# A Constrained Backpropagation Approach to Solving Partial Differential Equations in Non-stationary Environments

Gianluca Di Muro and Silvia Ferrari

*Abstract*— **A constrained-backpropagation (CPROP) training technique is presented to solve Partial Differential Equations (PDEs). The technique is based on constrained optimization and minimizes an error function subject to a set of equality constraints, provided by the boundary conditions of the differential problem. As a result, sigmoidal neural networks can be trained to approximate the solution of PDEs avoiding the discontinuity in the derivative of the solution, which may affect the stability of classical methods. Also, the memory provided to the network through the constrained approach may be used to solve PDEs on line when the forcing term changes over time, learning different solutions of the differential problem through a continuous nonlinear mapping. The effectiveness of this method is demonstrated by solving a nonlinear PDE on a circular domain. When the underlying process changes subject to the same boundary conditions, the CPROP network is capable of adapting online and approximate the new solution, while memory of the boundary conditions is maintained virtually intact at all times.**

## I. Introduction

PARTIAL Differential Equations (PDEs) occur in many problems in science and engineering, but in many cases they do not allow for a solution in closed analytic form. Therefore, a plethora of effective numerical methods have been established, such as the finite difference method (FDM) [1] and the finite element method (FEM) [2]. Although these methods provide good approximations to the solution, they require a domain discretization via meshing, which may be challenging in two or higher-dimensional problems. Also, the approximate solution's derivatives are discontinuous and may seriously impact its stability. Furthermore, in order to obtain a satisfactory solution accuracy it may be necessary to deal with fine meshes that significantly increase the computational cost.

Another approach to solving PDEs numerically is to adopt artificial neural networks (ANNs), exploiting their ability to provide universal function approximation for multivariate input/output spaces on a compact set. In this approach, the PDE solution is approximated by a feedforward ANN, and the adjustable parameters or weights are chosen to minimize an error function of the solution and its derivatives, based on the differential operator. Different techniques have been developed to take into account the boundary conditions. One line of research [3] expresses the solution as the sum of two functions. One function is problem dependent and is designed by the user to satisfy the boundary conditions (BCs) with no adjustable parameters. The second function is an ANN trained to approximate the differential operator. However, this approach only is applicable to PDEs defined on orthogonal box domains, and cannot be extended to non-stationary environments in which the BCs change over time because part of the solution must be designed by the user off-line. A more recent study [4] overcomes some of these limitations, but it adopts radial basis functions to correct the solution on the boundaries, making difficult to implement this technique in non-stationary environments, when the shape of the boundaries may change over time.

Another approach consists of embedding the BCs in the cost function [5], [6]. Although this approach has been shown effective in solving linear PDEs, in order to match BCs with high accuracy, which may be crucial in many engineering applications, it require many points on the integration domain boundaries, thereby increasing the computational cost dramatically. Also, it relies on evolutionary algorithms that are computationally expensive and, typically, can only be implemented reliably off line in non-stationary environments. A methodology based on radial basis functions ANNs was developed in [7], by learning the adjustable parameters using a two-stage gradient-descent strategy. One advantage of this methodology is that the ANN architecture is adapted over time using a node insertion strategy. However, because it includes the BCs in the cost function, this methodology suffers from the same aforemention limitations, as [5].

Recently, the authors presented a novel constrained-backpropagation (CPROP) approach to eliminate interference and preserve long-term memory (LTM) in fully-connected sigmoidal neural networks [8]. CPROP preserves LTMs by embedding them into a set of equality constraints that are formulated in terms of the neural weights by means of algebraic training [9]. In [8], the CPROP approach was illustrated by preserving LTM of gain scheduled controllers in a neural network controller that adapted to nonlinear plant dynamics on line, via an adaptive-critic architecture; whereas in [10], it was extended to other engineering applications, namely, function approximation, solution of ordinary differential equations, and system identification. The previous results demonstrated the ability of CPROP to retain *procedural memories*, which refer to memories of actions and motor sequences.

In this paper, the CPROP approach is extended to the problem of solving PDEs in non-stationary environments, in which the BCs constitute the LTMs to be retained during on-line learning. Also, new relations to deal with nonlinear

This work was supported by National Science Foundation, under grants ECS CAREER 0448906, and ECS 0823945.

G. Di Muro is a graduate student of Mechanical Engineering at Duke University, Durham, NC 27707, USA gianluca.dimuro@duke.edu.

