Computing and Informatics, Vol. 44, 2025, 1–25, doi: 10.31577/cai_2025_1_1

RESOURCE-EFFICIENT MODEL FOR DEEP KERNEL LEARNING

Luisa D'Amore

Department of Mathematics and Applications
University of Naples Federico II
Complesso Monte Sant'Angelo
80126 Napoli, Italy
e-mail: luisa.damore@unina.it

Abstract. According to Hughes phenomenon, the major challenges encountered in computations with learning models come from the scale of complexity, e.g. the so-called curse of dimensionality. Approaches for accelerated learning computations range from model- to implementation-level. The first type is rarely used in its basic form. Perhaps, this is due to the theoretical understanding of mathematical insights. We describe a model-level decomposition approach that combines both the decomposition of the objective function and of data. We perform a feasibility analysis of the resulting algorithm, both in terms of accuracy and scalability.

Keywords: Parallel machine learning, parallel and distributed deep learning, GPU parallelism, domain decomposition, problem and model reduction

Mathematics Subject Classification 2010: 68T07, 65K05

1 INTRODUCTION

Predictive Data Science is the paradigm shift of computational science tightly integrating numerical simulations with algorithms and techniques having the capability of extracting insights or useful knowledge from data (also known as learning-from-examples). Predictive Data Science – revolutionizing decision-making for high-consequence applications in science, engineering and medicine – aims not only to reproduce with high-fidelity the real world, but also to predict its behaviour in

situations for which the numerical simulation has not been specifically validated. Machine learning (ML) is part of predictive data science, dealing with statistics, algorithms, and scientific methods used to extract knowledge from data [1, 2]. Various types of ML algorithms such as supervised, unsupervised, semi-supervised, and reinforcement learning exist in the area. In addition, Deep Learning (DL), which is part of a broader family of machine learning methods, can analyze the data on a large scale by learning from subtasks. DL has having immense success in the recent past leading to state-of-art results in various domains (biotechnology, finance, ...). Anyway, the same flexibility that makes DL excellent at modeling diverse phenomena and outperforming other models (which indeed depend on large amounts of data and computation) also makes it dramatically more computationally expensive. If the current trend continues, the growing computational cost of DL will become technical and economically prohibitive [3]. The continuing appearance of research on distributed learning is due to the progress made by specialized high-performance architectures. However, the computational needs of DL grow so rapidly that it will quickly become computationally constrained again.

We present a model-level technique for tackling intensive DL problems which relies on the ideas of the Kernel decomposition approach. We call Domain Decomposition Deep Learning – D^3L. D^3L involves data reduction, localization of the predictive function and reduction of the error function. The main feature of D^3L is that local error functions are suitably modified, by imposing a regularization constraint to enforce the matching of their solutions between adjacent subproblems. As a consequence, instead of solving one DL problem, we can solve several smaller problems by improving the accuracy-per-parameter metric. Most importantly, subproblems can be solved in parallel, leading to a scalable algorithm where the workers locally exchange parameter updates via a nearest-neighbourhood communication scheme, which does not require a fully connected network.

1.1 Related Works

Resource-efficient DL research has vividly been carried out independently in various research communities including the machine learning, computer arithmetic, and computing system communities. There are various approaches to compress or accelerate DL methods with minimal loss of precision [4, 5, 6, 7]. Among them, the model-level techniques aim at reducing the problem size to fit the DL models to resource-constrained systems. On the contrary, implementation-level techniques aim at improving the computational speed of DL operations [8]. These approaches range from fine to coarse-grained parallelism. The first one corresponds to the standard fine-grained concurrency of the floating point operations (it exploits concurrency inside the parts that represent the main computational bottlenecks of the neural network layers to enhance the overall performance of the whole algorithm). In the realm of DL, this approach is often referred to as *concurrency in operators*. To implement fine-grained parallelism effectively, specialized hardware such as GPUs and TPUs are commonly used. These hardware accelerators are designed to handle

a multitude of parallel operations simultaneously, making them well-suited for DL workloads. Finally, parallel execution techniques such as multiprocessing and multithreading are employed to execute parallel operations concurrently. This approach significantly reduces the training time for deep neural networks, allowing for the training of larger and more complex models.

The second one is based on the problem decomposition and, in contrast to fine-grained parallelism introduces concurrency at a coarser level of computation. In the realm of DL, this approach is known as *concurrency in the network*, and involves partitioning the computational workload in various ways: by input samples (data parallelism), by network structure (model parallelism), and by layer (pipelining) [9, 10]. Data parallelism is a straightforward approach to parallelization. This method dates back to the first practical implementations of artificial neural networks [11]. Data parallelism partitions the dataset among processing units having a copy of the full model. Each unit calculates the gradients of different subsets, independently, and uses these gradients to update the global model concurrently. Most DL frameworks support data parallelism (Chainer [12], PyTorch [13], CNTK [14]). In the model parallelism, data are copied to all processors, each unit has a portion of the model and works in cooperation with others to do the calculation for one subset. Subsequently, different stages of the calculation of the global model are executed in the pipeline on different processors. In layer pipelining, different layers of the neural network are processed concurrently, with data flowing through the network in a pipeline fashion. It is a strategy to reduce the latency in DL inference.

Model parallelism has lower communication requirements, but because of its pipeline data dependency, model parallelism suffers from underutilization of the computing elements, while data parallelism does not have the data dependency issues, but requires heavier communication across processing units due to the synchronization with the other processing units. The combination of multiple parallelism schemes can overcome the drawbacks of each scheme giving rise to hybrid approaches. In this regard, the most notable frameworks supporting hybrid approaches are TensorFlow [15], MXNet [16] and SINGA [17].

1.2 The Present Work

According to the recent survey in [8], in the following we summarize main aspects of significance and novelty of D^3L.

Parallelization Model. We introduce a hybrid parallelism which starts with a decomposition of the global problem into overlapped smaller subproblems. On these subdomains we define local minimization problems and we prove that the solution of the (global) problem can be obtained by collecting the solutions of local problems.

Framework Architecture. The proposed approach works without any parameter services, e.g. it is decentralized, but it does not need to exchange parameter updates by using an all-reduce operation. The resulting algorithm consists of

several copies of the original one, each one requiring approximately the same amount of computations on each subdomain and an exchange of boundary conditions between adjacent subdomains.

Synchronization. The partitioning of the computational domain requires only interactions among two adjacent subdomains. Since data is flowing across the surfaces, the so-called surface-to-volume effect is produced. As the equations in the subdomains are solved concurrently the synchronization of local solutions is imposed iteratively. Such synchronization guarantees the model convergence, although it may slow down the complete work. In such cases, a static and or a priori decomposition could not ensure a well-balanced workload, while a way to re-partition the mesh so that the subdomains maintain a nearly equal number of observations plays an essential role in the success of any effective DD approach. A dynamic load balancing algorithm allows minimal data movement restricted to the neighbouring processors. It is achieved by considering a connected graph induced by the mesh partitioning whose vertices represent a subdomain associated with a scalar representing the number of observations on that subdomain. Once the domain has been partitioned a load balancing schedule (scheduling step) should make the load on each subdomain equal to the average load providing the amount of load to be sent to adjacent subdomains (migrations step). The most intensive kernel is the scheduling step which defines a schedule for computing the load imbalance (which we quantify in terms of number of observations) among neighbouring subdomains. Such quantity is then used to update the shifting the adjacent boundaries of subdomains which are finally re-mapped to achieve a balanced decomposition. We are assuming that load balancing is restricted to the neighbouring domains so that we reduce the overhead processing time. Finally, we use a diffusion type scheduling algorithm minimizing the Euclidean norm of data movement. The resulting constrained optimization problem leads to the solution of the related normal equations whose matrix is associated with the decomposition graph [18].

Communication. The approach we introduce is extremely easy to implement on emerging parallel architectures. This is due to the ability to exploit multiple levels of parallelism depending both on the granularity of the operations and on the mapping of the target architecture. In particular, theoretical performance analysis in terms of the scale-up tells us that if the application needs to reach a prescribed time-to-solution (strong scaling), we can exploit the high performance of emerging GPUs. In this case, the number of processing elements can increase while the local problem size is fixed according to the memory constraints of GPU; the scale-up factor increases with a fixed surface-to-volume ratio. On the other hand, for computationally intensive applications, it is preferable to exploit the weak scaling of clusters of distributed memory multiprocessors. By fixing the number of processors while the local problem size may increase, according to the requirements of the application, the scale-up factor is kept constant while the surface-to-volume ratio decreases supporting the overall efficiency.

