initial guess ($\alpha_{1,n} > 0$). The approximate minimum, $\tilde{\alpha}^*_n$, is computed as $\alpha^*_n = -k_{2,n}/(2k_{1,n})$, where $\hat{f}'(\tilde{\alpha}^*_n) = 0$. For the implementation, we compute $\tilde{\alpha}^*_n$ in the closed-form as follows:

$$\alpha^*_n = \alpha_{0,n} - \tilde{f}'_{0,n}\frac{\Delta\alpha_n}{\Delta\tilde{f}'_n} = \alpha_{0,n} - \tilde{f}'_{0,n}\frac{\alpha_{1,n} - \alpha_{0,n}}{\tilde{f}'_{1,n} - \tilde{f}'_{0,n}}. \tag{10}$$

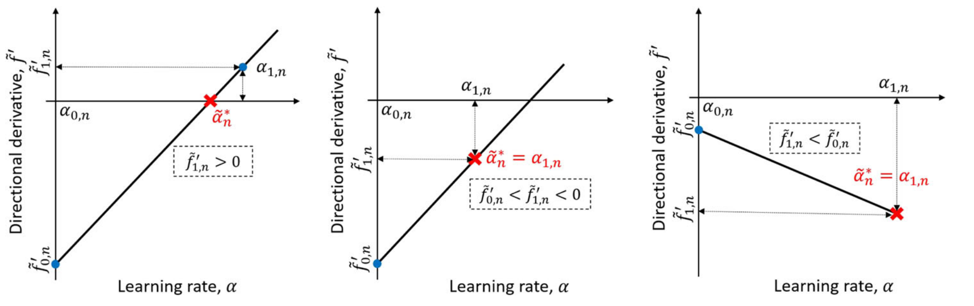
The heuristics of the vanilla line search algorithm using (10), proposed by [8], are recalled here:

1. If $\tilde{f}'_{1,n} > 0$, as shown in Fig. 2(a), we may perform bounded linear interpolations, (10), to resolve the learning rate, $\tilde{\alpha}^*_n$.
2. If $\tilde{f}'_{0,n} < \tilde{f}'_{1,n} < 0$, as shown in Fig. 2(b), we can perform bounded extrapolation using (10), but the prediction error is expected to be larger than the case of bounded interpolation. Hence, we immediately choose the initial guess as the resulting learning rate, i.e. $\tilde{\alpha}^*_n = \alpha_{1,n}$.
3. If $\tilde{f}'_{1,n} < \tilde{f}'_{0,n}$, as shown in Fig. 2(c), it would be unwise to perform unbounded extrapolation for the same reason as in the case of bounded extrapolation. Therefore, we immediately accept the initial guess α^*, as the learning rate of the current iteration $\tilde{\alpha}^*_n = \alpha_{1,n}$.

The examples studied by [8] used stochastic gradient descent (SGD) with the vanilla algorithm, $\boldsymbol{d}_n = -\tilde{\boldsymbol{g}}_n$. The initial guess, $\alpha_{1,n}$, for every iteration was chosen to be the inverse of the L-2 norm of the search direction vector, $\alpha_{1,n} = \|\boldsymbol{d}\|_2^{-1}$. This means that $\alpha_{1,n}$ is adapted to the magnitude of the search direction vector, $\boldsymbol{d}_n$, to prevent overly aggressive (potentially unstable) training behavior, when $\|\nabla\mathscr{L}_n\|_2 \approx 0$.

3 Relative robustness measure, *R*

Traditional performance measures only consider the top performances of strategies in DNNs. However, we observe rapid growth in the size of DNN problems, and the complexity of the problems also increases. This means that when we face an unseen problem, we might want to choose a more robust strategy across different problems and optimizers than a problem-specific top-performing strategy. Hence, we propose a new relative robustness measure (RRM) that considers both poor and excellent performances of learning rate strategies across different problems and



(a) Bounded Interpolation (b) Bounded extrapolation (c) Unbounded extrapolation

Fig. 2 Illustration of three possible cases when implementing the vanilla line search algorithm using the derivative-only approximation: (a) bounded interpolation, when $\tilde{f}'_{1,n} > 0$, (b) bounded extrapolation when $\tilde{f}'_{0,n} < \tilde{f}'_{1,n} < 0$ and (c) unbounded extrapolation when $\tilde{f}'_{1,n} < \tilde{f}'_{0,n}$

optimizers instead of only the top performances. We define the RRM for learning rate strategies as follows:

Definition 1 (Relative robustness) The relative robustness, R, of a strategy, y, is computed by summing the absolute differences, $\psi_{y,h,o}$, in the strategy's accuracy, $\eta_{y,h,o}$, and the best accuracy of all strategies, $\eta^*_{h,o}$, for all optimizers, $O \ni o$, and all problems, $H \ni h$. Hence, the less the measure, R_y, is, the more robust the strategy is. R_y is defined as

$$R_y = \sum_{h \in H} \sum_{o \in O} \psi_{y,h,o}, \text{ where } \psi_{y,h,o} = |\eta^*_{h,o} - \eta_{y,h,o}|. \quad (11)$$

The robustness measure, $R_{y,h}$, can be computed for a strategy, y; problem, h while applying a line search over all considered optimizers. We will compare our proposed algorithm's training and test performance, GOS-LSC, against the other learning rate strategies based on the RRM throughout the paper.

4 Conservative gradient-only approximation line search (GOS-LSC)

This section proposes our line search strategy, conservative gradient-only approximation line search (GOS-LSC), using the quadratic derivative-only approximation capable of locating the SNN-GPPs in the stochastic loss functions produced by dynamic MBSS. Unlike the vanilla algorithm, GOS-LSC requires consecutive function evaluations to converge to an interval of sign changes for a given descent direction, $\boldsymbol{d}_n$, update.

GOS-LSC comprises two main stages: 1) An immediate accept condition (IAC), and 2) a bracketing strategy. The IAC means that we accept the initial learning rate guess of the n-th iteration, $\alpha_{1,n}$, as the approximate solution, $\tilde{\alpha}^*_n$, when $\alpha_{1,n}$ falls within the accepted range set by the Wolfe condition [26]. If the IAC is not satisfied, the proposed bracketing strategy is used to locate SNN-GPPs.

4.1 Immediate accept condition (IAC)

The IAC aims to save computational costs. When the immediate accept condition (IAC) is satisfied, we immediately accept the initial guess, $\alpha_{1,n}$, and continue to the next search direction. The IAC is based on the Wolfe condition, consisting of 1) Armijo condition and 2) Wolfe curvature condition. The Armijo condition ensures that the function value at the accepted learning rate decreases monotonically as outlined,

$$\tilde{f}_{1,n} \le \tilde{f}_{0,n} + \omega\alpha_{1,n}\tilde{f}'_{0,n}. \quad (12)$$

Here, ω is a prescribed constant, often very small (e.g. $\omega = 10^{-4}$). Note that the Armijo condition limits the maximum learning rate by disallowing any increase in function value.

The Wolfe curvature condition ensures that the directional derivative at the initial guess, $\tilde{f}'_{1,n}$, is less than at the starting point, $\tilde{f}'_{0,n}$. The condition is given by:

$$-\tilde{f}'_{1,n} \le -c\tilde{f}'_{0,n}, \quad (13)$$

where $c \in (0, 1)$ is a prescribed curvature constant. The Wolfe curvature condition limits the minimum learning rate based solely on directional derivative information. In contrast, the Armijo condition, (12), requires function value information. Hence, it is not suitable for our derivative-only purpose. However, the Armijo condition is essential for limiting the maximum learning rates by not allowing growth in the function value. Therefore, we will implement the strong Wolfe condition as an alternative to (12) and (13):

$$|\tilde{f}'_{1,n}| \le c|\tilde{f}'_{0,n}|. \quad (14)$$

As a consequence of applying the strong Wolfe condition, we gain control over preventing overshooting, (12), and undershooting, (13), by changing the c constant.

Although we implement (13) as the IAC throughout the paper, note that one could also independently control undershoot and overshoot limits by splitting c into two positive constants, c_1 and c_2, respectively, as follows:

$$c_1\tilde{f}'_{0,n} \le \tilde{f}'_{1,n} \le -c_2\tilde{f}'_{0,n}, \quad c_1, c_2 \in [0, 1). \quad (15)$$

If the IAC in (14) is satisfied, just one directional derivative computation is required to determine the learning rate along a descent direction, $\boldsymbol{d}_n$. Figure 3 illustrates cases when the initial guess, $\tilde{f}'_{1,n}$, is either accepted (left) for satisfying the IAC condition or not accepted (right) for not satisfying the condition.

However, if the initial guess is not accepted, we employ the bracketing strategy introduced in the next section to search for SNN-GPPs. Note that the larger the c value becomes, the larger the range of the IAC becomes. Algorithm 1 lists the pseudocode for the main GOS-LSC algorithm with the IAC.

```
Input: $\tilde{\boldsymbol{g}}_{0,n} = \tilde{\boldsymbol{g}}^*_{n-1}$, $\tilde{\alpha}^*_{n-1}$, $\boldsymbol{d}_n$, $c$, $\alpha_{\min}$, $\alpha_{\max}$, $\gamma$ and $\varepsilon$
Output: $\tilde{\alpha}^*_n$, $\tilde{\boldsymbol{g}}^*_n$
1  Compute directional derivative at $\alpha_{0,n} \rightarrow \tilde{f}'_{0,n}$
   /* Magnitude check for avoiding
      numerical issues                                  */
2  if $|\tilde{f}'_{0,n}| < \varepsilon$ then
3      Recompute $\tilde{\boldsymbol{g}}^*_n$ for the next iteration.
4      return $\tilde{\alpha}^*_n := 0$, $\tilde{\boldsymbol{g}}^*_n$
   /* Setting the initial guess, $\alpha_{1,n}$          */
5  $\alpha_{1,n} \rightarrow \gamma$
6  Compute gradient and directional derivative at
   $\alpha_{1,n} \rightarrow \tilde{\boldsymbol{g}}_{1,n}$, $\tilde{f}'_{1,n}$
   /* Check whether the IAC satisfies
      */
7  if $|\tilde{f}'_{1,n}| \leq c|\tilde{f}'_{0,n}|$ then
8      $\tilde{\alpha}^*_n := \alpha_{1,n}$ and $\tilde{\boldsymbol{g}}^*_n := \tilde{\boldsymbol{g}}_{1,n}$
9  else
10     $\tilde{\alpha}^*_n, \tilde{\boldsymbol{g}}^*_n := Bracketing(\alpha_{0,n}, \alpha_{1,n}, \tilde{f}'_{0,n}, \tilde{f}'_{1,n})$
11 return $\tilde{\alpha}^*_n, \tilde{\boldsymbol{g}}^*_n$
```

Algorithm 1 GOS-LSC.

For every iteration, n, GOS-LSC requires the gradient vector at the starting point, $\tilde{\boldsymbol{g}}_{0,n}$. Therefore, for $n > 0$, the resulting gradient from the previous iteration, $\tilde{\boldsymbol{g}}^*_{n-1}$, can be used for $\tilde{\boldsymbol{g}}_{0,n}$ in the next iteration. In line 2, the tolerance value, ε, ensures that $\tilde{f}'_{0,n}$ is numerically positive. Otherwise, we recompute the gradient at the same point with the resulted learning rate, $\tilde{\alpha}^*_n = 0$, using a new mini-batch, $\mathscr{B}$, and continue to the next iteration.

