On a multilevel Levenberg-Marquardt method for the training of artificial neural networks and its application to the solution of partial differential equations

H. Calandra[a] and S. Gratton[b] and E. Riccietti[b] and X. Vasseur[c]

[a]TOTAL, Centre Scientifique et Technique Jean Féger, avenue de Larribau F-64000 Pau, France; [b]INPT-IRIT, University of Toulouse and ENSEEIHT, 2 Rue Camichel, BP 7122, F-31071 Toulouse Cedex 7, France; [c] ISAE-SUPAERO, University of Toulouse, 10, avenue Edouard Belin, BP 54032, F-31055 Toulouse Cedex 4, France

ARTICLE HISTORY
Compiled February 13, 2020

ABSTRACT
In this paper we propose a new multilevel Levenberg-Marquardt optimizer for the training of artificial neural networks with quadratic loss function. This setting allows us to get further insight into the potential of multilevel optimization methods. Indeed, when the least squares problem arises from the training of artificial neural networks, the variables subject to optimization are not related by any geometrical constraints and the standard interpolation and restriction operators cannot be employed any longer. A heuristic, inspired by algebraic multigrid methods, is then proposed to construct the multilevel transfer operators. We test the new optimizer on an important application: the approximate solution of partial differential equations by means of artificial neural networks. The learning problem is formulated as a least squares problem, choosing the nonlinear residual of the equation as a loss function, whereas the multilevel method is employed as a training method. Numerical experiments show encouraging results related to the efficiency of the new multilevel optimization method compared to the corresponding one-level procedure in this context.

KEYWORDS
Algebraic multigrid method; Artificial neural network; Levenberg-Marquardt method; Multilevel optimization method; Partial differential equation.

1. Introduction

Artificial neural networks (ANNs) have been recently successfully employed in different fields, such as classification and regression, image or pattern recognition, to name a few [5, 25, 27, 32]. Their promising performance encouraged also the spread of special neural networks' hardware implementations, to decrease the computational training times and to provide a platform for efficient adaptive systems [41].

One of the main issues with the use of artificial neural networks is to be able to successfully train them. The training is based on the solution of an optimization problem that can be large-scale as, for highly nonlinear problems, a network with a large number of weights may be necessary to approximate the solution with a sufficient accuracy. Gradient-based methods may exhibit a slow convergence rate, it may be

Contact E. Riccietti. Email: elisa.riccietti@enseeiht.fr

difficult to properly tune the learning rate and they are generally more efficient in the solution of convex problems. Recently, a new stream of research has emerged, on methods that make use of second-order derivative approximations, contributing to build the next generation of optimization methods for large-scale machine learning [1, 7]. These methods are anyway generally really expensive in their standard form, especially when the number of variables is large.

For such problems multilevel techniques have shown to be really effective, both in the context of pure multigrid methods for the solution of linear systems arising from the discretization of linear or nonlinear elliptic partial differential equations in two or higher dimensions [9, 23, 57] and in the context of nonlinear optimization [20, 21, 31, 35, 36, 42, 43, 58]. The main idea on which multilevel methods are based is the exploitation of a hierarchy of problems, approximating the original one. Usually this hierarchy is built exploiting the fact that the underlying infinite-dimensional problem may be described at several discretization levels. The use of more levels is beneficial, as the coarse problems will generally be cheaper to solve than the original one, and it is possible to exploit the fact that such problems exhibit multiple scales of behaviour [9].

Inspired by these developments, we propose a new multilevel Levenberg-Marquardt training method, which is a member of the family of multilevel optimization methods recently introduced by the authors in [10], aimed at reducing the cost of the standard one-level procedure. Our aim is then twofold: on one side we can investigate the usefulness of multilevel methods as training methods, inquiring if the multilevel nature of the proposed solver can help to speed up the training process with respect to the one level optimization strategy. On the other side, this context allows us to get more insight into the potential of multilevel strategies for nonlinear optimization, investigating their application on problems that do not possess an underlying geometrical structure. Multilevel techniques require the construction of transfer operators. Usually standard interpolation and restriction operators, commonly employed in the case of linear systems, are also used in the case of nonlinear problems. However, to employ such operators a geometrical structure of the variables is needed. In the case of a training problem, the variables subject to optimization are the weights and biases of the network, which do not possess any geometrical structure. To derive a multilevel method for the training problem, it is then necessary to design multilevel transfer operators differently.

It is clear that the network possesses an important structure, even if it is not geometric in nature. It rather presents an algebraic structure, as the one that is usually exploited in algebraic multigrid, see, e.g., [51] and [57, Appendix A]. We investigate then if we could exploit this structure to define both the hierarchy of problems and the multilevel transfer operators. We propose a technique inspired by classical algebraic multigrid methods to do so.

We then test the proposed procedure on an important and up-to-date problem: the approximate solution of partial differential equations (PDEs) by neural networks. ANNs are well known for their excellent flexibility in approximating complex high-dimensional nonlinear functions and are thus naturally suitable to approximate the solution of partial differential equations. The use of artificial neural networks in problems involving PDEs has attracted a lot of interest lately. ANNs have been used for many different purposes in this field: for the numerical solution of either direct problems [15, 40, 44, 46, 49] or of inverse problems [45, 47], to reconstruct the equation from given data [38, 50, 53], or to melt with standard solution techniques such as finite element or finite difference methods, see [34, 39, 48, 56].

These techniques have indeed been shown to have several advantages over conventional numerical methods, see for example [17, 34], including the following ones. The differential operator does not need to be discretized, and the approximate solution obtained possesses an analytical expression and is continuously differentiable. This is essential for example in control problems, where derivatives of the solution are needed. The solution is also meshless, which helps to solve problems in complex geometries, as these methods require only a sampling of the domain, which is more easily produced than meshes when the geometry in irregular. Using neural networks provides a solution with very good generalization properties, this allows not only to have an approximation to the solution at all the points of the training interval (with an accuracy usually comparable to that obtained at the training points) but also to perform extrapolation outside the interval. This approach is useful especially in case of high dimensional equations, as it allows to alleviate the curse of dimensionality as the computational cost of training is only weakly dependent on the problem dimension, see, e.g., [24, 28, 29] among others, while for classical methods the problem's size grows exponentially with its dimension. Temporal and spatial derivatives can be treated in the same way and the training is highly parallelizable on Graphics Processing Units (GPU). These techniques are easy to implement, they are general and can be therefore be applied to every kind of differential equation and can be easily modified to handle different types of boundary conditions. More importantly, this approach provides a natural way to solve problems with nonlinear operators, with no need of linearisation. On some problems, e.g. problems in high dimension and problems with intricate domains, approaches based on neural networks have already been shown to outperform state-of-the-art methods [2, 28, 29, 37]. However, further work can be done to further improve their performance.

We propose to use a feedforward neural network and to choose the residual of the (possibly nonlinear) differential equation as the training loss function. This gives rise to a nonlinear least squares problem that we solve by the multilevel Levenberg-Marquardt method.

This manuscript represents a first step in the use of multilevel techniques in the training of artificial neural networks. Indeed, here we focus on the simplest case, that of one-layer networks. This case is interesting on its own, motivated by the Hecht-Nielsen theorem [26], see Section 4.1. From this theorem, we know that a one-layer network is capable of approximating a wide class of functions, up to any given accuracy level. The necessary number of nodes may however be really large. We experimentally show that it is not really the case for the class of problems we consider. Numerical results show indeed that with a reasonable number of nodes we can reach the desired solution accuracy. Moreover, the strategy we present to cope with the large dimension of the input space may further encourage the use of such networks, the extension of this to a more complicated multilayer being deferred to a forthcoming paper.

Our contributions We summarize here the main novelties and contributions of the manuscript. Our aim is to study the applicability of multilevel optimization methods for problems that do not possess an underlying geometrical structure, and in particular we focus on the training of artificial neural networks. Two practical questions are addressed.

- *How to make the approach practical and face the lack of a geometrical structure of the underlying problem?* We propose the use of a heuristic inspired by classical algebraic multigrid methods to define the hierarchy of problems and the

multilevel transfer operators.

- *What is the performance of the proposed method as compared to the standard one-level version?* We consider the approximation of the solution of a partial differential equation by a neural network, problem lately widely addressed by the machine learning community. We show through numerical experiments the gains, in terms of floating point operations, arising from the use of multilevel solvers as compared to standard one level solvers.

The idea of exploiting multiple scales in learning is not new, we mention for example [22, 30, 52]. Therein the multilevel structure is introduced in the model architecture, while here the training strategy is a multilevel strategy, the network's architecture being unchanged.

Moreover, to our knowledge, multilevel optimization techniques have only been applied to problems in which the hierarchy of function approximations could be built by exploiting the underlying geometrical structure of the problem at hand. Thus this work represents an improvement in the study of multilevel optimization methods.

Structure of the manuscript The manuscript is organized as follows. In Section 2 we briefly review the standard Levenberg-Marquardt method, whereas in Section 3 we describe its multilevel extension detailed in [10]. Then, in Section 4, we describe the artificial neural network approximation of the solution of the partial differential equation we employ and the related least squares problem. We then discuss its solution by the multilevel solver. In particular, we introduce the heuristic we propose to build the multilevel transfer operators. Finally, in Section 5, we present detailed numerical experiments related to the solution of both linear and nonlinear partial differential equations. Conclusions are drawn in Section 6.

2. The Levenberg-Marquardt method

The Levenberg-Marquardt (LM) method is an iterative procedure for the solution of least squares problems.

Let us consider a least squares problem of the form:

$$\min_x f(x) = \frac{1}{2}\|F(x)\|^2, \tag{1}$$

with $F : \mathcal{D} \subseteq \mathbb{R}^n \to \mathbb{R}^m$, $m \geq n$, a twice continuously differentiable function. At each iteration k, given the current iterate x_k, the objective function f is approximated by the norm of an affine model of F, resulting in a quadratic Taylor model for f with approximated Hessian matrix:

$$m_k(x_k, s) = \frac{1}{2}\|F(x_k)\|^2 + (J(x_k)^T F(x_k))^T s + \frac{1}{2} s^T B_k s,$$

where J is the Jacobian matrix of F and $B_k = J(x_k)^T J(x_k)$ approximates the Hessian matrix $\nabla_x^2 f(x_k)$. We later denote $m_k(x_k) = m_k(x_k, 0)$. Its regularized counterpart:

$$m_k(x_k, s) + \frac{\lambda_k}{2}\|s\|^2, \tag{2}$$

for $\lambda_k > 0$ a positive value called regularization parameter, is minimized (possibly approximately) to find a step s_k that is used to define the new iterate $x_{k+1} = x_k + s_k$.