It is worth noting that the efficient implementation of any model-level techniques on given computing platforms is essential to improve physical resource efficiency. In this regard, we are mainly interested in investigating the integration of the STRADS interface into the framework [8]. Here we start presenting the feasibility analysis of the proposed approach and validate its scalability using the high-performance hybrid computing architecture of the SCoPE (Sistema Cooperativo Per Elaborazioni scientific multidisciplinary) data center, located at the University of Naples Federico II. The architecture is composed of 8 nodes consisting of distributed memory DELL M600 blades. The blades are connected by a 10 Gigabit Ethernet technology and each of them is composed of 2 Intel Xeon @ 2.33 GHz quadcore processors sharing the same local 16 GB RAM memory for a number of 8 cores per blade and of 64 total cores. We study the performance by using Parallel Computing Toolbox of MATLAB R2014b.

The rest of the article is described as follows. Section 2 introduces ML and DL as identification problems while in Section 3 by restricting the analysis on Reproducing Kernel Hilbert Spaces we cast DKL (Deep Kernel Learning) problems into a framework exploiting the connection with the theory of inverse and ill-posed problems. Finally, we formulate DKL problems as Concatenated Tikhonov Regularization functionals ($\mathcal{CTR}$). Section 4 focuses on the new approach to $\mathcal{CTR}$ reduction, while the algorithm is presented in Section 5 with its performance analysis. Section 6 discusses the main outcomes of this analysis.

2 BASIC MATHEMATICAL CONCEPTS OF LEARNING FROM EXAMPLES

Identification problems concern the formulation of models. ML in its universal expression can be regarded as an *identification* mathematical problem.

According to [19], we say that a model is identified if it is in a unique form that enables subsequently unique estimates of its parameters from the available data. In order to better clarify that ML is in fact an identification problem, we review basic mathematical definitions of ML [20, 1, 2, 21].

Definition 1 (ML – Problem I)**.** We are given the input space X, which we assume to be a compact subset of $\Re$, the output space Y, which is a subset of $\Re$ contained in $[-M, M]$ for some $M \geq 0$ and the (training) data set $S := (x_i, y_i)$, for $i = 1, \ldots, N$, which are samples in $X \times Y$.

Given the data set S the aim of any (supervised) learning problem is to find the function $\Phi : X \to Y$ (also known the predictor) which is able to well estimate any new output $y \in Y$ once a new input $x \in X$ is given (or, function Φ generalizes the output from unseen input).

Solving identification problems, e.g. finding the predictor, necessitates finding a way of incorporating information coming from data inside the most likely probable model. We recognize that identification problem is logically prior to data estimation prob-

lem. More precisely, we could not understand from the data set which specific relationship it is representing unless we get a particular form of it.

Moreover, as we will see later, the typical approach towards the solution of any identification problem is to define a metric (the loss function) depending on the data and on the most likely model Φ. Therefore the identification problem is treated as an approximation problem [22].

To understand how such an approximation problem comes out in any ML problems, we first note that ML Problem I belongs to the so-called Data Estimation problems, introduced by R. E. Kalman, in his pioneering work in 1960 [23].

Definition 2 (Data Estimation)**.** Given the points

$$(t_i, y_i), \quad i = 1, 2 \quad t_i \in [0, T],$$

where $y_i = x_1(t_i) + \epsilon_i$, to calculate $x_1(\widetilde{t})$, $\widetilde{t} \in [0, T]$.

Depending on the position of $\widetilde{t}$ with respect to t_1 and t_2 in [23] this problem was characterized as follows:

- $t_1 < \widetilde{t} < t_2$: data smoothing (fitting of data);
- $\widetilde{t} = t_i$: data filtering;
- $\widetilde{t} > \{t_1, t_2\}$: data prediction (data mining).

and, in general, it was called data estimation.

Since C. F. Gauss in 1795, in scientific computing Data Estimation problems are solved as approximation problems. C. F. Gauss, at age 18, in his study of the orbits of the planets stated that [24]:

> "[...] measurements are affected by errors and so are all obtained from these computations, therefore, the only way to get information about the problem at hand is to compute an approximation of the nearest and most practicable solution possible. This can be done by using a suitable combination of the experimental measurements, which must be in number than those of the unknown parameters, and starting from an approximate knowledge of the orbit (to be calculated), which will be corrected in order to describe as accurately as possible the experimental observations."

In other words, C. F. Gauss highlighted the main ingredients needed for the computation of the solution of an approximation problem:

1. the use of experimental measurements in a number higher than that of the unknown parameters;
2. the identification of the model linking the quantities and the known unknowns;
3. the calculation of the minimum distance between the known values and those obtained by solving the model.

Following Gauss, we now refine ML Problem I by contextualizing it into the Deep Learning models.

3 LEARNING FROM EXAMPLES: A LARGE SCALE INVERSE ILL-POSED PROBLEM

Deep Learning (DL) is the enhanced version of ML where models deal with complex tasks by learning from subtasks. In particular, several nonlinear functions are stacked in hierarchical architectures to learn multiple levels of representation from input data (the higher-level features are defined in terms of lower-level ones). Each function transforms the representation at one level into a much more abstract representation at a higher level. The core building block of DL are mathematical functions called artificial neurons [25].

Definition 3 (Artificial Neuron)**.** An artificial neuron with weights $w_1, \ldots, w_N \in \Re$, bias $b \in \Re$ and (activation) function $\rho : \Re \to \Re$ is defined as the function $f : \Re^N \to \Re$ given by

$$f(x_1, \ldots x_N) = \rho\left(\sum_{i=1}^{N} x_i w_i - b\right). \tag{1}$$

The simplest form of deep networks is the Deep Feedforward network (or deep neural network, DNN) described as a collection of artificial neurons which are organized in layers. Neural networks are basically made up of three layers: an input layer, a hidden layer, and an output layer. Adding two or more hidden layers to a traditional neural network we obtain the DNN. Each layer has a set of units. The units between adjacent layers are inter-connected and each connection is associated with a weight parameter. In each layer, the input vector first goes through an affine linear transformation and then pass through the activation function [26, 25].

Definition 4 (DL – Problem I)**.** Let $d \in \mathcal{N}$ be the dimension of the input layer, L the number of layers, $N_0 := d$, N_l, where $l = 1, \ldots, L$ the dimensions of the hidden and last layer, ρ, an activation function and, for $l = 1, \ldots, L$ T_l be the affine linear functions

$$T_l\mathbf{x} = W^{(l)}\mathbf{x} + b^{(l)} \tag{2}$$

with $W^{(l)} \in \Re^{N_l \times N_{l-1}}$ being the weight matrices, $\mathbf{x} = (x_1, \ldots, x_d)$, and $b^{(l)} \in \Re^{N_l}$ the bias vector of the l^{th} layer. Then the composite function $\Phi : \Re^d \to \Re^{N_L}$ given by

$$\Phi := T_L \circ \rho \cdots \circ \rho \circ T_1,$$

such that

$$\Phi(\mathbf{x}) := T_L\rho(T_{L-1}\rho(\ldots \rho(T_1(\mathbf{x}))))$$

is a (deep) neural network of depth L. The activation function is applied at each hidden unit to achieve the nonlinearity of neural network models. Commonly used activation functions include sigmoid, hyperbolic tangent, and rectified linear unit (ReLU) functions [26, 25].

3.1 The Learning Inverse Ill-Posed Problem

Kernel methods have been successfully applied to a wide variety of learning problems. These methods map data from the input space to a Reproducing Kernel Hilbert Space (RKHS) by using a kernel function which computes the scalar product between data points in the RKHS.

In this section, by restricting the analysis on RKHS, we cast Deep Learning models into a functional analysis framework exploiting the connection with the theory of Tikhonov Regularization [21, 27].

Formally, an RKHS is a Hilbert space of functions on some domain in which all evaluation functionals are bounded linear functionals [28].

Definition 5 (RKHS)**.** Let H be a Hilbert space of functions from $\Omega \subset \Re^N$ to $\Re$, equipped with the scalar product $\langle \cdot, \cdot \rangle$. H is a RKHS if exists a function

$$K : \Omega \times \Omega \to \Re, \tag{3}$$

which is called a Reproducing Kernel (RK) of H, satisfying:

1. $\forall \mathbf{x} \in \Omega$, $k_x = K(\cdot, \mathbf{x}) \in H$;
2. $\forall \mathbf{x} \in \Omega$, $\forall f \in H$, $\langle f, K(\cdot, \mathbf{x}) \rangle = f(\mathbf{x})$.

