ISSN 2281-4299

DIPARTIMENTO DI INGEGNERIA INFORMATICA
AUTOMATICA E GESTIONALE ANTONIO RUBERTI

SAPIENZA
UNIVERSITÀ DI ROMA

Block Layer decomposition applied to a watchdog controlled minibatch algorithm

Ilaria Ciocci
Corrado Coppola
Laura Palagi
Lorenzo Papa

Technical Report n. 02, 2024

Block Layer decomposition applied to a watchdog controlled minibatch algorithm

Ilaria Ciocci[1], Corrado Coppola[1], Laura Palagi[1], and Lorenzo Papa[1]

[1]Department of Computer, Control and Management Engineering, Sapienza University of Rome, Rome, Italy

Abstract

In this paper, we introduce a decomposed version of the CMA Light algorithm, leveraging block decomposition over variables to enhance computational efficiency by significantly reducing computational time while maintaining satisfactory performance. Our approach dynamically selects the blocks of variables to be updated at each iteration, based on both training performance conditions and architectural importance heuristics. Numerical results demonstrate that this strategy achieves a favorable trade-off between substantially reducing the computational cost while maintaining sufficient accuracy. This makes it a suitable and robust alternative in application where high precision is not essential or computational resources are limited.

Keywords: decomposition methods; block decomposition; computational efficiency

1 Introduction

In this paper, we consider the unconstrained finite-sum optimization problem:

$$\min_{w\in\mathbb{R}^n} \quad f(w) = \sum_{i=1}^{P} f_i(w) \tag{1}$$

where we assume that f_i: $\mathbb{R}^n \to \mathbb{R}, i = 1, \ldots, P$ are continuously differentiable and possibly non-convex functions. Problem (1) is a well-known minimization problem that arises in many applications where the number of components P is usually large, such as in Deep Learning [5].

One of the most important applications requiring solving a large-scale finite sum optimization problem is the training of a machine learning model. Here, the loss function f represents the regularized empirical risk, which measures the distance between the network prediction $\hat{y}_i$ and the real output y_i. The computational cost per iteration to solve Problem (1) depends on the size of the dataset P, which is typically very large when dealing with Deep Neural Networks. Consequently, the optimization problem may become too expensive to be tackled using traditional full-batch methods, which require the computation of the entire gradient $\nabla f(x)$. Furthermore, the number of variables n of the weight optimization problem becomes extremely large when using deep architectures, making the use of standard first-order methods less effective and the use of second-order methods practically infeasible. As a consequence, alternative strategies such as mini-batch methods have been developed to tackle this problem by exploiting the additive structure of the objective function. These methods perform each weight update using only a small set of samples p^k, being particularly efficient and suitable in Big Data applications, where the dataset is composed of a large number of samples P. Mini-batch methods can be classified as deterministic or stochastic according to the selection strategy of the samples used to estimate the gradient. In deterministic methods, such as the Incremental Gradient (IG) [2, 3], the samples are processed at each iteration in a specific order, unchanged over the epochs. The most common stochastic algorithm is the Stochastic Gradient Descent (SGD) [18], where at each iteration the samples p^k are selected randomly using a with-replacement rule, potentially excluding some of the P samples during an epoch. Another broadly used stochastic method is Random Reshuffling (RR), where the samples p^k are randomly selected without replacement, shuffling the order of the samples and obtaining a new permutation at the beginning of each epoch. This strategy usually performs better than SGD [16], as the use of permutations allows to process the samples more equally, ensuring that each component of the objective function is processed once per epoch. On the other hand, without-replacement sampling introduces correlations among the sampled gradients, making the convergence analysis almost impossible without introducing further and more complex assumptions [21, 11].