At each iteration it has to be decided whether to accept the step or not. This decision is based on the accordance between the decrease in the function (*actual reduction*, $ared = f(x_k) - f(x_k + s_k)$) and in the model (*predicted reduction*, $pred = m_k(x_k) - m_k(x_k, s_k)$):

$$\rho_k = \frac{ared}{pred} = \frac{f(x_k) - f(x_{k+1})}{m_k(x_k) - m_k(x_k, s_k)}. \tag{3}$$

If the model is a sufficiently accurate approximation to the objective function, ρ_k will be close to one. Then, the step is accepted if ρ_k is larger than a chosen threshold $\eta_1 \in (0, 1)$ and is rejected otherwise. In the first case the step is said to be *successful*, otherwise the step is *unsuccessful*.

After the step acceptance, the regularization parameter is updated for the next iteration. The update is still based on the ratio (3). If the step is successful the parameter λ is decreased, otherwise it is increased.

The whole process is stopped when a minimizer of f is reached. Usually, the stopping criterion is based on the norm of the gradient, i.e. given a threshold $\epsilon > 0$ the iterations are stopped as soon as $\|\nabla_x f(x_k)\| < \epsilon$.

The main computational work per iteration in this kind of methods is represented by the minimization of the model (2). This is the most expensive task, and the cost depends on the dimension of the problem. However, from the convergence theory of such methods, it is well known that it is not necessary to minimize the model exactly to obtain global convergence.

A well-known possibility is indeed to minimize the model until the Cauchy decrease is achieved, i.e. until a fraction of the decrease provided by the Cauchy step (the step that minimizes the model in the direction of the negative gradient) is obtained. Here, we will consider a different kind of stopping criterion for the inner iterations, initially proposed in [4, 12] and also used in [10]. In this case, the inner iterations (for the minimization of the model) are stopped as soon as the norm of the gradient of the regularized model becomes lower than a multiple of the norm of the step:

$$\|\nabla_s m_k(x_k, s_k) + \lambda_k s_k\| \leq \theta \|s_k\|, \tag{4}$$

for a chosen constant $\theta > 0$. In order to minimize the model approximately, it is possible to use a Krylov subspace method on the system

$$(B_k + \lambda_k I)s = -J(x_k)^T F(x_k),$$

(with I the identity matrix of order n) and stop it as soon as the inequality (4) is satisfied.

The Levenberg-Marquardt procedure is sketched in Algorithm 1.

In the next section we will briefly review the multilevel extension of the Levenberg-Marquardt method. This method is part of the family of methods introduced in [10], and corresponds to the case $q = 1$, but with a different norm for the regularization term. In [10], if $q = 1$, the regularized model is defined as

$$f(x_k) + \nabla_x f(x_k)^T s + \frac{\lambda_k}{2}\|s\|^2, \tag{5}$$

where, in case of a least squares problem, $\nabla_x f(x_k) = J(x_k)^T F(x_k)$. For a symmetric positive definite matrix $M \in \mathbb{R}^{n\times n}$ and $x \in \mathbb{R}^n$, we can define the following norm:

$$\|x\|_M = (x^T M x)^{1/2}.$$

If we define $M_k = \dfrac{B_k}{\lambda_k} + I$, then we have $\dfrac{\lambda_k}{2}\|s\|^2_{M_k} = \dfrac{1}{2}s^T B_k s + \dfrac{\lambda_k}{2}\|s\|^2$, so that the regularized model in (2) can be written as

$$f(x_k) + \nabla_x f(x_k)^T s + \frac{\lambda_k}{2}\|s\|^2_{M_k},$$

corresponding to the model in (5), just with a different norm for the regularization term.

The theory presented in [10] for the case $q = 1$ applies for the multilevel method presented in Section 3, because the $\|\cdot\|_{M_k}$ norm and the Euclidean norm are equivalent, if we assume that $\|B_k\|$ is bounded at each iteration k, which is a common assumption in optimization.

Algorithm 1 LM$(x_0, \lambda_0, \epsilon)$ (Standard Levenberg-Marquardt method)

Given $0 < \eta_1 \le \eta_2 < 1$, $0 < \gamma_2 \le \gamma_1 < 1 < \gamma_3$, $\lambda_{\min} > 0$, $\theta > 0$.
Input: $x_0 \in \mathbb{R}^n$, $\lambda_0 > \lambda_{\min}$, $\epsilon > 0$.
$k = 0$
while $\|\nabla_x f(x_k)\| > \epsilon$ **do**
  • **Initialization:** Define the model m_k as in (2).
  • **Model minimization:** Find a step s_k that sufficiently reduces the model, i.e. such that (4) holds.
  • **Acceptance of the trial point and regularization parameter update:** Compute $\rho_k = \dfrac{f(x_k) - f(x_k + s_k)}{m_k(x_k) - m_k(x_k, s_k)}$.
  if $\rho_k \ge \eta_1$ **then**

$$x_{k+1} = x_k + s_k, \quad \lambda_{k+1} = \begin{cases} \max\{\lambda_{\min}, \gamma_2\lambda_k\}, & \text{if } \rho_k \ge \eta_2, \\ \max\{\lambda_{\min}, \gamma_1\lambda_k\}, & \text{if } \rho_k < \eta_2. \end{cases}$$

  else

$$x_{k+1} = x_k, \quad \lambda_{k+1} = \gamma_3\lambda_k.$$

  end if
  $k = k + 1$
end while

3. Multilevel extension of the Levenberg-Marquardt method

We consider a least squares problem of the form (1). At each iteration of the standard method, the objective function is approximated by the regularized Taylor model (2).

The minimization of (2) represents the major cost per iteration of the methods, which crucially depends on the dimension n of the problem. We want to reduce this cost by exploiting the knowledge of alternative simplified expressions of the objective function. More specifically, we assume that we know a collection of functions $\{f^l\}_{i=0}^{l_{\max}}$ such that each f^l is a twice-continuously differentiable function from $\mathbb{R}^{n_l} \to \mathbb{R}$ and $f^{l_{\max}}(x) = f(x)$ for all $x \in \mathbb{R}^n$. We will also assume that, for each $i = 1, \ldots, l_{\max}$, f^l is more costly to minimize than f^{l-1}.

The method is recursive, so it suffices to describe the setting with two levels only. Then, for sake of simplicity, from now on we will assume that we have two approximations to our objective function f.

For ease of notation, we will denote by $f^h : \mathbb{R}^{n_h} \to \mathbb{R}$ the approximation at the highest level, then $n_h = n$ and $f^h = f^{l_{\max}}$ in the notation previously used, while for $n_H < n_h$, $f^H : \mathcal{D} \subseteq \mathbb{R}^{n_H} \to \mathbb{R}$ is the approximation that is cheaper to optimize. All the quantities on the fine level will be denoted by a superscript h and all the quantities on the coarse level will be denoted by a superscript H, respectively. The main idea is then to use f^H to construct, in the neighbourhood of the current iterate x_k^h an alternative model m_k^H to the Taylor model m_k^h for $f^h = f$

$$m_k^h(x_k^h, s) = f^h(x_k^h) + \nabla_x f^h(x_k^h)^T s + \frac{1}{2} s^T B_k s. \tag{6}$$

The alternative model m_k^H should be cheaper to optimize than the quadratic model m_k^h, and will be used, whenever suitable, to define the step for the Levenberg-Marquardt algorithm. Of course, for f^H to be useful at all in minimizing f^h, there should be some relation between the variables of these two functions. We henceforth assume that there exist two full-rank linear operators $R : \mathbb{R}^{n_h} \to \mathbb{R}^{n_H}$ and $P : \mathbb{R}^{n_H} \to \mathbb{R}^{n_h}$ such that

$$\sigma_R P = R^T, \quad \max\{\|R\|, \|P\|\} \leq \kappa_R$$

for some constants $\sigma_R > 0$ and $\kappa_R > 0$.

Let $x_{0,k}^H := R x_k^h$ be the starting point at coarse level. We define the lower level model m_k^H as a modification of the coarse function f^H. f^H is modified adding a linear term to enforce the relation:

$$\nabla_s m_k^H(x_{0,k}^H) = R \nabla_x f^h(x_k^h). \tag{7}$$

Relation (7) crucially ensures that the first-order behaviour of f and m_k^H, are coherent in a neighbourhood of x_k^h and $x_{0,k}^H$. Indeed, if $s^H \in \mathbb{R}^{n_H}$ and $s^h = P s^H$, it holds:

$$\nabla_x f^h(x_k^h)^T s^h = \nabla_x f^h(x_k^h)^T P s^H = \frac{1}{\sigma_R} (R \nabla_x f^h(x_k^h))^T s^H = \frac{1}{\sigma_R} \nabla_s m_k^H(x_{0,k}^H)^T s^H.$$

To achieve this, we define m_k^H as:

$$m_k^H(x_{0,k}^H, s^H) = f^H(x_{0,k}^H + s^H) + (R \nabla_x f^h(x_k^h) - \nabla_x f^H(x_{0,k}^H))^T s^H, \tag{8}$$

where $\nabla_x f^h$ and $\nabla_x f^H$ are the gradients of the respective functions. At each generic iteration k of our method, a step s_k^h has to be computed to decrease the objective function f. Then, two choices are possible: the Taylor model (6) or the lower level model

(8). Obviously, it is not always possible to use the lower level model. For example, it may happen that $\nabla_x f^h(x_k^h)$ lies in the nullspace of R and thus that $R\nabla_x f^h(x_k^h)$ is zero while $\nabla_x f^h(x_k^h)$ is not. In this case, the current iterate appears to be first-order critical for m_k^H while it is not for f. Using the model m_k^H is hence potentially useful only if $\|\nabla_s m_k^H(x_{0,k}^H)\| = \|R\nabla_x f(x_k^h)\|$ is large enough compared to $\|\nabla_x f(x_k^h)\|$. We therefore restrict the use of the model m_k^H to iterations where

$$\|R\nabla_x f^h(x_k^h)\| \geq \kappa_H \|\nabla_x f^h(x_k^h)\| \text{ and } \|R\nabla_x f^h(x_k^h)\| > \epsilon_H \tag{9}$$

for some constant $\kappa_H \in (0, \min\{1, \|R\|\})$ and where $\epsilon_H \in (0,1)$ is a measure of the first-order criticality for m_k^H that is judged sufficient at level H. Note that, given $\nabla_x f^h(x_k^h)$ and R, this condition is easy to check, before even attempting to compute a step at level H.