S. Ferrari is with Faculty of Mechanical Engineering at Duke University, Durham, NC 27707, USA sferrari@duke.edu.

PDEs and analytic expressions for the Jacobian of differential operators, approximated by ANNs, are analytically derived. The results clearly show the CPROP approach is effective at providing the solution to nonlinear PDEs and BCs are always satisfied during on-line learning of several forcing functions.

## II. Formulation of the problem

Consider the nonlinear PDE,

$$\mathcal{D}^k\left[u(\mathbf{y})\right] = f(\mathbf{y}) \tag{1}$$

where $\mathcal{D}^k$ is a non-linear differential operator of order $k$, $\mathbf{y} \in \mathcal{I} \subset \mathbb{R}^N$, and $\mathcal{I} \in \mathbb{R}^N$ is a compact set with associated boundary conditions (BCs),

$$\mathcal{G}^j\left[u(\mathbf{y})\right] = h(\mathbf{y}) \tag{2}$$

$\mathcal{G}$ is a linear operator of order $j < k$. The functions $\mathbf{y} \in \partial\mathcal{I} \subset \mathbb{R}^N$ and $f, h : \mathbb{R}^N \to \mathbb{R}$ are assumed to be continuous and known. Without the loss of generality, assume that $\mathcal{D}^k = \mathcal{L}^{k_1} + \mathcal{H}^{k_2}$, where $k = \max\{k_1, k_2\}$, $\mathcal{L}^{k_1}$ is a linear differential operator of order $k_1$, and $\mathcal{H}^{k_2}$ is a nonlinear differential operator of order $k_2$ of the form,

$$\mathcal{H}^{k_2} = \sum_{m=1}^{N}\sum_{l=1}^{k_2}\sum_{r=1}^{R_l} c_{lmr}\ \frac{\partial^l u}{\partial y_m^l}\ u_r(\mathbf{y})$$

The solution $\hat{u}(\mathbf{y})$ is provided by the output of an ANN with adjustable parameters $\mathbf{W}, \mathbf{d}, \mathbf{v}$, and the output bias is set to zero, for simplicity. The input-to-node operator is defined as,

$$\mathbf{\Phi}\left(\mathbf{W}, \mathbf{y}, \mathbf{d}\right) = \mathbf{W}\ \mathbf{y} + \mathbf{d} \tag{3}$$

where, $\mathbf{W} \in \mathbb{R}^{S\times N}, \mathbf{d} \in \mathbb{R}^S$ are the input weights and bias, respectively, $S$ is the number of the hidden nonlinear units, and $\mathbf{\Phi} : \mathbb{R}^n \to \mathbb{R}^S$ is the linear operator which maps the input space into the node space. Furthermore, assume that the hidden nonlinear units are sigmoidal, with a transfer function $\sigma(r) = (e^r - 1)/(e^r + 1)$. Then, define the sigmoidal operator

$$\boldsymbol{\mathcal{S}}\left(\mathbf{W}, \mathbf{y}, \mathbf{d}\right) = \sigma\left(\mathbf{\Phi}\right) \tag{4}$$

as the nonlinear mapping $\boldsymbol{\mathcal{S}} : \mathbb{R}^s \to \mathbb{R}^s$ which estimates the output of the nonlinear hidden units of the network. Hence, the approximation of the solution of the PDE (1) is given by the ANN output provided it satisfies the relationship,

$$\hat{u}(\mathbf{y}) = \boldsymbol{\mathcal{S}}\left(\mathbf{W}, \mathbf{y}, \mathbf{d}\right)\mathbf{v}^T \tag{5}$$

where $\mathbf{v} \in \mathbb{R}^{1\times S}$ is the vector of the output weights. It is then trivial to extend the approach to the case of multiple outputs' network because of their linearity with respect to the output weights. Substituting (5) into (1), the differential operator is applied to the ANN,

$$\mathcal{D}^k\left[\boldsymbol{\mathcal{S}}\left(\mathbf{W}, \mathbf{y}, \mathbf{d}\right)\mathbf{v}^T\right] = f(\mathbf{y}) \tag{6}$$

