Les Cahiers du GERAD ISSN: 0711–2440

PLSR1: A limited-memory partitioned quasi-Newton optimizer for partially-separable loss functions

P. Raynaud, D. Orban, J. Bigeon

G–2023–41

September 2023

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

PLSR1: A limited-memory partitioned quasi-Newton optimizer for partially-separable loss functions

Paul Raynaud [a, b]

Dominique Orban [a]

Jean Bigeon [a, b, c]

[a] *GERAD and Department of Mathematics and Industrial Engineering, Polytechnique Montréal, Montréal (Qc), Canada, H3T 2A7*

[b] *GSCOP, Univsité Grenoble Alpes, Grenoble, France*

[c] *LS2N, Centrale Nantes, France*

paul.raynaud@polymtl.ca
dominique.orban@gerad.ca
jean.bigeon@univ-nantes.fr

September 2023
Les Cahiers du GERAD
G–2023–41

Abstract : Improving neural network optimizer convergence speed is a long-standing priority. Recently, there has been a focus on quasi-Newton optimization methods, which have fewer hyperparameters compared to gradient-based methods and show improved convergence results in deterministic optimization. We introduce PLSR1, a limited-memory partitioned quasi-Newton optimizer designed for optimizing a partially separable loss function, which is a sum of element loss functions of smaller dimensions. PLSR1 aggregates limited-memory quasi-Newton approximations of individual element loss Hessians to better approximate the overall loss Hessian. To keep storage affordable, element function dimensions must be small compared to the total dimension. Thus, we adapt standard neural network architectures by incorporating separable layers, creating a partitioned architecture (PSNet). The numerical results compare the performance of several optimizers training the same partially-separable loss function on LeNet and PSNet architectures of similar sizes and effectiveness. The graphs exhibit the optimizer accuracies over epochs, on both the MNIST and CIFAR10 datasets. PLSR1 and an adaptative Nesterov variant show a training convergence comparable to Adam and outperforms LBFGS and SGD.

1 Introduction

Deep learning is now a cornerstone of our society. In supervised learning, neural networks are trained on datasets to solve data-aligned problems. For multiclass classification [12], neural networks seek to correctly identify the class where any input adhering to the network's architecture belongs. The network is trained using an optimizer on a specific dataset, trying iteratively to reduce misclassifications. The optimizer minimizes a loss function, which assesses the relevance of score for one input and its label. As a result, it enhances class determination and increase the neural network's accuracy.

Datasets have expanded to the point where assessing them entirely is impractical. For example, the ImageNet dataset comprises over 14 million images [32]. Consequently, training datasets are divided into smaller, randomly selected minibatches to ensure reasonable loss function evaluations. However, this approach introduces stochastic noise when evaluating the loss function on a minibatch.

In Robbins and Monro [31], the foundation of stochastic optimization was laid, leading to the first method rooted in gradient loss computations from data subsets, commonly known as stochastic gradient descent (SGD) [24]. SGD represents the prototype among gradient-based methods, where weight updates rely on scalar adjustments derived of first partial derivatives. Over time, gradient-based methods evolved. Momentum was introduced by [28], followed by more recent techniques like AdaGrad [13], RMSProp [18], and Adam [21]. These methods offer distinct strategies concerning the step size derived from sampled loss gradient moments. While a comprehensive exploration of these methods isn't within our scope, interested readers can delve into further insights through sources such as [3, 5, 24].

Within deterministic smooth convex optimization scope—where the full-size batch loss gradient is considered—gradient-based methods display linear convergence towards the minimum [5]. This linear convergence rate is surpassed by second-order methods, which also require fewer hyperparameter. In particular, quasi-Newton methods which seek to approximate the Hessian (or its inverse) from past gradients. Namely, BFGS and its limited-memory variant LBFGS [26] achieve local superlinear convergence and outperforms gradient-based methods significantly.

Within a stochastic setup, both gradient-based methods and limited quasi-Newton methods achieve sub-linear convergence rates [5]. However, quasi-Newton methods are likely to enhance the hidden constant behind theoretical asymptotic convergence results [5].

