OPT2022: 14th Annual Workshop on Optimization for Machine Learning

Optimization using Parallel Gradient Evaluations on Multiple Parameters

Yash Chandak YCHANDAK@CS.UMASS.EDU
University of Massachusetts

Shiv Shankar SSHANKAR@CS.UMASS.EDU
University of Massachusetts

Venkata Gandikota GANDIKOTA.VENKATA@GMAIL.COM
Syracuse University

Philip Thomas PTHOMAS@CS.UMASS.EDU
University of Massachusetts

Arya Mazumdar ARYA@UCSD.EDU
University of California San Diego

Abstract

We propose a first-order method for convex optimization, where instead of being restricted to the gradient from a single parameter, gradients from multiple parameters can be used during each step of gradient descent. This setup is particularly useful when a few processors are available that can be used in parallel for optimization. Our method uses gradients from multiple parameters in synergy to update these parameters together towards the optima. While doing so, it is ensured that the computational and memory complexity is of the same order as that of gradient descent. Empirical results demonstrate that even using gradients from as low as *two* parameters, our method can often obtain significant acceleration and provide robustness to hyper-parameter settings. We remark that the primary goal of this work is less theoretical, and is instead aimed at exploring the understudied case of using multiple gradients during each step of optimization.

1. Introduction

Ranging from personal computers to internet-of-things enabled hardware, there exists a plethora of local computational devices that have capabilities of executing a few parallel processes but are otherwise limited in their memory and computational capacities. Having optimization procedures designed for such constrained systems can, therefore, enable wider use of machine learning models. However, gradient descent based optimizers predominantly only use a single new gradient during each step of optimization. Perhaps one common way to leverage multiple processes is by splitting the computation of the gradient for that *single* parameter (e.g., by distributing data), if possible. In this paper, we focus on a complementary problem, where we leverage multiple processes for gradient computation of *multiple* parameters.

Particularly, we restrict our focus to optimization of convex functions. Formally, for any convex function $f : \mathbb{R}^d \to \mathbb{R}$, and a fixed domain $\Omega \subset \mathbb{R}^d$ our goal is to find

$$\theta^* \in \underset{\theta \in \Omega}{\arg\min}\ f(\theta).$$

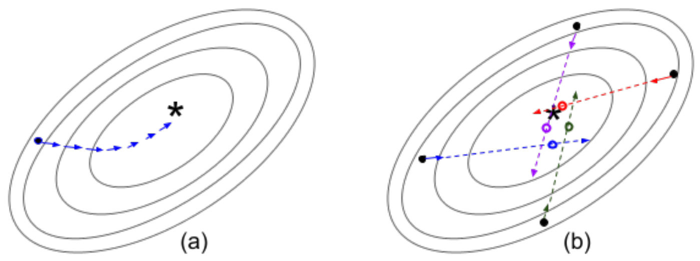

Figure 1: (a) Conventional gradient descent based methods have only one parameter vector which gets iteratively updated. (b) Proposed method maintains a set of N parameters (here, $N = 4$) and the gradients at each of those points are used to directly move the parameters near the minimizer of the function, marked as $\star$. Solid arrows depict the gradient and the proposed method uses them to jump to the points marked with empty circles in the direction of gradient.

Broadly, popular approaches for finding θ^* can be grouped into zeroth-order, first-order, or second-order methods. In the following paragraphs, we briefly review how these approaches can potentially be used for the desired problem setup. Detailed discussions are deferred to Appendix A.

Zeroth-order methods: Methods like genetic algorithms, evolutionary strategies, etc., work by evaluating the function $f(\theta)$ for multiple values of θ to estimate the optimization geometry and compute a direction that moves the parameters towards the optima [4, 6]. In our problem setup, each individual evaluation of $f(\theta)$ could ideally be done in parallel. However, especially in high dimensions, these methods may scale poorly [17], thereby requiring a lot of parallel resources, which are not often available.

First-order methods: Methods such as gradient descent, ADAM [7], etc., iteratively update a single parameter vector θ using the gradient $\nabla f(\theta)$ to move towards the optima. Multiple processes can be leveraged by performing independent runs from different initializations and taking the minimum among the runs. However, over here, optimizing from a given initialization does not make use of the information available from other parallel searches. Thereby not making an ideal utilization of the already limited resource. Alternatively, often $f : \mathbb{R}^d \times \mathcal{D} \to \mathbb{R}$ is a function of data $\mathcal{D}$. Multiple processes are often leveraged by distributing computation of $\nabla f(\theta; \mathcal{D})$, for a common θ, over mini-batches of data and then aggregating all gradients. This approach has the downside of not capturing information about the function at various points. In this work, we gain advantage over both the above mentioned approaches and show how $f(\theta; \mathcal{D})$, for *different* values of θ with mini-batches can be used to improve the search for θ^*.

Second-order methods: Optimizers like Newton's method make direct use of the Hessian $\nabla^2 f(\theta)$ to move θ towards θ^* [2]. Access to the curvature of the function makes these methods more robust to hyper-parameters. However, this requires inverting the Hessian, which can be computationally expensive for our setup. Several quasi-Newton methods exist that minimize the computation required for estimating the Hessian inverse but still need to keep a matrix having a large $O(d^2)$ memory footprint [12]. Limited memory versions of quasi-Newton methods have also been developed that only need to store past k *sequentially* generated gradient evaluations [8].

Contribution Summary: An ideal optimization procedure for our problem setup would meet the following criteria: **(a)** It should leverage the availability of the few parallel resources judiciously. **(b)** It only uses first-order gradients to be computationally cheap. **(c)** It does not have large memory footprint, so that it can gracefully scale to large problems. **(d)** Can provide quicker convergence to θ^*, like the second-order methods. **(e)** It is robust to hyper-parameters.

In this work, we propose a new optimization algorithm, called *Gradient Grouping* to satisfy the above requirements. The key idea behind the proposed algorithm is that it makes use of gradients from N *different* parameters in synergy, where N can be as low as 2. During each update, the minima is estimated by extrapolating the N gradients and finding their approximate point of intersection. All the parameters are then updated towards this point, and the process is repeated.

2. Proposed Method

In this section, we first motivate the underlying principle behind our approach and then derive an update rule. Unlike existing methods that use only a single gradient from the current timestep or use multiple gradients obtained in the past, what if we had access to several gradients, *from different parameters*, at any update step? In such a case, multiple gradients can provide additional information about the loss surface and can thus help in both acceleration and automatically choosing step sizes.

The core idea is illustrated in Figure 1. Given gradient evaluations from N different points, we propose estimating the optima by extrapolating these gradients and finding their approximate point of intersection. Formally, given N parameter $\{\theta_i\}$, we propose to update each $\theta_i \in \mathbb{R}^{1\times D}$ as

$$\tilde{\theta}_i = \theta_i + \eta_i \nabla f(\theta_i) \qquad\qquad \forall i \in [1, N], \tag{1}$$

where η_i is the corresponding step size and is set such that it directly makes the parameters θ_i jump to the (approximate) point of intersection of the gradient vectors $\{\nabla f(\theta_i)\}$.