In line 5, the initial guess, $\alpha_{1,n}$, is the default learning rate, γ, often the recommended learning rate for the chosen optimizer. Next, we compute the gradient vector, $\tilde{\boldsymbol{g}}_{1,n}$, and directional derivative, $\tilde{f}'_{1,n}$, at the initial guess, $\alpha_{1,n}$. In line 7, if the IAC satisfies, we choose the current learning rate, $\tilde{\alpha}^*_n = \alpha_{1,n}$, and the resulting gradient, $\tilde{\boldsymbol{g}}^*_n = \tilde{\boldsymbol{g}}_{1,n}$. If the IAC is not satisfied, we implement the bracketing strategy to compute $\tilde{\alpha}^*_n$ and $\tilde{\boldsymbol{g}}^*_n$. The bracketing strategy aims to minimize the model error using linear interpolation, introduced in the following section.

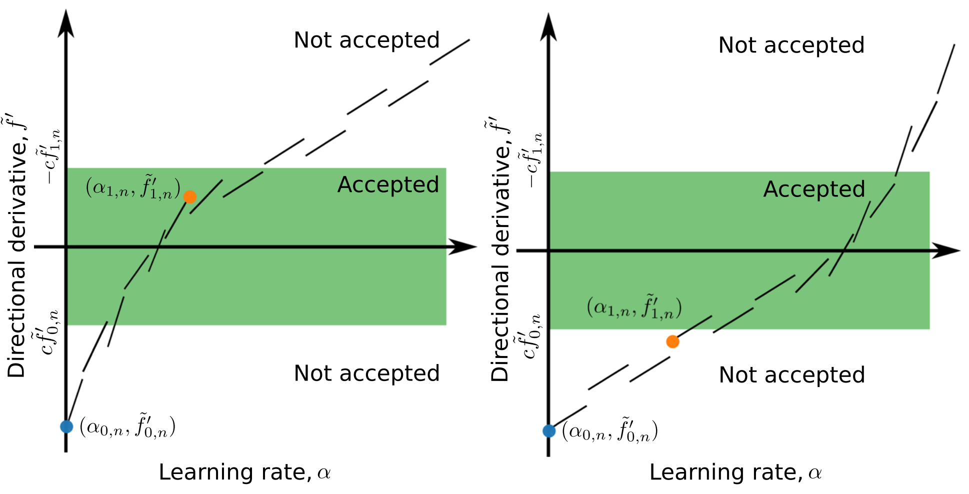
4.2 Bracketing strategy

The bracketing strategy aims to isolate an SNN-GPP inside an interval, $\mathscr{I} \in [\alpha_L, \alpha_U]$ with lower bound, α_L and upper bound, α_U, by updating $\mathscr{I}$ repeatedly, until the strong Wolfe condition, (15) of $\tilde{\alpha}^*_n$ is satisfied. Once the directional derivative signs at α_L and α_U are found to bracket an SNN-GPP, we reduce the interval by applying the Regula-Falsi method [12]. This is essentially a consecutive linear interpolation method until $\tilde{\alpha}^*_n$ satisfies (15). We provide the pseudocode for the bracketing strategy in Algorithm 2.

```
Input: $\alpha_{0,n}$, $\alpha_{1,n}$, $\tilde{f}'_{0,n}$, $\tilde{f}'_{1,n}$, $\boldsymbol{d}_n$, $\alpha_{\min}$, $\alpha_{\max}$, $c$ and $\varepsilon$
Output: $\alpha^*_n$, $\tilde{\boldsymbol{g}}^*_n$
1  Initialize $\alpha_L := \alpha_{0,n}$, $\tilde{f}'_L := \tilde{f}'_{0,n}$, $\alpha_U := \alpha_{1,n}$ and
   $\tilde{f}'_U := \tilde{f}'_{1,n}$
   /* Shifting the interval to larger
      learning rates                                    */
2  while $(\tilde{f}'_U < c\tilde{f}'_{0,n})$ and $(2\alpha_U < \alpha_{\max})$ do
3      Update $\alpha_L := \alpha_U$ and $\tilde{f}'_L := \tilde{f}'_U$
4      $\alpha_U := 2(\alpha_U)$ and recompute $\tilde{\boldsymbol{g}}_U$, $\tilde{f}'_U$
5  $\tilde{f}'_{temp} := \tilde{f}'_U$
6  $\alpha_{temp} := \alpha_U$
   /* Shrinking the interval using
      linear interpolations                             */
7  while $(\tilde{f}'_{temp} > -c\tilde{f}_{0,n})$ and $(\tilde{f}'_U\tilde{f}'_L <$
   $0)$ and $(|\tilde{f}'_U - \tilde{f}'_L| > \varepsilon)$ do
8      $\alpha_{temp} = \frac{\alpha_L\tilde{f}'_U - \alpha_U\tilde{f}'_L}{\tilde{f}'_U - \tilde{f}'_L}$
9      Recompute $\tilde{f}'_{temp}$ at $\alpha_{temp}$
10     if $\tilde{f}'_{temp}\tilde{f}'_L < 0$ then
11         $\tilde{f}'_U := \tilde{f}'_{temp}$
12         $\alpha_U = \alpha_{temp}$
13     else
14         $\tilde{f}'_L := \tilde{f}'_{temp}$
15         $\alpha_L = \alpha_{temp}$
16 $\alpha^*_n = \alpha_{temp}$
17 Compute gradient $\tilde{\boldsymbol{g}}^*_n$ at $\alpha^*_n$
18 return $\alpha^*_n$, $\tilde{\boldsymbol{g}}^*_n$
```

Algorithm 2 Bracketing.

In line 1, we begin with initialization of the lower, α_L, and upper, α_U, bounds, and their respective directional derivatives, $\tilde{f}'_L$ and $\tilde{f}'_U$. In line 2, the interpolation interval



(a) Accepted (b) Not accepted

Fig. 3 Illustration of immediate accept condition: (a) when the IAC (15) satisfies, the initial guess, $\alpha_{1,n}$, is accepted, and (b) when the IAC (15) does not satisfy, the initial guess, $\alpha_{1,n}$, is not accepted

grows by doubling α_U which is directly followed by α_L, until it reaches α_{max} or $\tilde{f}'_U \geq c\tilde{f}'_{0,n}$ is satisfied. In line 7, the size of the interpolation interval is reduced by consecutively performing linear interpolation until $\tilde{f}'_U \leq -c\tilde{f}_{0,n}$ is satisfied. The conditions in line 7 ensure that linear interpolation steps with the Regula-Falsi method in lines 8, 9, 10, 11, 12, 13, 14 and 15 do not cause any numerical instabilities. The second term, $\tilde{f}'_U\tilde{f}'_L < 0$, ensures that the signs of the two-directional derivatives are opposite. The last term, $|\tilde{f}'_U\tilde{f}'_L| < \varepsilon$, provides the denominator of line 8 to be non-zero.

The Wolfe curvature conditions are divided into two sections, as shown in lines 2 and 7, to prevent this algorithm from searching infinitely for points that do not satisfy the Wolfe curvature conditions. Due to the stochastic nature of dynamic MBSS loss functions, it is not guaranteed that continually reducing learning rates would find a negative directional derivative with less magnitude than the directional derivative at the origin, $\tilde{f}'_{n,0}$. This implies that we can not assure that the first condition in line 2 is still met after the second condition in line 7 is satisfied. The risk associated with undershooting is lower than that of overshooting because overshooting may cause divergence in training, while undershooting, in the worst case, causes slower training. The flowchart of GOS-LSC is shown in Fig. 4.

4.3 Proof of convergence

Let us assume that the full-batch loss function, $\mathscr{L}(\boldsymbol{x})$, is a smooth coercive function of a weight vector, $\boldsymbol{x} \in \mathbb{R}^p$, so that we can replace $\mathscr{L}(\boldsymbol{x})$ with a Lyapunov function, $\Gamma(\boldsymbol{x})$ [20]. The Lyapunov's global stability theorem states that a Lyapunov function, $\Gamma(\boldsymbol{x})$, results in $\lim_{n\to\infty} \boldsymbol{x}_n = 0$, $\forall\ \boldsymbol{x}_n \in \mathbb{R}^p$ under the following conditions:

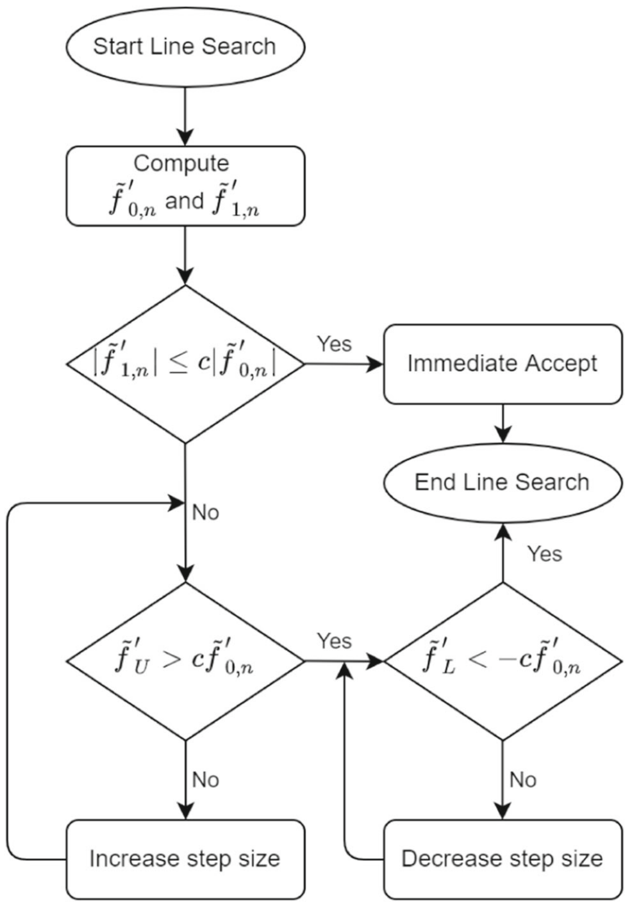

Start Line Search
Compute $\tilde{f}'_{0,n}$ and $\tilde{f}'_{1,n}$
$|\tilde{f}'_{1,n}| \leq c|\tilde{f}'_{0,n}|$
Yes
Immediate Accept
End Line Search
No
$\tilde{f}'_U > c\tilde{f}'_{0,n}$
Yes
$\tilde{f}'_L < -c\tilde{f}'_{0,n}$
Yes
No
No
Increase step size
Decrease step size

Fig. 4 The flowchart of the GOS-LSC line search strategy

1. Positivity: $\Gamma(\mathbf{0}) = 0$ and $\Gamma(\boldsymbol{x}) > 0, \forall\ \boldsymbol{x} \neq \mathbf{0}$
2. Coercive: $\lim_{\boldsymbol{x}\to\infty} \Gamma(\boldsymbol{x}) = \infty$
3. Strict descent: $\Gamma(\mathscr{D}(\boldsymbol{x})) < \Gamma(\boldsymbol{x}), \forall\ \boldsymbol{x} \neq \mathbf{0}$

where $\mathscr{D}(\boldsymbol{x})$ is a weight update function given by

$$\boldsymbol{x}_{n+1} := \mathscr{D}(\boldsymbol{x}_n); \quad \mathscr{D} : \mathbb{R}^p \to \mathbb{R}^p. \tag{16}$$

It is proven by [26] that locating an NN-GPP along a strictly descending direction, $\boldsymbol{d}_n$, is equivalent to minimizing along $\boldsymbol{d}_n$ when $\mathscr{L}(\boldsymbol{x}_n + \alpha\boldsymbol{d}_n)$ is a smooth function. Therefore, locating NN-GPPs along descent directions in consecutive iterations of a training algorithm behaves like $\mathscr{D}(\boldsymbol{x})$. Similarly, for loss functions resulting from dynamic MBSS, $\tilde{\mathscr{L}}$, we assume as point-wise discontinuous coercive.