Among quasi-Newton methods, a specific subclass, known as partitioned quasi-Newton methods, has demonstrated notably superior performance in deterministic optimization [16, 26]. While quasi-Newton methods are applicable to any $\mathcal{C}^2$ smooth function, the partitioned variants require the function to exhibit partial separability, i.e. a sum of smaller dimension element functions. In this work, we exploit a partially-separable loss function [1] to train a multiclass classification neural network with a partitioned quasi-Newton optimizer. To fully exploit partial separability and benefit its determinist performances, traditional neural network architectures must be adapted. To address this issue, [1] employs separable layers to construct partitioned architectures. [1] details how partitioned structure enlarged the panel of techniques distributing training computations by evaluating element loss computation on different devices. Federated learning benefits from this characteristic, as the training is distributed across less capable edge devices [34].

The structure of the paper is as follows: Section 2 provides a brief overview of the unsupervised learning context. Section 3 delves into the concept of partial separability, outlines the derivative partitioned structure and introduce a partially-separable loss function. Section 4 recalls quasi-Newton principles and details our limited-memory partitioned quasi-Newton approximation of the Hessian and the resulting optimization method [2]. Section 5 recaps separable layers and introduces the partitioned architectures from Anonymous [1]. Finally, Section 6 compares various training strategies applied to a partitioned architecture, discusses current limitations, and concludes in Section 7.

2 Supervised learning context and multiclass classification neural network

Neural networks for multiclass classification aim to categorize inputs among C distinct classes [12]. To achieve this, they compute a score c_j for each class and select the class with the highest score ($\arg\max_{j=1,\dots,C} c_j$). Parametrized by n weights, the neural network architecture can be seen as a function $c : \mathbb{R}^n \to \mathbb{R}^C$, applicable to inputs that conforms to its architectural specifications.

In supervised training, the dataset is divided into two segments: the training dataset and the test dataset. A dataset comprises observations denoted as $\mathbf{X}$, paired with their respective labels $\mathbf{Y}$. Each $x \in \mathbf{X}$ signifies an individual observation, while its corresponding label $y \in \mathbf{Y}$ is described by $1 \leq y \leq C$, $y \in \mathbb{N}$. A dataset containing $\mathbf{L}$ pairs of observation-label is expressed as $(\mathbf{X}, \mathbf{Y}) = \{(x^{(l)}, y^{(l)})\}_{l=1}^{\mathbf{L}}$.

In tandem with the architecture that generates the function c, the training problem requires an evaluation of its prediction accuracy. This role is fulfilled by the loss function $\mathcal{L}(x, y; w)$, which quantifies the appropriateness of the neural network's predictions for each observation-label pair (x, y). When applied to a minibatch $(X, Y) \subset (\mathbf{X}, \mathbf{Y})$ of size L, the loss function computes the mean loss of all observation-label :

$$\mathcal{L}(X, Y; w) = \frac{1}{L} \sum_{l=1}^{L} \mathcal{L}(x^{(l)}, y^{(l)}; w). \tag{1}$$

For the context of multiclass classification, a widely-used loss function combines the softmax layer and negative log-likelihood, another way to break down the cross-entropy loss [9]:

$$\mathcal{L}^{\text{NLL}}(x^{(l)}, y^{(l)}; w) = -\log \underbrace{\left(\frac{exp(c_{y^{(l)}}(x^{(l)}; w))}{\sum_{i=1}^{C} exp(c_i(x^{(l)}; w))} \right)}_{\text{softmax}}, \tag{2}$$

where $exp(c_{y^{(l)}})$ represents the score for the expected result $y^{(l)}$ given the input $x^{(l)}$. Observe that the denominator of the softmax layer in $\mathcal{L}^{\text{NLL}}$ encompasses all the scores produced by the neural network. Consequently, $\mathcal{L}^{\text{NLL}} : \mathbb{R}^{n^{\text{NLL}}} \to \mathbb{R}$ is fully parameterized by w, i.e. $n^{\text{NLL}} = n$.

3 Partially-separable training

Section 3.1 formalizes the partially-separable function concept. Then, Section 3.2 proposes a partially-separable loss function.