In high dimensions, there is unlikely to be a unique point of intersection. Therefore, we relax this exact intersection constraint and instead aim at making the parameters update to a point where they are closest to each other. Let $\Psi : \mathbb{R}^{d\times N} \to \mathbb{R}$ be the desired function of all the $\tilde{\theta}_i$ that is a (proxy) measure for getting towards the intersection point/area. Our goal is to find,

$$\underset{\{\eta_1,...,\eta_N\}}{\arg\min} \ \Psi(\tilde{\theta}_1, ..., \tilde{\theta}_N),$$

and subsequently use the obtained η_i's to update the respective parameters. Some candidate choices for Ψ can be a polytope/convex-hull of all the points of intersection, or the radius of the ball containing all the points of intersection. However, it can also quite likely be that, in high dimensions, there is *not even a single* point of intersection among the gradients. To still ensure that after the update to the parameters, $\{\theta_i\}$, the resulting set of parameters, $\{\tilde{\theta}_i\}$, are as close to each other as possible, we define Ψ as the following,

$$\Psi(\tilde{\theta}_1, ..., \tilde{\theta}_N) := \sum_{i=1}^{N} \left\| \tilde{\theta}_i - \frac{\sum_j \tilde{\theta}_j}{N} \right\|^2. \tag{2}$$

Before proceeding further, we now establish few notations that will be used throughout the paper. Let $\circ$ represent Hadamard (entrywise) product between two matrices of same dimensions defined as $(A \circ B)_{i,j} = A_{i,j}B_{i,j}$. Let, Θ and G corresponds to matrices with columns containing all the parameter vectors and their gradients, respectively. Let all the step-sizes, η_i for i^{th} parameters, be in a vector η. Let L represents a Laplacian matrix for a fully connected graph with N vertices and let $\mathbb{1}$ denote a ones vector of size N. That is,

$$\begin{aligned} \Theta &\coloneqq [\theta_1, ..., \theta_N] && \in \mathbb{R}^{d\times N}, && G \coloneqq [\nabla f(\theta_1), ..., \nabla f(\theta_N)] && \in \mathbb{R}^{d\times N}, \\ \eta &\coloneqq [\eta_1, ..., \eta_N]^\top && \in \mathbb{R}^{N\times 1}, && \mathbb{1} \coloneqq [1, ..., 1]^\top && \in \mathbb{R}^{N\times 1}, \\ L &\coloneqq N \cdot I - \mathbb{1}\mathbb{1}^\top && \in \mathbb{R}^{N\times N}. \end{aligned}$$

Theorem 1. *For any given $N > 1$*

$$\underset{\eta}{\textit{arg min}}\ \Psi(\tilde{\theta}_1, ..., \tilde{\theta}_N) = -(G^\top G \circ L)^{-1}(G^\top \Theta \circ L)\mathbb{1}.$$

Proof For brevity, we denote $\Psi(\tilde{\theta}_1, ..., \tilde{\theta}_N)$ as Ψ. We begin by expanding (2),

$$\begin{aligned} \Psi &= \sum_{i=1}^N \left\| \tilde{\theta}_i - \frac{\sum_{i=1}^N \tilde{\theta}_i}{N} \right\|^2 \\ &= \sum_{i=1}^N \left\| \theta_i + \eta_i \nabla f(\theta_i) - \frac{\sum_j (\theta_j + \eta_j \nabla f(\theta_j))}{N} \right\|^2 \\ &= \sum_{i=1}^N \left\| \frac{N-1}{N}\theta_i - \frac{\sum_{j\neq i}\theta_j}{N} + \eta_i \frac{N-1}{N}\nabla f(\theta_i) - \frac{\sum_{j\neq i}\eta_j \nabla f(\theta_j)}{N} \right\|^2. \end{aligned} \tag{3}$$

To solve for η that minimizes (3), we first convert (3) into matrix notations. To illustrate the intermediate steps, we define the following,

$$\begin{aligned} M_i &\coloneqq \left[-\frac{1}{N}, ..., \frac{N-1}{N}, ..., -\frac{1}{N}\right] && \in \mathbb{R}^{N\times 1}, \\ \Lambda_i &\coloneqq \text{diag}(M_i) && \in \mathbb{R}^{N\times N}, \end{aligned}$$

where i^{th} position in M_i is $(N-1)/N$ and the rest are $-1/N$. Therefore, (3) can be expressed as,

$$\Psi = \sum_{i=1}^N \|\Theta M_i + G\Lambda_i \eta\|^2. \tag{4}$$

Solving for η that minimizes (4),

$$\frac{\partial \Psi}{\partial \eta} = 2\sum_{i=1}^N (\Theta M_i + G\Lambda_i\eta)^\top (G\Lambda_i) = 2\sum_{i=1}^N M_i^\top \Theta^\top G \Lambda_i + 2\sum_{i=1}^N \eta^\top \Lambda_i^\top G^\top G \Lambda_i.$$

As $M_i = \Lambda_i \mathbb{1}$,

$$\frac{\partial \Psi}{\partial \eta} = 2\sum_{i=1}^N \mathbb{1}^\top \Lambda_i^\top \Theta^\top G\Lambda_i + 2\sum_{i=1}^N \eta^\top \Lambda_i^\top G^\top G \Lambda_i = \mathbb{1}^\top 2\sum_{i=1}^N \Lambda_i^\top \Theta^\top G \Lambda_i + \eta^\top 2 \sum_{i=1}^N \Lambda_i^\top G^\top G \Lambda_i. \tag{5}$$

Equating (5) to 0 gives $\eta^\top \sum_{i=1}^N \Lambda_i^\top G^\top G \Lambda_i = -\mathbb{1}^\top \sum_{i=1}^N \Lambda_i^\top \Theta^\top G \Lambda_i$. Therefore,

$$\eta^\top = -\mathbb{1}^\top \left(\sum_{i=1}^N \Lambda_i^\top \Theta^\top G \Lambda_i \right) \left(\sum_{i=1}^N \Lambda_i^\top G^\top G \Lambda_i \right)^{-1}. \tag{6}$$

To simplify (6), we make use of the following property (See Appendix B for a proof),

Property 1. *For any matrix* $B \in \mathbb{R}^{N\times N}$, $\sum_{i=1}^N \Lambda_i^\top B \Lambda_i = \frac{1}{N} B \circ L$.

Using Property 1 in (6),

$$\eta^\top = -\mathbb{1}^\top \left(\frac{1}{N} \Theta^\top G \circ L \right) \left(\frac{1}{N} G^\top G \circ L \right)^{-1} = -\mathbb{1}^\top \left(\Theta^\top G \circ L \right) \left(G^\top G \circ L \right)^{-1},$$

Therefore, $\eta = -(G^\top G \circ L)^{-1} \left(G^\top \Theta \circ L \right) \mathbb{1}$.

3. Exploratory Insights

One step convergence for quadratics with condition number 1**:** Perhaps interestingly, it can be shown that updating the parameters with step sizes $\eta = -(G^\top G \circ L)^{-1} \left(G^\top \Theta \circ L \right) \mathbb{1}$ converges to the optima in a **single** step for a quadratic function $f(\theta) := \frac{1}{2}\theta^\top A \theta$, where A is positive-semi-definite and has condition number $\rho = 1$. This result holds irrespective of the number of dimensions d of $\theta \in \mathbb{R}^d$, and even with $N = 2$. We refer the readers to Appendix C.1 for details.