Hence, the Lyapunov's global stability theorem is relaxed for the expected Lyapunov function, $E[\Gamma(\boldsymbol{x})]$, where:

1. Expected positivity: $E[\Gamma(\mathbf{0})] = 0$ and $E[\Gamma(\boldsymbol{x})] > 0$, $\forall\ \boldsymbol{x} \neq \mathbf{0}$
2. Expected coercive: $\lim_{\boldsymbol{x}\to\infty} E[(\Gamma(\boldsymbol{x})] = \infty$
3. Expected strict descent: $E[\Gamma(\mathscr{D}(\boldsymbol{x}))] < E[\Gamma(\boldsymbol{x})]$, $\forall\ \boldsymbol{x} \neq \mathbf{0}$

Subsequently, consecutively searching for SNN-GPPs along descent directions makes the training algorithm behave like $\mathscr{D}(\boldsymbol{x})$, which tends towards a ball, B_ε:

$$\lim_{n\to\infty} \boldsymbol{x}_n = \{\boldsymbol{q} \| \|\boldsymbol{q} - \boldsymbol{x}^*\| < \varepsilon\} \in B_\varepsilon \tag{17}$$

where B_ε is a ball function with the radius of ε, with the true optimum, $\boldsymbol{x}^*$, located at its center. Since our bracketing strategy searches for SNN-GPPs with weights, $\boldsymbol{x}_n \in \mathscr{B}_\varepsilon$, respectively, along a strictly descending direction, $\boldsymbol{d}_n$,

$$|E[\Gamma(\boldsymbol{x}_{n+1})] - E[\Gamma(\boldsymbol{x}^*)]| < |E[\Gamma(\boldsymbol{x}_n)] - E[\Gamma(\boldsymbol{x}^*)]|, \tag{18}$$

and weights outside the ball, $\boldsymbol{x}_n \in \mathscr{B}'_\varepsilon$, would eventually be inside the ball, $\boldsymbol{x}_n \in \mathscr{B}_\varepsilon$, as $n \to \infty$.

5 Numerical study design

We conducted two sets of numerical studies to investigate the performance of the proposed learning rate algorithm, GOS-LSC. First, we prepared GOS-LSC with three different hyperparameter settings, namely, GOS-LSC-1, 2, 3, and 4. Their details are explained in Section 5.1. We compared them against the fixed learning rates, which are recommended learning rates for different optimizers, and the vanilla GOS-LS on ResNet-18 and EfficientNet-B0 with the CIFAR-10 dataset for optimizers, including SGD, RMSProp, and Adam. We chose the batch size of 128 as implemented by [18]. From this experiment, we aim to investigate the following:

1. Relative robustness of GOS-LSC on the various optimizers that generate different descent directions;
2. Relative robustness of GOS-LSC compared to the fixed learning rates and vanilla GOS-LS;
3. Effect of various hyperparameter settings for GOS-LSC;
4. Generalizability of the different learning rate strategies.

Note that as previously motivated in Section 3, we measured the performance of a strategy using the RRM in (11) since we were interested in a learning rate strategy that operates well over different problems and optimizers.

Generalization is an ability of a problem to perform well on unobserved inputs [11]. Hence, to check generalizability, we measure the ratios of training to test accuracies. When the ratios approach one, the relative discrepancy between the training accuracy and test accuracy is small, indicating the problem generalizes well.

Second, we conducted a hyperparameter study for GOS-LSC using the only SGD on a shallower DNN, N-II, implemented by [21]. We tested all four hyperparameter settings for GOS-LSC and the most robust setting on our RRM was tested against other learning rate strategies. These included fixed learning rates, vanilla GOS-LS [8], GOLS-I [15], and cosine annealing with warm restarts [19]. We tested multiple batch sizes of 10, 100, 200, and 1000. The small batch size of 10 allows us to investigate the behaviors of strategies when the information is critically sparse. We only used SGD as the optimizer for this experiment because SGD is sensitive to different learning rates as we compare strategies. We aimed to investigate the following in the second experiment:

1. Relative robustness of GOS-LSC compared to the fixed learning rates and vanilla GOS-LS on different batch sizes;
2. Effect of various hyperparameter settings for GOS-LSC;
3. Generalizability of the different learning rate strategies.

5.1 Hyperparameter settings of GOS-LSC

GOS-LSC requires three hyperparameters to be selected:

1. The initial learning rate, $\alpha_{0,1}$;
2. The curvature hyperparameter, c;
3. Selecting to use the previous learning rate for the following initial learning rate, $\alpha_{0,n} = \alpha^*_{0,n-1}$, or resetting it to the default initial learning rate, $\alpha_{0,n} = \alpha_{0,1}$.

Table 1 lists the four different settings of hyperparameters for GOS-LSC. While all four settings keep the curvature hyperparameter, c, identical and large, the initial learning rates, $\alpha_{0,1}$, are the recommended learning rates, γ, for the selected optimizer or the inverse of the L-2 norm of the

Table 1 Comparison between the settings of vanilla GOS-LS and GOS-LSC that are tested in this paper

Algorithms	Settings	$\alpha_{0,1}$	$\alpha_{0,n} = \alpha^*_{0,n-1}$	c
GOS-LSC	GOS-LSC-1	γ	No	0.9
	GOS-LSC-2	γ	Yes	0.9
	GOS-LSC-3	$1/\lVert \boldsymbol{d}_n \rVert$	Yes	0.9
	GOS-LSC-4	$1/\lVert \boldsymbol{d}_n \rVert$	No	0.9
GOS-LS	–	$1/\lVert \boldsymbol{d}_n \rVert$	No	–

We choose the initial learning rates for the first iteration, $\alpha_{0,1}$, the curvature hyperparameter, c, and decide whether we want to use the final learning rate as the next initial learning rate, $\alpha_{0,n} = \alpha^*_{0,n-1}$

search direction, $1/\lVert \boldsymbol{d}_n \rVert$. We use GOS-LSC-1,2,3 for the first numerical study and GOS-LSC-1,2,3,4 for the second numerical study. We omit GOS-LSC-4 for the first numerical study because the loss function from deep network architectures is highly non-linear. Hence, constructing multiple quadratic approximations causes significant approximation errors and many gradient evaluations when taking an exponentially growing initial guess such as $1/\lVert \boldsymbol{d}_n \rVert$. Note that the hyperparameter setting closest to the vanilla GOS-LS is GOS-LSC-4 allowing for direct comparison of performance between them.

5.2 Numerical study 1 setup

For the first numerical study, we investigate the performance of GOS-LSC for different architectures and optimizers. It is compared against GOS-LS and fixed learning rates for various optimizers' search directions: SGD, RMSProp, and Adam. We chose ResNet-18 [13] and EfficientNet-B0 [28], which are implemented by [18] in PyTorch [24], for the test DNN models and the CIFAR-10 dataset [16] for this experiment. The details of the dataset are shown in Table 2. The chosen mini-batch size, $|\mathscr{B}|$, for this numerical study is 128.

The following learning rate strategies were trained for 350 epochs, repeated five times: fixed learning rate, GOS-LS, GOS-LSC-1, GOS-LSC-2, GOS-LSC-3, and GOS-LSC-4. The fixed learning rates, γ, for SGD, RMSProp, and Adam are 0.01, 0.01, and 0.001, respectively, which are the default values provided by PyTorch [24] and TensorFlow [1].

Note that because we adopt dynamic MBSS, a different mini-batch for every function evaluation, for the experiments, the fixed number of epochs also means the fixed number of gradient computations in training. Hence, some strategies may have fewer search directions when more gradient evaluations are required for each search direction or iteration.

5.3 Numerical study 2 setup

For the second numerical study, we first examine the performance of GOS-LSC with different hyperparameter settings: GOS-LSC-1, 2, 3, and 4 to choose the most robust hyperparameter setting, based on the RRM. Next, we test GOS-LSC with the best hyperparameter setting against various learning rate strategies such as GOS-LS, GOLS-I, fixed learning rate methods, cosine annealing with warm restarts [19] with different mini-batch sizes. We restrict ourselves to SGD directions on shallower neural network architecture.

We used similar neural network training problems to those proposed by [21], namely training on a fully connected feedforward neural network problem, N-II. This network's architecture involves fully connected layers with three hidden layers, $n_{input} - 1000 - 500 - 250 - n_{output}$. Hence, this architecture has shallower networks compared to the test problems in Numerical study 1. It contains the tanh activation functions, mean square loss, and Xavier initialization [10]. The dataset used for the problem is MNIST in Table 2.

The descriptions of the learning rate strategies compared against GOS-LSC are listed as follows:

1. Fixed learning rates: we test five sets of fixed learning rates, $\alpha = 10^{-3}$, 10^{-2}, 10^{-1}, 10^{0} and, 10^{1};
2. The cosine annealing scheduler with warm restarts [19]: starting learning rates used are $\alpha = 10^{-1}$ and 10^{0}, the initial restart period, the multiplying factor is chosen to be $T_0 = 1$ epoch, and $T_{mult} = 2$;

Table 2 Descriptions of datasets used in the numerical study

Datasets	Classes	Input sizes	Training samples	Test samples
MNIST [17]	10	28×28	6×10^4	1×10^4
CIFAR-10 [16]	10	32×32	5×10^4	1×10^4

3. The gradient-only line search that is Inexact (GOLS-I) [14];
4. The vanilla gradient-only surrogate line search (GOS-LS) [8];
5. The conservative gradient-only surrogate line search (GOS-LSC) allows various hyperparameter settings. For the comparison, we choose GOS-LSC-4 in Table 1.

This makes ten strategies in total. The training is limited in the number of directional derivative computations per training run, and the limit is 4×10^4. The mini-batch sizes chosen were $|\mathscr{B}| \in \{10, 100, 200, 1000\}$. We include the batch size of 10 to investigate how the strategies behave when insufficient information is provided for them. As mentioned earlier, we select the SGD direction as the search directions, computed using the same mini-batch size, $|\mathscr{B}|$. For each setting, we take ten runs for generating results.

6 Results of numerical study

6.1 Results of numerical study 1

Figures 5 and 6 show the results for SGD, RMSProp and Adam, respectively. The 5-step moving average values of the training errors, test errors, learning rates, and the number of gradient computations is plotted along 350 epochs using error bars on a $\log_{10}$ scale for each optimizer. The lower errors and upper errors represent the minimum and the maximum errors of the five runs. Note that dynamic MBSS requires a new mini-batch for every function evaluation. Hence, the larger the number of function evaluations computed per iteration, the fewer search direction updates per epoch.

A common phenomenon observed in most of the results in Figs. 5 and 6 is that the average learning rates of both GOS-LS and GOS-LSC increase as the epoch increases. This happens because the directional derivative at the origin, $\tilde{f}'_{0,n}$, decreases throughout training. This means that the average number of gradient computations for GOS-LSC may increase over epochs to satisfy the curvature condition on the flatter domains of the functions. On the other hand, the average number of gradient computations for GOS-LS decreases since the chance of observing a directional derivative at $\alpha_{1,n}$, $\tilde{f}'_{1,n}$ is less than the initial directional derivative, $\tilde{f}'_{0,n}$, grows. In this case, it is the immediate accept condition (IAC). Hence, we accept $\alpha_{1,n}$ as the final learning rate, α^*_n.