3.1 Partially-separable function

A partially separable function is defined as:

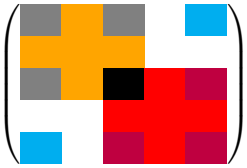
$$f(w) = \sum_{i=1}^{N} \widehat{f}_i(U_i w), \quad U_i \in \mathbb{R}^{n_i \times n}, w \in \mathbb{R}^n. \tag{3}$$

Here, f sums element functions of smaller dimensions $\widehat{f}_i : \mathbb{R}^{n_i} \to \mathbb{R}$, where $n_i < n$. The linear operator U_i selects the variables parametrizing $\widehat{f}_i$. The concept of partial separability is detailed in the early 1980s, originally motivated by partial differential equation discretized problems [10, 15]. This structure has since found applications across various optimization domains, including quasi-Newton methods [8, 10, 15], derivative-free methods [6, 30], and evolutionary methods [14]. Essentially, partial separability leads to partitioned derivatives:

$$\nabla f(w) = \sum_{i=1}^{N} U_i^\top \nabla \widehat{f}_i(U_i w), \ \nabla^2 f(w) \approx B = \sum_{i=1}^{N} U_i^\top \widehat{B}_i U_i, \tag{4}$$

where $\nabla \widehat{f}_i \in \mathbb{R}^{n_i}$ and $\widehat{B}_i = \widehat{B}_i^\top \in \mathbb{R}^{n_i \times n_i}$ approximates $\nabla^2 \widehat{f}_i$. For example: $f(w) = f_1(w_1, w_2, w_3) + f_2(w_3, w_4, w_5) + f_1(w_1, w_3, w_5)$ gets the partitioned Hessian:

$$\nabla^2 f = \left(\quad \right).$$



This property generates straightforward schemes to parallelize computations. In particular, [25, 29] explain it in the context of deterministic optimization context, while [1] specifics are more related to deep learning.

3.2 Partially-separable loss function

A loss function must yield a high value if the neural network's maximum score $\arg\max_{j=1,..,C} c_j$ differs from $y^{(l)}$ for a given input $x^{(l)}$. Conversely, if $c_{y^{(l)}}$ is correctly identified, the value returned must be low.

It's worth noting that despite the summation in (1), $\mathcal{L}^{\text{NLL}}(X, Y; w)$ doesn't exhibit partial separability, as $n = n^{\text{NLL}} \not< n$. To establish an element loss of smaller dimension, it needs to rely on a subset of the scores c_j, which depend only on a subset of weights. Formally, $\widehat{c}_j(x^{(l)}, y^{(l)}; U_j w) = c_j(x^{(l)}, y^{(l)}; w)$, where $\widehat{c}_j : \mathbb{R}^{n_j} \to \mathbb{R}$, with $n_j < n$ and parametrized by weights selected by U_j (as illustrated in Figure 2). Our approach introduces a partially separable loss (PSL), in which each element loss function focuses on a specific pair of scores:

$$\mathcal{L}^{\text{PSL}}(X, Y; w) := \frac{1}{L} \sum_{l=1}^{L} \sum_{j=1}^{C} e^{c_j(x^{(l)}; w) - c_{y^{(l)}}(x^{(l)}; w)}, \tag{5a}$$

$$= \sum_{p=1}^{C} \sum_{j=1 \neq p}^{C} h_{p,j}(X, Y; w), \tag{5b}$$

$$h_{p,j}(X, Y; w) := \frac{1}{L} \sum_{l=1}^{L} \delta_{p,j}(y^{(l)})\, e^{c_j(x^{(l)}; w) - c_p(x^{(l)}; w)}, \tag{5c}$$

where $\delta_{p,j}(y^{(l)}) = 1$ if $y^{(l)} = p$, and 0 otherwise. Each element function $h_{p,j}$ exclusively utilizes the weights that correspond to the two scores c_p and c_j:

$$\widehat{h}_{p,j}(X, Y; U_{p,j} w) = h_{p,j}(X, Y; w), \quad U_{p,j} \in \mathbb{R}^{n_{p,j} \times n}.$$

Here, the linear operator $U_{p,j}$ combines the selected weights from both U_p and U_j, $n_{p,j} \leq n_p + n_j$ and $n_{p,j} \leq n$. Section 5 discusses partitioned neural networks designed to minimize $n_{p,j}$ before n [1].

4 Limited-memory partitioned quasi-Newton training

Second-order methods differ from gradient-based methods, which update weights mainly from the sampled loss gradient. Section 4.1 recalls the basics of quasi-Newton methods while Section 4.2 presents our limited-memory partitioned variant. This method is tailored for partially-separable problems and can be applied in any neural network training problem minimizing $\mathcal{L}^{\text{PSL}}$.