If the Taylor model is chosen, then we just compute a standard Levenberg-Marquardt step, minimizing (possibly approximately) the corresponding regularized model. If the lower level model is chosen, then we minimize its regularized counterpart

$$m_k^H(x_{0,k}^H, s^H) + \frac{\lambda_k}{2}\|s^H\|^2 \tag{10}$$

(possibly approximately) and get a point $x_{*,k}^H$ such that (if the minimization is successful) the model is reduced, and a step $s_k^H = x_{*,k}^H - x_{0,k}^H$. This step has to be prolongated on the fine level. Then, the step is defined as $s_k^h = P s_k^H$.

In both cases, after the step is found, we have to decide whether to accept it or not. The step acceptance is based on the ratio:

$$\rho_k = \frac{f^h(x_k^h) - f^h(x_k^h + s_k^h)}{pred}. \tag{11}$$

where *pred* is defined as

- $pred = \frac{1}{\sigma_R}(m_k^H(Rx_k^h) - m_k^H(x_{*,k}^H)) = \frac{1}{\sigma_R}(m_k^H(Rx_k^h) - m_k^H(Rx_k^h, s_k^H))$ if the lower level model has been selected,
- $pred = m_k^h(x_k^h) - m_k^h(x_k^h, s_k^h)$ if the Taylor model has been selected.

As in the standard form of the methods, the step is accepted if it provides a sufficient decrease in the function, i.e. if given $\eta > 0$, $\rho_k \geq \eta$.

We sketch the whole procedure in Algorithm 2. As we anticipated, the procedure is recursive. For sake of clarity, allowing the possibility of having more than two levels, we denote the quantities on each level by a superscript l. We label our procedure MLM (Multilevel Levenberg-Marquardt). It is assumed to have at disposal a sequence of functions $\{f^l(\cdot) = \frac{1}{2}\|F^l(\cdot)\|^2\}_{l=1}^{l_{\max}}$ with $F^l : \mathbb{R}^{n_l} \to \mathbb{R}^m$ for $n_l > n_{l-1}$, with corresponding Jacobian J^l and we define $B_k^l = J^l(x_k^l)^T J^l(x_k^l)$. We provide additional comments to explain Step 6 in Algorithm 2. The generic framework sketched in Algorithm 2 comprises different possible methods. Specifically, one of the flexible features is that, to ensure convergence, the minimization at lower levels can be stopped after the first successful iteration, as shown in [10]. This therefore opens the possibility to consider both fixed form recursion patterns and free form ones. A free form pattern is obtained when Algorithm 2 is run carrying the minimization at each level out, until the norm of the gradient becomes small enough. The actual recursion pattern is then

uniquely determined by the progress of minimization at each level and may be difficult to forecast. By contrast, the fixed form recursion patterns are obtained by specifying a maximum number of successful iterations at each level, a technique directly inspired from the definitions of V- and W-cycles in multigrid algorithms [9].

4. Artificial neural network based approach for the approximate solution of partial differential equations

In this section we describe the strategy we propose for the approximate solution of a PDE.

Let us consider a stationary PDE written as:

$$D(z, u(z)) = g_1(z), \ z \in \Omega, \tag{12a}$$

$$BC(z, u(z)) = g_2(z), \ z \in \partial\Omega, \tag{12b}$$

where $\Omega \subset \mathbb{R}^N$, $N \geq 1$, is a connected subset, $\partial\Omega$ is the boundary of Ω, D is a differential operator, BC is an operator defining the boundary conditions, and $g_1, g_2 : \mathbb{R}^N \to \mathbb{R}$ are given functions. We remark that we do not need to make strong assumptions on D, we do not require it to be elliptic nor linear.

In the following, we describe the network's architecture that we have chosen to employ, how the training problem is formulated and how the multilevel Levenberg-Marquardt procedure is adapted to the solution of the specific problem.

4.1. Artificial neural network approximation of the solution of the partial differential equation

Our approach is based on the approximation of the solution $u(z)$ of the partial differential equation by an artificial neural network.

We assume the network to have just one hidden layer, as depicted in Figure 1. We have selected this network because the weights and biases are related in a simple way. It is therefore possible to devise a rather simple strategy to define the multilevel transfer operators. As already discussed in the Introduction, such networks have interest on their own, motivated by the Hecht-Nielsen theorem [26][1]. However the case of more hidden layers is at the same time an interesting and a challenging problem, due to the increased nonlinearity. For this case, it is not trivial to extend the developed strategy. We leave this as a perspective for future research. The aim of this manuscript is rather to make a first step toward a deeper understanding of the potential of multilevel techniques in speeding up the convergence of the corresponding one level methods, when the geometry of the underlying problem cannot be exploited.

The neural network takes the value of z as input and gives an approximation $\hat{u}(p, z)$ to $u(z)$ as output, for $p \in \mathbb{R}^n$, $n = (N + 2)r + 1$ with r the number of nodes in the hidden layer. For the sake of simplicity, we describe our approach in the simplest case $N = 1$, i.e. $z \in \mathbb{R}$. The generalization to the case $N > 1$ is straightforward and is reported in Appendix A.

The network is composed of three layers in total: one *input layer* composed of just one neuron as $z \in \mathbb{R}$, that receives the value of z as input; one *hidden layer* with r

[1]For any function in $L_2[(0, 1)^n]$ (i.e. square integrable on the n-dimensional unit cube) it exists a neural network with just one hidden layer that can approximate it, within any given accuracy.

Algorithm 2 MLM($l, \text{obj}^l, x_0^l, \lambda_0^l, \epsilon^l$) (Multilevel Levenberg-Marquardt method)

Given $0 < \eta_1 \leq \eta_2 < 1$, $0 < \gamma_2 \leq \gamma_1 < 1 < \gamma_3$, $\lambda_{\min} > 0$, $\theta > 0$.

Input: $l \in \mathbb{N}$ (index of the current level, $1 \leq l \leq l_{\max}$, $l_{\max}$ being the highest level), $\text{obj}^l : \mathbb{R}^{n_l} \to \mathbb{R}$ objective function to be optimized ($\text{obj}^{l_{\max}} = f$), $x_0^l \in \mathbb{R}^{n_l}$, $\lambda_0^l > \lambda_{\min}$, $\epsilon^l > 0$.

R_l denotes the restriction operator from level l to $l-1$, P_l the prolongation operator from level $l-1$ to l.

$k = 0$

while $\|\nabla_x f^l(x_k^l)\| > \epsilon^l$ **do**

• **Model choice:** If $l > 1$ compute $R_l \nabla_x f^l(x_k^l)$ and check (9). If $l = 1$ or (9) fails, go to Step 7. Otherwise, choose to go to Step 7 or to Step 8.

• **Taylor step computation:** Define $m_k^l(x_k^l, s^l)$ the Taylor series of order 2 of obj^l with approximated Hessian $B_k^l = J^l(x_k^l)^T J^l(x_k^l)$ and find a step s_k^l that sufficiently reduces $m_k^l(x_k^l, s^l) + \frac{\lambda_k^l}{2}\|s^l\|^2$, i.e. such that (4) holds. Go to Step 9.

• **Recursive step computation:** Define

$$m_k^{l-1}(R_l x_k^l, s^{l-1}) = \frac{1}{2}\|F^{l-1}(R_l x_k^{l-1} + s^{l-1})\|^2 + (R_l \nabla_x f^l(x_k^l) - \nabla_x f^{l-1}(R_l x_k^l))^T s^{l-1}.$$

Choose ϵ^{l-1} and call MLM($l-1$, m_k^{l-1},$R_l\ x_k^l$, λ_k^l, ϵ^{l-1}) yielding an approximate solution $x_{*,k}^{l-1}$ of the minimization of m_k^{l-1}. Define $s_k^l = P_l\ (x_{*,k}^{l-1} - R_l\ x_k^l)$ and $m_k^l(x_k^l, s^l) = \frac{1}{\sigma_R} m_k^{l-1}(R_l x_k^l, s^{l-1})$ for all $s^l = P_l s^{l-1}$.

• **Acceptance of the trial point and regularization parameter update:** Compute $\rho_k^l = \dfrac{f^l(x_k^l) - f^l(x_k^l + s_k^l)}{m_k^l(x_k^l) - m_k^l(x_k^l, s_k^l)}$.

if $\rho_k^l \geq \eta_1$ **then**

$$x_{k+1}^l = x_k^l + s_k^l, \quad \lambda_{k+1}^l = \begin{cases} \max\{\lambda_{\min}, \gamma_2 \lambda_k^l\}, & \text{if } \rho_k^l \geq \eta_2, \\ \max\{\lambda_{\min}, \gamma_1 \lambda_k^l\}, & \text{if } \rho_k^l < \eta_2. \end{cases}$$

else

$$x_{k+1}^l = x_k^l, \quad \lambda_{k+1}^l = \gamma_3 \lambda_k^l.$$

end if

$k = k + 1$

end while

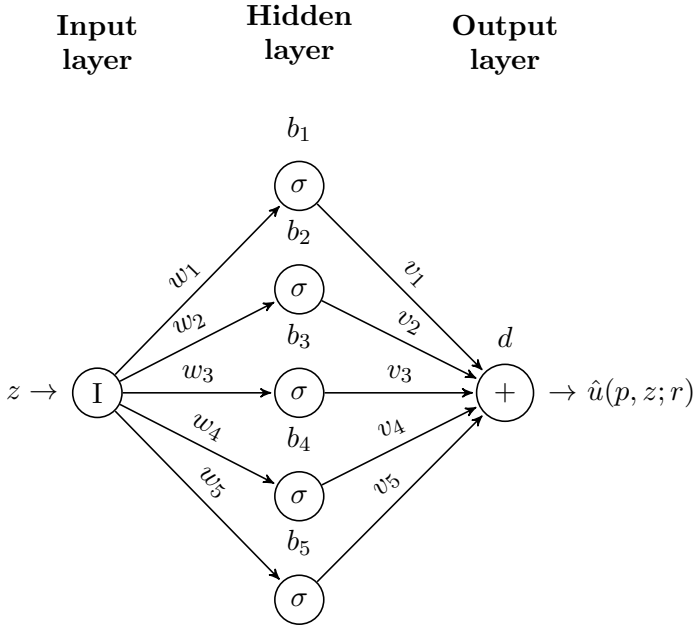

Figure 1. Artificial neural network architecture with weights and biases ($r = 5$, $N = 1$).

nodes, where r is a constant to be fixed. A bias b_i, $i = 1, \ldots, r$, is associated with each of these nodes. All of them are connected to the input node by edges, whose weights are denoted by w_i, $i = 1, \ldots, r$, as depicted in Figure 1. The w_i are called *input weights*. The last layer is the *output layer*, composed of just one node, as $u(z) \in \mathbb{R}$, associated with a bias $d \in \mathbb{R}$. The output node is connected to all the nodes in the hidden layer, the corresponding weights are called *output weights* and they are denoted by v_i, $i = 1, \ldots, r$.