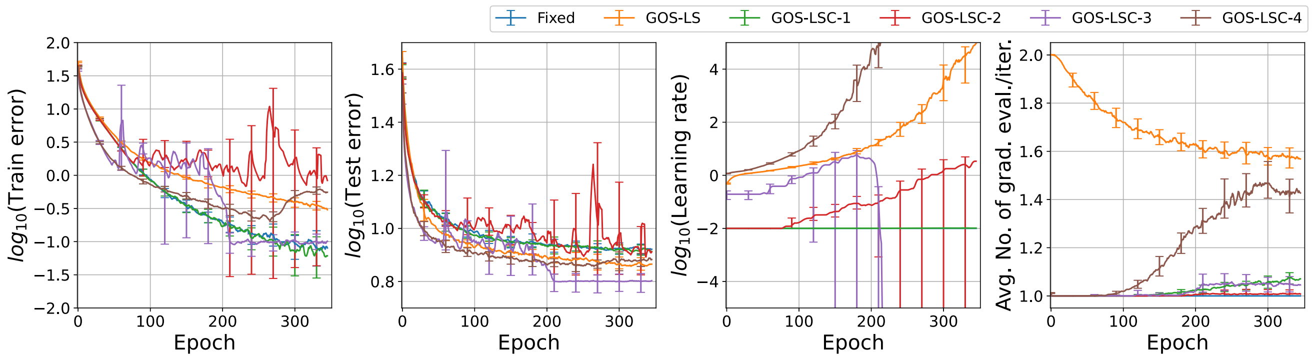
The ResNet results in Fig. 5 indicate that the initial convergence rate in GOS-LS's training is slightly lower than for the fixed learning rates and GOS-LSC. GOS-LS has a small initial learning rate since the inverse of the norm of search direction is small and does not extend the learning rate to be inside the ball like GOS-LSC. Fig. 5(a) shows that although the training error of GOS-LS is relatively high, its test error is one of the lowest. The performance of GOS-LSC-1 is similar to that of the fixed learning rate since the initial learning rate, $\alpha_{0,n} = \gamma$, happens to satisfy the Wolfe condition most of the time. Hence, note the slight increase in the gradient evaluations only toward the end of training. GOS-LSC-2 shows a large variance in the performance since using previous learning rates may cause a large model error as the previous learning rate might be far from the ball. Hence, it is more challenging to find the SNN-GPPs for possibly multimodal distributions of sign changes.

Note that the lower limit of GOS-LSC's learning rates is softly bounded since we satisfy the lower curvature condition before satisfying the upper independently. Consequently, when the model error is largely due to using the previous learning rates as initial guesses, it may allow learning rates to be numerically zero. For the same reason, we also observe the phenomenon of diminishing learning rate in GOS-LSC-3. However, it starts with larger learning rates and a steeper convergence rate in training, and it also shows the lowest test error for SGD. Although GOS-LSC-4 has a similar hyperparameter setting as GOS-LS, it shows quicker convergence in both training and test. The growth rate of its learning rate is also faster than GOS-LS, and the average number of gradient evaluations increases faster than in the other hyperparameter settings.

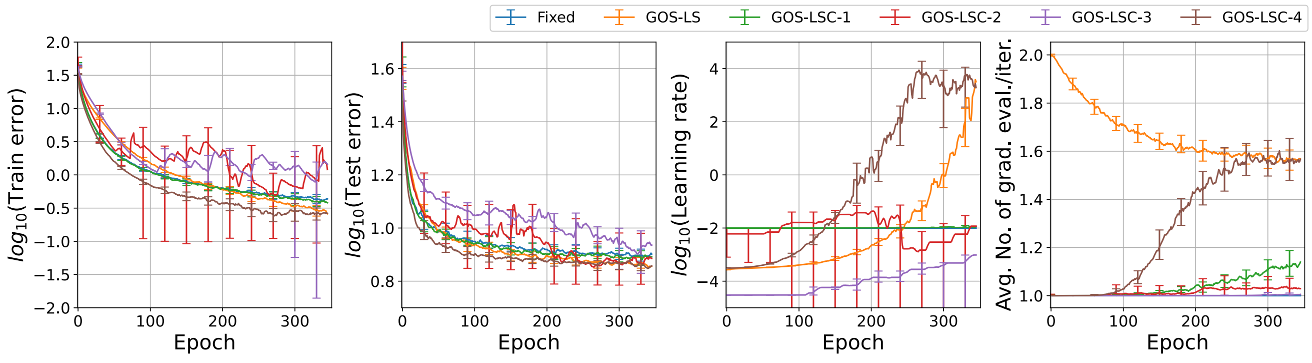
Figure 5(b) shows that the ResNet-18 results for RMSProp show that GOS-LSC-4 has the lowest training and test errors, while both GOS-LSC-2 and GOS-LSC-4 show large fluctuations in learning rates, GOS-LSC-1 and the fixed learning rate perform similarly since the recommended learning rate approximations SNN-GPPs well.

The ResNet-18 results for Adam in Fig. 5(c) show that while both GOS-LSC-2 and GOS-LSC-3 again perform poorly, both GOS-LS and the fixed learning rate perform similarly. Note that GOS-LSC-1 and GOS=LSC-4 show the best training and test performance. Also, note that the average number of gradient computations for GOS-LSC-4 increases as the epoch increases, showing the largest values among the optimizers. This means the recommended learning rate, γ, for Adam requires more adjustments to satisfy the curvature condition close to the end of training.

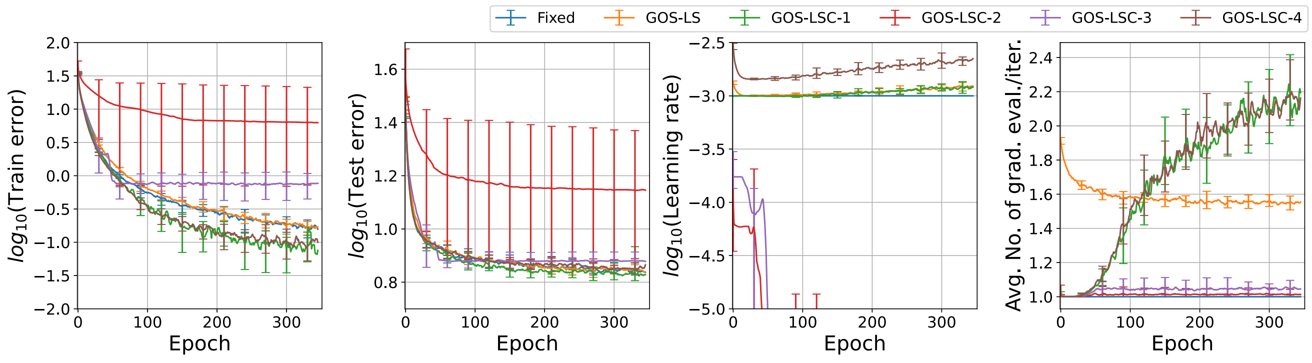
The EfficientNet-B0 results shown in Figs. 6(a) and (b) indicate that SGD and RMSProp are mostly unaffected by GOS-LSC-1 and GOS-LSC-2. This is because their default learning rates mostly satisfy the curvature condition. Hence, the IAC continues through the whole training. However, Adam with GOS-LSC affected the learning rates close to the end of training with slightly lower training and test errors. GOS-LSC-3 for SGD, shown in Fig. 6(a), has a high initial convergence rate. However, the large variance



(a) SGD



(b) RMProp

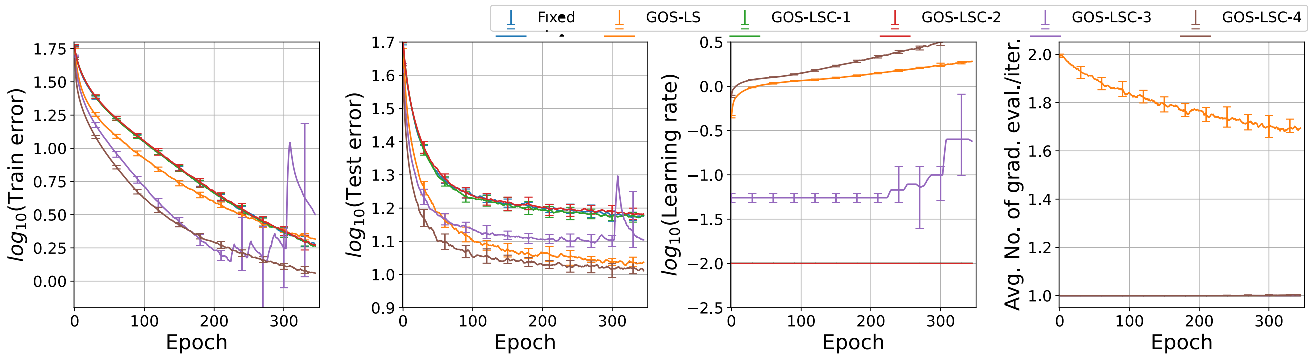


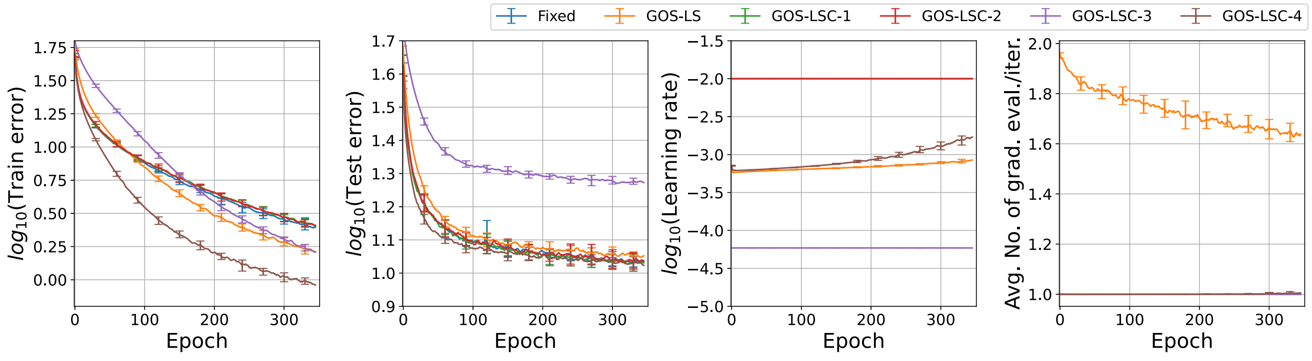
(c) Adam

Fig. 5 Comparisons of the performances of (a) SGD, (b) RMSProp, and (c) Adam between with and without GOS-LSC applied, tested on ResNet-18 for the CIFAR-10 dataset, the results are averaged over five runs and smoothened out with moving average over five epochs. Left to right presents the training errors, test errors, learning rates on the $\log_{10}$ scale, and the average number of gradient evaluations per iteration

in learning rates resulted in fluctuations in the training error. GOS-LSC-4 shows the lowest training and test errors for both SGD and RMSProp followed by GOS-LS. Yet, it requires fewer gradient computations. Figure 6(c) shows the average gradient computation numbers for both GOS-LSC-1 and GOS-LSC-2 increase for Adam, unlike SGD and RMSProp. This results in more adjustments in learning rates and improved train accuracies compared to the fixed learning rate.