Connections to approximate Newton's method: The proposed approach uses gradient evaluations at various points on the loss surface to estimate the optima. From one perspective, it can be seen as obtaining the curvature information of the loss surface and using it to obtain the step-sizes. This is reminiscent of (quasi-) Newton's methods which use Hessian to estimate the loss surface and get the appropriate direction and the step-size for parameter update. In Appendix C.2, we discuss this connection is more details.

4. Empirical Analysis

In this section, we present an empirical comparison on various convex optimization benchmarks between the proposed approach and popular optimization methods. We call our proposed method Gradient Grouping (GG). Exact algorithm is presented in Appendix D.

Robustness to Hyper-parameter: We use several multi-class logistic regression datasets to evaluate the performance of our method, and the following baselines: stochastic gradient descent, Nesterov's accelerated gradient method [2], ADAM [7], Rmsprop [18], and a first order quasi-Newton method L-BFGS [8]. Figure 2 provides a comparison of the performance of our method against performances of the baselines optimized with different values of learning rate.

The plots in Figure 2 correspond to the most challenging setting for our method where **(a)** the number of available parallel resources is only 2 and **(b)** while GG was developed for the deterministic gradient, it is typically infeasible to compute the full-batch gradient. Therefore, we use GG as-is in the setting where function evaluations are stochastic due to mini-batches. Further, as the proposed method makes use of 2 resources in parallel as opposed to the baselines that use 1, we

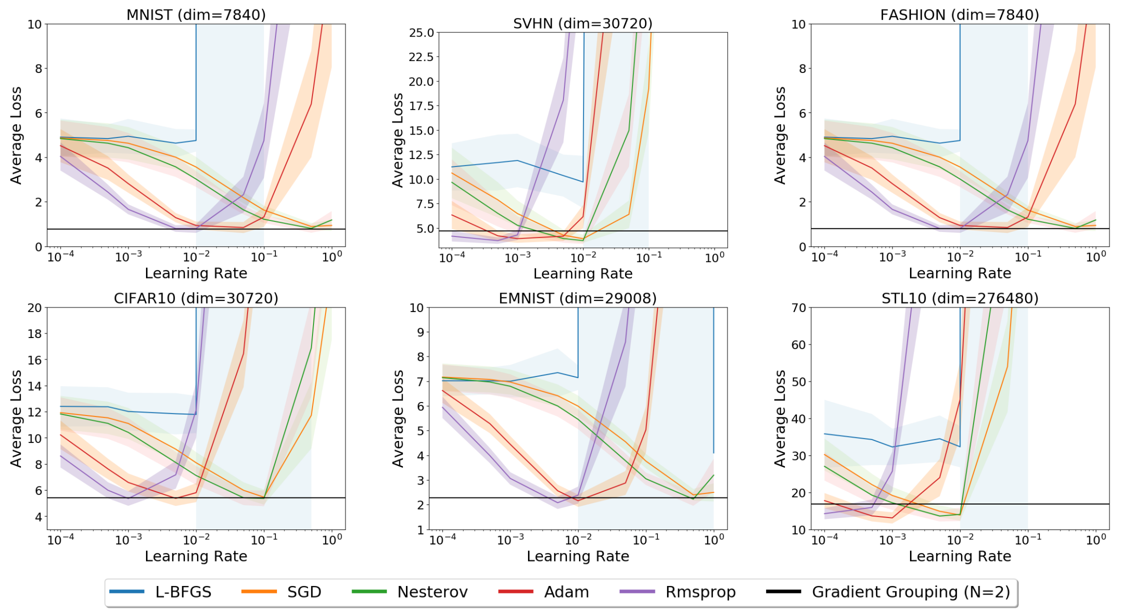

Figure 2: Title of the sub-plots mention the name of the dataset and the dimension of parameters θ used. The plots showcase sensitivity of the baselines to their learning rate hyper-parameter. Y-axis corresponds to the average loss over the entire training phase. The reference line (in black) corresponds to the performance of the proposed method obtained without performing any domain specific hyper-parameter tuning. Shaded regions correspond to standard deviation obtained using 10 trials.

consider a strict setup for a fair comparison by making use of only half the data per computation of gradient. This ensures that the proposed method neither uses more wall-time nor the total compute time. Specifically, GG uses a batch size of 32 for computing each gradient and the baselines use a batch-size of 64.

We notice that across all the domains, even without any hyper-parameter optimization, the proposed algorithm achieves nearly the same performance as that achieved by optimally tuned baselines. This showcases the advantage of leveraging parallel resources to compute gradients for *different* parameter vectors, even in the restricted case when only 2 gradients can be obtained.

In comparison, note that although there exists a learning-rate with which the baselines are able to achieve the same performance, the baselines are particularly sensitive to it and their performance deteriorates significantly when it is not set to the optimal value. Further, the optimal learning rates for these baselines vary across the domains, thereby necessitating a thorough hyper-parameter search.

Ablation: Impact of N: In problem setups, where more than 2 parallel resources are available, a natural question to ask is how well does the proposed method perform then? In Appendix F we present this ablation and showcase that while just 2 parallel gradients sufficed to provide acceleration in high-dimensional problems, our method further benefits when more resources are available.

5. Conclusion and Future Work

We presented a convex optimization algorithm that exploits the availability of parallel resources by computing only the first-order gradients of different parameters in parallel and using them in synergy to reach the optima. Through empirical results, we demonstrated that even with 2 gradients computed in parallel, acceleration similar to the best tuned baselines can be achieved. Further, using gradients in synergy achieves this accelerated performance without requiring any hyper-parameter tuning. Enjoying both low memory and computationally cost, it can be used for quickly optimizing large problems on devices with limited compute resources.

An interesting future direction would be to efficiently combine the proposed method that makes use of gradients computed in parallel, with methods that do curvature correction by making use of past gradients that are computed sequentially. Addressing this would allow bringing the best of both the paradigms. Another research question is to extend this approach to non-convex optimizations. The proposed approach relies upon extrapolating the gradients from N different points. If these points happen to be in different 'valleys', it is not clear how to optimally combine these gradients.

References

[1] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554, 2011.

[2] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[3] Sébastien Bubeck et al. Convex optimization: Algorithms and complexity. *Foundations and Trends® in Machine Learning*, 8(3-4):231–357, 2015.

[4] Lawrence Davis. Handbook of genetic algorithms. 1991.

[5] John R Dormand and Peter J Prince. A family of embedded runge-kutta formulae. *Journal of computational and applied mathematics*, 6(1):19–26, 1980.

[6] Nikolaus Hansen, Dirk V Arnold, and Anne Auger. Evolution strategies. 2015.

[7] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[8] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

[9] Agnes Lydia and Sagayaraj Francis. Adagrad—an optimizer for stochastic gradient descent. *Int. J. Inf. Comput. Sci*, 6(5):566–568, 2019.

[10] Niru Maheswaranathan, Luke Metz, George Tucker, Dami Choi, and Jascha Sohl-Dickstein. Guided evolutionary strategies: Augmenting random search with surrogate gradients. *arXiv preprint arXiv:1806.10230*, 2018.