A relevant version of the IG method is the incremental aggregated gradient (IAG) method [4], in which the gradient is computed incrementally and aggregated with the m most recent gradients previously computed and stored in memory. Although this method may require an excessive amount of memory when the number of components m is large, it provides an efficient stopping criterion (norm of the approximated gradient under a certain

threshold) and leads to convergence for fixed and sufficiently small stepsizes thanks to a more accurate estimate. A randomized variant of the IAG method is the Stochastic Aggregated Gradient (SAG) [19]. This method is identical to IAG, except for the selection rule of p^k, which is random in this case. SAG presents some advantages over IAG, as the possibility of allowing larger stepsizes, leading to faster convergence and better performance [20].
The most widely used state-of-the-art algorithms for training Neural Networks are the Adaptive Gradient methods where the estimate of the gradient is improved by using the gradient estimates computed in the previous iterations, such as Adam and Adamax [13], Nadam [7], Adadelta [22] and Adagrad [8].
Even though mini-batch methods allow to deal with problems where the number of data P is very large, they are still affected by the number of variables n, which is usually very large when using deep architectures. A possible solution consists in using decomposition methods, which break the optimization problem into separate subproblems of lower dimension that are easier to solve and may present a special structure in the remaining variables. These methods only update a subset of the whole variables at each iteration while keeping the remaining fixed, being particularly suitable when dealing with large-scale problems.
Decomposition methods are typically divided into sequential methods and parallel methods, depending on the scheme used to update the variables. One of the most common sequential methods is the Gauss-Seidel, in which the blocks of variables are updated in a specific order, and convergence can be proven under suitable assumptions [10, 9]. Parallel methods update different blocks of variables simultaneously and independently, obtaining the subsequent point by using some suitable rule to combine the different updates.
Decomposition methods have gained increasing attention for their ability to exploit parallel computation. When dealing with big data, the size of problem may be so large that decomposing into smaller and more manageable subproblems is essential. In this context, parallel and distributed coordinate descent methods have been developed to leverage modern parallel computing architectures [1, 15, 17].

2 Minibatch methods: an overview

Minibatch methods offer a good trade-off between the per-iteration cost and per-iteration improvement by leveraging the additive structure of the objective function. In fact, these methods only use a small subset of the P samples to perform a single update, i.e., the search direction is computed using only a subset of the gradient of $f(w)$. Without loss of generality, we will denote the elements of this subset of samples $\mathcal{I}_p \subset \{1, \dots, P\}$ with a single term p, therefore

$$\nabla f_p(w) = \sum_{h_i \in \mathcal{I}_p} \nabla f_{h_i}(w)$$

where $h_i, i = 1, \dots, P$ represents a permutation of the indexes $\{1, \dots, P\}$. In Incremental Gradient (IG) methods, this permutation is deterministic and known a-priori; instead, in Random Reshuffling (RR) methods the permutation is selected randomly at each iteration. The inner iterations of a mini-batch method are obtained by moving from the current point using only a small set of samples to compute the descent direction, leading to reduced cost

per iteration and faster convergence. For the sake of simplicity, we will consider the case in which only one sample at a time is processed.
Starting from the initial point $w_0^k = w^k$ at the beginning of the epoch k, and given a permutation $I^k = \{h_1^k \dots h_P^k\}$ of $\{1, \dots, P\}$, the inner iterations of a mini-batch method update the iteration by using an estimate of the gradient $\tilde{d}_i^k$, as follows: for $i = 1, ..., P$

$$\widetilde{w}_i^k = \widetilde{w}_{i-1}^k + \zeta^k \tilde{d}_i^k \qquad \text{with} \quad \tilde{d}_i^k = -\nabla f_{h_i^k}(\widetilde{w}_{i-1}^k)$$

where $\zeta^k > 0$ denotes the stepsize which is fixed during the course of an epoch. Consequently, starting from an initial point $\widetilde{w}_0^k$, the point obtained at the end of an epoch, which is the result of an outer iteration of a mini-batch method, can be expressed as:

$$\widetilde{w}_P^k = \widetilde{w}_0^k - \zeta^k \sum_{i=1}^{P} \nabla f_{h_i}(\widetilde{w}_{i-1}^k)$$

where $h_i, i = 1, \dots, P$ represents the permutation of the indexes $\{1, \dots, P\}$.