Tables 3 and 4 present the top average training and test accuracies, the performance differences, $\Psi_{y,h,o}$ for them, and the ratios of the training to test accuracies for Numerical study 1. The ratio measures the generalization of each learning rate strategy. The closer the ratio becomes to one, the smaller discrepancy between the training accuracy and test accuracy. This implies that the strategy generalizes well for the optimizer and problem. The tables also provide the RRM, $R_{y,h}$, over different optimizers, o, by summing the


(a) SGD, $\alpha = 0.01$


(b) RMSProp

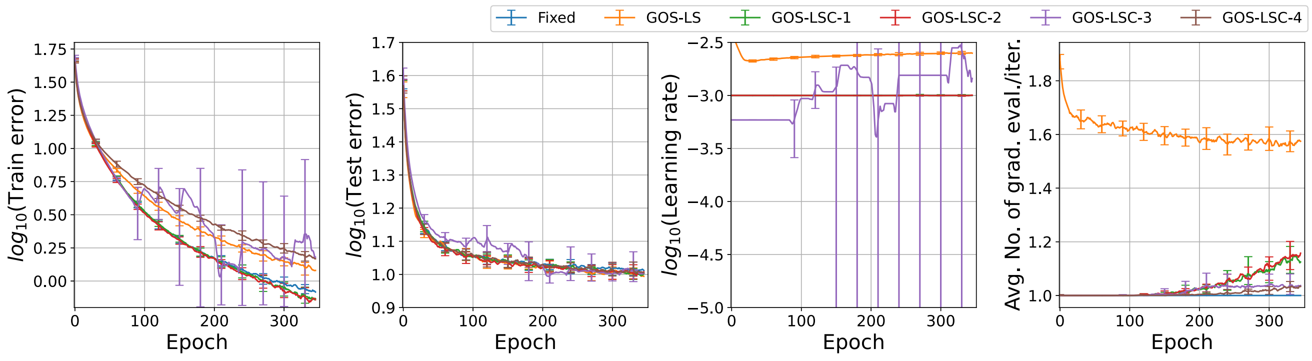

(c) Adam

Fig. 6 Comparisons of the performances of (a) SGD, (b) RMSProp, and (c) Adam between with and without GOS-LSC applied, tested on EfficientNet-B0 for the CIFAR-10 dataset, the results are averaged over five runs and smoothened out with moving average over five epochs. Left to right presents the training errors, test errors, learning rates on the $\log_{10}$ scale, and the average number of gradient evaluations per iteration

differences, $\Psi_{y,h,o}$, and the average ratios of test accuracies to training accuracies over the optimizers.

Table 3 shows the results of ResNet-18. The $R_{y,h}$ values indicate that GOS-LSC-1 and GOS-LSC-4 are the most robust strategy for training and test, respectively, over the optimizers for ResNet-18 among the six strategies. Concerning the ratios, GOS-LS and GOS-LSC-3 have the highest generalizability as their ratios are the closest ones, followed by GOS-LSC-4. On the other hand, the lowest ratio, shown by GOS-LSC-2, indicates that GOS-LSC-2 experiences more overfitting than others on ResNet-18. On average, the ResNet-18 results show that GOS-LSC-1 and 4 outperform GOS-LS in the training results, while only GOS-LSC-4 outperforms GOS-LS in test results.

Table 4 shows the results of EfficientNet-B0. The $R_{y,h}$ values indicate that GOS-LSC-4 is the most robust strategies for both training and test, over the optimizers for EfficientNet-B0 among the six strategies. GOS-LSC-1

Table 3 Top average training and test accuracies over the five runs tabulated for optimizers, including SGD, RMSPROP, and ADAM, with the fixed recommended learning rates, vanilla GOS-LS, and GOS-LSC with various settings on ResNet-18

Models	Optimizers	Strategies	Train acc. [%]	Diff., $\psi_{y,h,o}$	Test acc. [%]	Diff., $\psi_{y,h,o}$	Te./Tr.
ResNet-18	SGD	Fixed	99.94	0.02	91.88	1.9	0.919
		GOS-LS	99.72 (-0.22)	0.24	92.93 (+1.05)	0.85	0.932
		GOS-LSC-1	**99.96 (+0.02)**	0	91.96 (+0.08)	1.82	0.92
		GOS-LSC-2	99.43 (-0.51)	0.53	92.33 (+0.45)	1.45	0.929
		GOS-LSC-3	99.93 (-0.01)	0.03	**93.78 (+1.90)**	0	0.938
		GOS-LSC-4	99.8 (-0.14)	0.16	93.02 (+1.14)	0.76	0.932
	RMSPROP	Fixed	99.64	0.15	92.37	0.65	0.927
		GOS-LS	99.78 (+0.14)	0.01	**93.02 (+0.65)**	0	0.932
		GOS-LSC-1	99.65 (+0.01)	0.14	92.59 (+0.12)	0.43	0.929
		GOS-LSC-2	99.56 (-0.08)	0.23	**93.02 (+0.65)**	0	0.934
		GOS-LSC-3	99.25 (-0.39)	0.54	92.24 (-0.13)	0.78	0.929
		GOS-LSC-4	**99.79 (+0.15)**	0	93.01 (+0.64)	0.01	0.932
	ADAM	Fixed	99.87	0.09	93.3	0.28	0.934
		GOS-LS	99.86 (-0.01)	0.1	93.23 (-0.07)	0.35	0.934
		GOS-LSC-1	**99.96 (+0.09)**	0	**93.58 (+0.28)**	0	0.936
		GOS-LSC-2	93.79 (-6.08)	6.17	86.08 (-7.22)	7.5	0.918
		GOS-LSC-3	99.31 (-0.56)	0.65	92.53 (-0.77)	1.05	0.932
		GOS-LSC-4	99.94 (+0.07)	0.02	93.16 (-0.14)	0.42	0.932
	$R_{y,h}$	Fixed	–	0.26	–	2.83	–
		GOS-LS	–	0.35	–	1.2	–
		GOS-LSC-1	–	**0.14**	–	2.25	–
		GOS-LSC-2	–	6.93	–	8.95	–
		GOS-LSC-3	–	1.22	–	1.83	–
		GOS-LSC-4	–	0.18	–	**1.19**	–
	Avg. Te./Tr.	Fixed	–	–	–	–	0.927
		GOS-LS	–	–	–	–	0.933
		GOS-LSC-1	–	–	–	–	0.928
		GOS-LSC-2	–	–	–	–	0.927
		GOS-LSC-3	–	–	–	–	0.933
		GOS-LSC-4	–	–	–	–	0.932

The differences in performance compared to the fixed learning rate are given inside the brackets. It measures the relative robustness, $R_{y,h}$, by computing summing the differences, $\psi_{y,h,o}$, between the performance and the best one from the same optimizer. The ratios of test to training accuracies are given in the last column, and the average ratios for each optimizer are computed in the last row. The highest train, test accuracies, and the lowest robustness measures are indicated in bold

gave the most robust test result among GOS-LSC. GOS-LS shows the average ratio closest to one. Hence, GOS-LS is again the most generalizing strategy, followed by fixed recommended learning rates and GOS-LSC-1. Note that it is more challenging to generalize on EfficientNet-B0 compared to ResNet-18. GOS-LSC-3 shows the lowest ratio of test to training accuracies, meaning that GOS-LSC-3 experiences more overfitting. On average, the EfficientNet-B0 results show that GOS-LSC-3 and 4 outperform GOS-LS in the training results, while only GOS-LSC-4 outperforms GOS-LS in test results.

Table 5 computes the overall training and test robustness measures, R_y, and the average test to training accuracies over the two problems, ResNet-18 and EfficientNet-B0. Hence, the relative robustness, R_y, for a strategy is computed as the sum of $R_{y,h}$ from Tables 3, and 4. While both training and test R_y is the lowest with GOS-LSC-4, followed by GOS-LS, GOS-LS has the highest generalizability ratio, followed by GOS-LSC-4. According to the results, GOS-LSC-4 is the most robust hyperparameter setting for GOS-LSC in DNNs, while GOS-LSC-3 is the least robust in testing and generalization,

Table 4 Top average training and test accuracies over the five runs tabulated for optimizers, including SGD, RMSProp, and Adam, with the fixed recommended learning rates, vanilla GOS, and GOS-LSC with various settings on EfficientNet-B0

Models	Optimizers	Strategies	Train acc. [%]	Diff., $\psi_{y,h,o}$	Test acc. [%]	Diff., $\psi_{y,h,o}$	Te./Tr.
EfficientNet-B0	SGD	Fixed	98.18	0.7	85.37	4.48	0.87
		GOS-LS	97.94 (-0.24)	0.94	89.44 (+4.07)	0.41	0.913
		GOS-LSC-1	98.25 (+0.07)	0.63	85.39 (+0.02)	4.46	0.869
		GOS-LSC-2	98.15 (-0.03)	0.73	84.97 (-0.4)	4.88	0.866
		GOS-LSC-3	98.64 (+0.46)	0.24	87.8 (+1.43)	2.05	0.89
		GOS-LSC-4	**98.88 (+0.7)**	0	**89.85 (+4.48)**	0	0.909
	RMSProp	Fixed	97.6	1.53	89.44	0.15	0.916
		GOS-LS	98.43 (+0.83)	0.7	89.0 (-0.44)	0.59	0.904
		GOS-LSC-1	97.51 (-0.09)	1.62	89.55 (+0.11)	0.04	0.918
		GOS-LSC-2	97.46 (-0.14)	1.67	89.42 (-0.02)	0.17	0.918
		GOS-LSC-3	98.4 (+0.8)	0.73	81.51 (-7.93)	8.08	0.828
		GOS-LSC-4	**99.13 (+1.53)**	0	**89.59 (+0.15)**	0	0.904
	Adam	Fixed	99.21	0.15	89.96	0.34	0.907
		GOS-LS	98.85 (-0.36)	0.51	90.22 (+0.26)	0.08	0.913
		GOS-LSC-1	99.34 (+0.13)	0.02	90.22 (+0.26)	0.08	0.908
		GOS-LSC-2	**99.36 (+0.15)**	0	90.13 (+0.17)	0.17	0.907
		GOS-LSC-3	99.0 (-0.21)	0.36	**90.3 (+0.34)**	0	0.912
		GOS-LSC-4	98.57 (-0.64)	0.79	90.18 (+0.22)	0.12	0.915
	$R_{y,h}$	Fixed	–	2.38	–	4.97	–
		GOS-LS	–	2.15	–	1.08	–
		GOS-LSC-1	–	2.27	–	4.58	–
		GOS-LSC-2	–	2.4	–	5.22	–
		GOS-LSC-3	–	1.33	–	10.13	–
		GOS-LSC-4	–	**0.79**	–	**0.12**	–
	Avg. Te./Tr.	Fixed	–	–	–	–	0.898
		GOS-LS	–	–	–	–	0.91
		GOS-LSC-1	–	–	–	–	0.898
		GOS-LSC-2	–	–	–	–	0.897
		GOS-LSC-3	–	–	–	–	0.877
		GOS-LSC-4	–	–	–	–	0.909

The differences in performance compared to the fixed learning rate are given inside the brackets. It measures the relative robustness, $R_{y,h}$, by computing summing the differences, $\psi_{y,h,o}$, between the performance and the best one from the same optimizer. The ratios of test to training accuracies are given in the last column, and the average ratios for each optimizer are computed in the last row. The highest train, test accuracies, and the lowest robustness measures are indicated in bold

leading to overfitting. The only difference between the GOS-LSC-3 and GOS-LSC-4 is that GOS-LSC-4 resets the next initial learning rate to the default initial learning rate.