4.1 Quasi-Newton methods

Secant quasi-Newton operators aim to approximate $\nabla^2 f(x_k)$ (or $\nabla^2 f(x_k)^{-1}$) using linear operators $B_k = B_k^\top$ (or H_k) throughout optimization iterations. These operators rely on gradient evaluations

and satisfy the secant equation:

$$B_{k+1}s_k = y_k,\ s_k = w_{k+1} - w_k,\ y_k = \nabla f(w_k + s_k) - \nabla f(w_k). \tag{6}$$

Among the most prevalent quasi-Newton updates are BFGS:

$$B_{k+1}^{\text{BFGS}} = B_k - \frac{(B_k s_k)(B_k s_k)^\top}{s_k^\top B_k s_k} + \frac{y_k y_k^\top}{s_k^\top y_k}\ , \tag{7}$$

which maintains $B_{k+1}^{\text{BFGS}} \succ 0$ if $B_k \succ 0$ and $s_k^\top y_k > 0$, and SR1 [11]:

$$B_{k+1}^{\text{SR1}} = B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^\top}{s_k^\top (y_k - B_k s_k)}, \tag{8}$$

which might result in an indefinite B_{k+1}^{SR1}.

Historically, these linear operators were implemented using dense matrices. However, as problem dimensions grow, dense matrices become impractical. Limited-memory variants emerged: LBFGS [7, 26] and LSR1 [27]. In these variants, B_k doesn't use a matrix; instead, it employs a linear operator $v \to B_k v$, based from the last corresponding m updates retained as $\{s_{k-j}, y_{k-j}\}_{j=1}^m$. For an operator of memory m, $v \to B_k v$ is accomplished with a double loop over $\{s_{k-j}, y_{k-j}\}_{j=1}^m$, whose computational complexity is $\Theta(mn)$. Limited-memory variants are commonly employed within a conjugate gradient linear solver [17], requiring only $v \to B_k v$ to (approximately) solve a linear system.

4.2 Partitioned quasi-Newton methods

Partitioned quasi-Newton methods approximate every $\nabla^2 \widehat{f}_i$ using matrices $\widehat{B}_i$ updated through BFGS or SR1 formulas (7)–(8). In these formulas, y and s are replaced with $\widehat{y}_i = \nabla \widehat{f}_i(U_i w_{k+1}) - \nabla \widehat{f}_i(U_i w_k)$ and $\widehat{s}_i = U_i(w_{k+1} - w_k)$. By updating all $\widehat{B}_i$ at each iteration, the matrix $B_{k+1} - B_k$ becomes of relatively high rank, i.e. $\min(N, n)$, in contrast to (L)BFGS or (L)SR1 updates (e.g. rank 1 or rank 2). As a result, partitioned quasi-Newton operators tend to provide more accurate approximations of $\nabla^2 f$ in fewer updates. Limited-memory variants of partitioned quasi-Newton updates are introduced by [2], where each $\nabla^2 \widehat{f}_i$ is approximated using either a LBFGS or a LSR1 operator. These variants significantly reduce B's storage requirements from $\Theta(\sum_{i=1}^N \frac{n_i(n_i+1)}{2})$ to $\Theta(m \sum_{i=1}^N n_i)$.

In the numerical results, we employ PLSR1, where each $\widehat{B}_{k+1,i} = \widehat{B}_{k+1,i}^{\text{LSR1}}$. In contrast to PLBFGS, where each $\widehat{B}_{k+1,i} = \widehat{B}_{k+1,i}^{\text{LBFGS}}$, PLSR1 does not necessitate the fulfillment of the element curvature condition $\widehat{s}_i^\top \widehat{y}_i > 0$ to update $\widehat{B}_{k,i}$. Consequently, $\widehat{B}_{k,i}$ is more likely to incorporate recent element updates compared to PLBFGS, allowing it to more frequently capture local element function curvatures.