Since the aim is to approximate the solution of the problem (1) on $\mathcal{D}$, let the set $\mathcal{T}_{\text{STM}} = \{\mathbf{y}^k \in \mathcal{D}, k = 1, 2, ..., P\}$ provide the training set for the PDE (1). The discretized version of the input-to-node operator (3) is given by the input-to-node matrix,

$$\mathbf{N} = \left[\mathbf{W}\mathbf{Y} + \mathbf{D}\right]^T \tag{7}$$

where, $\mathbf{Y} = \left[\mathbf{y}^1\ \mathbf{y}^2 \cdots \mathbf{y}^P\right]$ is an $N \times P$ matrix of output samples, and $\mathbf{D} = [\mathbf{d}\ \mathbf{d} \cdots \mathbf{d}]$ is an $S \times P$ matrix of input biases, and $\mathbf{N} \in \mathbb{R}^{P\times S}$. Similarly, the discrete version of the sigmoidal operator (4) is a $P \times S$ matrix that, from (7), can be written as,

$$\mathbf{S}^0 = \boldsymbol{\mathcal{S}}\left(\mathbf{N}\right) \tag{8}$$

Consequently, when the ANN is fed the training set input samples from $\mathcal{T}_{\text{STM}}$ its output is given by,

$$\hat{u}(\mathbf{y})|_{\mathbf{y}\in\mathcal{T}_{\text{STM}}} = \mathbf{S}^0\mathbf{v}^T \tag{9}$$

In order to solve (1), the ANN output must be differentiated with respect to its inputs up to the $k^{th}$-order derivative.

After some manipulations it is possible to extend the scalar equations presented in [3], adopting the operators previously introduced. The operators,

$$\mathbf{T} = \prod_{i=1}^{n}\mathbf{W}_i^{m_i} \quad \text{and} \tag{10}$$

$$\mathbf{R}_j = \begin{cases} \mathbf{W}_j^{m_j-1}\prod_{i=1,\ i\neq j}^{n}\mathbf{W}_i^{m_i} & \text{if } m_j \geq 1 \\ \mathbf{0} \in \mathbb{R}^{S\times S} & \text{otherwise} \end{cases} \tag{11}$$

are introduced to obtain a more compact notation. Then, the derivative of the ANN evaluated at the samples in $\mathcal{T}_{\text{STM}}$ is given by,

$$\frac{\partial^{m_1}}{\partial y_1^{m_1}}\cdots\frac{\partial^{m_j}}{\partial y_j^{m_j}}\cdots\frac{\partial^{m_n}}{\partial y_n^{m_n}}\left[\hat{u}(\mathbf{y})\right]|_{\mathbf{y}\in\mathcal{T}_{\text{STM}}} = \mathbf{S}^\lambda\ \mathbf{T}\ \mathbf{v}^T \tag{12}$$

where $\mathbf{S}^\lambda$ denotes the $\lambda^{\text{th}}$ derivative of the $\sigma$ function with respect to its scalar argument evaluated at the input-to-node matrix (7), and from hereon will be referred to as *transfer function matrix* of the $\lambda^{\text{th}}$ order, where $\lambda = \sum_{i=1}^{n} m_i$, and $d^0\sigma/dr^0 = \sigma$. $N$ diagonal matrices $\mathbf{W}_j \in \mathbb{R}^{S\times S}$, with $j = 1, 2, ..., N$ are defined such that the $j^{\text{th}}$ component on the diagonal is given by the $j^{\text{th}}$ input weight of the ANN.

We are now ready to compute the Jacobian of the error with respect to the ANN adjustable parameters required in order to train the ANN by backpropagation. Making use of equations (10)-(11), the Jacobian can be written as,

$$\mathbf{J} = [\ \mathbf{J}_{\mathbf{W}^1}\ |\ \cdots\ |\ \mathbf{J}_{\mathbf{W}^N}\ |\ \mathbf{J}_{\mathbf{d}}\ |\ \mathbf{J}_{\mathbf{v}}] \tag{13}$$

where,

$$\mathbf{J}_{\mathbf{W}^i} = \left(m_i\ \mathbf{S}^\lambda\mathbf{R}_i + \mathbf{Y}_i\mathbf{S}^{\lambda+1}\mathbf{T}\right)\ \mathbf{V}\ \ i = 1, ..., N \tag{14}$$

$$\mathbf{J}_{\mathbf{d}} = \mathbf{S}^{\lambda+1}\mathbf{T}\ \mathbf{V} \tag{15}$$

$$\mathbf{J}_{\mathbf{v}} = \mathbf{S}^\lambda\mathbf{T} \tag{16}$$

and $\mathbf{Y}_i \in \mathbb{R}^{P\times P}, i = 1, 2, ..., N$ are diagonal matrices, defined as: $[\mathbf{Y}_i]_l^k = y_i^k\delta_{kl}$, where $\delta_{kl}$ is the Kronecker delta and $y_i^k$ are the $i^{\text{th}}$ components of the $k^{\text{th}}$ samples' inputs, with no implied summation over indices. Similarly $\mathbf{V} \in \mathbb{R}^{S\times S}$ refers to a diagonal matrix assembled with the components of the $\mathbf{v}$ vector, or in components $[\mathbf{V}]_{mn} = v_m\delta_{mn}\ m = 1, ..., S$, and $v_m$ is the $m^{\text{th}}$ component of the output weights.