The network is also characterized by an activation function σ, which is a given nonlinear function. Different choices are possible for σ, e.g. the sigmoid, hyperbolic tangent, logistic and softplus functions, respectively:

$$\sigma(z) = \frac{e^z - 1}{e^z + 1}, \;\; \sigma(z) = \frac{e^{2z} - 1}{e^{2z} + 1}, \;\; \sigma(z) = \frac{e^z}{e^z + 1}, \;\; \sigma(z) = log(e^z + 1).$$

We will denote by:

$$w = [w_1, \ldots, w_r]^T, \;\; v = [v_1, \ldots, v_r]^T, \;\; b = [b_1, \ldots, b_r]^T, \;\; p = [v, w, b, d]^T,$$

the vectors of input weights, output weights and biases of the hidden nodes and the stacked vector of weights and biases, respectively. The output of the neural network is a function of the weights and biases, but it also depends on the number of nodes r, which is a parameter fixed before the training. We then denote the output as $\hat{u}(p, z; r)$, which can be expressed as

$$\hat{u}(p, z; r) = \sum_{i=1}^{r} v_i \sigma(w_i z + b_i) + d, \text{ for all } z \in \Omega. \tag{13}$$

The training phase consists then in the minimization of a chosen loss function, which depends on the network's output and is a function of p, while z is constrained to the training set $\mathcal{T}$:

$$\min_p \mathcal{L}(p, z) = \mathcal{F}(\hat{u}(p, z; r)), \qquad z \in \mathcal{T}, \tag{14}$$

for $\mathcal{F} : \mathbb{R}^{3r+1} \to \mathbb{R}$.

We choose the nonlinear residual of the equation as a loss function, plus a penalty term with parameter $\lambda_p > 0$, to impose the boundary conditions as in [16, 33, 55], so that for $z \in \mathcal{T}$:

$$\mathcal{L}(p, z) = \frac{1}{2t} \left(\|D(z, \hat{u}(p, z; r)) - g_1(z)\|^2 + \lambda_p \|BC(z, \hat{u}(p, z; r)) - g_2(z)\|^2 \right), \tag{15}$$

with $\mathcal{T}$ a training set such that $|\mathcal{T}| = t$, in this case a set of points z_i in Ω, $i = 1, \ldots, t$.

Remark 1. The proposed strategy does not require the discretization of the operator D, $D(z, \hat{u}(p, z; r))$ can be analytically computed using the known expression (13) of $\hat{u}(p, z; r)$.

Remark 2. The proposed strategy, coupled with a discretization scheme in time, can be generalized to nonstationary equations.

4.2. Solution of the training problem

Optimizing (15) may be really expensive for certain problems. If the solution $u(z)$ is highly nonlinear, a really large number of nodes r may be necessary to approximate it with a sufficient accuracy. Problem (15) becomes then a large scale problem. We solve it thanks to the multilevel Levenberg-Marquardt strategy described in Section 3.

We remark that, from the analysis in [10], we do not need to make strong assumptions on the operator D nor on the functions g_1, g_2 to have a convergent training method. It is just sufficient that the resulting objective function (15) has Lipschitz continuous gradient, as well as its coarse approximations. We do not need to require the operator to be elliptic nor linear.

To be able to employ the multilevel Levenberg-Marquardt method in the solution of problem of the form (14), we need to define a strategy to build the hierarchy of coarse problems and the multilevel transfer operators.

Usually, when the optimization problem to be solved directly arises from the discretization of an infinite dimensional problem, the approximations to the objective function are simply chosen to be the functions arising from the discretization on coarser levels, and P and R are chosen to be the interpolation and the restriction operators, see [9]. However, we remind that in our case the variables subject to optimization are the weights and biases of the network, and not the components of the solution. Consequently, there is no geometric structure that can be exploited to construct a hierarchy, and the grids will not be real geometric grids as in classical geometric multigrid, but rather just sets of variables of different sizes. We have then to decide how to construct a hierarchy of sets of variables.

Inspired by the fact that the network possesses an intrinsic algebraic structure, as the one that is typically exploited in algebraic multigrid (AMG) [57], we propose here a coarsening strategy based on AMG, to both explore and exploit the structure of the

network, to build the hierarchy of problems and the multilevel operators. First, we give a brief overview of classical AMG techniques, and then we present the coarsening strategy we propose.

4.2.1. Algebraic multigrid Ruge and Stüben coarsening strategy

AMG techniques are multilevel strategies used for the solution of linear systems of the form $Ax = b$. The goal in AMG is to generalize the multilevel method used in geometric multigrid to target problems where the correct coarse problem is not apparent, since a geometric structure is not present. While, in geometric multigrid, a multilevel hierarchy is directly determined from structured coarsening of the problem, in standard AMG the coarse problems, together with the transfer operators, are automatically constructed exploiting exclusively the information contained in the entries of the matrix. Specifically, the variables on the fine level are split into two sets, C and F, of *coarse* and *fine* variables, respectively. The variables in C are selected to be the variables of the coarse problem, those in F are all the others. The coarse variables are chosen to be *representative* of the fine ones, i.e. they are chosen to be the variables to which many of the remaining ones are connected. The connection among variables is only based on the entries of the matrix. There are many strategies to build this C/F splitting. We decided to use the *Ruge and Stüben* strategy, one of the most classical AMG coaesening strategies [8, 13, 57]. This is a first attempt and we do not claim that this is the best strategy to use. Other options are possible, that could be more effective.

The method relies on theoretical results if it is applied to a specific class of matrices [57], but it is commonly also used for different problems, for which there are no theoretical guarantees. If applied to systems arising from the discretization of simple elliptic differential operators, as the Laplace operator, this strategy is known to recover the structure exploited by geometric variants.

The splitting is built based on the notion of coupling. Two variables indexed by i and j are said to be *coupled* if the corresponding entry of the matrix is different from zero, i.e. $a_{i,j} \neq 0$. The coupling is said to be negative if $a_{i,j} < 0$, positive otherwise. The splitting is usually made considering first negative couplings, as typically in the applications AMG is used for, the negative couplings are more than the positive ones. Then, the notion of *strong negative coupling* is introduced, i.e. we say that a variable i is strongly negatively coupled to another variable j, if

$$-a_{i,j} \geq \epsilon_{AMG} \max_{a_{i,k}<0} |a_{i,k}| \tag{16}$$

for a fixed $0 < \epsilon_{AMG} < 1$. This measure is used to construct the splitting in practice. Each F variable is required to have a minimum number of its strong couplings be represented in C. The C/F splitting is usually made choosing some first variable i to become a coarse variable. Then, all variables strongly coupled to it become F variables. The process is repeated until all variables have been split. In order to avoid randomly distributed C/F patches, more sophisticated procedures can be designed, for example the process can be performed in a certain order, based on a measure of importance of the variables, for more details see [57, §A.7.1]. Then, also positive couplings are taken into account. After the coarsening process has been applied, a pass checks if there are

strong positive F/F couplings. If

$$a_{i,j} \geq \epsilon_{AMG} \max_{k \neq i} |a_{i,k}|$$

for some $j \neq i$, the variable j is added to the set of variables strongly connected to i and the variable corresponding to the largest positive coupling becomes a C variable [57].

Based on this splitting, the transfer operators are built to interpolate the components of the error corresponding to the variables in F. The components of the error corresponding to the variables in C are transferred to the higher level by the identity operator, while the others are transformed by an interpolation formula, so that we define the i-th variable at fine level as:

$$x_i^h = (Px^H)_i = \begin{cases} x_i^H & \text{if } i \in C, \\ \sum_{k \in P_i} \delta_{i,k} x_k^H & \text{if } i \in F, \end{cases}$$

with

$$\delta_{i,k} = \begin{cases} -\alpha_i a_{i,k}/a_{i,i} & \text{if } k \in P_i^-, \\ -\beta_i a_{i,k}/a_{i,i} & \text{if } k \in P_i^+, \end{cases} \qquad \alpha_i = \frac{\sum_{j \in N_i} a_{i,j}^-}{\sum_{k \in P_i} a_{i,k}^-}, \qquad \beta_i = \frac{\sum_{j \in N_i} a_{i,j}^+}{\sum_{k \in P_i} a_{i,k}^+},$$

where $a_{i,j}^+ = \max\{a_{i,j}, 0\}$, $a_{i,j}^- = \min\{a_{i,j}, 0\}$, N_i is the set of variables connected to i (i.e. all j such that $a_{i,j} \neq 0$), P_i the set of coarse variables strongly connected to i, which is partitioned in P_i^- (negative couplings) and P_i^+ (positive couplings). The interpolation operator, assuming to have regrouped and ordered the variables to have all those corresponding to indexes in C at the beginning, is then defined as $P = \begin{bmatrix} I \\ \Delta \end{bmatrix}$ where I is the identity matrix of size $|C|$ and Δ is the matrix such that $\Delta_{i,j} = \delta_{i,j}$, for $i = 1, \ldots, |F|$ and $j = 1, \ldots, |C|$.