From the results of Numerical study 1, we discovered that using the previous resultant learning rates from the previous iteration, α^*_{n-1}, as the next initial guess, $\alpha_{0,n}$, is not a reliable plan for using quadratic approximations. When for a highly non-linear problem, we expect a large discrepancy in the shapes of the approximation in different iterations. Hence, starting the initial guess with the recommended learning rate, γ, is much more conservative for the chosen optimizer at every iteration. This phenomenon was apparent when the optimizer was Adam.

As a result, we recommend the user employ the GOS-LSC-4 hyperparameter setting, as it resets the initial guess as the fixed recommended learning rates, γ, at every iteration. Among the different hyperparameter settings, GOS-LSC-4 had the best relative robustness measures (RRMs) for both training and test accuracies, and it also had the best generalizability among the different hyperparameter settings. This means that GOS-LSC-4 is least likely to fail to an unseen problem.

Table 5 The training and test relative robustness, R_y, for different strategies are given by summing the training and test relative robustness, $R_{y,h}$, over different problems, given in Tables 3 and 4

Strategies	Training R_y	Test R_y	Avg. Te./Tr.
Fixed	2.64	7.8	0.913
GOS-LS	2.5	2.28	0.922
GOS-LSC-1	2.41	6.83	0.913
GOS-LSC-2	9.33	14.17	0.912
GOS-LSC-3	2.55	11.96	0.905
GOS-LSC-4	**0.97**	**1.31**	0.921

The average ratios of training to test accuracies for each strategy are given as the average values of the ratios (Te./Tr.) from the two problems in Tables 3 and 4. The lowest train and test accuracies robustness measures are indicated in bold

GOS-LSC-4 tended to show more aggressive learning rates followed by GOS-LS, which led to better performance than GOS-LS. However, GOS-LS does not have any convergence proof enforced, and it is not robust in terms of convergence.

6.2 Results of numerical study 2

This section studies GOS-LSC on a shallower DNN problem, N-II. We start with the hyperparameter study of four different settings, as shown in Table 1. We expect GOS-LSC-4 to outperform other settings because the setting does not use the previously resolved learning rate, α^*_{n-1}, as the next initial guess, $\alpha_{0,n}$. This reduces the model error based on the previous study. Additionally, we use the $1/\|\boldsymbol{d}_n\|_2$ value as the initial guess, which elongates as the magnitudes of gradients reduce. This helps to increase the model accuracy.

Next, we determine the most robust GOS-LSC setting based on the RRM and compare it to nine other strategies only on SGD, which is an optimizer that is sensitive to the choice of learning rates. The results we obtain from this numerical study help us to determine which strategies are robust in training DNNs.

6.2.1 Performance test for different hyperparameter settings

Figure 7 shows the training error, test error, and learning rates for the various hyperparameter settings on the N-II architecture with MNIST. Note that the training and test errors for MNIST are plotted on the $\log_{10}$ scale, averaged over ten runs.

GOS-LSC-1, which starts every iteration with the initial learning rate of 0.01, did not modify the learning rates. This means that the fixed recommended learning rate of SGD continuously satisfied the curvature condition, resulting in the IAC condition. Hence, it would perform almost exactly like the fixed learning rate of 0.01. Both GOS-LSC-2 and GOS-LSC-3 reuse the previously resolved learning rates. As a result, both settings experience significant approximation errors, and the learning rates are unstable, often approaching zeros.

On the other hand, GOS-LSC-4, whose setting is similar to GOS-LSC-1 except for the initial guess equals $1/\|\boldsymbol{d}_n\|_2$, outperforms other hyperparameter settings, and the learning rates are close to the vanilla GOS-LS. The differences in learning rates tend to increase as the larger batch sizes grow. GOS-LSC-4 shows its best performance when its learning rates are closest to GOS-LS. This is when the batch size is 100.

Table 6 shows the top average training and test accuracies for each hyperparameter setting and the differences, $\psi_{y,|\mathscr{B}|}$, for each batch size on N-II. The ratios of test to training accuracies are given in the last column. Note that there are ratios greater than one. This might happen due to biases existing in the test dataset when the problem is simple to generalize. The lowest ratios are given by GOS-LSC-3 except for the batch size, $|\mathscr{B}|$, of 100, meaning that GOS-LSC-3 has the least generalizability. Except when the batch size, $|\mathscr{B}|$, is 10, the difference values, $\psi_{y,|\mathscr{B}|}$, show that the aggressive vanilla GOS-LS tends to outperform all GOS-LSC settings again because no convergence condition restricts its learning rate.

Table 7 shows the training and test relative robustness, R_y, and the average ratios of test to training accuracies of the results in Table 6 over the batch sizes. The measures indicate that GOS-LS is the most robust strategy in training and test results for this problem, followed by GOS-LSC-4. Note that this result is consistent with the previous numerical study. Since the N-II architecture is not challenging to generalize, the average ratios are practically alike. Still, the results show that the least generalizable strategies are GOS-LS and GOS-LSC-3 on this problem with the smallest average ratios. Based on these results, we chose GOS-LSC-4 for further comparison against other strategies.

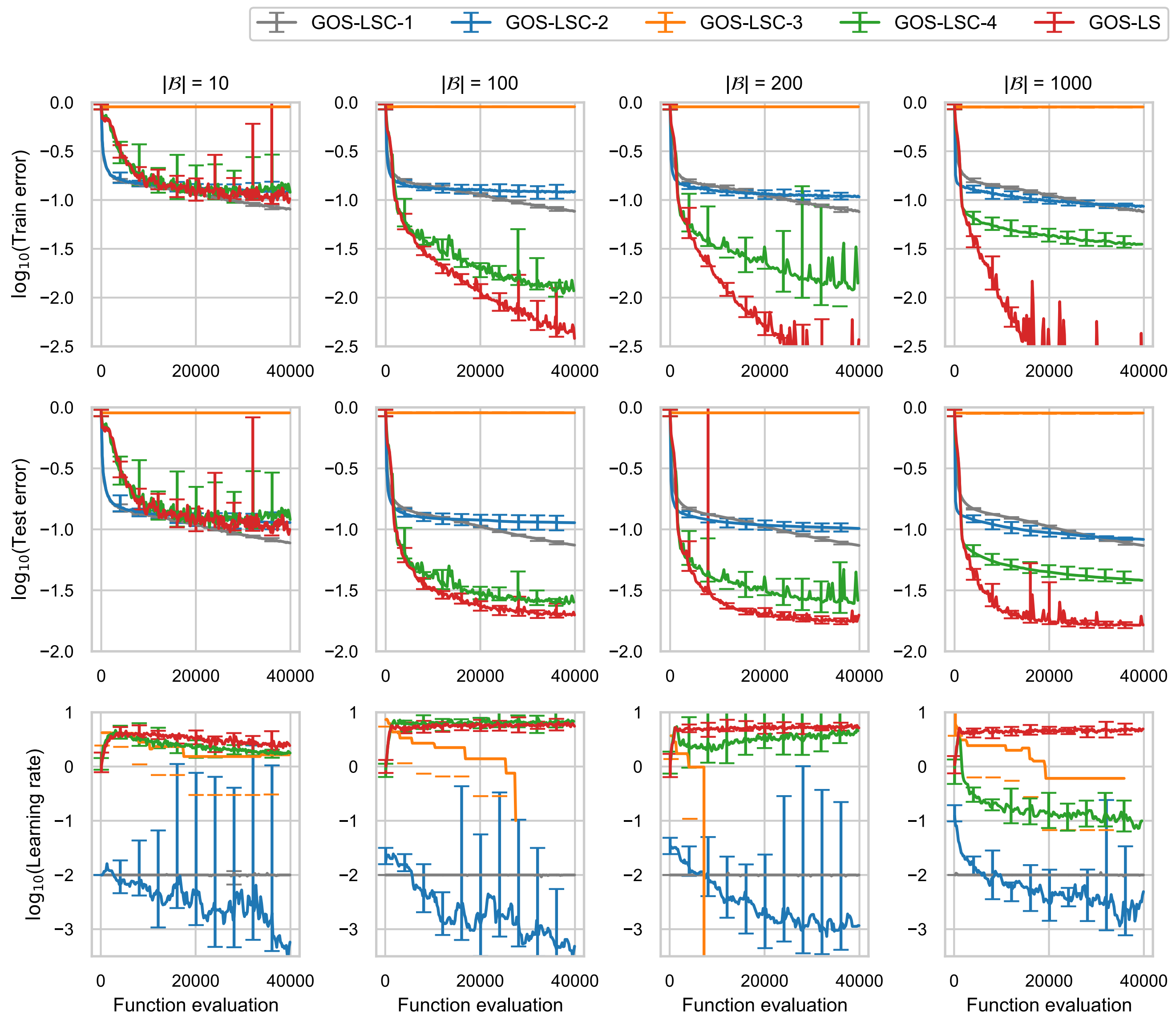


Fig. 7 N-II MNIST dataset with batch size, $|\mathscr{B}|$ = 10, 100, 200, and 1000 from left to right for various hyperparameters settings of GOS-LSC which are listed in Table 1. The comparison of training dataset error (the 1st row), test set error (the 2nd row), and learning rate (the 3rd row) on a log_{10} scale versus the number of function evaluations

6.2.2 Performance comparison for different strategies

Next, Fig. 8 shows the training error, test error, and learning rates for the ten strategies on the N-II architecture with MNIST. The constant learning rates generally show low variance in error during training. Its error noticeably reduces from $|\mathscr{B}| = 10$ to $|\mathscr{B}| = 100$ since the SGD direction becomes more representative of the exact (full-batch) SGD direction. The learning rates are independent of noisy information when $|\mathscr{B}|$ is small. Hence, the more straightforward strategies may perform better than more sophisticated ones.

The best performing strategy for SGD overall is GOS-LS, which is consistent with Section 5. Although the learning rates of GOS-LS are continuously larger than others, SGD benefits from overshooting because it causes conjugacy in search directions. The training performance of GOS-LS noticeably improves as $|\mathscr{B}|$ increases.

Cosine annealing with warm restarts changes the learning rates over epoch periods and resets them to the initial learning rates. Hence, we observe slight fluctuations in both training and test errors. Although this method sweeps through an extensive range of learning rates, the results show that initial learning rate choice significantly affects the strategy's performance.

Both GOS-LSC and GOLS-I try to locate SNN-GPPs. However, GOS-LSC, which uses the quadratic model, outperforms GOLS-I without any approximation model.