The RK is always symmetric and positive definite. Every RKHS has a unique RK. Conversely, for every positive definite function K there exists a unique RKHS with K as its RK.

Definition 6 (ML in RKHS – Problem II)**.** Let H be a Hilbert space of functions from X to $\Re$. Given the setup as in Definition 1 of ML Problem I, the (supervised) learning problem concerns the computation of the predictor Φ in an hypothesis space H which is a RKHS on the set X.

Following result leads to solution of ML in RKHS – Problem II for noiseless data (e.g. data interpolation) [29, 30].

Theorem 1 (Representer Theorem)**.** The function

$$\Phi = \sum_{k=1}^{N} \alpha_k K(\cdot, x_k)$$

is the unique minimizer of the Hilbert space norm in H under all functions $f \in H$ such that

$$f(x_i) = y_i.$$

The coefficients α_k can be calculated from the linear system

$$\mathbf{A}\alpha = \mathbf{y},$$

where $A_{ij} = K(x_i, x_j)$, $\alpha = (\alpha_1, \dots \alpha_N)^T$, $\mathbf{y} = (y_1, \dots, y_N)^T$ and $\mathbf{A} \in \Re^{N \times N}$.

This result states that ML Problem II in RKHS leads to the inverse problem consisting in computing the coefficients $\alpha_1, \ldots \alpha_N$ inverting the kernel matrix $\mathbf{A}$; since $\mathbf{A}$ is positive definite, it is also invertible, and the solution is unique.

Definition 7 (ML in RKHS – An Inverse Problem)**.** Let H be a Hilbert space of functions from X to $\Re$. Given the setup as in Definition 1 of ML Problem I, the (supervised) learning problem consists in the solution of the linear system

$$\mathbf{A}\alpha = \mathbf{y}, \tag{4}$$

where $\mathbf{A} \in \Re^{N \times N}$ is the kernel matrix of H.

Solving inverse problems can be very challenging for the following reasons: small changes in the data values may lead to changes in Φ i.e., the kernel matrix A can be very ill-conditioned (in the learning context this is the so-called overfitting problem) and the ill-posed problem.

The characterization of ill-posed mathematical problems dates back to the early years of the last century (J. Hadamard, 1902) and reflects the belief of the mathematicians of that time to be able to describe uniquely and completely each physics problem.

As a result, a problem was ill-posed when, from the mathematical point of view, it presents anomalies and for this reason it could certainly not correspond to a physical event. Therefore, for some years, ill-posed problems were not taken into consideration by mathematicians. The first comprehensive treatment of ill-posed problems is due in 1965, to A. N. Tikhonov, which described the concept of solution for ill-posed problem and introduced the regularization methods [31]. This solution is obtained by solving a best approximation problem minimizing the sum of two terms: the first is a combination of the residual between data and predicted outputs (the so-called data fitting term) in an appropriate norm, and the second is a regularization term that penalizes unwanted features of the solution. The inverse problem thus leads to a nonlinear variational problem in which the forward simulation model is embedded in the residual term. Hence, regularization methods are used to introduce prior knowledge [32].

Definition 8 (RML – Regularized ML Problem III)**.** Given the setup of ML Problem II as in Definition 6, the predictor Φ is defined such that

$$\Phi^* = argmin_{\Phi \in H} \sum_{i=1}^{N} \mathcal{L}(\Phi(x_i), y_i), \tag{5}$$

where the operator $\mathcal{L}$ is the regularization operator measuring the goodness of fit to data.

This viewpoint does not guarantee to compute acceptable solutions, because continuous dependence of the solution on the data (guaranteed by the regularization

methods) is necessary but not sufficient to get numerical stability. In 1988, James Demmel discussed the relationship between an ill-posed and conditioning of a problem, investigating the probability that a numerical analysis problem is difficult. In the meanwhile, P. C. Hansen introduced the so-called discrete ill-posed problems, to emphasize the huge condition number of rank-deficient discrete problems arising from the discretization of ill-posed problems. The key point is the computation of regularization parameters which are able to balance the accuracy of the solution and the stability of its computation [33, 34, 35, 36].

A classical choice for $\mathcal{L}$ is Tikhonov Regularization (TR). Standard TR method consists in replacing the linear system in (4) with the constrained least square problem

$$\alpha^* = \underset{\alpha}{argmin}\, \mathcal{L} = \underset{\alpha}{argmin}\, \|\mathbf{A}\alpha - \mathbf{y}\|_2 + \lambda \|\mathbf{Q}\alpha\|_2, \tag{6}$$

where $\mathbf{Q}$ is referred to as the regularization matrix and the scalar λ is known as the regularization parameter. The matrix $\mathbf{Q}$ is commonly chosen to be the identity matrix; however, if the desired solution has particular known properties, then it may be meaningful to let $\mathbf{Q}$ be a scaled finite difference approximation of a differential operator or a scaled orthogonal projection. Finally, $\|\cdot\|_2$ denotes the L^2-norms in $\Re^N$. For an introduction to the solution of this kind of problems we refer to [31].

Regularization parameter λ influences condition number of $\mathcal{L}$. By using Theorem 1, the function

$$\Phi = \sum_{k=1}^{N} \alpha_k^\lambda K(\cdot, x_k)$$

is the unique minimizer of (5), where coefficients α_k^λ can be calculated from the normal equations arising from (6):

$$(\mathbf{A}^T\mathbf{A} + \lambda \mathbf{Q}^T)\alpha^\lambda = \mathbf{A}^T\mathbf{y},$$

where $A_{ij} = K(x_i, x_j)$, $\alpha^\lambda = (\alpha_1^\lambda, \ldots . \alpha_N^\lambda)^T$ and $\mathbf{y} = (y_1, \ldots, y_N)^T$.

By proceeding in the same way, e.g. by considering DL models in RKHS, we can even employ TR [37] to solve DL problems.

Definition 9 (RDL – Regularized DL Problems)**.** Given the DL problem setup as in Definition 4, given L nonlinear functions $g_i(\mathbf{x}) := \rho \circ T_i(g_{i-1}(\mathbf{x}))$, where $g_0(\mathbf{x}) = \mathbf{x}$, the Regularized DL (RDL) consists in computing the function g_i^* such that

$$\{g_i^*\}_{i=1,\ldots,L} = \underset{g_i \in H}{argmin}\, \mathcal{J}(g_i) \tag{7}$$

with

$$\mathcal{J}(g_i) = \sum_{i=1}^{L} \Theta_i \|g_i\|_2 + \sum_{i=1}^{L} \|\mathcal{L}(g_L \circ g_{L-1} \circ \ldots g_1(\mathbf{x}), \mathbf{y})\|, \tag{8}$$

where $\mathcal{L}$ is defined in (6).

In particular, L-layer Deep Kernel Learning (DKL) models are hybrid DL problems on RKHS. They combine the flexibility of kernel methods with the structural properties of DL methods. DKL methods build a kernel by non linearly transforming the input vector of data before applying an outer kernel [38, 30, 29, 39, 40, 20, 41, 42]. Formally, we assume that each function $g_i \in H_i$ for $i = 1, \ldots, L$ where H_i are RKHS with associated kernel K_i.

In this case, the solution of a L-layer DKL model is given by a linear combination of at most N basis functions in each layer and the following Concatenated Representer Theorem subsists:

Theorem 2 (Concatenated Representer Theorem)**.** Let $H_1, \ldots, H_L$ be RKHS of functions with domain D_i, and range $R_i \subseteq \Re^{d_i}$ where $R_i \subseteq D_{i-1}$ and $D_L = \Omega$ for $i = 2,\ldots, L$. Let $\mathcal{L}$ be the regularization functional in (8). Then, a set of minimizer $f_i \in H_i$, $i = 1, \ldots, L$ of

$$J(f_1, \ldots, f_L) = \sum_{i=1}^{L} \mathcal{L}(y_i, f_1 \circ \cdots \circ f_L(x_i)) + \sum_{i=1}^{L} \|f_l\|_2^2 \tag{9}$$

fulfills $f_i \in V_i \subset H_i$ with

$$V_i = span\{K_i(f_{i+1} \circ \ldots f_L(x_j)), \cdot)e_{k_i}), for\, j = 1 \ldots N \; and \, k_i = 1, \ldots d_i\}.$$

This result enables us to consider the Regularized DKL (RDKL) Problem instead of DKL, and, by using TR in (8), we arrive at the computation of the minimizer of the following functional which we call Concatenated Tikhonov Regularization ($\mathcal{CTR}$):

$$\mathcal{CTR}(f_1 \circ f_2 \circ \cdots \circ f_L)(\mathbf{x})) = \mathcal{TR}(f_1) + \mathcal{TR}(f_2) + \ldots \mathcal{TR}(f_{L-1}). \tag{10}$$

In particular, each quadratic term in (6) can be expressed in terms of the kernel matrix A^l of each RKHS, for $l = 1, \ldots, L$:

$$A^l_{ij} = K(f_1 \circ f_2 \circ \cdots \circ f_L(x_i), f_1 \circ f_2 \circ \cdots \circ f_L(k_j)$$

leading to a nonlinear least square problem [43] whose dimension is $\sum_{i=1}^{L} N \cdot d_i$ Interested readers can found the detailed analysis for a two-layer least square problem in [44].