[11] Elizabeth Million. The hadamard product. *Course Notes*, 3(6), 2007. URL http://buzzard.ups.edu/courses/2007spring/projects/million-paper.pdf.

[12] Jorge Nocedal and Stephen J Wright. Quasi-newton methods. *Numerical optimization*, pages 135–163, 2006.

[13] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[14] Jssai Schur. Bemerkungen zur theorie der beschränkten bilinearformen mit unendlich vielen veränderlichen. 1911.

[15] Damien Scieur, Alexandre dAspremont, and Francis Bach. Regularized nonlinear acceleration. In *Advances in Neural Information Processing Systems 29*, pages 712–720. 2016.

[16] Wei Shi, Qing Ling, Gang Wu, and Wotao Yin. Extra: An exact first-order algorithm for decentralized consensus optimization. *SIAM Journal on Optimization*, 25(2):944–966, 2015.

[17] Dirk Thierens. Scalability problems of simple genetic algorithms. *Evolutionary computation*, 7(4):331–352, 1999.

[18] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2): 26–31, 2012.

[19] Homer F Walker and Peng Ni. Anderson acceleration for fixed-point iterations. *SIAM Journal on Numerical Analysis*, 49(4):1715–1735, 2011.

[20] Wikipedia contributors. Schur product theorem — Wikipedia, the free encyclopedia, 2021. URL https://en.wikipedia.org/w/index.php?title=Schur_product_theorem&oldid=1016337944. [Online; accessed 25-May-2022].

[21] Qiang Yang, Yang Liu, Yong Cheng, Yan Kang, Tianjian Chen, and Han Yu. Federated learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 13(3):1–207, 2019.

[22] Jingzhao Zhang, Aryan Mokhtari, Suvrit Sra, and Ali Jadbabaie. Direct runge-kutta discretization achieves acceleration. In *Advances in neural information processing systems*, pages 3900–3909, 2018.

[23] Ruiliang Zhang and James Kwok. Asynchronous distributed admm for consensus optimization. In *International Conference on Machine Learning*, pages 1701–1709, 2014.

Supplementary Material

Contents

Appendix A. Detailed Related Work

The literature on convex optimization is vast and no effort is enough to thoroughly review it. Here, we briefly touch upon the popular paradigms in convex optimization that are related to our approach and discuss their differences from our method.

Federated Learning: In many use-cases, due to either large amounts of data, or for accessing data only at a local server, distributed optimization is of prime importance. In such a scenario, data is partitioned into multiple sub-sets and for a given parameter, gradient computation with respect to each sub-set is computed in a decentralized fashion. The primary challenge over here includes how to aggregate these gradients effectively [16, 21, 23]. The proposed method presents a complementary procedure, where gradient with respect to different set of *parameters* are used in synergy to update those parameters towards the optima.

Center of mass/gravity method: In this method, the center of mass (CoM) of a valid search space is computed and then used to partition the valid search space for narrowing down the space consisting the minima. This process is iteratively carried out to reduce the domain of search and obtain the final solution [3]. This method is similar to ours as we also iteratively use gradients at various points in the domain to narrow down to the region consisting the optima. However, calculating the center of mass is computationally expensive and makes the CoM method dimension dependent. Therefore, unlike our approach, this method cannot be scaled to large problems.

Genetic algorithms (GAs)/Evolutionary strategy (ES): Methods along this idea are for settings where computing the gradient is infeasible or computationally expensive. This method maintains a set of parameters, similar to our method, where the function value is evaluated. These evaluations are then used to compute an update direction for all the parameters in the set [4]. In several cases, this procedure is also combined with gradients. For example, in hyper-parameter optimization (like architecture search for neural networks) function evaluation is itself an optimization procedure, where the quality of a hyper-parameter is estimated based on the quality of parameters obtained after performing gradient-descent using those hyper-parameters [1]. Another line of work first uses ES as an approximate global search procedure and then runs gradient descent for fine-tuning with a hope to more accurately estimate the global minima [4]. To make such evolutionary strategies feasible in high-dimension space, [10] propose to only search in a meaningful sub-space of parameters that is obtained using recent history of (surrogate) gradients. These still requires a large population size of parameters.

Quasi-Newton methods: Newton's method is a second-order optimization method which requires pre-conditioning the gradient with the inverse of the Hessian matrix. Due to this operation, they scale poorly to problems with large dimensions. Quasi-Newton methods (like SR1, LBFGS, etc.) aim to use only first-order gradients to approximate Newton's method and trade-off computational complexity with convergence rates [12]. While this is more scale-able than Newton's method, they still require pre-multiplication of the gradient with a $d \times d$ matrix, which can be expensive for large scale problems. L-BFGS further trades off performance quality by keeping only the past k gradients that were obtained sequentially [8].

Conjugate gradients: In this update procedure, for any d dimensional space, d orthogonal vectors are generated (iteratively) and then line search is used to minimize the objective in each of those directions [3]. Our method can be seen to be doing approximate line search by estimating the minima along the direction of gradient using N separate gradient evaluations. This avoids the expensive computation required for both obtaining and iterating over d orthogonal directions, and doing linear search each time.

Accelerated methods: Accelerated optimization procedures like heavy-ball method, Nesterov's method, RMSprop, ADAM, AdaGrad, etc. leverage past gradients to accelerate search [7, 9]. They are also related to acceleration procedures for ordinary differential equations (ODE), like Runge-Kutta [5] and Andersons acceleration [19]. Optimizers built around these ODE techniques have also been developed [15, 22]. Notice that all these methods use past gradients and function values that were obtained *sequentially*. In contrast, our method uses multiple *parallel* gradient evaluations at any update and is thus complementary to these approaches. For instance, update directions obtained using any of the above approaches can be substituted in place of $\nabla f(\theta)$ in our framework to leverage parallel processing. We leave such combinations for future work.

Appendix B. Remaining Proofs

Property 1. *For any matrix $B \in \mathbb{R}^{N \times N}$,*

$$\sum_i \Lambda_i^\top B \Lambda_i = \frac{1}{N} B \circ L.$$

Proof

As Λ_i is a diagonal matrix consisting of M_i in the diagonal, ΛB corresponds to row broadcast using M_i and $B\Lambda$ corresponds to column broadcast using M_i. Therefore,

$$\begin{aligned}\sum_i \Lambda_i^\top B \Lambda_i &= \sum_i B \circ M_i M_i^\top \\ &= B \circ \sum_i M_i M_i^\top. \end{aligned} \tag{7}$$

In the above equations, $M_iM_i^\top$ corresponds to a matrix where $(N-1)^2/N^2$ is in the (i,i) cell of the matrix, and both row i and column i contain $-(N-1)/N^2$,

$$M_iM_i^\top = \begin{bmatrix} 1/N^2 & .. & -(N-1)/N^2 & ... & 1/N^2 \\ & & ... & & \\ -(N-1)/N^2 & .. & (N-1)^2/N^2 & ... & -(N-1)/N^2 \\ & & ... & & \\ 1/N^2 & .. & -(N-1)/N^2 & ... & 1/N^2 \end{bmatrix},$$