3 CMA and CMALight

Controlled Minibatch Algorithm (CMA) is an algorithmic framework developed in [14], that leverages mini-batch iterations and batch derivative-free linesearch procedures. This approach allows for convergence under weak and standard assumptions while maintaining good numerical performance. CMA is an eased-controlled modification of Random Reshuffling/Incremental Gradient schemes, and its underlying idea is to perturbe as less as possible the iterations of a RR/IG gradient method. The acceptance of the point produced by these outer iterations is determined using a watchdog rule, that checks whether a sufficient decrease of the objective function has been attained during the course of an epoch. If this condition is not met, either the stepsize is adjusted or, if needed, an Extrapolation Derivative-Free Linesearch (EDFL) that guarantees convergence is performed. The control over the update of the stepsize prevents it from decreasing too quickly, thereby slowing convergence. While this algorithm requires an additional computational effort due to the evaluation of the objective function $f(w)$, which is necessary for the watchdog condition, its global convergence can be proven under the sole assumptions of coerciveness of $f(w)$ and Lipschitz continuity of the gradient components $\nabla f_i(w)$.
An outer iteration of the CMA scheme is defined as

$$w^{k+1} = w^k + \alpha^k d^k,$$

where d^k is the direction obtained from the inner iterations of a mini-batch method over an epoch (i.e. $d^k = -\sum_{i=1}^{P} \nabla f_{h_i^k}(\tilde{w}_{i-1})$) and the stepsize α^k is selected as follows:

- $\alpha^k = \zeta^k$ if the tentative point $\widetilde{w}_P$ produced in the outer iterations of a mini-batch method is accepted
- $\alpha^k = 0$ to restart from w^k, if the produced point w^{k+1} would be outside the Level set $\mathcal{L}(w_0)$

- $\alpha^k > \zeta^k$ to extrapolate along d^k

When performed, the EDFL procedure computes the stepsize α^k used in the outer iteration of CMA. This procedure checks, without using first-order information, if a sufficient decrease condition is satisfied. If this condition is met, the stepsize α is iteratively increased as long as the objective function decreases sufficiently. Each of these steps requires an evaluation of the entire objective function, making this procedure computationally expensive.

A modified version of CMA, aimed at improving its efficiency, was presented in [6]. The main bottleneck of CMA is the need of computing at least one function evaluation on the whole dataset per iteration, a task that may become computationally too heavy, especially when the number of samples P is large. This new algorithm, referred to as CMA Light, addresses this issue by exploiting the sum of the batch losses produced by a mini-batch method as an estimate of the objective function $f(w^k)$ to check the sufficient decrease condition. The estimate of the objective function is computed in the following way:

$$\tilde{f}^k = \sum_{i=1}^{P} f_i(\tilde{w}_{i-1}^k) = f_1(\tilde{w}_0^k) + f_2(\tilde{w}_1^k) + \cdots + f_P(\tilde{w}_{P-1}^k)$$

The EDFL was also modified to further reduce the number of function evaluations. This revised version, referred to as EDFL Light, is reported in Algorithm 1.

Algorithm 1 Extrapolation Derivative-Free Linesearch (`EDFL Light`)

Input $(\tilde{f}^k, w^k, d^k, \zeta^k; \gamma, \delta)$: $w^k \in \mathbb{R}^n, d^k \in \mathbb{R}^n, \zeta^k > 0$, $\gamma \in (0,1), \delta \in (0,1)$
Set $j = 0, \alpha = \zeta^k, f_j = \tilde{f}^k$
if $\tilde{f}^k > f(w^k) - \gamma\alpha\|d^k\|^2$ **then**
    Set $\alpha^k = 0$ and **return**
end if
while $f(w^k + (\alpha/\delta)d^k) \leq \min\{f(w^k) - \gamma\alpha\|d^k\|^2, f_j\}$ **do**
    Set $f_{j+1} = f(w^k + (\alpha/\delta)d^k)$
    Set $j = j + 1$
    Set $\alpha = \alpha/\delta$
end while
Set $\alpha^k = \alpha$ and **return**
Output: $\alpha^k, f(w^k + \alpha^k d^k)$

Global convergence of CMA Light was proven under the same assumption of coerciveness of $f(w)$ and Lipschitz-smoothness of $\nabla f_i(w)$, with the additional hypothesis of non-negativity and coerciveness of the components $f_p, p = 1, \ldots, P$.

4 A decomposition version of CMALight

In this section we introduce a decomposed version of CMA Light, in which at each iteration only a subset of the variables is updated, while the others are kept fixed at their current value. To leverage the additive structure of the objective function, we embed

the decomposition scheme into the minibatch framework, exploiting both sample-wise decomposition and block-wise decomposition. In other words, at each inner iteration, a subset of variables is updated using only a mini-batch.