Introducing the vector $\mathbf{f} = f(\mathbf{y})|_{\mathbf{y}\in\mathcal{T}_{\text{STM}}}$ with $\mathbf{f} \in \mathbb{R}^P$, the cost function to be minimized may be defined as,

$$V = \frac{1}{2}\mathbf{e}^T\mathbf{e} \tag{17}$$

where

$$\mathbf{e} = \mathcal{D}^k\left[\hat{u}(\mathbf{y})\right]|_{\mathbf{y}\in\mathcal{T}_{\text{STM}}} - \mathbf{f} \tag{18}$$

## III. Extension to Nonlinear PDEs

The methodology presented in the previous section can be extended to *nonlinear* PDEs by adopting a useful property of the Hadamard product. The Hadamard product of two matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{M\times N}$, also known as entry-wise product, is defined as,

$$(\mathbf{A}\circ\mathbf{B})_{ij} = a_{ij}b_{ij} \tag{19}$$

As shown in [11], the Hadamard product obeys the following property,

$$\frac{\partial\left(\mathbf{A}\circ\mathbf{B}\right)}{\partial\alpha} = \frac{\partial\mathbf{A}}{\partial\alpha}\circ\mathbf{B} + \mathbf{A}\circ\frac{\partial\mathbf{B}}{\partial\alpha} \tag{20}$$

where, $\alpha$ is a scalar parameter, and $\mathbf{A}$ and $\mathbf{B}$ are matrices or matrix functions. In order to derive the Jacobian for the nonlinear part, we extend the property in (20) to the case of differentiation with respect to vectors. After some algebraic manipulations it may be shown that the following expression holds,

$$\frac{\partial\left(\mathbf{A}\circ\mathbf{B}\right)}{\partial\mathbf{a}} = \frac{\partial\mathbf{A}}{\partial\mathbf{a}}\circ(\mathbf{B}\otimes\boldsymbol{\gamma}) + \frac{\partial\mathbf{B}}{\partial\mathbf{a}}\circ(\mathbf{A}\otimes\boldsymbol{\gamma}) \tag{21}$$

where in (21) $\mathbf{a} \in \mathbb{R}^l$ and $\boldsymbol{\gamma} \in \mathbb{R}^l$ are row-vectors, with $\boldsymbol{\gamma}$ defined as $\boldsymbol{\gamma} = [1\ 1\cdots\ 1]$. The symbol $\otimes$ denotes the Kronecker product between tensors. Expression (21) may be adopted to extend the constrained methodology to estimate the Jacobian operator in case of nonlinear equation. Although still some class of nonlinear functions may not be directly treated, it is possible to extend this approach, through subsequent Taylor's expansion.

## IV. Partition of the weights: enforcing boundary conditions

Imposing only the minimization of the cost function defined in (17) is not sufficient to have an approximate solution of the problem (1), since it has to satisfy the boundary conditions provided by (2). In order to achieve that, we are going to sample $\partial\mathcal{I}$ and impose (2) at these discrete points. According to the developed methodology we will able to deal with any kind of boundary conditions (Dirichlet, Neumann, mixed type) as long as the relations to be imposed on the boundary are linear, with respect to the unknowns. For simplicity we consider Dirichlet's boundary conditions; first of all let us define the set $\mathcal{T}_{\text{LTM}} = \{\mathbf{y}^j \in \partial\mathcal{D}, j = 1, 2, ..., \bar{P}\}$ where we are going to enforce the satisfaction of (2).

Using the approach in [8], partition the hidden nonlinear units into Long Term Memory (LTM) and Short Term Memory (STM) nodes and similarly we can refer to their weights as LTM and STM weights. The former are to enforce the constraints which have to be satisfied at any time by the ANN, which -in this case- are constituted by the boundary conditions; whereas the latter are used to acquire new knowledge and minimize the cost function (17). In agreement with this approach, we can claim that $S = S_{\text{STM}} + S_{\text{LTM}}$ where $S_{\text{STM}}$ and $S_{\text{LTM}}$ are the number of STM and LTM connections, respectively. In order to deal with linear equations we keep constant input-to-node LTM matrices, so that we can easily invert constraints equations using linear algebra and LTM output weights will be computed to enforce (2).