$\sum_i M_iM_i^\top$ would imply that every off diagonal entry is a summation of $N-2$ terms of $1/N^2$ and two terms of $-(N-1)/N^2$. This is because for any entry (i,j), where $i \neq j$, it has $-(N-1)/N^2$ either when it is $(N-1)^2/N^2$ in the cell (i,i) or in the cell (j,j). Therefore, each off-diagonal element will sum to $(N-2)/N^2 - 2(N-1)/N^2 = -1/N$. Similarly, each diagonal entry will be a sum $N-1$ terms of $1/N^2$ and one term of $(N-1)^2/N^2$, which is equal to $(N-1)/N^2 + (N-1)^2/N^2 = (N-1)/N$. Therefore,

$$\sum_i M_iM_i^\top = \frac{1}{N}\begin{bmatrix} N-1 & -1 & ... & -1 \\ -1 & N-1 & ... & -1 \\ & & ... & \\ -1 & -1 & ... & N-1 \end{bmatrix} = \frac{1}{N}L. \tag{8}$$

Using (8) in (7),

$$\sum_i \Lambda_i^\top B \Lambda_i = \frac{1}{N} B \circ L.$$

Appendix C. Exploratory Insights

In this section, we explore some properties of the proposed method for the fundamental case of quadratic functions, and draw some connections to prior methods.

C.1. One step convergence for quadratics with condition number $\rho = 1$

Perhaps interestingly, it can be shown that even with $N = 2$, GG can converge to the optima in a single step for a quadratic function $f(\theta) := \frac{1}{2}\theta^\top A\theta$, where A is positive-semi-definite and has condition number $\rho = 1$. This result holds irrespective of the number of dimensions d of $\theta \in \mathbb{R}^d$.

Recall from (1), the update rule in closed form can be expressed as,

$$\Theta_{t+1} \leftarrow \Theta_t + G\texttt{diag}(\eta),$$

where for the proposed GG,

$$\eta = -(G^\top G \circ L)^{-1}\left(G^\top\Theta \circ L\right)\mathbb{1}.$$

Further, as $\nabla f(\theta) = A\theta$, η can be expressed as,

$$\eta = -(G^\top G \circ L)^{-1}\left(G^\top\Theta \circ L\right)\mathbb{1} = -(\Theta^\top A^\top A\Theta \circ L)^{-1}\left(\Theta^\top A^\top\Theta \circ L\right)\mathbb{1}. \tag{9}$$

Now notice that as the condition number of A is one, there exists some c such that $A = cI$. To see this, For a quadratic function we can presume A to be symmetric because

$$\theta^\top A\theta = \frac{1}{2}\theta^\top(A + A^\top)\theta,$$

and so we can replace with A with a symmetric matrix $\bar{A} \coloneqq \frac{1}{2}(A + A^\top)$. Furthermore, convexity implies that the Hessian of $f(\theta)$ must be PSD, therefore A must be PSD.

Now as real-symmetric matrices are diagonalizable, using eigenvalue decomposition we get

$$A = Q\Lambda Q^\top,$$

where Q is an orthogonal matrix and Λ is a diagonal matrix consisting of the eigenvalues of A. Because the PSD condition ensures that all eigenvalues are nonnegative, and the condition number is 1, $(\text{max eigen value of } A)/((\text{min eigen value of } A)) = 1$. This implies that $\Lambda = cI$ for some constant c. Therefore,

$$A = Q\Lambda Q^\top = Q(cI)Q^\top = cQQ^\top = cI.$$

Therefore, (9) can be simplified as the following,

$$\eta = -(c^2\Theta^\top I^\top I\Theta \circ L)^{-1}\left(c\Theta^\top I^\top\Theta \circ L\right)\mathbb{1} = -\frac{1}{c}(\Theta^\top\Theta \circ L)^{-1}\left(\Theta^\top\Theta \circ L\right)\mathbb{1} = -\frac{1}{c}\mathbb{1}, \tag{10}$$

when the inverse exists. Here the inverse may not exist when the two gradients are exactly parallel (for e.g., the parameters in Θ are exactly the same). While this is unlikely in high dimensions, in Section 4 we discuss a simple practical trick to avoid this. Now, substituting the value of η from (10) in the update step,

$$\Theta_{t+1} = \Theta_t + G\texttt{diag}(\eta) = \Theta_t - A\Theta_t\texttt{diag}(\frac{1}{c}\mathbb{1}). \tag{11}$$

Finally, the result can be obtained by further simplifying (11),

$$\Theta_{t+1} = \Theta_t - cI\Theta_t\frac{1}{c} = \Theta_t - \Theta_t = 0.$$

As the minima of $f(\theta)$ is at $\theta = 0$, the update procedure finds the minima in **one** update (two gradient calls when $N = 2$). This is in contrast with the existing gradient based optimizers that can solve this in one step *if and only if* the step-size is chosen appropriately. Further, notice that the above result can be generalized for any $N > 1$, and also other functions f with condition number $\rho = 1$.

C.1.1. FOR SETTING WITH $\rho \neq 1$

The goal of this section is to understand the behavior of gradient grouping for the fundamental case of quadratics when $\rho \neq 1$. Particularly, in the following paragraphs we provide analysis using Eigen decomposition for one step update using the gradient grouping algorithm with $N = 2$. We consider $f(\theta) \coloneqq \frac{1}{2}\theta^\top A\theta$, where A is a symmetric positive definite matrix. As the optima of f is at 0, by looking at the norm of the parameters in Θ we can analyze the improvement towards the optima.

To do so, we first expand η's formula for $N = 2$. For brevity, we use g_1 and g_2 to represent the gradients at the two points θ_1 and θ_2. From Theorem 1,

$$\begin{aligned}
\eta &= -(G^\top G \circ L)^{-1} \left(G^\top \Theta \circ L\right) \mathbb{1} \\
&= -\begin{bmatrix} g_1^\top g_1 & -g_1^\top g_2 \\ -g_2^\top g_1 & g_2^\top g_2 \end{bmatrix}^{-1} \begin{bmatrix} g_1^\top \theta_1 & -g_1^\top \theta_2 \\ -g_2^\top \theta_1 & g_2^\top \theta_2 \end{bmatrix} \mathbb{1} \\
&\overset{(a)}{=} -\frac{1}{D} \begin{bmatrix} g_2^\top g_2 & g_1^\top g_2 \\ g_2^\top g_1 & g_1^\top g_1 \end{bmatrix} \begin{bmatrix} g_1^\top(\theta_1 - \theta_2) \\ -g_2^\top(\theta_1 - \theta_2) \end{bmatrix} \\
&= -\frac{1}{D} \begin{bmatrix} (g_2^\top g_2)(g_1^\top(\theta_1 - \theta_2)) - (g_1^\top g_2)(g_2^\top(\theta_1 - \theta_2)) \\ (g_2^\top g_1)(g_1^\top(\theta_1 - \theta_2)) - (g_1^\top g_1)(g_2^\top(\theta_1 - \theta_2)) \end{bmatrix} \\
&= -\frac{1}{D} \begin{bmatrix} ((g_2^\top g_2)g_1^\top - (g_1^\top g_2)g_2^\top)(\theta_1 - \theta_2) \\ ((g_2^\top g_1)g_1^\top - (g_1^\top g_1)g_2^\top)(\theta_1 - \theta_2) \end{bmatrix} \\
&= -\frac{1}{D} \begin{bmatrix} ((g_2^\top g_2)g_1 - (g_1^\top g_2)g_2)^\top A^{-1}(g_1 - g_2) \\ ((g_2^\top g_1)g_1 - (g_1^\top g_1)g_2)^\top A^{-1}(g_1 - g_2) \end{bmatrix} \\
&\overset{(b)}{=} -\frac{1}{D} \begin{bmatrix} ((\theta_2^\top A^\top A\theta_2)\theta_1^\top A^\top - (\theta_1^\top A^\top A\theta_2)\theta_2^\top A^\top)(\theta_1 - \theta_2) \\ ((\theta_2^\top A^\top A\theta_1)\theta_1^\top A^\top - (\theta_1^\top A^\top A\theta_1)\theta_2^\top A^\top)(\theta_1 - \theta_2) \end{bmatrix} \\
&= -\frac{1}{D} \begin{bmatrix} ((\theta_2^\top A^2\theta_2)\theta_1 - (\theta_1^\top A^2\theta_2)\theta_2)^\top A^\top(\theta_1 - \theta_2) \\ ((\theta_2^\top A^2\theta_1)\theta_1 - (\theta_1^\top A^2\theta_1)\theta_2)^\top A^\top(\theta_1 - \theta_2) \end{bmatrix},
\end{aligned}$$