Notation: For the sake of clearness, we state the notation used throughout this section. $w \in \mathbb{R}^n$ is the vector of all the network parameters, $w_\ell \in \mathbb{R}^{|\ell|}$ is the subvector of the weights belonging to layer $\ell \in \{1, \dots, L\}$. Notice that we use the term *layer* in a broad sense, denoting any subset of network parameters composing a block of variables. The set of all the network layers, denoted by $\mathcal{L} = \{1, \dots, L\}$, is a generic partition of the network parameters. Therefore $\sum_{\ell=1}^{L} |w_\ell| = n$.
$\nabla_w f_i \in \mathbb{R}^n$ represents the estimate of the gradient considering only the samples belonging to batch i, instead of all the P samples. We denote with $\nabla_{w_\ell} f_i \in \mathbb{R}^{|\ell|}$ the partial derivative computed with respect to the block of variables w_ℓ. We denote with $[\cdot]_\ell$ the vector where all the components are set to zero except for those of the ℓ-th layer. Consequently, $\nabla_w f_i$ can be expressed as $\sum_{\ell=1}^{L} [\nabla_{w_\ell} f_i]_\ell$.
In our proposed scheme, we consider the possibility of updating only a subset of the variables at each iteration, denoting the set of layers that are updated at iteration k with $\widetilde{\mathcal{L}}^k$. Consequently, the direction d^k is evaluated using only the gradient computed with respect to the variables $w_\ell, \ell \in \widetilde{\mathcal{L}}^k$. The set of the layers not updated at epoch k is denoted as $\bar{\mathcal{L}}^k = \mathcal{L} \setminus \widetilde{\mathcal{L}}^k$.

The aim of this strategy is to reduce the computational cost of the optimization process by mitigating not only the issues caused by a large number of samples P through mini-batch methods, but also the difficulties posed by a huge number of variables n through variable decomposition. In order to do that, we replace the standard inner iterations of a mini-batch method with a Decomposed Inner Cycle, reported in Algorithm 2, where only a subset of variables is updated at each epoch. During each inner iteration of the Decomposed Inner Cycle, this subset of variables $w_\ell, \ell \in \widetilde{\mathcal{L}}^k$ is iteratively updated using each time a mini-batch of samples.

Algorithm 2 `Decomposed Inner Cycle`

Input: w^k, ζ^k, $\widetilde{\mathcal{L}}^k$
Set $w_0^k = w^k$
Let $I^k = \{h_1^k \dots h_P^k\}$ be a permutation of $\{1, \dots P\}$
for $i = 1, \dots, P$ **do**
    $\widetilde{f}_i^k = f_{h_i}(w_{i-1}^k)$
    $\widetilde{d}_i^k = -\sum_{\ell \in \widetilde{\mathcal{L}}^k} [\nabla_{w_\ell} f_{h_i}^k(w_{i-1}^k)]_\ell$
    $w_i^k = w_{i-1}^k + \zeta^k \widetilde{d}_i^k$
end for
Output $w^k = w_P^k$, $d^k = \sum_{i=1}^{P} \widetilde{d}_i^k$ and $\widetilde{f}^k = \sum_{i=1}^{P} \widetilde{f}_i$

This approach aims to reduce the computational effort of the optimization process by alleviating the gradient evaluation, which is its heaviest computational task. By updating only a subset of the weights, we reduce the number of variables by considering some of them