Table 6 Top average training and test accuracy over the ten runs for the GOS-LSC settings, GOS-LSC-1, GOS-LSC-2, GOS-LSC-3, GOS-LSC-4, and GOS-LS on the N-II architecture with different batch sizes, $|\mathscr{B}| = 10, 100, 200, 1000$ for the SGD optimizer

Batch size, $\vert\mathscr{B}\vert$	Strategies	Train acc.	Diff., $\psi_{y,\vert\mathscr{B}\vert}$	Test acc.	Diff., $\psi_{y,\vert\mathscr{B}\vert}$	Te./Tr.
10	GOS-LSC-1	**91.95**	0	**92.26**	0	1.003
	GOS-LSC-2	88.05	3.9	88.63	3.63	1.007
	GOS-LSC-3	10.05	81.9	10.02	82.24	0.997
	GOS-LSC-4	89.4	2.55	89.55	2.71	1.002
	GOS-LS	90.62	1.33	90.93	1.33	1.003
100	GOS-LSC-1	92.33	7.29	92.57	5.49	1.003
	GOS-LSC-2	88.05	11.57	88.66	9.4	1.007
	GOS-LSC-3	9.95	89.67	9.93	88.13	0.998
	GOS-LSC-4	98.89	0.73	97.55	0.51	0.986
	GOS-LS	**99.62**	0	**98.06**	0	0.984
200	GOS-LSC-1	92.43	7.44	92.62	5.64	1.002
	GOS-LSC-2	89.23	10.64	89.83	8.43	1.007
	GOS-LSC-3	9.98	89.89	9.78	88.48	0.98
	GOS-LSC-4	98.8	1.07	97.53	0.73	0.987
	GOS-LS	**99.87**	0	**98.26**	0	0.984
1000	GOS-LSC-1	92.44	7.49	92.62	5.75	1.002
	GOS-LSC-2	91.49	8.44	91.74	6.63	1.003
	GOS-LSC-3	10.31	89.62	10.12	88.25	0.982
	GOS-LSC-4	96.51	3.42	96.18	2.19	0.997
	GOS-LS	**99.93**	0	**98.37**	0	0.984

It also measures the difference, $\psi_{y,|\mathscr{B}|}$, between the performance and the best one from the different batch sizes. The ratios of test to training accuracies are given in the last column. The highest train and test accuracies are indicated in bold

Both strategies double the learning rates to grow, but GOS-LSC uses quadratic approximations to shrink, while GOLS-I halves the learning rates to shrink. Hence, GOS-LSC may perform worse if the quadratic approximation has a large approximation error due to noisy information.

Table 8 shows each strategy's top average training and test accuracies and the differences measured for the same experiment. The last column in the table shows the ratios of test to training accuracies. The fixed learning rate results indicate that the performance of SGD differs considerably with the choice of learning rates. The fixed learning rate of 0.1 presents the best training and test accuracies for the batch size of 10. For the larger batch sizes, GOS-LS shows the highest accuracies of all strategies with SGD. The average ratios show that some are greater than one. This could mean the test data is biased when the problem is not challenging to generalize. The highest and lowest ratios are shown by the fixed learning rate of 0.001 and 10, respectively. When the ratios are lower, the strategy is more overfitted.

Table 9 shows the training and test relative robustness, R, and the average ratios of test to training accuracies of the results in Table 8 over the different batch sizes. The most robust training and test results are obtained from GOS-LS

Table 7 The training and test relative robustness, R_y, for different strategies are given by summing the differences, $\Psi_{y,|\mathscr{B}|}$, given in Table 6

Strategies	Training R_y	Test R_y	Avg. Te./Tr.
GOS-LSC-1	22.22	16.88	1.003
GOS-LSC-2	34.55	28.09	1.006
GOS-LSC-3	351.08	347.1	0.989
GOS-LSC-4	7.77	6.14	0.993
GOS-LS	**1.33**	**1.33**	0.989

The average ratios of training to test accuracies for each strategy are given as the average values of the ratios listed in Table 6 across the different batch sizes. The lowest robustness measures are indicated in bold

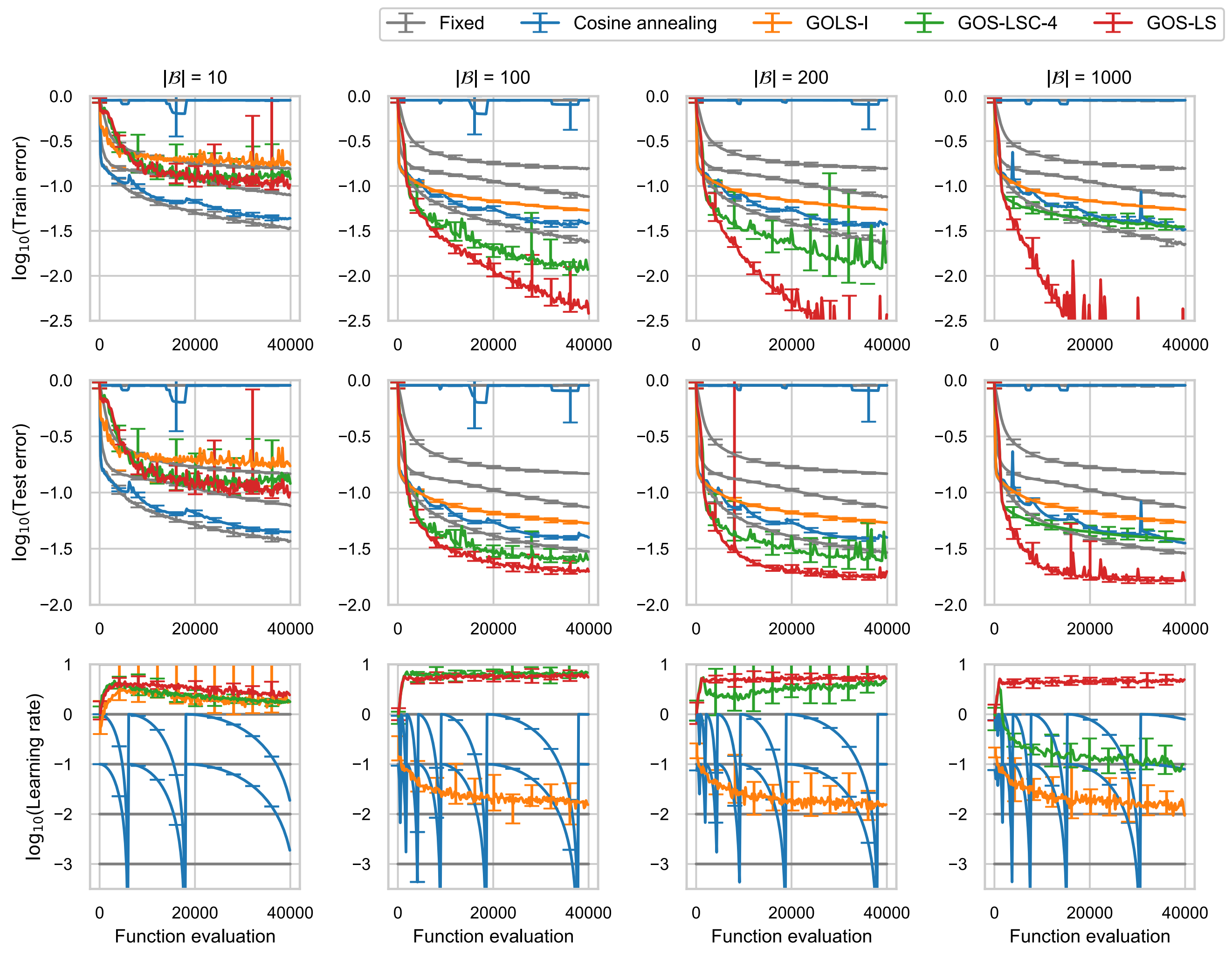


Fig. 8 N-II MNIST dataset with batch size, $|\mathscr{B}|$ = 10, 100, 200, and 1000 from left to right for various line search methods: constant learning rates, cosine annealing, GOLS-I, vanilla GOS-LS, and GOS-LSC-4. The comparison of training dataset error (the 1st row), test set error (the 2nd row), and learning rate (the 3rd row) on a $\log_{10}$ scale versus the number of function evaluations

and the fixed learning rate of 0.1, respectively. GOS-LSC-4 ranked fourth in both training and test results, led by GOS-LS, with a fixed learning rate of 0.1 and cosine annealing with $\alpha = 0.1$. However, note that the results of learning rate and cosine annealing are highly responsive to learning rates.

The table also computes the RRMs of each strategy when the impractical mini-batch size of 10 is excluded from computing the measures. Note that GOS-LSC-4 now ranks third and second in both training and test robustness, respectively. This happens because the information for the batch size of 10 makes the information sparse, affecting the approximation accuracy for GOS-LSC. The fixed learning rates and cosine annealing are not sensitive to the sparsity of the information like GOS-LSC, but GOS-LSC can adopt learning rates based on the available information.

7 Conclusions

Dynamic mini-batch sub-sampling (MBSS) in deep neural network problems causes the loss of functions point-wise discontinuous. This makes the function value minimization approach impractical as line searches because it would find infinitely many local minima in discontinuous settings. Dynamic MBSS also causes sampling errors, manifesting as small bias and large variance. To minimize the variance, a recent study introduced gradient-only surrogates (GOS) to resolve learning rates. GOS is a quadratic function approximation model constructed using only directional derivative information. It approximates the spatial locations of sign changes in directional derivatives to choose learning rates. The previous study showed the competitive

Table 8 Top average training and test accuracies over the ten runs for the SGD optimizer with various learning rate strategies, y, including the fixed learning rates, cosine annealing with warm restart, GOLS-I, GOS-LSC-4, and GOS-LS on the N-II architecture with different batch sizes, $|\mathscr{B}| = 10, 100, 200, 1000$

Batch size, $\|\mathscr{B}\|$	Strategies	Train acc.	Diff., $\psi_{y,\|\mathscr{B}\|}$	Test acc.	Diff., $\psi_{y,\|\mathscr{B}\|}$	Te./Tr.
10	Fixed, $\alpha = 0.001$	84.54	12.13	85.3	11.06	1.009
	Fixed, $\alpha = 0.01$	92.1	4.57	92.4	3.96	1.003
	Fixed, $\alpha = 0.1$	**96.67**	0	**96.36**	0	0.997
	Fixed, $\alpha = 1$	10.09	86.58	10.14	86.22	1.005
	Fixed, $\alpha = 10$	10.05	86.62	9.74	86.62	0.969
	Cosine, $\alpha = 0.1$	95.73	0.94	95.55	0.81	0.998
	Cosine, $\alpha = 1$	36.2	60.47	36.4	59.96	1.006
	GOLS-I	83.29	13.38	83.39	12.97	1.001
	GOS-LSC-4	89.4	7.27	89.55	6.81	1.002
	GOS-LS	90.62	6.05	90.93	5.43	1.003
100	Fixed, $\alpha = 0.001$	84.59	15.03	85.27	12.79	1.008
	Fixed, $\alpha = 0.01$	92.41	7.21	92.61	5.45	1.002
	Fixed, $\alpha = 0.1$	97.64	1.98	97.02	1.04	0.994
	Fixed, $\alpha = 1$	10.05	89.57	9.95	88.11	0.99
	Fixed, $\alpha = 10$	10.03	89.59	9.86	88.2	0.983
	Cosine, $\alpha = 0.1$	96.35	3.27	96.03	2.03	0.997
	Cosine, $\alpha = 1$	36.68	62.94	36.84	61.22	1.004
	GOLS-I	94.66	4.96	94.68	3.38	1
	GOS-LSC-4	98.89	0.73	97.55	0.51	0.986
	GOS-LS	**99.62**	0	**98.06**	0	0.984
200	Fixed, $\alpha = 0.001$	84.46	15.41	85.28	12.98	1.01
	Fixed, $\alpha = 0.01$	92.53	7.34	92.65	5.61	1.001
	Fixed, $\alpha = 0.1$	97.72	2.15	97.08	1.18	0.993
	Fixed, $\alpha = 1$	10.07	89.8	9.89	88.37	0.982
	Fixed, $\alpha = 10$	10.02	89.85	9.84	88.42	0.982
	Cosine, $\alpha = 0.1$	96.42	3.45	96.11	2.15	0.997
	Cosine, $\alpha = 1$	19.14	80.73	19.07	79.19	0.996
	GOLS-I	94.56	5.31	94.6	3.66	1
	GOS-LSC-4	98.8	1.07	97.53	0.73	0.987
	GOS-LS	**99.87**	0	**98.26**	0	0.984
1000	Fixed, $\alpha = 0.001$	84.42	15.51	85.29	13.08	1.01
	Fixed, $\alpha = 0.01$	92.39	7.54	92.65	5.72	1.003
	Fixed, $\alpha = 0.1$	97.79	2.14	97.12	1.25	0.993
	Fixed, $\alpha = 1$	10.31	89.62	10.14	88.23	0.984
	Fixed, $\alpha = 10$	10.06	89.87	9.94	88.43	0.988
	Cosine, $\alpha = 0.1$	96.77	3.16	96.45	1.92	0.997
	Cosine, $\alpha = 1$	18.5	81.43	18.52	79.85	1.001
	GOLS-I	94.55	5.38	94.57	3.8	1
	GOS-LSC-4	96.51	3.42	96.18	2.19	0.997
	GOS-LS	**99.93**	0	**98.37**	0	0.984

It measures the difference, $\psi_{y,|\mathscr{B}|}$, between the performance and the best one from the different batch sizes. The ratios of test to training accuracies are given in the last column. The highest train and test accuracies are indicated in bold

performance of GOS line search (GOS-LS) in training DNNs. However, it does not have a convergence proof developed, and it makes GOS-LS not conservative.