### A. Notation

From now on we will refer to the STM connections of the network using Latin letters, whereas Greek letters will address LTM connections. Also, inputs with a bar ($\bar{}$) will belong to the $\mathcal{T}_{\text{LTM}}$ training set, whereas inputs with a *breve* ($\breve{}$) will refer to the $\mathcal{T}_{\text{STM}}$ training set. For example $\bar{\mathbf{S}}^0$ refers to the STM transfer function matrix of zeroth order fed with the LTM inputs and similarly $\breve{\boldsymbol{\Sigma}}^2$ refers to the LTM transfer function matrix of second order fed with STM inputs.

### B. The Constraints: Imposing Boundary Conditions

Given (2), we want the Boundary Conditions to be always satisfied. Thus we can use (12) to evaluate (2); also, since (2) is linear for hypothesis, choosing $S_{\text{LTM}} = \bar{C}$ we can deal with equations linear with respect to the LTM output weights, where $\bar{C}$ is the total number of boundary conditions. Without loss of generality, let us consider the case of Dirichlet boundary conditions to be imposed for $\bar{P}$ points, then the ANN's output has to satisfy,

$$\bar{\mathbf{S}}^0\mathbf{v}^T + \bar{\boldsymbol{\Sigma}}^0\boldsymbol{\omega}^T = \mathbf{h} \tag{22}$$

where $\bar{\mathbf{S}}^0 \in \mathbb{R}^{S_{\text{STM}}\times\bar{P}}, \bar{\boldsymbol{\Sigma}}^0 \in \mathbb{R}^{S_{\text{LTM}}\times\bar{P}}, \mathbf{h} = h(\mathbf{x}^k), k = 1, 2, ..., \bar{P},\ \mathbf{h} \in \mathbb{R}^{\bar{P}}$; finally $\mathbf{v} \in \mathbb{R}^{1\times S_{\text{STM}}}$ and $\boldsymbol{\omega} \in \mathbb{R}^{1\times S_{\text{LTM}}}$ are the STM and LTM output weights, respectively. In this case $\bar{C} = \bar{P}$ and we have imposed $S_{\text{LTM}} = \bar{C}$, hence $\bar{\boldsymbol{\Sigma}}^0 \in \mathbb{R}^{S_{\text{LTM}}\times S_{\text{LTM}}}$ is a square-matrix. According to our assumptions we have fixed the input-to-node LTM matrix, therefore $\bar{\boldsymbol{\Sigma}}^0$ is completely known; moreover we can predesign it in order to make it non-singular, for the given boundary discretization. With these premises the constraints' equation (22) may be inverted and the LTM output weights may be expressed as a function of the STM weights; for Dirichlet's boundary conditions we would have explicitly,

$$\boldsymbol{\omega}^T = \left[\bar{\boldsymbol{\Sigma}}^0\right]^{-1}\left(\mathbf{h} - \bar{\mathbf{S}}^0\mathbf{v}^T\right) \tag{23}$$

Thus the solution of the PDE may be obtained, minimizing (17) subject to (23). In order to apply gradient-descent methods we have to evaluate the augmented Jacobian, which will be function only of the STM weights, since the LTM internal matrix is fixed and the LTM output weights are explicitly expressed in terms of the STM weights, through equation (23).

## V. Applications and Results

In order to illustrate how the ANN is capable to adapt and provide new solutions in a non-stationary environment, we consider a nonlinear equation, forced by a known term, which is subjected to some changes over time. Given the

following bidimensional PDE on the unit circle centred at the origin,

$$\nabla^2 u + u \, \frac{\partial u}{\partial y_2} = f^j(y_1, y_2) \quad j = 1, 2, 3 \tag{24}$$

we assume that the solution has to satisfy the following boundary conditions,

$$u(y_1, y_2) = 1 \tag{25}$$

on the unit circle, for which $y_1^2 + y_2^2 = 1$. Note that in (24) we have omitted the dependence of $u$ from $y_1, y_2$ for simplicity of notation; $f^j(y_1, y_2) \quad j = 1, 2, 3$ are continuous functions, whose values are supposed to be known on the discrete grid, where we are aiming to solve (24). With the position (9), it is straightforward to show that the error equation takes the form,

$$\mathbf{e}(\mathbf{X}) = \mathbf{e}_{\text{LIN}}(\mathbf{X}) + \mathbf{e}_{\text{NLIN}}(\mathbf{X}) \tag{26}$$

where $\mathbf{f} \in \mathbb{R}^P$ is defined as: $\mathbf{f}^k = f(\mathbf{y})|_{\mathbf{y} \in \mathcal{T}_{\text{STM}}}$. In (26), the explicit computation of the two contributions may be easily made, using (12), therefore we have,