fixed, meaning that the corresponding gradient components will be set to zero. Differently from a traditional decomposition method, we do not update the variables in a sequential order or in parallel. Instead, we dynamically select the blocks of variables to be updated at each iteration, while leaving the others unchanged. This can be considered as a training method that leverages the decomposition over variables to enhance computational efficiency performances. At each epoch, after having executed the Decomposed Inner Cycle, we add some additional controls to determine whether it is beneficial to deactivate or reactivate some blocks of variables. The selection of the parameters to be updated at each epoch aims to reduce the computational effort by considering only the most relevant variables at that step of the training procedure. When a block is active, its variables are updated during the optimization step of the current epoch. Instead, when convenient, some blocks of variables are left unchanged during one or more epochs, reducing the dimensionality of the optimization problem and decreasing its computational time.
Specifically, at each epoch we consider one candidate block (denoted as b^-) for deactivation and another candidate block (denoted as b^+) for reactivation. These candidate blocks are determined based on an architectural importance heuristic, which assesses the relevance of each block within the neural network architecture considering its impact on both computational time and accuracy. The process used to identify these candidate blocks is discussed more in detail later. The underlying idea is that if removing a block has minimal impact on the accuracy loss while leading to a significant computational gain, it should be considered as a candidate for deactivation. Conversely, if deactivating a block caused a substantial decrease in accuracy without significant reduction in computational time, it should not be deactivated; rather, it should be a prime candidate for reactivation. These considerations will be reflected in an ordered list prioritizing blocks considering their overall importance, based on their impact on accuracy and computational time.
This decision on whether to deactivate the candidate block b^- or not, is based on a certain deactivation threshold, computed as a function of the norm of the parameter updates of the currently active blocks. If the norm of the update of block b^- is below this threshold, then it can be assumed that these parameters do not play a major role at that training stage and are not significantly contributing to the current training iteration. Therefore, the corresponding variables can be deactivated in order to reduce the computational time. Subsequently, to avoid an excessive loss in accuracy, we compute the validation accuracy at each iteration to determine whether some variables should be reactivated. We use a counter to keep track of the number of consecutive epochs without sufficient improvement in the validation accuracy. If there is no sufficient improvement for a certain number of consecutive epochs (defined by the *patience* parameter), the current set of active blocks may not be sufficient for an effective training. Consequently, the candidate block b^+ is reactivated. This possibility of deactivating and reactivating blocks of parameters based on conditions on the training performance allows to find a good trade-off between reducing the computational cost and maintaining accuracy.

For the implementation of the proposed scheme, we considered Residual Neural Networks (ResNets), whose architecture is briefly described in Section 5.1. To select the blocks of parameters, we made use of the inherent division of ResNets into four main blocks (or stages, as commonly referred to in literature), which do not include the initial and final

layers. These layers remain always active and are excluded from the deactivation process, as their parameters are essential for processing input data and classifying instances. The importance order of the blocks was established heuristically, by running various ablation tests on the ResNet18 architecture. In each test, we trained the network while removing one block of parameters during the entire training procedure. Later, we observed the resulting values in terms of computational time and value of accuracy to rank the blocks in ascending importance order, based on the trade-off between computational time gain and reduction in test accuracy.
Given $\mathcal{L}$, which represents the set of all the network blocks ordered by their importance, we initialized the sets $\widetilde{\mathcal{L}}^0 = \mathcal{L}$ and $\bar{\mathcal{L}}^0 = \emptyset$, updating all the network parameters for a certain number of epochs determined by the parameter $\bar{k}$. After a number $\bar{k}$ of epochs, in which all the network parameters are updated, we start to check whether it is convenient to deactivate or reactivate some blocks of parameters using the scheme reported in Algorithm 3.

Algorithm 3 `Deactivation/activation scheme over blocks of variables`

Input: $\widetilde{\mathcal{L}}^k, \text{count}^k$
deactivation threshold $= \beta \frac{1}{|\widetilde{\mathcal{L}}^k|} \sum_{\ell \in \widetilde{\mathcal{L}}^k} \psi(\|w_\ell^k - w_\ell^{k-1}\|)$
$\bar{\mathcal{L}}^k = \mathcal{L} \setminus \widetilde{\mathcal{L}}^k$
$b^- = \widetilde{\mathcal{L}}_1^k$, $b^+ = \bar{\mathcal{L}}_{-1}^k$
$\widetilde{\mathcal{L}}^{k+1} = \widetilde{\mathcal{L}}^k$
if $\|w_{b^-}^k - w_{b^-}^{k-1}\| <$ deactivation threshold **then**
    $\widetilde{\mathcal{L}}^{k+1} = \widetilde{\mathcal{L}}^k \setminus \{b^-\}$
end if
Compute A^k
if $A^k < \rho A^{k-1}$ **then**
    $\text{count}^k = \text{count}^k + 1$
    $A^k = A^{k-1}$
    if $\text{count}^k = \text{patience}$ **then**
        $\text{count}^{k+1} = 0$
        $\widetilde{\mathcal{L}}^{k+1} = \widetilde{\mathcal{L}}^{k+1} \cup \{b^+\}$
    else
        $\text{count}^{k+1} = \text{count}^k$
    end if
else
    $\text{count}^{k+1} = 0$
end if
Output $\widetilde{\mathcal{L}}^{k+1}$, count^{k+1}