Algorithm 1 outlines an inexact line search method minimizing a partially separable function f by exploiting a limited-memory partitioned quasi-Newton approximation of $\nabla^2 f$. To adapt Algorithm 1 to the deep learning context, f and ∇f correspond respectively to the sampled loss function $\mathcal{L}^{\text{PSL}}(X, Y; w)$ and its gradient $\nabla \mathcal{L}^{\text{PSL}}(X, Y; w)$. Each element function $\widehat{f}_i$ refers to an element loss function $\widehat{h}_{p,j}$. Note that to calculate $\widehat{y}_{k,i}$, the element function must compute $\nabla \widehat{h}_{p,j}(X, Y; U_{p,j} w_{k+1})$ on the same minibatch X, Y as $\nabla \widehat{h}_{p,j}(X, Y; U_{p,j} w_k)$. Additionally, we also incorporated an adaptive Nesterov accelerated gradient, inspired by [19, 20] which adds a momentum in $\nabla f(w_k)$ computation from Algorithm 1. This variant is denoted as PLSR1_AdaN. Both partitioned quasi-Newton training methods are compared Figure 4 in Section 6.

5 Separable layer and partitioned architecture

Our partially separable loss function can be applied to different multiclass classification neural networks. However, not all architectures lead to significantly smaller $n_{p,j}$ compared to n. To address

Algorithm 1 Inexact partitioned quasi-Newton line search.

Choose $w_0 \in \mathbb{R}^n$, $\tau_1, \tau_2 \in \mathbb{R}^+$,
Choose $B_0 = B_0^\top = \sum_{i=1}^N U_i^\top \widehat{B}_{0,i} U_i \approx \nabla^2 f(w_0)$.
$k = 0$
repeat
    Compute an inexact solution d_k

$$\min_{d_k} m_k(d_k) = f(w_k) + \nabla f(w_k)^\top d_k + \tfrac{1}{2} d_k^\top B_k d_k$$

by using the conjugate gradient, such that $\|B_k d_k + \nabla f_k\|_2 \leq \tau_1$.
    Perform a line search to retrieve $\alpha > 0$ such that

$$f(w_k) - f(w_k + \alpha s_k) \geq \alpha * \tau_2 \nabla f(w_k)$$

    set $w_{k+1} = w_k + \alpha_k d_k$
    update every $\widehat{B}_{k,i}$ given $\widehat{s}_{k,i} = \alpha_k U_i d_k$ and $\widehat{y}_{k,i}$ satisfying $\widehat{B}_{k+1,i}\widehat{s}_{k,i} = \widehat{y}_{k,i}$, e.g. (**??**).
    $k = k + 1$
until convergence

this, [1] introduces the separable layer (see Figure 1). It divided neural layers into C groups, where each group interact with its counterpart as a standard dense layer. The stacking of multiple separable layers builds a partitioned architecture, referred as PSNet. Figure 2 provides a visual comparison of a simplified LeNet [24] architecture with a PSNet architecture. The figure also illustrates the weight dependencies for each score in both architectures using different colors. Specifically, blue, yellow, and red denote weights unique to individual scores. Shared weights across all scores are depicted in green.

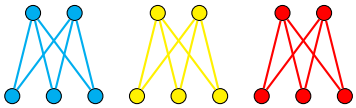


Figure 1: A separable layer, 9 x 6, considering $C = 3$ classes [1].

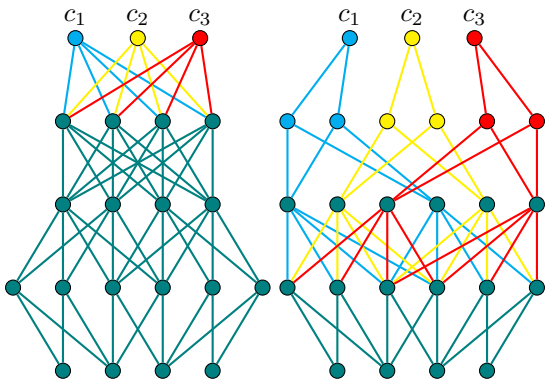


Figure 2: Simplified LeNet (left) and PSNet (right) score's dependencies [1].

When combining a partitioned architecture with $\mathcal{L}^{\text{PSL}}$, training computations can be evenly distributed among element functions $\widehat{h}_{p,j}$ [1]. Each $\widehat{h}_{p,j}$ is assigned to a specific worker. The worker computes $\widehat{h}_{p,j}(X, Y; w)$, $\nabla\widehat{h}_{p,j}(X, Y; w)$, and the approximation of $\nabla^2\widehat{h}_{p,j}(X, Y; w)$ or $v \to \nabla^2\widehat{h}_{p,j}v$. Hence, the partitioned linear product $Bv = \sum_{i=1}^N U_i^\top \widehat{B}_i U_i v$ can be distributed by having each worker calculates $\widehat{B}_i U_i v$ transmitted and aggregated by the master(s). In that case, it becomes a decentralized partitioned quasi-Newton method. By distributing each element loss function computation to a worker, the partitioned architecture aligns with the needs of federated learning. In particular, it structurally reduces computation and enhances the privacy of edge devices. Further insights regarding

the PSNet architectures can be found in [1]. Figure 3 in Section 6 report comparisons between LeNet and PSNet architectures of similar size trained with Adam [21] on the MNIST [23] and CIFAR10 [22] datasets.