4.2.2. Algebraic coarsening strategy when solving optimization problems related to the training of artificial neural networks

In this section, we propose a possible strategy to define both R and P, required in the solution of problem (14). As in classical AMG, we rely on a heuristic strategy. In our procedure, we have the nonlinear minimization problem (14) to solve. At each iteration at fine level, the minimization process requires the solution of a linear system with matrix $B_k = J(x_k)^T J(x_k)$, where J is the Jacobian matrix of F at x_k. Then, a possibility to build the C/F splitting is to apply the algebraic multigrid technique we just described to B_k. However, we cannot apply the procedure directly to this matrix. Indeed, while in the coarsening process of standard AMG all the variables are treated in the same way, in our application the variables are coupled [13]. We are actually optimizing with respect to triples $\{v_i, w_i, b_i\}$ of input weights, output weights and biases. The bias d being a scalar is treated separately. If no distinction is made, a weight/bias could be removed, without the other components of the triple being removed, leading to a network that would not be well defined. Consequently, instead of considering the strength of connections among the variables, we will consider the strength of connections *among the triples*. We propose therefore to apply the AMG splitting to the matrix $A \in \mathbb{R}^{r \times r}$ resulting from a weighted sum of the submatrices of

B_k containing the derivatives of F taken with respect to the same kind of variables, so that the contributions of the three different variables are not melted. More precisely, $B_k = J^T(x_k)J(x_k)$ reads:

$$B_k = \begin{bmatrix} F_v^T F_v & F_v^T F_w & F_v^T F_b & F_v^T F_d \\ F_w^T F_v & F_w^T F_w & F_w^T F_b & F_w^T F_b \\ F_b^T F_v & F_b^T F_w & F_b^T F_b & F_b^T F_d \\ F_d^T F_v & F_d^T F_w & F_d^T F_b & F_d^T F_d \end{bmatrix}, \quad F_\xi = \begin{bmatrix} \frac{\partial F_1(x_k)}{\partial \xi_1} & \cdots & \frac{\partial F_1(x_k)}{\partial \xi_r} \\ & \cdots & \\ \frac{\partial F_m(x_k)}{\partial \xi_1} & \cdots & \frac{\partial F_m(x_k)}{\partial \xi_r} \end{bmatrix}, F_d = \begin{bmatrix} \frac{\partial F_1(x_k)}{\partial d} \\ \vdots \\ \frac{\partial F_m(x_k)}{\partial d} \end{bmatrix},$$

for each variable $\xi \in \mathbb{R}^r$ and for $d \in \mathbb{R}$.

We then apply the Ruge and Stüben splitting strategy [57] to the following matrix:

$$A = \frac{F_v^T F_v}{\|F_v\|_\infty} + \frac{F_w^T F_w}{\|F_w\|_\infty} + \frac{F_b^T F_b}{\|F_b\|_\infty}.$$

In this way, we first obtain a C/F slitting of the triples and then deduce the corresponding interpolation operator $P \in \mathbb{R}^{r \times r_c}$ with $r_c \leq r$ and the restriction operator $R \in \mathbb{R}^{r_c \times r}$ to use in the multilevel Levenberg-Marquardt method. These operators are used to project the vector $s_k^h \in \mathbb{R}^{3r}$ of fine weights and biases on the coarse level and to prolongate the coarse level step $s_k^H \in \mathbb{R}^{3r_c}$ to the fine level (in both cases omitting the contribution of the bias d that is a scalar variable and is therefore left unchanged when changing levels). In both cases, the operators P and R are applied to each of the three components of s_k^h, s_k^H, corresponding to the three different kinds of variables. In case of more than two levels, the operators R_l and P_l, on each level, can be built with the same technique, applied to the matrix that approximates the Hessian matrix of the coarse function approximating f on level l.

Remark 3. We remark that the strategy we propose is not specifically tailored for (15), it can be used for all problems of the form (14).

5. Numerical experiments

In this section, we report on the practical performance of our multilevel approach.

5.1. Setting

The whole procedure has been implemented in Julia [3]. We set the following values for the parameters in Algorithms 1 and 2, respectively: $\eta_1 = 0.1$, $\eta_2 = 0.75$, $\gamma_1 = 0.85$, $\gamma_2 = 0.5$, $\gamma_3 = 1.5$, $\theta = 10^{-2}$, $\lambda_0 = 0.05$ and $\lambda_{\min} = 10^{-6}$. We compare the standard (one level) Levenberg-Marquardt method with a two-level Levenberg-Marquardt variant. We have decided to rely on just two levels, as in the experiments the cardinality of the coarse set of variables is much lower than that of the fine set (just a few dozens of parameters rather than more than 500 or 1000, depending on the problem).

The construction of the coarse set of variables is performed through the Julia's algebraic multigrid package (AMG), that implements the classical Ruge and Stüben method[2]. The operators R and P are built just once at the beginning of the optimization procedure, using matrix B_0. In (9), we choose $\kappa_H = 0.1$ and ϵ_H is equal to the

[2]Available at: https://github.com/JuliaLinearAlgebra/AlgebraicMultigrid.jl

tolerance chosen at the fine level. In (16), we set $\epsilon_{AMG} = 0.9$, but numerical experiments highlighted that the cardinality of the coarse set does not strongly depend on this choice. For the procedure to be effective, we noticed that it is also beneficial to scale the operators R and P, yielded by the AMG package, by their infinity norm. We choose to perform a fixed form of recursion pattern, inspired by the V-cycle of multigrid methods [9]. Hence we alternate a fine and a coarse step, and we impose a maximum number of 10 iterations at the coarse level.

The linear systems arising from the minimization of the models on the fine level are solved by a truncated CGLS method [6, §7.4], while those on the coarse level, due to the really low number of parameters, are solved with a direct method. However, it is worth mentioning that such systems have a peculiar structure. This is similar to that of normal equations, but is fundamentally different due to the presence of the linear correction term $(R\nabla_x f^h(x_k^h) - \nabla_x f^H(x_{0,k}^H))^T s^H$ on the right-hand side, which prevents the use of methods based on normal equations. We refer the reader to [11] for a discussion on how to exploit this structure in case of higher dimensional coarse level problems.

5.2. Definition of the test problems

We consider both partial differential equations in one- and two-dimensions. In (12a) we obtain g_1 as the result of the choice of the true solution u_T, and we choose $u_T = u_T(z, \nu)$, depending on a parameter ν, that controls the oscillatory behaviour of the solution. As ν increases, the true solution becomes more oscillatory and thus harder to approximate. A network with a larger number of nodes (r) is thus necessary, and consequently the size of the fine problem increases. We set $\Omega = (0, 1)^N$ for the problems corresponding to the first six lines in Table 1 and for the rest of the problems we choose domains with more intricate geometries, which are specified in Table 1. In all the experiments we impose Dirichlet boundary conditions. We choose $\mathcal{T}$ as a Cartesian uniform grid with $h = \frac{1}{2\nu}$, according to the Nyquist-Shannon sampling criterion [54], so that the cardinality of the training set is $t = 2\nu + 1$. Finally, in (15) we have set the penalty parameter to $\lambda_p = 0.1\, t$.

In many practical applications, as in seismology for example, a really accurate solution is not required. We stress that this setting is typical when solving partial differential equations by artificial neural networks: a high accuracy is never sought, but rather an approximate solution is looked for, see, e.g., [40, 46, 47]. Therefore, we look for an approximate solution and stop the procedure as soon as $\|\nabla\mathcal{L}(p_k^h, z)\| \leq 10^{-4}$ for one-dimensional problems, $\|\nabla\mathcal{L}(p_k^h, z)\| \leq 10^{-3}$ for two-dimensional problems, respectively. Indeed in the one-dimensional case, the convergence is quite fast at the beginning, allowing to quickly reach an approximate solution, while it requires far more iterations to obtain a more accurate solution for two-dimensional problems. We remark that the selected test problems are quite difficult, due to the high nonlinearity of the solutions. We outline that additional tests performed on problems with smaller value of ν (not reported here) allowed to reach a tighter accuracy level with a smaller amount of iterations.

All selected problems are listed in Table 1, where we report in which table the corresponding results are reported, which equation is considered, the imposed true solution and its frequency ν, respectively. We consider both linear and nonlinear partial differential equations (such as Liouville equation). For the two-dimensional Helmholtz equation, we have chosen different choices for $c(z)$: a constant function, a piecewise

Table 1. Problems considered in Tables 2 to 9. We report: the equation considered, the true solution u_T, the frequency considered ν and the domain chosen Ω. $\mathcal{D}_m$ is the domain depicted in Figure 2 and $\mathcal{D}_s$ is the domain depicted in Figure 3. For the two-dimensional Helmholtz equation (Table 5) the reference solution is computed by finite differences.

	Equation	$u_T(z,\nu)$	ν	Ω
Table 2	$-\Delta u = g_1$ (1D)	$\cos(\nu z)$	$20, 25$	$(0,1)^N$
Table 3	$-\Delta u = g_1$ (2D)	$\cos(\nu(z_1+z_2))$	$5, 6$	$(0,1)^N$
Table 4	$-\Delta u - \nu^2 u = 0$ (1D)	$\sin(\nu z) + \cos(\nu z)$	5	$(0,1)^N$
Table 5	$-\Delta u - \left(\frac{2\pi\nu}{c(z)}\right)^2 u = g_1$ (2D)	-	$1, 2$	$(0,1)^N$
Table 6 (left)	$\Delta u + \sin u = g_1$ (1D)	$0.1\cos(\nu z)$	20	
Table 6 (right)	$\Delta u + e^u = g_1$ (2D)	$\log\left(\frac{\nu}{z_1+z_2+10}\right)$	1	$(0,1)^N$
Table 7	$-\Delta u + \nu^2 u = g_1$ (2D)	$\sin(\nu(x+y))$	$3, 7$	$\mathcal{D}_m$
Table 8	$-\Delta u + \nu^2 u = g_1$ (2D)	$(x^2+y^2) + \sin(\nu(x^2+y^2))$	$3, 5$	$\mathcal{D}_s$
Table 9	$-\Delta u + \nu u^2 = g_1$ (2D)	$(x^2+y^2) + \sin(\nu(x^2+y^2))$	0.5	$\mathcal{D}_s$

Table 2. One-dimensional Poisson problem. Solution of the minimization problem (15) with the one level Levenberg-Marquardt method (LM) and the two-level Levenberg-Marquardt method (MLM), respectively. `iter` denotes the averaged number of iterations over ten simulations, `RMSE` the root-mean square error with respect to the true solution u_T and `save` the ratio between the total number of floating point operations required for the matrix-vector products in LM and MLM, r is the number of nodes in the hidden layer.