When a block is deactivated, it is removed from the list of the blocks to be updated in the subsequent iteration $\widetilde{\mathcal{L}}^{k+1}$ and added to the list of the deactivated blocks $\bar{\mathcal{L}}^{k+1}$. Both of these lists are built containing the blocks in ascending importance order. Consequently, at

epoch k, the deactivation candidate b^- is the first block of the list $\widetilde{\mathcal{L}}^k$, while the reactivation candidate b^+ is the last element of the list $\bar{\mathcal{L}}^k$.
In this scheme, A^k represents the validation accuracy computed at iteration k, while β is a parameter used to regulate the inclination to deactivate some blocks of parameters, so that increasing β leads to a higher degree of parameter deactivation. The parameter ρ defines the minimum increase in validation accuracy we aim to achieve at each iteration. Finally, we use the counter *count* to keep track of the number of consecutive epochs without sufficient improvement in the validation accuracy. When the counter reaches the *patience*, the candidate block b^+ is reactivated and the counter is set to zero. The deactivation threshold is computed as the average of a function ψ of the norm update of the active blocks, multiplied by the parameter β. In our tests we used ψ as the identity function.

Finally, we report the complete scheme of the algorithm, which incorporate the decomposed inner cycle and the additional controls into CMA Light. The algorithm starts by initializing the parameters and variables. It enters the main loop over the epochs, beginning with the Decomposed Inner Cycle, where the variables belonging to $\widetilde{\mathcal{L}}^k$ are updated. If the iteration count exceeds the number $\bar{k}$ of epochs during which all the parameters are updated, the blocks to be updated in the following epoch are selected through the Deactivation/Activation scheme. In addition, when necessary, the Extrapolation Derivative-Free Linesearch is executed. In the final scheme below, the monotonically non-increasing sequence $\{\phi^k\}$ introduced in CMA Light, is used to check the watchdog condition of sufficient decrease. The use of this sequence compensates the fact that $\{\tilde{f}^k\}$ is not assumed to be non-increasing and was utilized to prove the convergence of CMA Light.

Algorithm 4 `Decomposed CMA Light (CMALBD)`

Let $\zeta^0 > 0$, $w^0 \in \mathbb{R}^n$, $\theta \in (0,1)$, $k = 0$, $\bar{k}$, $\text{count}^0 = 0$, β, ρ, patience and $\widetilde{\mathcal{L}}^0 = \mathcal{L}$
for $k = 0, 1, 2, \ldots$ **do**
  Compute $(w^k, d^k) =$ `Decomposed Inner Cycle`$(w^k, \zeta^k, \widetilde{\mathcal{L}}^k)$
  if $k > \bar{k}$ **then**
    Compute $(\widetilde{\mathcal{L}}^{k+1}, \text{count}^{k+1}) =$ `Deactivation/Activation scheme` $(\widetilde{\mathcal{L}}^k, \text{count}^k)$
  end if
  if $\tilde{f}^{k+1} \leq \min\{\phi^k - \gamma\zeta^k, f(\omega^0)\}$ **then**
    Set $\zeta^{k+1} = \zeta^k$, $\alpha^k = \zeta^k$ and $\phi^{k+1} = \tilde{f}^{k+1}$
  else
    Set $(\tilde{\alpha}^k, \hat{f}^{k+1}) =$ `Extrapolation Derivative-Free Linesearch`
    Set $\alpha^k = \begin{cases} \tilde{\alpha}^k & \text{if LS successful} \\ 0 & \text{otherwise} \end{cases}$, $\zeta^{k+1} = \begin{cases} \zeta^k & \text{if LS is successful} \\ \theta\zeta^k & \text{otherwise} \end{cases}$
    Set $\phi^{k+1} = \min\{\hat{f}^{k+1}, \tilde{f}^{k+1}, \phi^k\}$
  end if
  $w^{k+1} = \begin{cases} w^k + \alpha^k d^k & \text{the RR/IG point} \\ w^k & \text{restart} \\ w^k + \tilde{\alpha}^k d^k & \text{longer step along } d_k \end{cases}$
end for

5 Computational Experiments

In this section, we report numerical results comparing the implementations of CMA Light and its block decomposed version described in Algorithm 4, denoted as CMALBD. Tests were conducted on non-pretrained state-of-the-art neural architecture, in particular ResNet18, to assess the impact on block deactivation starting from untrained weights and to evaluate the robustness of the proposed method.