where in (a) the notation D represents determinant when computing the inverse, and (b) follows from expanding the gradient of $f(\theta)$ and using the fact that A is a symmetric matrix.

As the labeling of parameters are arbitrary, without loss of generality let us consider only the first row (step-size for updating θ_1) to analyze one (of the two) parameters in the set. Using (1),

$$\widetilde{\theta_1} = \theta_1 - \frac{1}{D}((\theta_2^\top A^2\theta_2)\theta_1 - (\theta_1 A^2\theta_2)\theta_2)^\top A(\theta_1 - \theta_2)A\theta_1),$$

where we have used $\widetilde{\theta_1}$ to denote the value of θ_1 after one update. Now evaluating the norm of $\widetilde{\theta_1}$,

$$\begin{aligned}
||\widetilde{\theta_1}|| &= ||\theta_1 - \frac{1}{D}((\theta_2^\top A^2\theta_2)\theta_1 - (\theta_1 A^2\theta_2)\theta_2)^\top A(\theta_1 - \theta_2)A\theta_1)|| \\
&= \frac{1}{D}||(D)\theta_1 - ((\theta_2^\top A^2\theta_2)\theta_1 - (\theta_1 A^2\theta_2)\theta_2)^\top A(\theta_1 - \theta_2)A\theta_1)|| \\
&= \frac{1}{D}||((\theta_1^\top A^2\theta_1)(\theta_2^\top A^2\theta_2) - (\theta_1^\top A^2\theta_2)^2)\theta_1 - ((\theta_2^\top A^2\theta_2)\theta_1 - (\theta_1 A^2\theta_2)\theta_2)^\top A(\theta_1 - \theta_2)A\theta_1)|| \\
&= \frac{1}{D}||(\theta_1^\top A^2\theta_1)(\theta_2^\top A^2\theta_2)\theta_1 - (\theta_1^\top A^2\theta_2)^2\theta_1 - (\theta_2^\top A^2\theta_2)(\theta_1^\top A\theta_1)A\theta_1 \\
&\qquad + (\theta_2^\top A^2\theta_2)(\theta_1^\top A\theta_2)A\theta_1 + (\theta_1 A^2\theta_2)(\theta_2^\top A\theta_1)A\theta_1 - (\theta_1 A^2\theta_2)(\theta_2^\top A\theta_2)A\theta_1||
\end{aligned}$$

To simplify, we now regroup terms with the matching colors,

$$\begin{aligned}
||\widetilde{\theta_1}|| = \frac{1}{D}||((\theta_2^\top A^2\theta_2)\underbrace{((\theta_1^\top A^2\theta_1)I - (\theta_1^\top A\theta_1)A)\theta_1}_{\text{S1}} + (\theta_1^\top A^2\theta_2)\underbrace{((\theta_2^\top A\theta_1)A - (\theta_1^\top A^2\theta_2)I)\theta_1}_{\text{S2}} \\
\underbrace{((\theta_2^\top A^2\theta_2)(\theta_1^\top A\theta_2) - (\theta_1^\top A^2\theta_2)(\theta_2^\top A\theta_2))}_{\text{S3}} A\theta_1)||. \qquad (12)
\end{aligned}$$

Now we bound the behavior of S1, S2, and S3 in terms of the eigenvalues of A. For the following, we denote the Eigenvalues of A as λ_i and the corresponding Eigenvector v_i. Note that as A is positive definite, $\lambda_i > 0$. We decompose both θ_1, θ_2 in the Eigenbasis and call the coefficients associated with each basis vector as α_i, and β_i, respectively. That is, $\theta_1 = \sum_i \alpha_i v_i$ and $\theta_2 = \sum_i \beta_i v_i$. Under this expansion we get the following useful expressions:

$$\begin{aligned}
\theta_1^\top A\theta_1 &= (\sum_i \alpha_i v_i)^\top A(\sum_i \alpha_i v_i) = (\sum_i \alpha_i v_i)^\top(\sum_i \lambda_i\alpha_i v_i) = \sum_i \lambda_i \alpha_i^2\\
\theta_1^\top A\theta_2 &= (\sum_i \alpha_i v_i)^\top A(\sum_i \beta_i v_i) = (\sum_i \alpha_i v_i)^\top(\sum_i \lambda_i\beta_i v_i) = \sum_i \lambda_i \alpha_i\beta_i\\
\theta_1^\top A^2\theta_2 &= (\sum_i \alpha_i v_i)^\top A^2(\sum_i \beta_i v_i) = (\sum_i \alpha_i v_i)^\top(\sum_i \lambda_i^2\beta_i v_i) = \sum_i \lambda_i^2 \alpha_i\beta_i\\
\theta_1^\top A^2\theta_1 &= (\sum_i \alpha_i v_i)^\top A^2(\sum_i \alpha_i v_i) = (\sum_i \alpha_i v_i)^\top(\sum_i \lambda_i^2\alpha_i v_i) = \sum_i \lambda_i^2 \alpha_i^2.
\end{aligned}$$