	$\nu = 20$		$r = 2^9$	$\nu = 25$		$r = 2^{10}$
Method	`iter`	`RMSE`	`save`	`iter`	`RMSE`	`save`
LM	869	10^{-4}		1439	10^{-3}	
MLM	507	10^{-4}	1.1-2.6-4.3	1325	10^{-3}	1.2-1.7-2.8

Table 3. Two-dimensional Poisson problem. Solution of the minimization problem (15) with the one level Levenberg-Marquardt method (LM) and the two-level Levenberg-Marquardt method (MLM), respectively. `iter` denotes the averaged number of iterations over ten simulations, `RMSE` the root-mean square error with respect to the true solution u_T and `save` the ratio between the total number of floating point operations required for the matrix-vector products in LM and MLM, r is the number of nodes in the hidden layer.

	$\nu = 5$		$r = 2^{10}$	$\nu = 6$		$r = 2^{11}$
Method	`iter`	`RMSE`	`save`	`iter`	`RMSE`	`save`
LM	633	10^{-3}		1213	10^{-3}	
MLM	643	10^{-3}	1.1-1.5-2.1	1016	10^{-3}	1.2-1.9-2.4

Table 4. One-dimensional Helmholtz problem. Solution of the minimization problem (15) with the one level Levenberg-Marquardt method (LM) and two-level Levenberg-Marquardt method (MLM), respectively. `iter` denotes the averaged number of iterations over ten simulations, `RMSE` the root-mean square error with respect to the true solution u_T and `save` the ratio between total number of floating point operations required for the matrix-vector products in LM and MLM, r is the number of nodes in the hidden layer.

	$\nu = 5$		$r = 2^{10}$
Method	`iter`	`RMSE`	`save`
LM	1159	10^{-3}	
MLM	1250	10^{-3}	1.2-1.9-3.1

Table 5. Two-dimensional Helmholtz problem. Solution of the minimization problem (15) with the one level Levenberg-Marquardt method (LM) and two-level Levenberg-Marquardt method (MLM), respectively. `iter` denotes the averaged number of iterations over ten simulations, `RMSE` the root-mean square error with respect to the solution computed by finite differences and `save` the ratio between total number of floating point operations required for the matrix-vector products in LM and MLM. With respect to the notation in Table 1, in all the tests $g_1([z_1, z_2]) = (0.25 < z_1 < 0.75)(0.25 < z_2 < 0.75)$, and $c(z)$ has been chosen as: $\bar{c}_1([z_1, z_2]) = 40$ (up, left); $\bar{c}_1([z_1, z_2]) = 20\,(0 \le z_1 < 0.5) + 40\,(0.5 \le z_1 \le 1)$ (up right); $\bar{c}_2([z_1, z_2]) = 20\,(0 \le z_1 < 0.25) + 40\,(0.25 \le z_2 \le 0.5) + 60\,(0.5 \le z_3 < 0.75) + 80\,(0.75 \le z_4 \le 1)$ (bottom, left); $\bar{c}_2([z_1, z_2]) = 0.1\sin(z_1 + z_2)$ (bottom, right).

		$\nu = 1$	$r = 2^9$		$\nu = 2$	$r = 2^9$
Method	`iter`	`RMSE`	`save`	`iter`	`RMSE`	`save`
LM	200	10^{-3}		200	10^{-2}	
MLM	200	10^{-3}	1.7-1.8-1.9	200	10^{-2}	1.7-1.8-1.9
		$\nu = 2$	$r = 2^9$		$\nu = 2$	$r = 2^9$
Method	`iter`	`RMSE`	`save`	`iter`	`RMSE`	`save`
LM	200	10^{-2}		200	$5\,10^{-3}$	
MLM	200	10^{-2}	1.7-1.8-1.8	200	$5\,10^{-3}$	1.7-1.8-1.9

Table 6. Nonlinear partial differential equations (see Table 1). Solution of the minimization problem (15) with the one level Levenberg-Marquardt method (LM) and two-level Levenberg-Marquardt method (MLM), respectively. `iter` denotes the averaged number of iterations over ten simulations, `RMSE` the root-mean square error with respect to the true solution u_T and `save` the ratio between total number of floating point operations required for the matrix-vector products in LM and MLM, r is the number of nodes in the hidden layer.

		$\nu = 20$	$r = 2^9$		$\nu = 1$	$r = 2^9$
Method	`iter`	`RMSE`	`save`	`iter`	`RMSE`	`save`
LM	950	10^{-5}		270	10^{-3}	
MLM	1444	10^{-5}	0.8-2.9-5.3	320	10^{-3}	1.2-1.7-1.8

Table 7. Two-dimensional Screened Poisson problem on domain $\mathcal{D}_m$. Solution of the minimization problem (15) with the one level Levenberg-Marquardt method (LM) and the two-level Levenberg-Marquardt method (MLM), respectively. `iter` denotes the averaged number of iterations over ten simulations, `RMSE` the root-mean square error with respect to the true solution u_T and `save` the ratio between the total number of floating point operations required for the matrix-vector products in LM and MLM, r is the number of nodes in the hidden layer.

		$\nu = 3$	$r = 2^9$		$\nu = 7$	$r = 2^{10}$
Solver	`iter`	`RMSE`	`save`	`iter`	`RMSE`	`save`
LM	395	3.e-4		1500	2.4e-3	
MLM	110	2.e-4	1.3-5.6-10.0	1039	1.6e-3	1.6-2.1-2.5

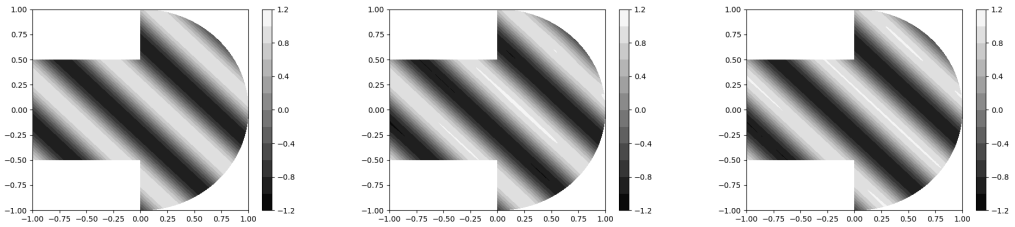

Figure 2. Contour plot corresponding to the problem in Table 7 for $\nu = 7$, of true solution (left), solution approximation computed by LM (center) and MLM (right).

Table 8. Two-dimensional Screened Poisson problem on domain $\mathcal{D}_s$. Solution of the minimization problem (15) with the one level Levenberg-Marquardt method (LM) and the two-level Levenberg-Marquardt method (MLM), respectively. `iter` denotes the averaged number of iterations over ten simulations, `RMSE` the root-mean square error with respect to the true solution u_T and `save` the ratio between the total number of floating point operations required for the matrix-vector products in LM and MLM, r is the number of nodes in the hidden layer.

	$\nu = 3$		$r = 2^9$	$\nu = 5$		$r = 2^{10}$
Solver	`iter`	`RMSE`	`save`	`iter`	`RMSE`	`save`
LM	1482	5.0e-3		2143	1.2e-2	
MLM	952	3.7e-3	1.4-2.4-3.5	2076	1.0e-2	1.4-1.6-1.8

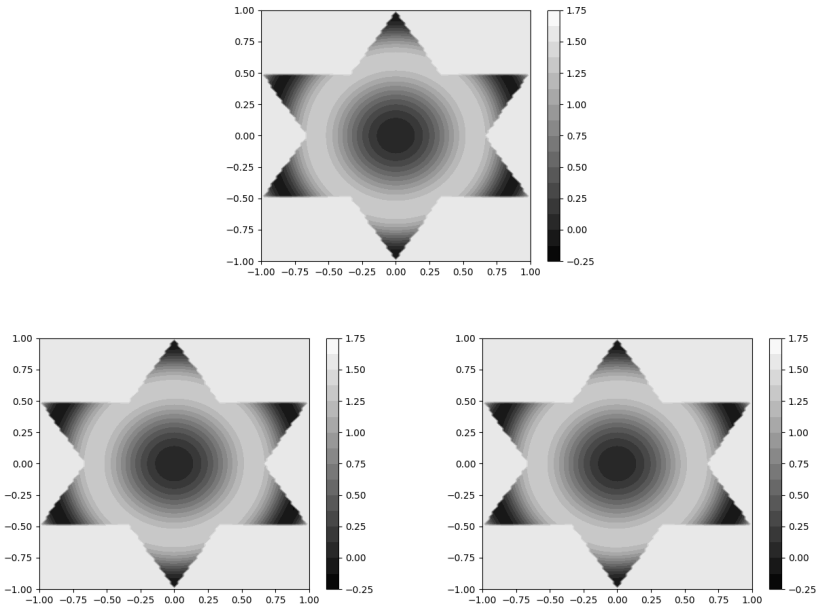

Figure 3. Contour plot corresponding to the problem in Table 8 for $\nu = 5$, of true solution (top), solution approximation computed by LM (bottom left) and MLM (bottom right).

Table 9. Nonlinear problem on domain $\mathcal{D}_s$ (see Table 1). Solution of the minimization problem (15) with the one level Levenberg-Marquardt method (LM) and the two-level Levenberg-Marquardt method (MLM), respectively. `iter` denotes the averaged number of iterations over ten simulations, `RMSE` the root-mean square error with respect to the true solution u_T and `save` the ratio between the total number of floating point operations required for the matrix-vector products in LM and MLM, r is the number of nodes in the hidden layer. Over the 10 runs, LM is successful in only 4 for $\nu = 0.5$ and 3 for $\nu = 1$, while RLM is successful in 8 and 9 runs respectively. The results refer just to the successful runs.

	$\nu = 0.5$		$r = 2^9$	$\nu = 1$		$r = 2^9$
Solver	`iter`	`RMSE`	`save`	`iter`	`RMSE`	`save`
LM	1408	3.2e-3		1450	1.5e-2	
MLM	1301	3.6e-3	0.9-1.7-2.4	1295	1.9e-2	1.1-1.7-2.1

constant function and a sinusoidal function, see Table 5. For this test case, we do not choose the true solution, but rather impose the right-hand side. The reference solution (needed for the computation of the root mean squared error) is then computed by finite differences.