$$\mathbf{e}_{\text{LIN}}(\mathbf{X}) = \breve{\mathbf{S}}^2 \mathbf{B} \mathbf{v}^T + \breve{\mathbf{\Sigma}}^2 \mathbf{C} \boldsymbol{\omega}^T - \mathbf{f}^k \;\; k = 1, 2, 3 \tag{27}$$

with $\mathbf{B} = (\mathbf{W}_1^2 + \mathbf{W}_2^2)$ and $\mathbf{C} = (\mathbf{\Omega}_1^2 + \mathbf{\Omega}_2^2)$ and for the nonlinear part we have the following expression,

$$\mathbf{e}_{\text{NLIN}}(\mathbf{X}) = (\breve{\mathbf{\Sigma}}^0 \boldsymbol{\omega}^T + \breve{\mathbf{S}}^0 \mathbf{v}^T) \circ (\breve{\mathbf{\Sigma}}^1 \mathbf{\Omega}_2 \boldsymbol{\omega}^T + \breve{\mathbf{S}}^1 \mathbf{W}_2 \mathbf{v}^T) \tag{28}$$

We are considering the case of a forcing term $f$, transitioning from $f^1$ to $f^2$ and finally to $f^3$. We have predesigned the analytic solution, to have an exact estimate of the error, and have computed the $f^k$ subsequently. The solutions are: $u^1 = y_1^2 + y_2^2 \rightarrow u^2 = 1.15 \; u^1 - 0.15 \rightarrow u^3 = e^{-u^1} + 1.1 \; u^1 - e^{-1} - 0.1$ and all satisfy (25). In order to solve (24) we need to evaluate the Jacobian. Thus we have,

$$\mathbf{J} = \mathbf{J}_{\text{LIN}} + \mathbf{J}_{\text{NLIN}}$$

And the explicit expressions of the two parts are respectively:

$$\mathbf{J}_{\text{LIN}} = \left[ \mathbf{J}_{\nabla^2}^{\mathbf{W}_1} \mid \mathbf{J}_{\nabla^2}^{\mathbf{W}_2} \mid \mathbf{J}_{\nabla^2}^{\mathbf{d}} \mid \mathbf{J}_{\nabla^2}^{\mathbf{v}} \right] + \mathbf{\Pi}_1 \mathbf{J}_2 \tag{29}$$

where the different terms in (29) are,

$$\begin{aligned} \mathbf{J}_{\nabla^2}^{\mathbf{W}_i} &= 2 \breve{\mathbf{S}}^2 \mathbf{W}_i \mathbf{V} + \breve{\mathbf{Y}}_i \breve{\mathbf{S}}^3 \mathbf{B} \; \mathbf{V} \qquad i = 1, 2 \\ \mathbf{J}_{\nabla^2}^{\mathbf{d}} &= \breve{\mathbf{S}}^3 \mathbf{B} \; \mathbf{V} \\ \mathbf{J}_{\nabla^2}^{\mathbf{v}} &= \breve{\mathbf{S}}^2 \mathbf{B} \end{aligned}$$

and $\mathbf{\Pi}_1$ is responsible to ensure that the BCs are respected, for the part pertinent to the Laplacian, and is defined as,

$$\mathbf{\Pi}_1 = -\breve{\mathbf{\Sigma}}^2 \; \mathbf{B} \; \left[ \bar{\mathbf{\Sigma}}^0 \right]^{-1}$$

similarly $\mathbf{J}_2$ takes into account the interaction between LTM and STM weights, for the computation of the Laplacian operator and it is defined accordingly as,

$$\mathbf{J}_2 = \left[ \bar{\mathbf{Y}}_1 \bar{\mathbf{S}}^1 \mathbf{V} \mid \bar{\mathbf{Y}}_2 \bar{\mathbf{S}}^1 \mathbf{V} \mid \bar{\mathbf{S}}^1 \mathbf{V} \mid \bar{\mathbf{S}} \right]$$

whereas the nonlinear contribution to the jacobian takes the expression,

$$\mathbf{J}_{\text{NLIN}} = [\hat{\mathbf{u}} \otimes \boldsymbol{\gamma}] \circ [\mathbf{J}_3 + \mathbf{\Pi}_3 \mathbf{J}_4] + [\hat{\mathbf{u}}_{,y_2} \otimes \boldsymbol{\gamma}] \circ [\mathbf{J}_5 + \mathbf{\Pi}_2 \mathbf{J}_4] \tag{30}$$

where $\hat{\mathbf{u}}$ and $\hat{\mathbf{u}}_{,y_2}$ refer to the ANN's output and its partial derivative with respect to $y_2$, computed on the STM training set respectively. Therefore they may be expressed through the network weights as,

$$\begin{aligned} \hat{\mathbf{u}} &= \breve{\mathbf{S}}^0 \mathbf{v}^T + \breve{\mathbf{\Sigma}}^0 \boldsymbol{\omega}^T \\ \hat{\mathbf{u}}_{,y_2} &= \breve{\mathbf{S}}^1 \mathbf{W}_2 \mathbf{v}^T + \breve{\mathbf{\Sigma}}^1 \mathbf{\Omega}_2 \boldsymbol{\omega}^T \end{aligned}$$