6 Numerical results

Figure 3 record Adam trainings performed on LeNet and PSNet architectures using $\mathcal{L}^{\text{NLL}}$ and $\mathcal{L}^{\text{PSL}}$ [1], on both MNIST and CIFAR10 datasets. MNIST's LeNet and PSNet architectures are adapted to CIFAR10 input dimensions. The architecture specifics are summarizes in Table 1. Every convolutional layer use an average pooling and is parametrized by $k_1 \times k_2 \times i \times o$, where k_j specify the kernel dimensions, i the input channels and o the output channels.

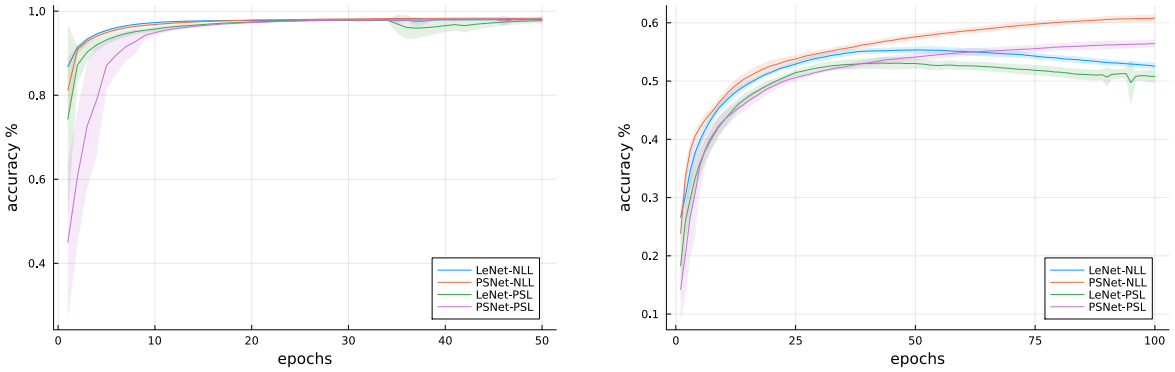


Figure 3: LeNet and PSNet training accuracies over epochs on MNIST (left) and CIFAR10 (right) [1].

Table 1: Architecture details.

LeNet			PSNet		
type	MNIST	CIFAR10	type	MNIST	CIFAR10
Conv	$5 \times 5 \times 1 \times 6$	$5 \times 5 \times 3 \times 6$	Conv	$5 \times 5 \times 1 \times 30$	$5 \times 5 \times 3 \times 30$
Conv	$5 \times 5 \times 6 \times 16$	$5 \times 5 \times 6 \times 16$	Conv	$5 \times 5 \times 30 \times 40$	$5 \times 5 \times 30 \times 60$
Dense	256×120	400×200	Separable	480×240	750×350
Dense	120×84	200×100	Separable	240×150	350×150
Dense	84×10	100×10	Separable	150×10	150×10
n	44426	103882		53780	81750
$n_j, n_{p,j}$	-	-		$6340, 11588$	$12279, 19998$

In both Figure 3 and Figure 4, each curve represents the mean accuracy while its shaded region is the standard deviation error. MNIST trainings run for 50 epochs whereas CIFAR10 training run for 100 epochs. All results are produced in Julia [4], and gradient-based method implementation are from Knet.jl [33]. All methods were running on a Nvidia A100 Tensor Core GPU, using minibatches of size 100. Adam (resp. SGD) learning rate is fixed at (resp. 0.025), while $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. The main objective of Figure 3 is to highlight the comparable performance between LeNet, coupled with $\mathcal{L}^{\text{NLL}}$, and PSNet, combined with $\mathcal{L}^{\text{PSL}}$.