5.3. Analysis of the numerical results

In what follows, the numerical results refer to ten simulations for different random initial guesses. For each simulation however, the starting guess is the same for the two solvers. The first line refers to the standard (one level) Levenberg-Marquardt method (LM), while the second one to the two-level Levenberg-Marquardt method (MLM), respectively. We report the average number of iterations (`iter`) over the ten simulations, the root mean squared error (`RMSE`) of the computed solution evaluated on a grid of 100^N testing points, inside the training interval and other than the training points, with respect to the true solution evaluated at the same points and (`save`), the ratio between the total number of floating point operations required for the matrix-vector products in the two methods (min-mean-max values are given), respectively.

As expected, we can notice that the problems become more difficult to solve as ν increases, and that the number of nodes r in the neural network has to be increased to obtain an approximate solution as accurate as for lower values of ν.

Most often, the number of iterations required by the two-level procedure is lower than that required by the one-level procedure, which is a behaviour typically observed when using classical multilevel methods. This effect is particularly evident in the problem corresponding to Table 7, where the ratios are found to be really large, as the one-level procedure seldom manages to satisfy the stopping criterion within the maximum number of iterations. Contrarily to the standard Levenberg-Marquardt method, the new method is able to reach lower values of the stopping tolerance than those chosen in the test, thus being able to find more accurate solution approximations.

Moreover, due also to the lower dimension of the linear systems at the coarse level, the number of floating point operations required for the matrix-vector products is always *considerably* lower, even when MLM performs more iterations. The computational gain in terms of floating point operations is on average a factor around 2 on all the experiments, and the maximum values of the ratio is much larger for certain problems (see Tables 6 and 7). More importantly, the quality of the approximate solution is not affected. This is a rather satisfactory result.

Finally, the method manages to solve problems defined on domains with non-standard geometries, such as the ones in Figures 2-3, both for linear (cf. Tables 7-8) and nonlinear (cf. Table 9) equations. For such domains ANN-based approaches are easier to use than standard mesh-based methods, as the domain just needs to be sampled, which is much easier than building a mesh. The advantage of the multilevel procedure over the standard method is particularly evident for the problem in Table 9, for which the new method shows to be more robust. The problem is particularly difficult and the standard method fails to reach the stopping criterion and to correctly approximate the true solution in 6 runs for $\nu = 0.5$ and in 7 runs for $\nu = 1$, while the new method only fails in 2 and 1 run(s), respectively. The results in Table 9 are then referred just to the successful runs.

We remark also that the two-dimensional problems are more difficult than one-dimensional ones, especially those corresponding to the Helmholtz equation, which is a particularly difficult problem with Dirichlet boundary conditions, for which standard

multigrid methods are known not to be effective, see for example [14, 18, 19]. In this case, a rough approximation is obtained in really few iterations, a maximum number of 200 iterations is imposed, as iterating further is not useful to improve the approximation. This is already a satisfactory result. To improve the solution accuracy, a network with a more complex topology shall be more efficient, as the problem itself possesses a more complex structure.

6. Conclusions

We have investigated the potential of the multilevel Levenberg-Marquardt method in the solution of training problems. This is chosen as representative of a class of problems in which the variables subject to optimization are not related by any explicit geometrical structure. To the best of our knowledge, this is also the first attempt at using multilevel optimization for the training of artificial neural networks.

We have chosen the approximation of the solution of partial differential equations by a neural network as an applicative context to test our procedure. We remark however that the approach introduced here is not specific to the solution of partial differential equations. It can generally be used to solve least squares problems of the form (14), in which the solution can be expressed by an artificial neural network.

We have proposed a possible heuristic strategy based on standard algebraic multigrid methods, to both explore and exploit the structure of the neural network to build a hierarchy of problems and the multilevel transfer operators.

The performance of the multilevel optimization method has been tested and compared to that of the standard one-level version. The numerical results are quite satisfactory, showing the potential of the multilevel strategy. The new method is shown to provide considerable savings in terms of number of floating point operations, and on specific problems to be more robust and to be capable of driving the stopping tolerance to lower values, thus providing better solution approximations.

These preliminary numerical results and the perspective for improvements encourage us to investigate further on multilevel training methods. This strategy has been designed for networks with one hidden layer. Considering a multilayer network is a natural and challenging extension due to the increased nonlinearity. We currently consider the extension of the procedure to multilayer networks as a significant perspective, as we believe that this could lead to a competitive learning method, especially if coupled with a strategy to make the approach purely matrix-free.

Funding

This work was funded by TOTAL.

References

[1] A. Berahas, J. Nocedal, and M. Takáč, *A Multi-batch L-BFGS Method for Machine Learning*, in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, USA. Curran Associates Inc., NIPS'16, 2016, pp. 1063–1071. Available at http://dl.acm.org/citation.cfm?id=3157096.3157215.

[2] J. Berg and K. Nyström, *A unified deep artificial neural network approach to partial differential equations in complex geometries*, Neurocomputing 317 (2018), pp. 28–41.

[3] J. Bezanson, A. Edelman, S. Karpinski, and V. Shah, *Julia: A fresh approach to numerical computing*, SIAM Review 59 (2017), pp. 65–98. Available at https://doi.org/10.1137/141000671.

[4] E.G. Birgin, J.L. Gardenghi, J.M. Martínez, S.A. Santos, and P.L. Toint, *Worst-case evaluation complexity for unconstrained nonlinear optimization using high-order regularized models*, Mathematical Programming 163 (2017), pp. 359–368. Available at https://doi.org/10.1007/s10107-016-1065-8.

[5] C. Bishop, *Pattern recognition and machine learning*, Information Science and Statistics, Springer-Verlag New York, 2006, Available at https://www.springer.com/us/book/9780387310732.

[6] A. Björck, *Numerical Methods for Least Squares Problems*, Vol. 51, Siam, 1996, Available at https://epubs.siam.org/doi/book/10.1137/1.9781611971484?mobileUi=0.

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

[8] A. Brandt, *General highly accurate algebraic coarsening*, Electronic Transactions on Numerical Analysis 10 (2000), pp. 1–20. Available at http://eudml.org/doc/120680.

[9] W. Briggs, V. Henson, and S. McCormick, *A Multigrid Tutorial, Second Edition*, 2nd ed., Society for Industrial and Applied Mathematics, 2000, Available at https://epubs.siam.org/doi/abs/10.1137/1.9780898719505.

[10] H. Calandra, S. Gratton, E. Riccietti, and X. Vasseur, *On high-order multilevel optimization strategies*, in *arXiv preprint*. 2019. Available at https://arxiv.org/abs/1904.04692.

[11] H. Calandra, S. Gratton, E. Riccietti, and X. Vasseur, *On the solution of the extended normal equations*, in *arXiv preprint*. 2019. Available at https://arxiv.org/abs/1911.00026.

[12] C. Cartis, N.I.M. Gould, and P.L. Toint, *Adaptive cubic regularisation methods for unconstrained optimization. Part I: motivation, convergence and numerical results*, Mathematical Programming 127 (2011), pp. 245–295. Available at https://doi.org/10.1007/s10107-009-0286-5.

[13] T. Clees, *AMG strategies for PDE systems with applications in industrial semiconductor simulation*, Ph.D. diss., University of Cologne, Germany, 2005. Available at https://kups.ub.uni-koeln.de/1575/.

[14] P. Cocquet and M. Gander, *How large a shift is needed in the shifted Helmholtz preconditioner for its effective inversion by multigrid?*, SIAM Journal on Scientific Computing 39 (2017), pp. A438–A478. Available at https://doi.org/10.1137/15M102085X.

[15] G. Di Muro and S. Ferrari, *A constrained-optimization approach to training neural networks for smooth function approximation and system identification*, in *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*. 2008, pp. 2353–2359. Available at http://dx.doi.org/10.1109/IJCNN.2008.4634124.

[16] M. Dissanayake and N. Phan-Thien, *Neural-network-based approximations for solving partial differential equations*, Communications in Numerical Methods in Engineering 10 (1994), pp. 195–201. Available at https://onlinelibrary.wiley.com/doi/abs/10.1002/cnm.1640100303.

[17] W. E and B. Yu, *The deep Ritz method: A deep learning-based numerical algorithm for solving variational problems*, eprint arXiv 1710.00211 (2017). Available at https://arxiv.org/abs/1710.00211.

[18] O.G. Ernst and M.J. Gander, *Why it is Difficult to Solve Helmholtz Problems with Classical Iterative Methods*, in *Numerical Analysis of Multiscale Problems*, Springer Berlin Heidelberg, Berlin, Heidelberg (2012), pp. 325–363. Available at https://doi.org/10.1007/978-3-642-22061-6_10.

[19] M. Gander and H. Zhang, *A class of iterative solvers for the Helmholtz equation: Factorizations, sweeping preconditioners, source transfer, single layer potentials, polarized traces, and optimized Schwarz methods*, SIAM Review 61 (2019), pp. 3–76. Available at https://doi.org/10.1137/16M109781X.

[20] S. Gratton, A. Sartenaer, and P.L. Toint, *Recursive trust-region methods for multiscale nonlinear optimization*, SIAM Journal on Optimization 19 (2008), pp. 414–444. Available at https://doi.org/10.1137/050623012.

[21] C. Groß and R. Krause, *On the convergence of recursive trust-region methods for multiscale nonlinear optmization and applications to nonlinear mechanics*, SIAM Journal on Numerical Analysis 47 (2009), pp. 3044–3069.

[22] E. Haber, L. Ruthotto, E. Holtham, and S. Jun, *Learning across scales-Multiscale methods for convolution neural networks*, in *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[23] W. Hackbusch, *Multi-grid methods and applications*, Springer Series in Computational Mathematics Vol. 4, Springer-Verlag, Berlin, Heidelberg, 1985, Available at https://www.springer.com/us/book/9783540127611.

[24] J. Han, A. Jentzen, and W. E, *Solving high-dimensional partial differential equations using deep learning*, Proceedings of the National Academy of Sciences 115 (2018), pp. 8505–8510. Available at https://doi.org/10.1073/pnas.1718942115.

[25] S. Haykin, *Neural networks: a comprehensive foundation*, 1st ed., Prentice Hall, 1994, Available at https://dl.acm.org/citation.cfm?id=541500.