4 PROBLEM REDUCTION

This nonlinear least-squares problem is typically considered data intensive with N larger than 10^{10} and L larger than 1000 (ResNet comprises 1202 layers and the number of layers grew about 2.3× each year). So how to speed the time-to-solution is an interesting and active research direction. In this context, we provide a mathematical approach based on domain decomposition of $\mathcal{CTR}$ which starts from data decomposition then uses a partitioning of the solution and of the modified functional.

Starting from the $\mathcal{CTR}$ loss functional, we define local $\mathcal{CTR}$ functional on sub sets of data and we prove that the minimum of the "global" functional can be obtained by collecting the minimum of each "local" functional. We prove that the "local" inverse problems are equivalent. As a result, we may say that the proposed approach is loseless keeping the reliability of the global solution [45, 46].

4.1 Basic Concepts

In this section we introduce some concepts and notations we need to use. We give a precise mathematical setting for space and function decomposition then we state some notations used later. In particular, we first introduce the function and domain decomposition, then by using restriction and extension operators, we associate to the domain decomposition a functional decomposition. So, we may prove the following result: the minimum of the global functional, defined on the entire domain can be regarded as a piecewise function obtained by collecting the minimum of each local functional.

Definition 10 (Data Decomposition)**.** Let Ω be a finite numerable set such that $card(\Omega) = N$. Let

$$\Omega = \bigcup_{i=1}^{p} \Omega_i \quad card(\Omega_i) = r_i \tag{11}$$

be a decomposition of the domain Ω into a sequence of overlapping sub-domains Ω_i, where $r_i \leq N$ and $\Omega_i \cap \Omega_j = \Omega_{ij} \neq \emptyset$ when the subdomains are adjacent.

Associate to the decomposition (11), we give the following:

Definition 11 (The Restriction and the Extension Operator)**.** If $\mathbf{w} = (w_i)_{i\in\Omega} \in \Re^N$ then

$$RO_i(\mathbf{w}) := (w_i)_{i\in\Omega_i} \in \Re^{r_i}$$

is the restriction operator acting on $\mathbf{w}$. In the same way, if $\mathbf{z} = (z_i)_{i\in\Omega_i}$, then it is

$$EO_i(\mathbf{z}) := (\tilde{z}_k)_{k\in\Omega} \in \Re^N,$$

where

$$\tilde{z}_k := \begin{cases} z_k, & k \in \Omega_i, \\ 0, & \text{elsewhere} \end{cases} \tag{12}$$

is the extension operator acting on $\mathbf{z}$.

We shall use the notations $RO_i(\mathbf{w}) \equiv \mathbf{w}^{\mathbf{RO_i}}$ and $EO_i(\mathbf{z}) \equiv \mathbf{z}^{\mathbf{EO_i}}$.

Remark 1. For any vector $\mathbf{w} \in \Re^N$, associated to the domain decomposition (11), it results that

$$\mathbf{w} = \sum_{i=1,p} EO_i \left[\mathbf{w}^{RO_i}\right]. \tag{13}$$

The summation

$$\mathbf{w} := \sum_{i=1,p} \mathbf{w}^{EO_i} \tag{14}$$

is such that, for any $j \in \Omega$:

$$RO_j[\mathbf{w}] = RO_j \left[\sum_{i=1,p} \mathbf{w}^{EO_i} \right] = \mathbf{w}^{RO_j}.$$

Definition 12 (The Functional Restriction Operator)**.** Let

$$J(\mathbf{w}) : \mathbf{w} \in \Re^N \mapsto J(\mathbf{w}) \in \Re$$

be the least square operator as defined in (6)

$$J(\mathbf{w}) = \|\mathbf{A}\mathbf{w} - \mathbf{y}\|_2 + \lambda \|\mathbf{w}\|_2 \tag{15}$$

defined in $\Re^N$, where $\lambda > 0$ is the regularization parameter. For simplicity of notations we let $\mathbf{Q} = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix. We generalize the definition of the restriction operator RO_i acting on J, as follows:

$$RO_i[J] : J(\mathbf{w}) \mapsto RO_i[J(\mathbf{w})], \tag{16}$$

where $RO_i[J(\mathbf{w})] = \|\mathbf{A}^{RO_i}\mathbf{w}^{RO_i} - \mathbf{y}^{RO_i}\|_2 + \lambda \|\mathbf{w}^{RO_i}\|_2$ and

$$J(\mathbf{w}^{RO_i}) \mapsto \begin{cases} J(\mathbf{w}^{RO_i}), & \forall j,\ \Omega_i \cap \Omega_i = 0, \\ \frac{1}{2} J(\mathbf{w}^{RO_i}), & \exists j : i \in \Omega_i \cap \Omega_j. \end{cases}$$

For simplicity of notations, and also for underlining that the restriction operator is associated to the domain decomposition (11) we pose:

$$RO_i[J] \equiv J_{\Omega_i}.$$

Definition 13 (The Functional Extension Operator)**.** We generalize the definition of the extension operator EO_i acting on J_{Ω_i} as

$$EO_i : J_{\Omega_i} \mapsto J_{\Omega_i}^{EO_i},$$

where

$$EO_i[J_{\Omega_i}] : \mathbf{w} \mapsto \begin{cases} J(EO_i(\mathbf{w}^{RO_i})), & i \in \Omega_i, \\ 0, & \text{elsewhere.} \end{cases} \tag{17}$$

We note that the (17) can be written as

$$J_{\Omega_i}^{EO_i}(\mathbf{w}) = EO_i[J_{\Omega_i}](\mathbf{w}) = J(EO_i(RO_i[\mathbf{w}])). \tag{18}$$

Proposition 1 (Functional Reduction)**.** Let $\Omega = \bigcup_{i=1}^{p} \Omega_i$ be the decomposition defined in (11) and let J be the functional defined in (15). It holds:

$$J \equiv \sum_{i=1,p} J_{\Omega_i}^{EO_i}, \tag{19}$$

where

$$J_{\Omega_i} : \Re^{r_i} \mapsto \Re.$$

Proof. From the (18) it follows that, if $\mathbf{w} \in \Re^N$, then

$$\sum_{i=1,p} J_{\Omega_i}^{EO_i}(\mathbf{w}) = \sum_{i=1,p} EO_i[J_{\Omega_i}](\mathbf{w}) = \sum_{i=1,p} J_{\Omega_i}(RO_i[\mathbf{w}]). \tag{20}$$

From (13), (17) and (20) it results that

$$\sum_{i=1,p} [J_{\Omega_i}(RO_i[\mathbf{w}])]^{EO_i} = \sum_{i=1,p} J\left(EO_i((\mathbf{w}^{RO_i}))\right) = J\left[\sum_{i=1,p} (\mathbf{w}^{RO_i})^{EO_i}\right] = J(\mathbf{w}). \tag{21}$$

From (20) and (21), the (19) follows. □

4.2 TR Reduction

Let:

$$\mathbf{w}_{\lambda}^{TR} = argmin\, J(\mathbf{w}) \tag{22}$$

We now introduce the *local* $\mathcal{CTR}$ functionals which describes the local problems on each sub-domain Ω_i.