Figure 4 illustrates the comparison between several optimizers: SGD, Adam, LBFGS, and PLSR1 and PLSR1_AdaN. All are employed to minimize $\mathcal{L}^{\text{PSL}}$ using a PSNet architecture over MNIST and CIFAR10 datasets. In this setup, Adam achieves the fastest training convergence, followed closely by both PLSR1 variants, LBFGS and SGD. The Nesterov accelerated gradient incorporation doesn't bring a substantial improvement on final accuracy, compare to the standard PLSR1. SGD's slower convergence can be attributed to its small learning rate, which is chosen to prevent numerical instability

of the gradients. For both dataset, PLSR1 variants minimizing $\mathcal{L}^{\text{PSL}}$ are able to reach after all the epochs the asymptotic the accuracy of LeNet where Adam minimizes $\mathcal{L}^{\text{NLL}}$ Figure 3.

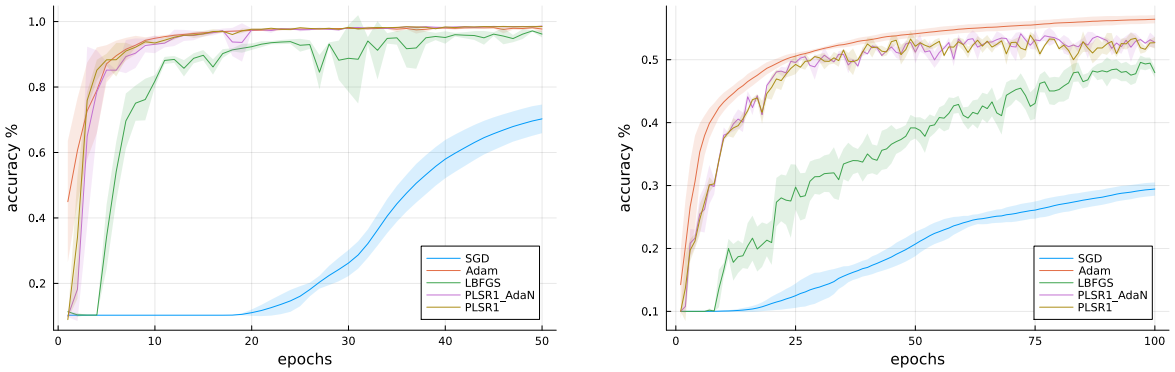


Figure 4: Comparison of optimizer accuracies over epochs during PSNet training when minimizing $\mathcal{L}^{\text{PSL}}$ on MNIST (left) and CIFAR10 (right).

7 Conclusion

We introduce a novel partitioned limited-memory quasi-Newton training (PLSR1), tailored for partially separable loss functions. To take full advantage of partial separability within the loss, it's crucial to incorporate separable layers into the architecture, creating a partitioned architecture. In this setup, our empirical results demonstrate that PLSR1 outperforms SGD and LBFGS and is competitive with Adam. Regarding future work, we aim to integrate this partitioned training into a distributed computing framework to lean toward federated learning.

References

[1] X. Anonymous. Partially-separable loss to parallellize partitioned neural network training. 2023.

[2] X. Anonymous. Anonymous title. 2023.

[3] T. Ben-Nun and T. Hoefler. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. ACM Comput. Surv., 52(4), aug 2019. doi: 10.1145/3320060. URL https://doi.org/10.1145/3320060.

[4] J. e. a. Bezanson. Julia: A fresh approach to numerical computing. SIAM Review, 59(1):65–98, 2017. doi: 10.1137/141000671.

[5] L. Bottou, F. E. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. SIAM Review, 60(2):223–311, 2018. doi: 10.1137/16M1080173. URL https://doi.org/10.1137/16M1080173.

[6] Z. Bouzarkouna, A. Auger, and D. Ding. Local-meta-model CMA-ES for partially separable functions. In Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, GECCO '11, pages 869—-876, New York, NY, USA, 2011. ISBN 9781450305570. doi: 10.1145/2001576.2001695.

[7] R. H. Byrd, J. Nocedal, and R. B. Schnabel. Representations of quasi-Newton matrices and their use in limited memory methods. 63(1):129–156, Jan 1994. doi: 10.1007/BF01582063.

[8] H. Cao and L. Yao. A partitioned PSB method for partially separable unconstrained optimization problems. 290:164–177, 2016. doi: 10.1016/j.amc.2016.06.009.