Hence, we extended the vanilla GOS-LS to a more robust line search by enforcing GOS-LS with the convergence proof and the bracketing strategy. We implemented the

Table 9 The training and test relative robustness, R_y, for different strategies are given by summing the differences, $\Psi_{y,|\mathscr{B}|}$, given in Table 8

Strategies	Training R_y	Test R_y	Avg. Te./Tr.
Fixed, $\alpha = 0.001$	58.08 (30.54)	49.91 (38.85)	1.009
Fixed, $\alpha = 0.01$	26.66 (14.75)	20.74 (16.78)	1.002
Fixed, $\alpha = 0.1$	6.27 (4.12)	**3.47** (3.47)	0.994
Fixed, $\alpha = 1$	355.57 (179.19)	350.93 (264.71)	0.99
Fixed, $\alpha = 10$	355.93 (179.46)	351.67 (265.05)	0.981
Cosine, $\alpha = 0.1$	10.82 (6.43)	6.91 (6.1)	0.997
Cosine, $\alpha = 1$	285.57 (144.37)	280.22 (220.26)	1.002
GOLS-I	29.03 (10.34)	23.81 (10.84)	1
GOS-LSC-4	12.49 (4.15)	10.24 (3.43)	0.993
GOS-LS	**6.05** (0)	5.43 (0)	0.989

The average ratios of training to test accuracies for each strategy are given as the average values of the ratios listed in Table 8 across the different batch sizes. The lowest robustness measures are indicated in bold. The relative robustness, R_y, measured excluding $|\mathscr{B}| = 10$, is listed in brackets

Wolfe curvature condition as the convergence proof because it only requires directional derivative information. The bracketing strategy is empowered by the Regular-Falsi method. It aims to restrict the domain of GOS-LS for higher model accuracy because the accuracy could have been reduced by implementing the simplistic quadratic models for computational efficiency. Unlike GOS-LS, which constructs an approximation once for a descent direction, GOS-LSC consecutively constructs the GOS models until the curvature condition is satisfied. This makes GOS-LSC more robust in terms of convergence. However, in return for the robustness, the curvature condition also restricts the large learning rates, making the convergence rate of GOS-LSC slightly lower.

We introduced a new relative robustness measure (RRM) for assessing the performance of GOS-LSC. The traditional performance measure for a learning rate strategy only considers the top performance for a specific problem and optimizer. On the other hand, our RRM considers all accounts, including poor performances. Hence, the robustness measure favors a strategy that performs well overall across different optimizers and problems.

We conducted hyperparameter studies for GOS-LSC on ResNet-18 and EfficieintNet-B0 with the CIFAR-10 dataset using various optimizers, including SGD, RMSProp, and Adam. Testing on various optimizers showed the adaptability of GOS-LSC as a learning rate strategy. We learned it is essential to choose the data point close to the origin based on the robustness measure to reduce the model error. Using the fixed recommended learning rate for the specific optimizer to determine the data point for every descent direction turned out to be a better choice to reduce the approximation error than using the previously resolved learning rate as the next initial guess. The experimental results showed that one hyperparameter setting close to GOS-LS outperformed GOS-LS in both training and test accuracy for both test problems. It also showed that GOS-LSC, led by GOS-LS, generalizes better and is more robust than the recommended learning rates for each optimizer.

We further compared the performance of GOS-LSC against nine other learning rate strategies on a shallower DNN, N-II, using only the SGD optimizer, as the performance of SGD is sensitive to learning rates. For this less non-linear problem compared to ResNet-18 and EfficientNet-B0, extending the initial data point for GOS-LSC farther than the recommended learning rates helped with performances based on the RRMs. On the N-II architecture, GOS-LSC ranked third and second for the training and test robustness based on the RRMs, respectively, led by less conservative GOS-LS among ten learning rate strategies in total. Hence, the overall investigation shows that GOS-LSC outperforms GOS-LS when the architecture is more complex. The results warrant future investigations of approximation approaches to resolve learning rates because it is competitive against other well-known learning rate strategies.

Acknowledgements This research was supported by the National Research Foundation (NRF), South Africa, and the Center for Asset Integrity Management (C-AIM), Department of Mechanical and Aeronautical Engineering, University of Pretoria, Pretoria, South Africa. We would like to express our special thanks to Nvidia Corporation for supplying the GPUs on which this research was conducted.

References

1. Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, Corrado GS, Davis A, Dean J, Devin M, Ghemawat S, Goodfellow I, Harp A, Irving G, Isard M, Jia Y, Jozefowicz R, Kaiser L, Kudlur M, Levenberg J, Mané D, Monga R, Moore S, Murray D, Olah C, Schuster M, Shlens J, Steiner B, Sutskever I, Talwar K, Tucker P, Vanhoucke V, Vasudevan V, Viégas F, Vinyals O, Warden P, Wattenberg M, Wicke M, Yu

Y, Zheng X (2015) TensorFlow: large-scale machine learning on heterogeneous systems. software available from tensorflow.org. https://www.tensorflow.org/. Accessed 10 Aug 2021

2. Agarwal N, Bullins B, Hazan E (2017) Second-order stochastic optimization for machine learning in linear time. J Mach Learn Res 18(1):4148–4187
3. Bengio Y (2012) Practical recommendations for gradient-based training of deep architectures. In: Neural networks tricks of the trade. Springer, pp 437–478
4. Bergou Eh, Diouane Y, Kunc V, Kungurtsev V, Royer CW (2018) A subsampling line search method with second-order results. arXiv:181007211
5. Bollapragada R, Byrd R, Nocedal J (2018) Adaptive sampling strategies for stochastic optimization. SIAM J Optim 28(4):3312–3343
6. Brown T, Mann B, Ryder N, Subbiah M, Kaplan JD, Dhariwal P, Neelakantan A, Shyam P, Sastry G, Askell A, Agarwal S, Herbert-Voss A, Krueger G, Henighan T, Child R, Ramesh A, Ziegler D, Wu J, Winter C, Hesse C, Chen M, Sigler E, Litwin M, Gray S, Chess B, Clark J, Berner C, McCandlish S, Radford A, Sutskever I, Amodei D (2020) Language models are few-shot learners. In: Larochelle H, Ranzato M, Hadsell R, Balcan MF, Lin H (eds) Advances in neural information processing systems, Curran associates, Inc., vol 33, pp 1877–1901. https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf. Accessed 04 Oct 2021
7. Byrd RH, Chin GM, Nocedal J, Wu Y (2012) Sample size selection in optimization methods for machine learning. Math Program 134(1):127–155. https://doi.org/10.1007/s10107-012-0572-5
8. Chae Y, Wilke DN (2019) Empirical study towards understanding line search approximations for training neural networks. arXiv:190906893
9. Friedlander MP, Schmidt M (2012) Hybrid deterministic-stochastic methods for data fitting. SIAM J Sci Comput 34(3):A1380–A1405
10. Glorot X, Bengio Y (2010) Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the thirteenth international conference on artificial intelligence and statistics, JMLR workshop and conference proceedings, pp 249–256
11. Goodfellow I, Bengio Y, Courville A (2016) Deep learning (Adaptive computation and machine learning series). The MIT Press
12. Gupta RK (2019) Numerical methods: fundamentals and applications. Cambridge University Press
13. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 770–778
14. Kafka D, Wilke DN (2019) Gradient-only line searches: an alternative to probabilistic line searches. arXiv:190309383
15. Kafka D, Wilke DN (2021) Resolving learning rates adaptively by locating stochastic non-negative associated gradient projection points using line searches. J Glob Optim 79(1):111–152
16. Krizhevsky A (2009) Learning multiple layers of features from tiny images. Tech rep, Department of Computer Science, University of Toronto
17. LeCun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. Proc IEEE 86(11):2278–2324
18. Liu K (2020) 95.16% on CIFAR10 with PyTorch. https://github.com/kuangliu/pytorch-cifar. Accessed 12 June 2021
19. Loshchilov I, Hutter F (2017) SGDR: stochastic gradient descent with warm restarts. In: 5th international conference on learning representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference track proceedings, OpenReview.net. https://openreview.net/forum?id=Skq89Scxx
20. Lyapunov AM (1992) The general problem of the stability of motion. Int J Control 55(3):531–534
21. Mahsereci M, Hennig P (2017) Probabilistic line searches for stochastic optimization. J Mach Learn Res 18(1):4262–4320
22. Masters D, Luschi C (2018) Revisiting small batch training for deep neural networks. arXiv:180407612
23. Mutschler M, Zell A (2020) Parabolic approximation line search for dnns. Adv Neural Inf Process Syst 33:5405–5416
24. Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L, Desmaison A, Kopf A, Yang E, DeVito Z, Raison M, Tejani A, Chilamkurthy S, Steiner B, Fang L, Bai J, Chintala S (2019) PyTorch: an imperative style, high-performance deep learning library. In: Wallach H, Larochelle H, Beygelzimer A, dAlché-Buc F, Fox E, Garnett R (eds) Advances in neural information processing systems 32, Curran associates, Inc., pp 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf. Accessed 06 Aug 2021
25. Robbins H, Monro S (1951) A stochastic approximation method. Ann Math Stat, pp 400–407
26. Snyman J, Wilke D (2018) Practical mathematical optimization: basic optimization theory and gradient-based algorithms. Springer optimization and its applications, Springer international publishing. https://books.google.co.kr/books?id=n1dLswEACAAJ. Accessed 27 July 2021
27. Strubell E, Ganesh A, McCallum A (2020) Energy and policy considerations for modern deep learning research. In: Proceedings of the AAAI conference on artificial intelligence vol 34, pp 13693–13696
28. Tan M, Le Q (2019) Efficientnet: rethinking model scaling for convolutional neural networks. In: International conference on machine learning, PMLR, pp 6105–6114
29. Yedida R, Saha S, Prashanth T (2021) LipschitzLR: using theoretically computed adaptive learning rates for fast convergence. Appl Intell 51(3):1460–1478