Definition 14. Let $\Omega = \bigcup_{i=1}^{p} \Omega_i$ be the domain decomposition in (11). For any vector $\mathbf{w} \in \Re^N$, let: $J_{\Omega_i} = RO_i[J]$ be the operator as defined in (16) and let

$$\tilde{J}_{\Omega_{ij}} : \mathbf{w}^{RO_i} \mapsto \tilde{J}_{\Omega_{ij}}(\mathbf{w}^{RO_i}) \in \Re$$

be a quadratic operator defined in $\Re^{card(\Omega_{ij})}$. The operator

$$J_{\Omega_i}(\mathbf{w}^{RO_i}) := J_{\Omega_i}(\mathbf{w}^{RO_i}) + \omega_i \tilde{J}_{\Omega_{ij}}(\mathbf{w}^{RO_{ij}}) \tag{23}$$

is the local $\mathcal{CTR}$ functional defined on Ω_i and on $\Omega_i \cap \Omega_j$. Parameters ω_i are local regularization parameters. Then

$$\mathbf{w}_{\lambda,\omega_i}^{TR_i} := \underset{\mathbf{w}^{RO_i}}{argmin}\, J_{\Omega_i}\left(\mathbf{w}^{RO_i}\right). \tag{24}$$

is the solution of the local TR problem. Since the local functional is quadratic this solution is also unique, once index i has been fixed.

Remark 2. From (19) it follows that

$$J_{D^3L} := \sum_{i=1,p} J_{\Omega_i}^{EO_i} = \underbrace{\sum_{i=1,p} J_{\Omega_i}^{EO_i}}_{J} + \underbrace{\sum_{i=1,p} \omega_i \tilde{J}_{\Omega_{ij}}^{EO_i}}_{\mathcal{C}}. \tag{25}$$

In practice, J_{D^3L} is obtained from a restriction of the $\mathcal{CTR}$ functional J_Ω in (22), and adding a *local* functional defined on the overlapping regions in Ω_{ij}. This is done in order to enforce a sufficient continuity of local solutions onto the overlap region between adjacent domains Ω_i and Ω_j.

Operator $\tilde{J}_{\Omega_{ij}}$ can be suitably defined according to the specific requirements of the solution of the $\mathcal{CTR}$ problem.

The following result relates the solution of $\mathcal{CTR}$ problem in (22) to the solutions of the local TR problems in (23). From simplicity of notations, in the following theorem we assume that $\mathcal{C}$ is quadratic functional. The result still holds if this is a more general convex functional.

Theorem 3. Let

$$\Omega = \bigcup_{i=1,p} \Omega_i$$

be a domain decomposition of Ω defined in (11), and let (25) be the associated functional decomposition. Then let $\mathbf{w}_\lambda^{TR}$ be defined in (22) and let $\widehat{\mathbf{w}}_\lambda^{TR}$ be defined as follows:

$$\widehat{\mathbf{w}}_\lambda^{TR} = \sum_i (\mathbf{w}_{\lambda,\omega_i}^{TR_i})^{EO_i}.$$

It is:

$$\widehat{\mathbf{w}}_\lambda^{TR} = \mathbf{w}_\lambda^{TR}.$$

Proof. J_Ω is convex, as well as all the functionals J_{Ω_i}, so their (unique) minimum, $\mathbf{w}_\lambda^{TR}$ and $\mathbf{w}_{\lambda,\omega_i}^{TR_i}$, respectively, are obtained as zero of their gradients, i.e.:

$$\nabla J[\mathbf{w}_\lambda^{TR}] = 0, \quad \nabla J_{\Omega_i}[\mathbf{w}_{\lambda,\omega_i}^{TR_i}] = 0. \tag{26}$$

From (23) it follows that

$$\nabla J_{\Omega_i}[\mathbf{w}_{\lambda,\omega_i}^{TR_i}] = \nabla J_{\Omega_i}[\mathbf{w}_{\lambda,\omega_i}^{TR_i}]. \tag{27}$$

From (12) it is:

$$\mathbf{w}_{\lambda,\omega_i}^{RO_i} = \sum_{j=1,p} (\mathbf{w}_{\lambda,\omega_i}^{TR_j})^{EO_j}, \quad on \quad \Omega_i. \tag{28}$$

From (26), (27) and (28), it follows that

$$0 = \nabla J_{\Omega_i}(\mathbf{w}_{\lambda,\omega_i}^{TR_i}) = \nabla J_{\Omega_i}^{EO_i} \left(\sum_{j=1,p} (\mathbf{w}_{\lambda,\omega_i}^{TR_i})^{EO_i} \right). \tag{29}$$

By summing each equation in (29) for $i = 1, \ldots, p$ on all sub-domains Ω_i, from (29) it follows that:

$$\sum_i \nabla J^{EO_i}_{\Omega_i} \left(\sum_j (\mathbf{w}^{EO_j}_{\lambda,\omega_j})^{TR} \right) = 0 \Leftrightarrow \sum_i \nabla J^{EO_i}_{\Omega_i}(\hat{\mathbf{w}}^{TR}\lambda) = 0. \tag{30}$$

From the linearity of the gradients of J_{Ω_i}, it is

$$\sum_i \nabla J^{EO_i}_{\Omega_i}(\widehat{\mathbf{w}}^{TR}_{\lambda}) = \nabla \sum_i J^{EO_i}_{\Omega_i}(\widehat{\mathbf{w}}^{TR}_{\lambda}) = \nabla J(\widehat{\mathbf{w}}^{TR}_{\lambda}). \tag{31}$$

Hence, from (31) it follows

$$\sum_i \nabla J_{\Omega_i}(\hat{\mathbf{w}}^{TR}_{\lambda}) = 0 \Leftrightarrow \nabla J(\hat{\mathbf{w}}^{TR}_{\lambda}) = 0.$$

Finally,

$$\nabla J(\widehat{\mathbf{w}}^{TR}_{\lambda}) = 0 \Rightarrow \widehat{\mathbf{w}}^{TR}_{\lambda} \equiv \mathbf{w}^{TR}_{\lambda},$$

where the last equality holds because the minimum is unique. □

In conclusion, the minimum of J_Ω, can be obtained by patching together the minima of J_{Ω_i}. This means that the accuracy per parameter metric is highly improved in this way.

5 THE D^{3}L PARALLEL ALGORITHM

Definition 15. Let $\mathcal{A}^{loc}_{D^3L}$ denote the algorithm solving the local $\mathcal{CTR}$ functional. Parallel D^{3}L algorithm is symbolically denoted as

$$\mathcal{A}^{D^3L} := \bigcup_{i=1,p} \mathcal{A}^{loc}_{D^3L}(\Omega_i). \tag{32}$$

Parallel D^{3}L algorithm is described by Algorithm 1. Similarly, the $\mathcal{A}^{loc}_{D^3L}$ is described by Algorithm 2. $\mathcal{A}^{loc}_{D^3L}$ computes a local minimum of $\mathbf{J}_{\Omega_i}$ by solving the normal equations arising from the linear least squares (LLS) problem [47]. We note, in line 6, the exchange of $\mathbf{w}^l_i$. This is done in order to enforce a sufficient continuity of local solutions onto the overlap region between adjacent domains Ω_i and Ω_j.

5.1 Performance Analysis

We use time complexity and scalability as performance metrics. Our aim is to highlight the benefits which arise from using the decomposition approach instead of solving the problem on the whole domain.

Algorithm 1

procedure D^3L(in: $\mathbf{A}$, $\mathbf{y}$; out: $\mathbf{w}_\lambda^{\mathbf{TR}}$)
  % Domain Decomposition Step
  repeat
    $l := l + 1$
    Call Loc_LLS (in: $\mathbf{A}^{RO_i}$, $\mathbf{y}^{RO_i}$; out: $\mathbf{w}_i^l$)
    Exchange $\mathbf{w}_i^l$ between adjacent subdomains
  until $\|\mathbf{w}_i^l - \mathbf{w}_i^{l-1}\| < eps$
  % End Domain Decomposition Step
  Gather of $\mathbf{w}_i^l : \mathbf{w}^{\mathbf{TR}} = argmin_i \left\{ J\left(\mathbf{w}_i^l\right) \right\}$
end procedure

Algorithm 2 Algorithm 2

procedure Loc-LLS($\mathbf{A}^{\mathbf{RO_i}}$, $\mathbf{y}^{RO_i}$; out: $\mathbf{u}_i^l$)
  Initialize $l := 0$;
  repeat
    Compute $\delta\mathbf{u}_i^l = argmin\, \mathbf{J}_{\Omega_i}$ by solving the normal equations system
    Update $\mathbf{u}_i^l = \mathbf{u}_i^l + \delta\mathbf{u}_i^l$
    Update $l = l + 1$
  until (convergence is reached)
end procedure

Definition 16. A uniform decomposition of Ω is such that if we let

$$size(\Omega) = N$$

be the size of the whole data domain, then each subdomain Ω_i is such that

$$size(\Omega_i) = N_{loc}, \quad i = 1, \ldots, p,$$

where $N_{loc} = \frac{N}{p} \geq 1$.