Plugging in these expressions to expand S1,S2 and S3,

$$\begin{aligned}
||\mathrm{S1}|| &= ||((\theta_1^\top A^2\theta_1)I - (\theta_1^\top A\theta_1)A)\theta_1||\\
&= ||(\sum_i \alpha_i^2\lambda_i^2)(\sum_i \alpha_i v_i) - (\sum_i \alpha_i^2\lambda_i)(\sum_i \alpha_i\lambda_i v_i)||\\
&= ||\sum_i\sum_j(\alpha_i^2\alpha_j\lambda_i^2 - \alpha_i^2\alpha_j\lambda_i\lambda_j)v_j||\\
&= ||\sum_i\sum_j \alpha_i^2\alpha_j\lambda_i^2(1-\frac{\lambda_j}{\lambda_i})v_j||\\
&= ||\sum_i\sum_j \alpha_i^2\alpha_j\lambda_i^2(\frac{\lambda_j}{\lambda_i}-1)v_j||\\
&\le ||\sum_i\sum_j \alpha_i^2\alpha_j\lambda_i^2(\rho-1)v_j||\\
&= ||\sum_i\sum_j \alpha_i^2\alpha_j\lambda_i^2 v_j||(\rho-1),
\end{aligned} \tag{13}$$

where $\rho := \frac{\lambda_{\max}}{\lambda_{\min}}$. Similarly,

$$\begin{aligned}
||\mathrm{S2}|| &= ||((\theta_2^\top A\theta_1)A - (\theta_1^\top A^2\theta_2)I)\theta_1||\\
&= ||(\sum_i \alpha_i\beta_i\lambda_i)(\sum_i \alpha_i\lambda_i v_i) - (\sum_i \alpha_i\beta_i\lambda_i^2)(\sum_i \alpha_i v_i)||\\
&= ||\sum_i\sum_j \beta_i\alpha_i\alpha_j\lambda_i^2(\frac{\lambda_j}{\lambda_i}-1)v_j||\\
&\le ||\sum_i\sum_j \beta_i\alpha_i\alpha_j\lambda_i^2 v_j||(\rho-1).
\end{aligned} \tag{14}$$

Similarly,

$$\begin{aligned}
||\mathrm{S3}|| &= ||(\theta_2^\top A^2\theta_2)(\theta_1^\top A\theta_2) - (\theta_1^\top A^2\theta_2)(\theta_2^\top A\theta_2)|| \\
&= ||(\sum_i \beta_i^2\lambda_i^2)(\sum_i \beta_i\alpha_i\lambda_i) - (\sum_i \alpha_i\beta_i\lambda_i^2)(\sum_i \beta_i^2\lambda_i)|| \\
&= ||\sum_i\sum_j \beta_i^2\beta_j\alpha_j(\lambda_i^2\lambda_j - \lambda_i\lambda_j^2)|| \\
&= ||\sum_i\sum_j \beta_i^2\beta_j\alpha_j\lambda_i^2\lambda_j(1-\frac{\lambda_j}{\lambda_i})|| \\
&\leq ||\sum_i\sum_j \beta_i^2\beta_j\alpha_j\lambda_i^2\lambda_j||(\rho-1). \qquad (15)
\end{aligned}$$

Using (13), (14), and (15) in (12), observe that distance of $||\theta_1||$ from the optimum can be bounded in terms proportional to $(\rho - 1)$. Notice that similar to discussions in C.1, this also shows that if $\rho = 1$ we reach the optimum in 1 step, and that if $\rho = 1 + \delta$, for some small δ, the parameters can approach the optima quickly. Extending this analysis to characterize the evolution of parameters for the entire sequence of updates remain an interesting future direction.

C.2. Gradient post-conditioning (instead of pre-conditioning)

The proposed Gradient Grouping approach uses gradient evaluations at various points on the loss surface to estimate the optima. From one perspective, it can be seen as obtaining the curvature information of the loss surface and using it to obtain the step-sizes. This is reminiscent of (quasi-) Newton's methods which use Hessian to estimate the loss surface and get the appropriate direction and the step-size for parameter update.

Given a quadratic of the form $f(\theta) := \frac{1}{2}\theta^\top A\theta$, the first and second order derivatives are $\nabla f(\theta) = A\theta$, and $\nabla^2 f(\theta) = A$, respectively. According to the Newton method's update rule,

$$\theta_{t+1} \leftarrow \theta_t - A^{-1}g_t.$$

However, direct pre-multiplication with A^{-1} requires both matrix inversion and a $d \times d$ matrix-vector multiplication, which can be computationally expensive. Several prior methods have studied quasi-Newton methods to minimize the computational requirements [2].

In the following, we discuss a complementary method that draws some connections to the proposed GG method. Let the Newton method's update for a set of points $\Theta \in \mathbb{R}^{d\times N}$ using $G \in \mathbb{R}^{d\times N}$ be,

$$\Theta_{t+1} \leftarrow \Theta_t - A^{-1}G_t, \qquad (16)$$

where $A^{-1}G_t \in \mathbb{R}^{d\times d} \times \mathbb{R}^{d\times N}$. To reduce the computational cost, instead of pre-multiplying with A^{-1}, we aim to post multiply G with another matrix $B \in \mathbb{R}^{N\times N}$, such that,

$$A^{-1}G_t \approx G_tB.$$

To do so, we can estimate B such that,

$$B = \underset{B}{\mathrm{argmin}}\|A^{-1}G_t - G_tB\|^2.$$

Solving for the above equation,

$$\frac{\partial \|A^{-1}G_t - G_t B\|^2}{\partial B} = G_t^\top (A^{-1}G_t - G_t B) = G_t^\top A^{-1} G_t - G_t^\top G_t B.$$

Equating the derivative to 0,

$$\begin{aligned} G_t^\top G_t B &= G_t^\top A^{-1} G_t, \\ B &= (G_t^\top G_t)^{-1}(G_t^\top A^{-1} G_t). \end{aligned} \tag{17}$$

Now as $\nabla f(\theta) = A\theta$, we know $A^{-1}\nabla f(\theta) = \theta$. Substituting this in (17),

$$B = (G_t^\top G_t)^{-1}(G_t^\top \Theta_t). \tag{18}$$

Perhaps interestingly, B in (18) shares a striking resemblance with η from Theorem 1. Particularly, as Θ_t is updated by $A^{-1}G_t \approx G_t B$ in (16), B plays a role similar to η in Line 9 of Algorithm 1. Nonetheless, there are important differences. For instance, B does not have the Laplacian and the product with the ones vector. This makes B a full-matrix, unlike η.

Notice that while (18) is useful to draw some connections with GG, it is not practically useful. Unlike how η was obtained from Theorem 1, B was obtained by enforcing a specific quadratic structure, which is crucial for the above steps. For instance, if $f(\theta) := \theta^\top A \theta + C^\top \theta$, where C is some vector, then B will depend on additional terms dependent on C which we do not typically have access to.

Appendix D. Algorithm

In this section, we discuss the proposed algorithm. As we maintain a group of parameters that compute gradients in parallel and use it to help each other obtain a common goal, we call our approach *Gradient Grouping*. A pseudo code is presented in Algorithm 1.

Algorithm 1: Gradient Grouping

```
Input $N \in \mathbb{N}_{\geq 2}$                                        ▷ Parameter set size
Randomly Initialize $\Theta \in \mathbb{R}^{d \times N}$                ▷ Parameter set
Zero Initialize $G \in \mathbb{R}^{d \times N}$                         ▷ Gradient set
$L = N \cdot I - \mathbb{1}\mathbb{1}^\top$
for $t = 1, 2, 3, ...$ do
    for $idx = 1, 2, ..., N$ (in parallel) do
        $G_t[:, idx] = \nabla f(\Theta_t[:, idx])$                      ▷ Compute gradients
    end
    $\eta_t = -({G_t}^\top G_t \circ L)^{-1} \left({G_t}^\top \Theta_t \circ L\right) \mathbb{1}$.   ▷ Lemma 1.
    $\Theta_{t+1} \leftarrow \Theta_t + G_t \mathtt{diag}(\eta_t)$        ▷ Update parameters
end
```

In Line 1, the number of parameters, N, to be considered in parallel is taken as the input. In Lines 2 and 3, the parameter matrix and the gradient matrix are initialized. As the Laplacian matrix, L, will always be fixed during updates, Line 4 creates and stores it beforehand. Lines 5 to

9 correspond to the main optimization loop. To avoid overloading the notation, we use super-script t to denote the values in the t^{th} iteration of optimization. In Lines 6 and 7, gradients are evaluated for all the N parameters in parallel. Using Lemma 1, the step-sizes are obtained in Line 8. Finally, in Line 9, the parameters are updated using the gradients and the step sizes obtained in the Line 8.