[26] R. Hecht-Nielsen, *Theory of the backpropagation neural network*, in *International 1989 Joint Conference on Neural Networks*, Vol. 1. 1989, pp. 593–605. Available at https://doi.org/10.1109/IJCNN.1989.118638.

[27] C. Higham and D. Higham, *Deep learning: An introduction for applied mathematicians*, SIAM Review 61 (2019), pp. 860–891.

[28] M. Hutzenthaler, A. Jentzen, T. Kruse, T. Nguyen, and P. Wurstemberger, *Overcoming the curse of dimensionality in the numerical approximation of semilinear parabolic partial differential equations*, eprint arXiv 1807.01212 (2018). Available at https://arxiv.org/abs/1807.01212.

[29] A. Jentzen, D. Salimova, and T. Welti, *A proof that deep artificial neural networks overcome the curse of dimensionality in the numerical approximation of Kolmogorov partial differential equations with constant diffusion and nonlinear drift coefficients*, eprint arXiv 1809.07321 (2018). Available at https://arxiv.org/abs/1809.07321.

[30] T. Ke, M. Maire, and S.X. Yu, *Multigrid Neural Architectures*, in *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 4067–4075. Available at https://doi.org/10.1109/CVPR.2017.433.

[31] M. Kočvara and S. Mohammed, *A first-order multigrid method for bound-constrained convex optimization*, Optimization Methods and Software 31 (2016), pp. 622–644. Available at https://doi.org/10.1080/10556788.2016.1146267.

[32] A. Krizhevsky, I. Sutskever, and G.E. Hinton, *Imagenet classification with deep convolutional neural networks*, in *NIPS'12 Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. 2012, pp. 1097–1105. Available at https://dl.acm.org/citation.cfm?id=2999257.

[33] I.E. Lagaris, A. Likas, and D.I. Fotiadis, *Artificial neural networks for solving ordinary and partial differential equations*, IEEE Transactions on Neural Networks 9 (1998), pp. 987–1000. Available at https://doi.org/10.1109/72.712178.

[34] H. Lee and I.S. Kang, *Neural algorithm for solving differential equations*, Journal of Computational Physics 91 (1990), pp. 110–131. Available at https://doi.org/10.1016/0021-9991(90)90007-N.

[35] R.M. Lewis and S.G. Nash, *Using inexact gradients in a multilevel optimization algorithm*, Computational Optimization and Applications 56 (2013), pp. 39–61. Available at https://doi.org/10.1007/s10589-013-9546-7.

[36] R. Lewis and S. Nash, *Model problems for the multigrid optimization of systems governed by differential equations*, SIAM journal on Scientific Computing 26 (2005), pp. 1811–1837. Available at https://doi.org/10.1137/S1064827502407792.

[37] X. Li, J. Lowengrub, A. Rätz, and A. Voigt, *Solving PDEs in complex geometries: a diffuse domain approach*, Communications in Mathematical Sciences 7 (2009), p. 81.

[38] Z. Long, Y. Lu, X. Ma, and B. Dong, *PDE-Net: Learning PDEs from data*, in *Proceedings of the 35th International Conference on Machine Learning, PMLR 80*. 2018, pp. 3208–3216. Available at http://proceedings.mlr.press/v80/long18a.html.

[39] L. Manevitz, A. Bitar, and D. Givoli, *Neural network time series forecasting of finite-element mesh adaptation*, Neurocomputing 63 (2005), pp. 447 – 463. Available at https://doi.org/10.1016/j.neucom.2004.06.009.

[40] S. Mishra, *A machine learning framework for data driven acceleration of computations of differential equations*, Tech. Rep. 2018-28, Seminar for Applied Mathematics, ETH Zürich, 2018. Available at https://www.sam.math.ethz.ch/sam_reports/reports_final/reports2018/2018-28.pdf.

[41] J. Misra and I. Saha, *Artificial neural networks in hardware: A survey of two decades of progress*, Neurocomputing 74 (2010), pp. 239 – 255. Available at https://doi.org/10.1016/j.neucom.2010.03.021.

[42] S. Nash, *A multigrid approach to discretized optimization problems*, Optimization Methods and Software 14 (2000), pp. 99–116. Available at https://doi.org/10.1080/10556780008805795.

[43] S. Nash, *Properties of a class of multilevel optimization algorithms for equality constrained problems*, Optimization Methods and Software 29 (2014), pp. 137–159. Available at https://doi.org/10.1080/10556788.2012.759571.

[44] M. Raissi, P. Perdikaris, and G. Karniadakis, *Numerical Gaussian processes for time-dependent and nonlinear partial differential equations*, SIAM Journal on Scientific Computing 40 (2018), pp. A172–A198. Available at https://doi.org/10.1137/17M1120762.

[45] M. Raissi and G.E. Karniadakis, *Hidden physics models: Machine learning of nonlinear partial differential equations*, Journal of Computational Physics 357 (2018), pp. 125 – 141. Available at https://doi.org/10.1016/j.jcp.2017.11.039.

[46] M. Raissi, P. Perdikaris, and G.E. Karniadakis, *Physics informed deep learning (part I): data-driven solutions of nonlinear partial differential equations*, eprint arXiv 1711.10561 (2017). Available at http://arxiv.org/abs/1711.10561.

[47] M. Raissi, P. Perdikaris, and G.E. Karniadakis, *Physics informed deep learning (part II): data-driven discovery of nonlinear partial differential equations*, eprint arXiv 1711.10566 (2017). Available at http://arxiv.org/abs/1711.10566.

[48] P. Ramuhalli, L. Udpa, and S.S. Udpa, *Finite-element neural networks for solving differential equations*, IEEE Transactions on Neural Networks 16 (2005), pp. 1381–1392. Available at https://doi.org/10.1109/TNN.2005.857945.

[49] K. Rudd, *Solving partial differential equations using artificial neural networks*, Ph.D. diss., Cornell University, 2013. Available at http://lisc.mae.cornell.edu/PastThesis/KeithRuddPhD.pdf.

[50] S.H. Rudy, S.L. Brunton, J.L. Proctor, and J.N. Kutz, *Data-driven discovery of partial differential equations*, Science Advances 3 (2017). Available at http://advances.sciencemag.org/content/3/4/e1602614.

[51] J.W. Ruge and K. Stüben, *Algebraic multigrid*, in *Multigrid methods*, chap. 4, SIAM, 1987, pp. 73–130. Available at https://epubs.siam.org/doi/10.1137/1.9781611971057.ch4.

[52] E. Sadrfaridpour, T. Razzaghi, and I. Safro, *Engineering fast multilevel support vector machines*, eprint arXiv 1707.07657 (2017). Available at https://arxiv.org/abs/1707.07657.

[53] H. Schaeffer, *Learning partial differential equations via data discovery and sparse optimization*, Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences 473 (2017). Available at https://doi.org/10.1098/rspa.2016.0446.

[54] C.E. Shannon, *Communication in the presence of noise*, Proceedings of the IEEE 86 (1998), pp. 447–457. Available at https://doi.org/10.1109/JPROC.1998.659497.

[55] Y. Shirvany, M. Hayati, and R. Moradian, *Multilayer perceptron neural networks with novel unsupervised training method for numerical solution of the partial differential equations*, Applied Soft Computing 9 (2009), pp. 20 – 29. Available at http://www.sciencedirect.com/science/article/pii/S1568494608000276.

[56] J. Takeuchi and Y. Kosugi, *Neural network representation of finite el-*

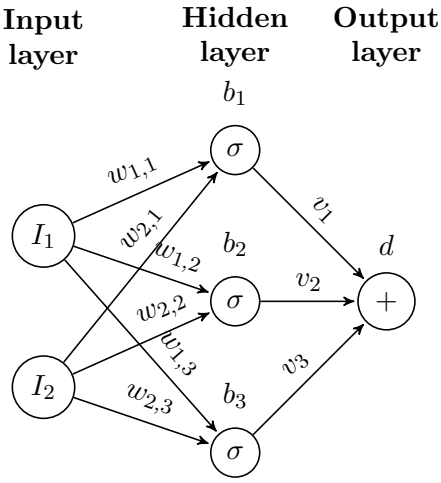

Figure A1. Artificial neural network architecture with weights and biases (Case of $r = 3$ and $N = 2$).

ement method, Neural Networks 7 (1994), pp. 389 – 395. Available at http://www.sciencedirect.com/science/article/pii/0893608094900310.

[57] U. Trottenberg, C.W. Oosterlee, and A. Schuller, *Multigrid*, Elsevier, New-York, 2000, Available at https://www.elsevier.com/books/multigrid/trottenberg/978-0-08-047956-9.

[58] Z. Wen and D. Goldfarb, *A line search multigrid method for large-scale nonlinear optimization*, SIAM J. on Optimization 20 (2009), pp. 1478–1503. Available at http://dx.doi.org/10.1137/08071524X.

Appendix A.

In Section 4.1, we have considered the simplest case of network's architecture, corresponding to $N = 1$. However, our method is also applicable when $N > 1$, as shown in Section 5. Here, we describe this extension (still considering the case of just one hidden layer). The network is still composed of three layers in total, but the input layer is composed of N nodes, one for each component of the input. The input nodes are connected to all the nodes in the hidden layer. Instead of having just r input weights, we have N groups of r input weights, $\{w_{i,1}, \dots, w_{i,r}\}$, $i = 1, \dots, N$, where the weights in group i are located on the edges departing from the i-th input node. The right part of the network remains the same, with one output node and r output weights. The input space of the objective function $\mathcal{F}$ in (14) will be $\mathbb{R}^{(N+2)r+1}$ rather than $\mathbb{R}^{3r+1}$. As an example, we depict the case $N = 2$ in Figure A1.

The training procedure is exactly the same, with the only difference that $p = [v, w_1, \dots, w_N, b, d] \in \mathbb{R}^{(N+2)r+1}$, with $w_i = [w_{i,1}, \dots, w_{i,r}]^T$, $i = 1, \dots, N$. In the construction of the matrix required for the application of the AMG coarsening strategy, the contribution of the input weights is now represented by N submatrices, corresponding to the derivatives with respect to couples (w_i, w_i), $i = 1, \dots, N$:

$$A = \frac{F_v^T F_v}{\|F_v\|_\infty} + \sum_{i=1}^{N} \frac{F_{w_i}^T F_{w_i}}{\|F_{w_i}\|_\infty} + \frac{F_b^T F_b}{\|F_b\|_\infty}.$$