Various metrics have been developed to assist in evaluating the scalability of a parallel algorithm: speedup, throughput, efficiency are the most used. Each one highlights specific needs and limits to be answered by the parallel software. Since we mainly focus on the algorithm's scalability arising from the proposed framework, we consider the so-called scale-up factor first introduced in [45].

Let $T(\mathcal{A}^{D^3L}(\Omega))$ denote time complexity of $\mathcal{A}^{D^3L}(\Omega)$. The major computational tasks to be performed are as follows:

1. Computation of the kernel $RO_{ji}[\mathbf{A}]$ (the time complexity of such an operation scales as LN^2).
2. Solution of the normal equations, involving at each iteration two matrix-vector products with $RO_{ji}[(\mathbf{A}^T)]$ and $RO_{ji}[\mathbf{A}]$ (whose time complexity scales as L^2N^4).

We pose $d = 4$. We now provide an estimate of the time complexity of each local algorithm, denoted as $T(\mathcal{A}^{Loc}(\Omega_i))$.

The first result straightforwardly derives from the definition of the scale-up factor:

Proposition 2 (Scale-up factor)**.** The (relative) scale-up factor of $\mathcal{A}^{D^3L}(\Omega)$ related to $\mathcal{A}^{loc}(\Omega_i)$, denoted as $Sc_p(\mathcal{A}^{D^3L}(\Omega))$, is

$$Sc_p(\mathcal{A}^{D^3L}(\Omega)) := \frac{1}{p} \times \frac{T(\mathcal{A}^{D^3L}(\Omega))}{T(\mathcal{A}^{loc}(\Omega_i))},$$

where p is the number of subdomains. It is

$$Sc_p(\mathcal{A}^{D^3L}) \geq \alpha(N_{loc}, p)\,(p)^{d-1}, \tag{33}$$

where

$$\alpha(N_{loc}, p) = \frac{a_d + a_{d-1}\frac{1}{N} + \cdots + \frac{a_0}{N_{loc}^d}}{a_d + a_{d-1}\frac{p}{N_{loc}} + \cdots + \frac{a_0(p)^d}{N_{loc}^d}}$$

and

$$\lim_{p \to N_{loc}} \alpha(N_{loc}, p) = \beta \in]0, 1].$$

Corollary 1. If $a_i = 0 \quad \forall i \in [0, d-1]$, then $\beta = 1$, that is,

$$\lim_{p \to N_{loc}} \alpha(N_{loc}, p) = 1.$$

Then,

$$\lim_{N_{loc} \to \infty} \alpha(N_{loc}, p) = 1.$$

Corollary 2. If N_{loc} is fixed, then

$$\lim_{p \to N_{loc}} Sc_p(\mathcal{A}^{D^3L}) = \beta \cdot N_{loc}^{d-1},$$

while if p is fixed, then

$$\lim_{N_{loc} \to \infty} Sc_p(\mathcal{A}^{D^3L}) = const \neq 0.$$

From (33) it results that, considering one iteration of the whole parallel algorithm, the growth of the scale-up factor is essentially one order less than the time complexity of the reduced kernel. In other words, the time complexity of the reduced kernel impacts mostly the scalability of the parallel algorithm. In particular, since parameter d is equal to 4, it follows that the asymptotic scaling factor of the parallel algorithm, with respect to p, is bounded above by three.

Besides the time complexity, scalability is also affected by the communication overhead of the parallel algorithm. The surface-to-volume ratio is a measure of

the amount of data exchange (proportional to surface area of domain) per unit operation (proportional to volume of domain). It is straightforward to prove that the surface-to-volume ratio of the uniform decomposition of Ω is

$$\frac{\mathcal{S}}{\mathcal{V}} = \mathcal{O}\left(\frac{1}{N_{loc}}\right). \tag{34}$$

Theoretical performance analysis in terms of the scale up factor tell us that if the application needs to reach a prescribed time-to-solution (strong scaling), we can exploit the high performance of emerging GPU. In this case, p can increase while N_{loc} is fixed according to memory constraints of GPU; the scale-up factor increases with a fixed surface-to-volume ratio. On the other hand, for computationally intensive applications, it is preferable to exploit the weak scaling of clusters of distributed memory multiprocessors. In fact, by fixing p while N_{loc} may increase according to application requirements, the scaling factor is kept constant, while the surface-to-volume ratio decreases supporting overall efficiency. Summarizing, performance analysis tells us that by looking at the scale-up factor one may find the appropriate mapping of the specific application on the target architecture [48].

6 VALIDATION

The proposed approach has a straightforward application to Convolution Neural Networks (CNN) which are specialized in processing data that has a grid-like topology, such as an image. In this case, the domain Ω is the image while the core building block of the CNN is the convolutional layer. This layer performs a dot product between two matrices, where one matrix is the kernel and the other matrix is a restricted portion of the image. As a result, by decomposing Ω according to the standard two-dimensional block cyclic distribution of the image matrix, which is implemented in linear algebra libraries for parallel matrix computation (MATLAB, ScaLAPACK, PBLAS, ...), we get to the standard data layout of dense matrix on distributed-memory architectures which permits the use of the Level 3 of BLAS during computations on a single node [49].

We validate the algorithm on the DIGITS dataset of MATLAB R2014b, by using the high-performance hybrid computing architecture of the SCoPE (Sistema Cooperativo Per Elaborazioni scientifiche multidiscipliari) data center, located at the University of Naples Federico II. The architecture is composed of 8 nodes consisting of distributed memory DELL M600 blades. The blades are connected by 10 gigabit Ethernet technology and each of them is composed of 2 Intel Xeon @ 2.33 GHz quadcore processors sharing the same local 16 GB RAM memory for a number of 8 cores per blade and of 64 total cores. We validate the algorithm using the Parallel Computing Toolbox of MATLAB R2014b. In Table 1 we report strong scaling results computed using the scale-up factor $Sc_p(\mathcal{A}^{D^3L}(\Omega))$ as given in Proposition 21 and, just to make a comparison with a standard metric, we also report the classical speed-up metric indicated as $S_p(\mathcal{A}^{D^3L}(\Omega))$. Weak scaling scalability is reported in

Table 2 using the scale-up factor.

p	N	$T(\mathcal{A}^{D^3L}(\Omega))$			
1	1.23×10^4	1.56×10^3			
	S/V	$T\left(\mathcal{A}^{loc}(\Omega_i)\right)$	$T^p\left(\mathcal{A}^{D^3L}(\Omega)\right)$	$S_p\left(\mathcal{A}^{D^3L}(\Omega)\right)$	$Sc_p\left(\mathcal{A}^{D^3L}(\Omega)\right)$
2	1.06×10^{-7}	9.8×10^1	0.8×10^3	1.8	1.5
4	6.7×10^{-5}	3.0×10^1	0.48×10^3	3.2	6.5
8	2.5×10^{-5}	2.7×10^{-1}	0.24×10^3	6.5	7.2
16	1.4×10^{-5}	1.2×10^{-2}	0.1×10^3	14.2	8.1

Table 1. Strong scaling results

p	2	4	8	16	32	64
N	3.07×10^3	1.23×10^4	4.90×10^4	1.97×10^5	7.86×10^5	3.15×10^6
Sc_p	5.96×10^0	2.39×10^1	9.54×10^1	3.81×10^2	1.53×10^3	6.10×10^3

Table 2. Weak scaling scalability of $\mathcal{A}^{D^3L}(\Omega)$

7 FUTURE WORK

We underline that we referred to DKL problems which gave rise to convex optimization. More in general, we could address DL problems where, in case of the non convex loss function, we could consider a surrogate convex loss function. An example of such surrogate loss functions is the hinge loss, $\Phi(t) = \max(1 - t, 0)$, which is the loss used by Support Vector Machines SVMs. Another example is the logistic loss, $\Phi(t) = 1/(1+\exp(-t))$, used by the logistic regression model. A natural questions to ask is how much have we lost by this change. The property of whether minimizing the surrogate function leads to a function that also minimizes the loss function is often referred to as consistency [50]. This property will depend on the surrogate function. One of the most useful characterizations was given in [50] and states that if Φ is convex then it is consistent if and only if it is differentiable at zero and $\Phi'(0) < 0$. This includes most of the commonly used surrogate loss functions, including hinge, logistic regression and Huber loss functions.