5.1 Residual Neural Networks (ResNets)

Residual Neural Networks (ResNets) are a type of deep networks proposed in [12], where the concept of residual learning was introduced in Convolutional Neural Networks.
These networks allow to overcome the *degradation* problem, which consists in saturation and subsequent decrease of the accuracy when progressively increasing the depth of the network. This phenomenon is not caused by overfitting, as both the training and test errors increase when adding more layers to a deep architecture after saturation has been reached, indicating that deeper models are more difficult to train. The idea behind residual learning consists in using *residual blocks*, where the network layers compute an additive change to the current representation instead of transforming it directly. This is done by adding back the input of a residual block to its output, by using shortcut connections that add neither computational complexity nor extra parameters, as represented in Figure 1. Introducing skip connections enables the gradients and the feature learned to propagate more directly through the network. This facilitates the flow of information through the network layers, improving the performance of deeper networks and mitigating the vanishing gradient issue. More formally, we denote the underlying mapping to be fit by some stacked layers with $\mathcal{H}(x)$, and with x the inputs of the first of these layers. We explicitly make these layers learn the residual mapping $\mathcal{F}(x) = \mathcal{H}(x) - x$ by using residual connections which perform identity mapping, under the hypothesis that this residual mapping is easier to optimize compared to the underlying unreferenced mapping.

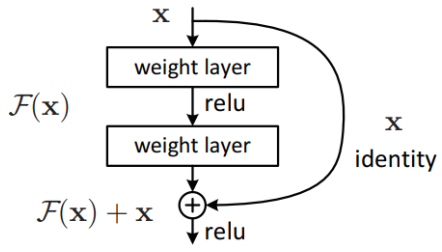

Figure 1: Residual Block. Picture from [12]

Empirical results show that extremely deep residual networks are easy to optimize, and can easily obtain accuracy gains from greatly increased depth, leading to substantially better results than previous networks.

5.2 Results

CMA Light and CMALBD have been implemented on Python 3.9, using the automatic differentiation tools for backward propagation provided by the open-source library Pytorch 2.1.2. Numerical tests have been carried out on a GPU NVIDIA Quadro P2200 4GB. We conducted tests using state-of-the-art neural architecture ResNet18, on the Image Classification datasets CIFAR10 and CIFAR100. CIFAR10 consists of 60.000 colored images, each of size 32x32, with 50.000 images for training and 10.000 for testing, belonging to ten different classes. CIFAR100 has the same image dimensions but contains 100 different classes, increasing the complexity of the classification task.
The target problem is the unconstrained minimization of the cross-entropy loss. All the algorithms were run for 50 epochs. Since the problems are highly non-convex, numerical results might be affected by the initial point used to initialize the algorithms. Therefore, we performed 5 multi-start runs for each dataset, initializing each run with a randomly selected starting point w_0. We used the default hyperparameters setting for both CMAL and CMALBD to ensure a fair comparison, while the additional parameters of the deactivation/activation control have been selected as follows:

β	ρ	$\bar{k}$	*patience*
1.25	1.005	5	2

Table 1: CMALBD parameters settings

The obtained results are summarized in the following tables, which report performance metrics at epoch 50 of average training loss, test accuracy, training accuracy and computational time across 5 multi-start runs:

Algorithm	Avg. Train Loss	Avg. Test Acc.	Avg. Train Acc.	Avg. Elapsed Time (s)
CMAL	0.00005781	63.10%	99.99%	3114.27
CMALBD	0.00034593	59.49%	99.94%	1375.05

Table 2: Results on CIFAR10 with ResNet18

Algorithm	Avg. Train Loss	Avg. Test Acc.	Avg. Train Acc.	Avg. Elapsed Time (s)
CMAL	0.00041152	29.45%	99.99%	3052.89
CMALBD	0.00201622	29.29%	99.94%	1750.02