Explicitly, in (30) we have the following expressions for the different components of the nonlinear term of the Jacobian,

$$\mathbf{J}_3 = \left[ \mathbf{J}_3^{\mathbf{W}_1} \mid \mathbf{J}_3^{\mathbf{W}_2} \mid \mathbf{J}_3^{\mathbf{d}} \mid \mathbf{J}_3^{\mathbf{v}} \right]$$

with

$$\begin{aligned} \mathbf{J}_3^{\mathbf{W}_1} &= \breve{\mathbf{Y}}_1 \breve{\mathbf{S}}^2 \mathbf{W}_2 \mathbf{V} \\ \mathbf{J}_3^{\mathbf{W}_2} &= (\breve{\mathbf{Y}}_2 \breve{\mathbf{S}}^2 \mathbf{W}_2 + \breve{\mathbf{S}}^1) \mathbf{V} \\ \mathbf{J}_3^{\mathbf{d}} &= \breve{\mathbf{S}}^2 \mathbf{W}_2 \mathbf{V} \\ \mathbf{J}_3^{\mathbf{v}} &= \breve{\mathbf{S}}^1 \mathbf{W}_2 \end{aligned}$$

where $\mathbf{\Pi}_2 = -\breve{\mathbf{\Sigma}}^0 \; \left[ \bar{\mathbf{\Sigma}}^0 \right]^{-1}$, $\mathbf{\Pi}_3 = -\breve{\mathbf{\Sigma}}^1 \mathbf{\Omega}_2 \left[ \bar{\mathbf{\Sigma}}^0 \right]^{-1}$ and analogously the other terms are defined as,

$$\begin{aligned} \mathbf{J}_4 &= \left[ \bar{\mathbf{Y}}_1 \bar{\mathbf{S}}^1 \mathbf{V} \mid \bar{\mathbf{Y}}_2 \bar{\mathbf{S}}^1 \mathbf{V} \mid \bar{\mathbf{S}}^1 \mathbf{V} \mid \bar{\mathbf{S}}^0 \right] \\ \mathbf{J}_5 &= \left[ \breve{\mathbf{Y}}_1 \breve{\mathbf{S}}^1 \mathbf{V} \mid \breve{\mathbf{Y}}_2 \breve{\mathbf{S}}^1 \mathbf{V} \mid \breve{\mathbf{S}}^1 \mathbf{V} \mid \breve{\mathbf{S}}^0 \right] \end{aligned}$$

For the numerical simulations, we have adopted a $100$ points grid, posed on $10$ different circles -equidistant from the center of the domain of integration- and subdividing each of them into $10$ equidistant arcs. In order not to have a homogeneous radial distribution of points, we have imposed a $\theta = \pi/8$ swirl (positive counter-clockwise) from one circle to another, beginning from the external one (of unitary radius). We have used a $7$ STM ANN architecture and have adopted $25$ LTM nonlinear hidden nodes, to impose the boundary conditions on the unit circle (subdividing it, again, into equal parts). Results are shown from adaptation to the known terms of equation (24) where the function $f$ is supposed to have changed after a certain number of epochs (namely: $50$). The final training has been run until satisfactory convergence has occurred and no further improvement seemed possible (after $450$ epochs). Again, as validation set, we have chosen a much denser grid of $90$ circles subdivided into twenty arcs. Figures (1 - 2), (3 - 4), (5 - 6) show the ANN output and the error surface respectively for the case of adaptation to $f^1, f^2, f^3$. To obtain surface polar plots we have used POLAR3D [12].