Following [51, 52] we suggest the employment of the proposed approach on the plentiful literature of Physical Informed Neural Network (PINN) applications, where data are constrained according to a specific physical model generally modelled by evolutive Partial Differential Equations.

8 CONFLICT OF INTEREST

The author declares no conflicts of interests in this paper.

REFERENCES

[1] Poggio, T.—Banburski, A.—Liao, Q.: Theoretical Issues in Deep Networks. Proceedings of the National Academy of Sciences, Vol. 117, 2020, No. 48, pp. 30039–30045, doi: 10.1073/pnas.1907369117.

[2] Poggio, T.—Smale, S.: The Mathematics of Learning: Dealing with Data. Notices of the AMS, Vol. 50, 2003, No. 5, pp. 537–544, https://www.ams.org/journals/notices/200305/fea-smale.pdf.

[3] Thompson, N. C.—Greenewald, K.—Lee, K.—Manso, G. F.: The Computational Limits of Deep Learning. CoRR, 2022, doi: 10.48550/arXiv.2007.05558.

[4] Ben-Nun, T.—Hoefler, T.: Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis. CoRR, 2018, doi: 10.48550/arXiv.1802.09941.

[5] Hestness, J.—Narang, S.—Ardalani, N.—Diamos, G.—Jun, H.—Kianinejad, H.—Patwary, M. M. A.—Yang, Y.—Zhou, Y.: Deep Learning Scaling Is Predictable, Empirically. CoRR, 2017, doi: 10.48550/arXiv.1712.00409.

[6] Mayer, R.—Jacobsen, H. A.: Scalable Deep Learning on Distributed Infrastructures: Challenges, Techniques and Tools. CoRR, 2019, doi: 10.48550/arXiv.1903.11314.

[7] Shazeer, N.—Cheng, Y.—Parmar, N.—Tran, D.—Vaswani, A.—Koanantakool, P.—Hawkins, P.—Lee, H.—Hong, M.—Young, C.—Sepassi, R.—Hechtman, B.: Mesh-TensorFlow: Deep Learning for Supercomputers. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (Eds.): Advances in Neural Information Processing Systems 31 (NeurIPS 2018). 2018, pp. 2135–2135, https://proceedings.neurips.cc/paper_files/paper/2018/file/3a37abdeefe1dab1b30f7c5c7e581b93-Paper.pdf.

[8] Lee, J.—Mukhanov, L.—Molahosseini, A. S.—Minhas, U.—Hua, Y.—Martinez del Rincon, J.—Dichev, K.—Hong, C. H.—Vandierendonck, H.: Resource-Efficient Convolutional Networks: A Survey on Model-, Arithmetic-, and Implementation-Level Techniques. ACM Computing Surveys, Vol. 55, 2023, No. 13s, Art. No. 276, doi: 10.1145/3587095.

[9] Liu, J.—Dutta, J.—Li, N.—Kurup, U.—Shah, M.: Usability Study of Distributed Deep Learning Frameworks for Convolutional Neural Networks. 24th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '18 Deep Learning Day), 2018, https://www.kdd.org/kdd2018/files/deep-learning-day/DLDay18_paper_29.pdf.

[10] Jäger, S.—Zorn, H. P.—Igel, S.—Zirpins, C.: Parallelized Training of Deep NN: Comparison of Current Concepts and Frameworks. Proceedings of the Second Workshop on Distributed Infrastructures for Deep Learning (DIDL '18), ACM, 2018, pp. 15–20, doi: 10.1145/3286490.3286561.

[11] Zhang, X.—McKenna, M.—Mesirov, J.—Waltz, D.: An Efficient Implementation of the Back-Propagation Algorithm on the Connection Machine CM-2. In: Touretzky, D. (Ed.): Advances in Neural Information Processing Systems 2 (NIPS 1989). Morgan-Kaufmann, 1990,

pp. 801–809, https://proceedings.neurips.cc/paper_files/paper/1989/file/e3796ae838835da0b6f6ea37bcf8bcb7-Paper.pdf.

[12] Tokui, S.—Oono, K.—Hido, S.—Clayton, J.: Chainer: A Next-Generation Open Source Framework for Deep Learning. Proceedings of Workshop on Machine Learning Systems (LearningSys) in the Twenty-Ninth Annual Conference on Neural Information Processing Systems (NIPS), 2015, pp. 1–6, http://learningsys.org/papers/LearningSys_2015_paper_33.pdf.

[13] Paszke, A.—Gross, S.—Chintala, S.—Chanan, G.—Yang, E.—DeVito, Z.—Lin, Z.—Desmaison, A.—Antiga, L.—Lerer, A.: Automatic Differentiation in PyTorch. 2017, https://openreview.net/pdf/25b8eee6c373d48b84e5e9c6e10e7cbbbce4ac73.pdf?ref=blog.premai.io.

[14] Seide, F.—Agarwal, A.: CNTK: Microsoft's Open-Source Deep-Learning Toolkit. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'16), 2016, pp. 2135–2135, doi: 10.1145/2939672.2945397.

[15] Mayer, R.—Mayer, C.—Laich, L.: The TensorFlow Partitioning and Scheduling Problem: It's the Critical Path! CoRR, 2017, doi: 10.48550/arXiv.1711.01912.

[16] Chen, T.—Li, M.—Li, Y.—Lin, M.—Wang, N.—Wang, M.—Xiao, T.—Xu, B.—Zhang, C.—Zhang, Z.: MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems. CoRR, 2015, doi: 10.48550/arXiv.1512.01274.

[17] Ooi, B. C.—Tan, K. L.—Wang, S.—Wang, W.—Cai, Q.—Chen, G.—Gao, J.—Luo, Z.—Tung, A. K. H.—Wang, Y.—Xie, Z.—Zhang, M.—Zheng, K.: SINGA: A Distributed Deep Learning Platform. Proceedings of the 23rd ACM International Conference on Multimedia (MM'15), 2015, pp. 685–688, doi: 10.1145/2733373.2807410.

[18] D'Amore, L.—Cacciapuoti, R.: Parallel Framework for Dynamic Domain Decomposition of Data Assimilation Problems: A Case Study on Kalman Filter Algorithm. Computational and Mathematical Methods, Vol. 3, 2021, No. 6, doi: 10.1002/cmm4.1145.

[19] Linz, P.: Theoretical Numerical Analysis: An Introduction to Advanced Techniques. Dover Publications, 2001.

[20] Anselmi, F.—Rosasco, L.—Tan, C.—Poggio, T.: Deep Convolutional Networks Are Hierarchical Kernel Machines. CoRR, 2015, doi: 10.48550/arXiv.1508.01084.

[21] Rosasco, L.—Caponnetto, A.—Vito, E.—Odone, F.—Giovannini, U.: Learning, Regularization and Ill-Posed Inverse Problems. In: Saul, L., Weiss, Y., Bottou, L. (Eds.): Advances in Neural Information Processing Systems 17 (NIPS 2004). The MIT Press, Cambridge, Massachusetts, USA, 2004, pp. 1145–1152, https://proceedings.neurips.cc/paper/2004/hash/33267e5dc58fad346e92471c43fcccdc-Abstract.html.

[22] D'Amore, L.—Murli, A.: Regularization of a Fourier Series Method for the Laplace Transform Inversion with Real Data. Inverse Problems, Vol. 18, 2002, No. 4, pp. 1185–1205, doi: 10.1088/0266-5611/18/4/315.

[23] Kalman, R. E.: A New Approach to Linear Filtering and Prediction Problems. Transactions of the ASME – Journal of Basic Engineering, Vol. 82, 1960, No. 1, pp. 35–45, doi: 10.1115/1.3662552.

[24] Gauss, C. F.: Theoria Motus Corporum Coelestium in Sectionibus Conicis Solem Ambientium. 1809, doi: 10.3931/e-rara-522 (in Latin).

[25] LeCun, Y.—Bengio, Y.—Hinton, G.: Deep Learning. Nature, Vol. 521, 2015, No. 7553, pp. 436–444, doi: 10.1038/nature14539.

[26] Berner, J.—Grohs, P.—Kutyniok, G.—Petersen, P.: The Modern Mathematics of Deep Learning. CoRR, 2021, doi: 10.48550/arXiv.2105.04026.

[27] Schölkopf, B.—Smola, A. J.: Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. The MIT Press, 2002.

[28] Aronszajn, N.: Theory of Reproducing Kernels. Transactions of the American Mathematical Society, Vol. 68, 1950, No. 3, pp. 337–404, doi: 10.1090/S0002-9947-1950-0051437-7.