Table 3: Results on CIFAR100 with ResNet18

These results show that the proposed CMALBD method allows for a significant reduction in computational time compared to CMAL, with a slight decrease in test accuracy, on both CIFAR10 and CIFAR100 datasets. The computational gain is remarkably larger compared to the loss in test accuracy, showing a positive trade-off between efficiency and model performance. Although the training loss on CMALBD is higher than the loss on CMAL, it

still indicates a successful learning procedure. The values of the training accuracy show the success of the optimization procedure and the ability of both CMAL and CMALBD to effectively learn from the training data. However, updating only a subset of variables may lead to a reduced generalization capability on new instances, reflected in the lower values of test accuracy. Despite this, the reduced computational time demonstrates the efficiency advantage provided by the reduced number of variables of the block decomposition approach, making CMALBD particularly suitable in large-scale settings.

6 Conclusions

In this paper, we presented a decomposed version of CMA Light, aimed at improving the efficiency of the training procedure by substantially reducing its computational time while maintaining satisfactory performance. Numerical results show the effectiveness of the proposed scheme in finding a good trade-off between the gain in computational time and reduction in the quality of the obtained solution. For tasks requiring high precision, CMA Light is preferable. However, in applications where computational resources are limited or high precision is not required, CMALBD offers a valid and efficient alternative. In future research, we aim at conducting extensive evaluations of CMALBD across various architectures, assessing its performance and scalability in diverse settings.

References

[1] Mahmoud Assran, Arda Aytekin, Hamid Feyzmahdavian, Mikael Johansson, and Michael Rabbat. Advances in asynchronous parallel and distributed optimization, 2020.

[2] Dimitri P Bertsekas. Incremental gradient, subgradient, and proximal methods for convex optimization: A survey. 2011.

[3] Dimitri P. Bertsekas and John N. Tsitsiklis. Gradient convergence in gradient methods with errors. *SIAM Journal on Optimization*, 10(3):627–642, 2000.

[4] Doron Blatt, Alfred O. Hero, and Hillel Gauchman. A convergent incremental gradient method with a constant step size. *SIAM Journal on Optimization*, 18(1):29–51, 2007.

[5] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM review*, 60(2):223–311, 2018.

[6] Corrado Coppola, Giampaolo Liuzzi, and Laura Palagi. Cma light: a novel minibatch algorithm for large-scale non convex finite sum optimization, 2023.

[7] Timothy Dozat. Incorporating Nesterov momentum into Adam. 2016.

[8] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159, 2011.

[9] Luigi Grippo and Marco Sciandrone. On the convergence of the block nonlinear gauss–seidel method under convex constraints. *Operations research letters*, 26(3):127–136, 2000.

[10] Luigi Grippof and Marco Sciandrone. Globally convergent block-coordinate techniques for unconstrained optimization. *Optimization Methods and Software*, 10(4):587–637, 1999.

[11] M. Gürbüzbalaban, A. Ozdaglar, and P. A. Parrilo. Why random reshuffling beats stochastic gradient descent. *Mathematical Programming*, 186(1–2):49–84, October 2019.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[13] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[14] Giampaolo Liuzzi, Laura Palagi, and Ruggiero Seccia. Convergence under lipschitz smoothness of ease-controlled random reshuffling gradient algorithms, 2023.

[15] Jakub Mareček, Peter Richtárik, and Martin Takáč. *Distributed Block Coordinate Descent for Minimizing Partially Separable Functions*, page 261–288. Springer International Publishing, 2015.

[16] Konstantin Mishchenko, Ahmed Khaled, and Peter Richtárik. Random reshuffling: Simple analysis with vast improvements. *Advances in Neural Information Processing Systems*, 33:17309–17320, 2020.

[17] Peter Richtárik and Martin Takáč. Distributed coordinate descent method for learning with big data, 2013.

[18] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[19] Nicolas Le Roux, Mark Schmidt, and Francis Bach. A stochastic gradient method with an exponential convergence rate for finite training sets, 2013.

[20] Mark Schmidt, Nicolas Le Roux, and Francis Bach. Minimizing finite sums with the stochastic average gradient, 2016.

[21] Ohad Shamir. Without-replacement sampling for stochastic gradient methods: Convergence results and application to distributed optimization, 2016.

[22] Matthew D. Zeiler. ADADELTA: An adaptive learning rate method, 2012.