[9] L. Ciampiconi, A. Elwood, M. Leonardi, A. Mohamed, and A. Rozza. A survey and taxonomy of loss functions in machine learning, 2023.

[10] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. An introduction to the structure of large scale nonlinear optimization problems and the LANCELOT project. Computing Methods in Applied Sciences and Engineering, pages 42–54, 1990. doi: 10.1007/BF02592099.

[11] W. C. Davidon. Optimally conditioned optimization algorithms without line searches. 9(1):1–30, Dec 1975. doi: 10.1007/BF01681328.

[12] T. G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. J. Artif. Int. Res., 2(1):263–286, jan 1995.

[13] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. 12, 2011. doi: 10.5555/1953048.2021068.

[14] N. Durand and J.-M. Alliot. Genetic crossover operator for partially separable functions. In GP 1998, 3rd annual conference on Genetic Programming, Madison, United States, 1998. URL https://hal-enac.archives-ouvertes.fr/hal-00937718.

[15] A. Griewank and Ph. L. Toint. Partitioned variable metric updates for large structured optimization problems. 39:119–137, 1982. doi: 10.1007/BF01399316.

[16] A. Griewank and Ph. L. Toint. Numerical experiments with partially separable optimization problems. In D. F. Griffiths, editor, Numerical Analysis, pages 203–220, Berlin, Heidelberg, 1984. Springer Berlin Heidelberg. ISBN 978-3-540-38881-4. doi: 10.1007/BFb0099526.

[17] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. 49(6):409–436, 1952. doi: 10.6028/jres.049.044.

[18] G. Hinton. Neural networks for machine learning, lecture 6a: Overview of mini-batch gradient descent. https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf, June 2012.

[19] S. Indrapriyadarsini, S. Mahboubi, H. Ninomiya, T. Kamio, and H. Asai. A modified limited memory nesterov's accelerated quasi-newton, 2021.

[20] S. Indrapriyadarsini, S. Mahboubi, H. Ninomiya, T. Kamio, and H. Asai. Accelerating symmetric rank-1 quasi-newton method with nesterov's gradient for training neural networks. Algorithms, 15(1), 2022. doi: 10.3390/a15010006. URL https://www.mdpi.com/1999-4893/15/1/6.

[21] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2017.

[22] A. Krizhevsky. Learning multiple layers of features from tiny images. pages 32–33, 2009. URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

[23] Y. LeCun, C. Cortes, and C. Burges. Mnist handwritten digit database. ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist, 2, 2010.

[24] Y. e. a. Lecun. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86 (11):2278–2324, Nov 1998. doi: 10.1109/5.726791.

[25] M. Lescrenier. Partially separable optimization and parallel computing. 14(1):213–224, 1988. doi: 10.1007/BF02186481.

[26] D. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. 45(1-3): 503–528, 1989. doi: 10.1007/BF01589116.

[27] X. Lu. A computational study of the limited memory SR1 method for unconstrained optimization. University of Colorado, Boulder, 1996.

[28] Y. Nesterov. A method for solving the convex programming problem with convergence rate $o(1/k^2)$. Proceedings of the USSR Academy of Sciences, 269:543–547, 1983. URL https://www.mathnet.ru/eng/dan46009.

[29] M. Porcelli and Ph. L. Toint. Exploiting problem structure in derivative free optimization. ACM Trans. Math. Softw., 48(1), feb 2022. doi: 10.1145/3474054.

[30] C. J. Price and Ph. L. Toint. Exploiting problem structure in pattern search methods for unconstrained optimization. 21(3):479–491, 2006. doi: 10.1080/10556780500137116.

[31] H. Robbins and S. Monro. A Stochastic Approximation Method. The Annals of Mathematical Statistics, 22 (3):400 – 407, 1951. doi: 10.1214/aoms/1177729586. URL https://doi.org/10.1214/aoms/1177729586.

[32] O. e. a. Russakovsky. ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision (IJCV), 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

[33] D. e. a. Yuret. Knet.jl: Koç university deep learning framework., 2020. URL https://github.com/denizyuret/Knet.jl.

[34] C. Zhang, Y. Xie, H. Bai, B. Yu, W. Li, and Y. Gao. A survey on federated learning. Knowledge-Based Systems, 216:106775, 2021. doi: https://doi.org/10.1016/j.knosys.2021.106775. URL https://www.sciencedirect.com/science/article/pii/S0950705121000381.