D.1. Practical Modifications

Notice that the inverse of $(G^\top G \circ L)$ may not exist if the gradients are parallel (for e.g., Θ contains exactly the same parameters, or if the parameters are different but the gradients are facing each other, etc.) To ensure that the inverse always exists in practice, we start all the parameters from a different initialization. To make this condition also holds during the learning process, we multiply η with a scaling factor $0 < \alpha < 1$, such that, $\eta \leftarrow \alpha\eta$. This ensures that even when the extrapolated gradients are intersecting, the parameters will never be at the same point after an update. More generally, we ensure invertibility of $(G^\top G \circ L)$ by clipping its minimum Eigen value to a small positive constant ϵ. As the $(G^\top G \circ L)$ matrix belongs to $\mathbb{R}^{N\times N}$, the computation cost of this operation is negligible. Specifically, leveraging the Eigen value decomposition, let

$$G^\top G \circ L = V^\top \Lambda V.$$

Note that $G^\top G$ is PSD and it is known that a Laplacian matrix L is also PSD. Now using Schur's theorem, which asserts that the Hadamard product of two PSD matrices is also a PSD matrix, it can be established that $G^\top G \circ L$ is also a PSD matrix [11, Theorem 3.4], [14, 20]. To convert $G^\top G \circ L$ into a PD matrix, we create a $\Lambda^+ := \texttt{clip}(\Lambda, \min = \epsilon, \max = \infty)$ and use $V^\top \Lambda^+ V$ instead of $G^\top G \circ L$ every time. As the $G^\top G \circ L$ matrix belongs to $\mathbb{R}^{N\times N}$, where $N < 10$ for all our experiments, the computation cost of this operation is negligible.

The two practical modifications to our method introduced the hyper-parameters: α and ϵ. In our experiments, we keep $\alpha = 0.9$ and $\epsilon = 1e - 4$ throughout and perform no problem specific hyper-parameter tuning.

D.2. Computational and Memory Complexity

An important property of our method is that it is cheap in both computational and memory requirements. Notice that $(G^\top G \circ L)$ is a $\mathbb{R}^{N\times N}$ matrix, irrespective of the number of dimensions d of the parameters space. As N can be as low as 2, and in general $N \ll d$, computational cost of inverting $(G^\top G \circ L)$ is negligible. Apart from that, as each processor only stores one copy of the parameter θ and its corresponding gradient $\nabla f(\theta)$, its memory and computational cost is in the same order as that of gradient descent. Further, the proposed method leverages curvature information by using the N individual parameters and their respective first-order gradients, and thus second-order derivatives, $\nabla^2 f(\theta)$, are never required to be computed.

Appendix E. Experimental Details

Implementations of the baseline algorithms: L-BFGS, SGD, Nesterov's method, Adam, and Rmsprop were based on the default routines available in PyTorch [13]. For the experiments, convexity was ensured using a single layer neural network implementation in PyTorch [13] with softmax classification loss.

The following datasets were used to report the empirical results.

Dataset	Input dimensions
CIFAR[1]	30720
MNIST[2]	7840
Extended-MNIST [3]	29008
STL10[4]	276480
SVHN[5]	30720
Fashion-MNIST[6]	7840

For Figures 2 and 3, each hyper-parameter setting was run till 100 epochs. In total 10 different seeds were used for each hyper-parameter setting to get the standard error. The authors had shared access to a computing cluster, consisting of 50 compute nodes with 28 cores each, which was used to run all the experiments.

1. https://www.cs.toronto.edu/ kriz/cifar.html
2. http://yann.lecun.com/exdb/mnist/
3. https://www.westernsydney.edu.au/bens/home/reproducible_research/emnist
4. https://cs.stanford.edu/ acoates/stl10/
5. http://ufldl.stanford.edu/housenumbers/
6. https://github.com/zalandoresearch/fashion-mnist

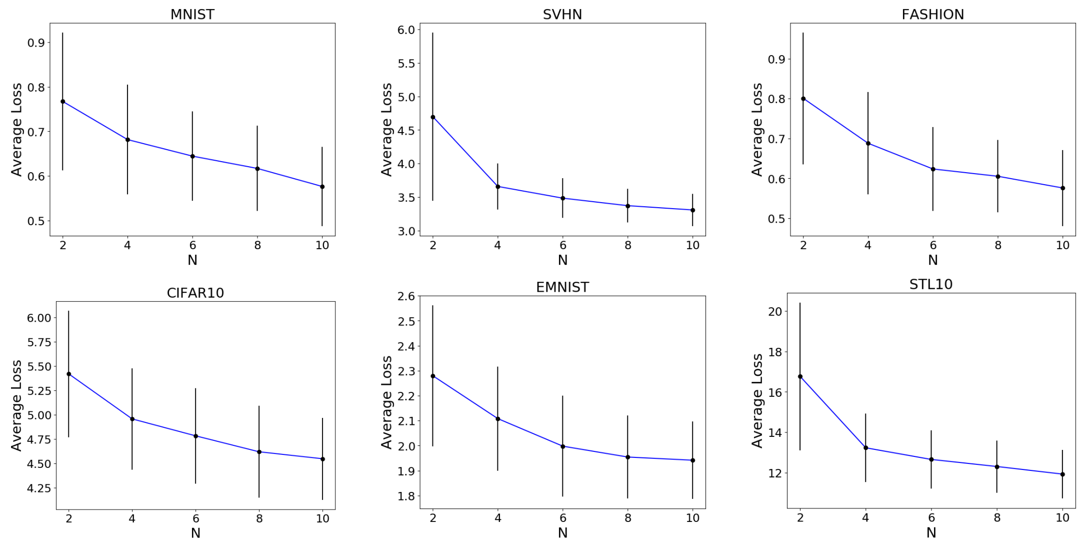

Figure 3: Performance improvement of the proposed Gradient Grouping algorithm for all the problems as the number of available parallel resources increase. The error bars correspond to standard deviation obtained using 10 trials.

Appendix F. Ablation Study

Impact of N: In problem setups, where more than 2 parallel resources are available, a natural question to ask is how well does the proposed method perform then? To answer this question, we conducted experiments with $N = (2, 4, 6, 8, 10)$ for all of the previous domains. The results are presented in Figure 3. We observe a consistent improvement in performance as the number of available resources increase. This showcases that while just 2 parallel gradients sufficed to provide acceleration in high-dimensional problems, our method further benefits when more resources are available.