[29] Bohn, B.—Griebel, M.—Rieger, C.: A Representer Theorem for Deep Kernel Learning. CoRR, 2018, doi: 10.48550/arXiv.1709.10441.

[30] Cho, Y.—Saul, L.: Kernel Methods for Deep Learning. Advances in Neural Information Processing Systems 22 (NIPS 2009), Curran Associates, Inc., 2009, pp. 342–350, https://proceedings.neurips.cc/paper_files/paper/2009/file/5751ec3e9a4feab575962e78e006250d-Paper.pdf.

[31] Tikhonov, A. N.: Solution of Incorrectly Formulated Problems and the Regularization Method. Soviet Mathematics Doklady, Vol. 4, 1963, pp. 1035–1038.

[32] Antonelli, L.—Carracciuolo, L.—Ceccarelli, M.—D'Amore, L.—Murli, A.: Total Variation Regularization for Edge Preserving 3D SPECT Imaging in High Performance Computing Environments. Computational Science – ICCS 2002, Springer Berlin Heidelberg, 2002, pp. 171–180, doi: 10.1007/3-540-46080-2_18.

[33] Arcucci, R.—D'Amore, L.—Carracciuolo, L.: On the Problem-Decomposition of Scalable 4D-Var Data Assimilation Models. 2015 International Conference on High Performance Computing & Simulation (HPCS), 2015, pp. 589–594, doi: 10.1109/HPCSim.2015.7237097.

[34] Arcucci, R.—D'Amore, L.—Celestino, S.—Laccetti, G.—Murli, A.: A Scalable Numerical Algorithm for Solving Tikhonov Regularization Problems. In: Wyrzykowski, R., Deelman, E., Dongarra, J., Karczewski, K., Kitowski, J., Wiatr, K. (Eds.): Parallel Processing and Applied Mathematics (PPAM 2015). Springer, Cham, Lecture Notes in Computer Science, Vol. 9574, 2016, pp. 45–54, doi: 10.1007/978-3-319-32152-3_5.

[35] Arcucci, R.—D'Amore, L.—Pistoia, J.—Toumi, R.—Murli, A.: On the Variational Data Assimilation Problem Solving and Sensitivity Analysis. Journal of Computational Physics, Vol. 335, 2017, pp. 311–326, doi: 10.1016/j.jcp.2017.01.034.

[36] D'Amore, L.: Remarks on Numerical Algorithms for Computing the Inverse Laplace Transform. Ricerche Di Matematica, Vol. 63, 2014, No. 2, pp. 239–252, doi: 10.1007/s11587-013-0176-2.

[37] Unser, M.: A Representer Theorem for Deep Neural Networks. Journal of Machine Learning Research, Vol. 20, 2019, No. 110, pp. 1–30, http://jmlr.org/papers/v20/18-418.html.

[38] Aitchison, L.—Yang, A.—Ober, S. W.: Deep Kernel Processes. In: Meila, M., Zhang, T. (Eds.): Proceedings of the 38th International Conference on Machine Learning. Proceedings of Machine Learning Research (PMLR), Vol. 139, 2021, pp. 130–140, https://proceedings.mlr.press/v139/aitchison21a.html.

[39] Dinuzzo, F.: Learning Functions with Kernel Methods. Ph.D. Thesis. University of Pavia, Pavia, Italy, 2011.

[40] Montavon, G.—Braun, M. L.—Müller, K. R.: Kernel Analysis of Deep Networks. Journal of Machine Learning Research, Vol. 12, 2011, No. 78, pp. 2563–2581, http://jmlr.org/papers/v12/montavon11a.html.

[41] Wang, T.—Zhang, L.—Hu, W.: Bridging Deep and Multiple Kernel Learning: A Review. Information Fusion, Vol. 67, 2021, pp. 3–13, doi: 10.1016/j.inffus.2020.10.002.

[42] Wilson, A. G.—Hu, Z.—Salakhutdinov, R.—Xing, E. P.: Deep Kernel Learning. In: Gretton, A., Robert, C. C. (Eds.): Artificial Intelligence and Statistics. Proceedings of Machine Learning Research (PMLR), Vol. 51, 2016, pp. 370–378, https://proceedings.mlr.press/v51/wilson16.html.

[43] Dennis Jr., J. E.—Schnabel, R. B.: Numerical Methods for Unconstrained Optimization and Nonlinear Equations. SIAM, 1996.

[44] Zhuang, J.—Tsang, I. W.—Hoi, S. C. H.: Two-Layer Multiple Kernel Learning. In: Gordon, G., Dunson, D., Dudík, M. (Eds.): Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research (PMLR), Vol. 15, 2011, pp. 909–917, https://proceedings.mlr.press/v15/zhuang11a/zhuang11a.pdf.

[45] D'Amore, L.—Arcucci, R.—Carracciuolo, L.—Murli, A.: A Scalable Approach for Variational Data Assimilation. Journal of Scientific Computing, Vol. 61, 2014, No. 2, pp. 239–257, doi: 10.1007/s10915-014-9824-2.

[46] Arcucci, R.—D'Amore, L.—Carracciuolo, L.—Scotti, G.—Laccetti, G.: A Decomposition of the Tikhonov Regularization Functional Oriented to Exploit Hybrid Multilevel Parallelism. Journal of Parallel Programming, Vol. 45, 2017, No. 5, pp. 1214–1235, doi: 10.1007/s10766-016-0460-3.

[47] D'Amore, L.—Laccetti, G.—Romano, D.—Scotti, G.—Murli, A.: Towards a Parallel Component in a GPU-CUDA Environment: A Case Study with the L-BFGS Harwell Routine. International Journal of Computer Mathematics, Vol. 92, 2014, No. 1, pp. 59–76, doi: 10.1080/00207160.2014.899589.

[48] D'Amore, L.—Casaburi, D.—Galletti, A.—Marcellino, L.—Murli, A.: Integration of Emerging Computer Technologies for an Efficient Image Sequences Analysis. Integrated Computer-Aided Engineering, Vol. 18, 2011, No. 4, pp. 365–378, doi: 10.3233/ica-2011-0382.

[49] Bertero, M.—Bonetto, P.—Carracciuolo, L.—D'Amore, L.—Formiconi, A.—Guarracino, M.—Laccetti, G.—Murli, A.—Oliva, G.: MedIGrid: A Medical Imaging Application for Computational Grids. Proceedings

International Parallel and Distributed Processing Symposium, IEEE, 2003, 8 pp., doi: 10.1109/ipdps.2003.1213457.

[50] Bartlett, P. L.—Jordan, M. I.—McAuliffe, J. D.: Convexity, Classification, and Risk Bounds. Journal of the American Statistical Association, Vol. 101, 2006, No. 473, pp. 138–156, doi: 10.1198/016214505000000907.

[51] D'Amore, L.—Cacciapuoti, R.: Model Reduction in Space and Time for ab initio Decomposition of 4D Variational Data Assimilation Problems. Applied Numerical Mathematics, Vol. 160, 2021, pp. 242–264, doi: 10.1016/j.apnum.2020.10.003.

[52] D'Amore, L.—Costantinescu, E.—Carracciuolo, L.: A Scalable Space-Time Domain Decomposition Approach for Solving Large Scale Nonlinear Regularized Inverse Ill Posed Problems in 4D Variational Data Assimilation. Journal of Scientific Computing, Vol. 91, 2022, No. 2, Art. No. 59, doi: 10.1007/s10915-022-01826-7.

Luisa D'Amore received her degree in mathematics in 1988 and her Ph.D. in applied mathematics and computer science in 1995. Since 1996, she has worked as a Researcher in numerical analysis, and since 2001, she has been Associate Professor of numerical analysis. Since 2011, she has been working as an Associate Staff Researcher of the ASC (Advanced Scientific Computing) Division of CMCC (Euro Mediterranean Center on Climate Changes). She is a member of the Academic Board for the Ph.D. in mathematics and informatics at the University of Naples Federico II and a Teacher of courses in numerical analysis, scientific computing and parallel computing. Her research activity focuses on scientific computing and the numerical solution of ill-posed inverse problems with applications in image analysis, medical imaging, astronomy, digital restoration of films and data assimilation. The need of computing the numerical solution in a suitable time, induced by the applications, often requires the use of advanced computing architectures. It involves designing and developing algorithms and software capable of exploiting the high performance of emerging computing infrastructures. She is (co)author of about 150 publications in refereed journals and conference proceedings.