## VI. Conclusions

A constrained-backpropagation approach is presented to solve PDEs on non-rectangular domains and in non-stationary environments. The methodology utilizes constrained optimization to minimize an error function provided by the PDE operator, subject to equality constraints obtained from the PDE's boundary conditions. Since the equality

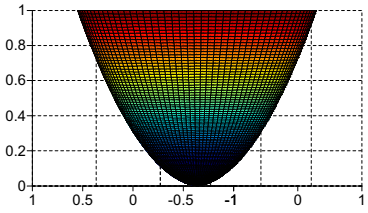


Fig. 1. Neural Network output after being trained with $f^1$

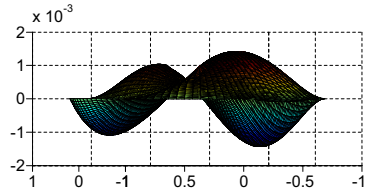


Fig. 2. Error surface comparing the analytic solution and the NN output when trained with $f^1$

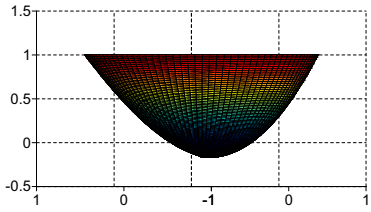


Fig. 3. Neural Network output after being trained with $f^2$

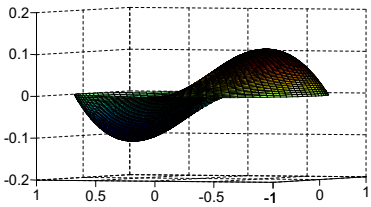


Fig. 4. Error surface comparing the analytic solution and the NN output when trained with $f^2$

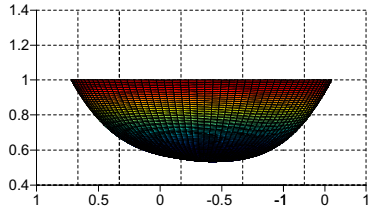


Fig. 5. Neural Network output after being trained with $f^3$

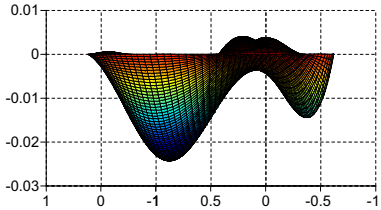


Fig. 6. Error surface comparing the analytic solution and the NN output when trained with $f^3$

constraints are derived analytically via an algebraic training approach, it is possible to satisfy the boundary conditions exactly, up to machine precision. This feature is crucial especially in the case of nonlinear PDEs, whose solution may be greatly affected by even small errors on the boundaries. The numerical results presented in this paper show that high precision in matching the boundary conditions is obtained using far fewer sample points on the boundaries than existing methods based on unconstrained backpropagation. Another important advantage of the proposed methodology is that the boundary conditions are satisfied during on-line learning of short-term memories comprised of new forcing functions, as brought about by non-stationary processes. Future work will investigate the computing requirements in case of larger system models. Also, the methodology will be further applied to study real-world data.

## REFERENCES

[1] G. D. Smith, *Numerical solution of partial differential equations: Finite difference methods*. Oxford: ClarendonPress, 1978.

[2] T. J. R. Hughes, *The finite element method*. New Jersey: Prentice Hall, 1987.

[3] I. Lagaris, A. Likas, and D. I. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE Trans. On Neural Networks*, vol. 9, no. 5, pp. 987–1000, 1998.

[4] I. Lagaris, A. Likas, and D. Papageorgiou, "Neural-network methods for boundary value problems with irregular boundaries," *Neural Networks, IEEE Transactions on*, vol. 11, no. 5, pp. 1041–1049, Sep 2000.

[5] L. P. Aarts and P. V. D. Veer, "Neural network method for solving partial differential equations," *Neural Process. Lett.*, vol. 14, no. 3, pp. 261–271, 2001.

[6] Y. Shirvany, M. Hayati, and R. Moradian, "Numerical solution of the nonlinear schrodinger equation by feedforward neural networks," *Communications in Nonlinear Science and Numerical Simulation*, vol. 13, no. 10, pp. 2132 – 2145, 2008.

[7] L. Jianyu, L. Siwei, Q. Yingjian, and H. Yaping, "Numerical solution of elliptic partial differential equation using radial basis function neural networks," *Neural Networks*, vol. 16, no. 5-6, pp. 729–734, 2003.

[8] S. Ferrari and M. Jensenius, "A constrained optimization approach to preserving prior knowledge during incremental training," *IEEE Transactions On Neural Networks*, vol. 19, no. 6, pp. 996–1009, 2008.

[9] S. Ferrari and R. Stengel, "Smooth function approximation using neural networks," *Neural Networks, IEEE Transactions on*, vol. 16, no. 1, pp. 24–38, Jan. 2005.

[10] G. Di Muro and S. Ferrari, "A constrained-optimization approach to training neural networks for smooth function approximation and system identification," in *Proc. International Joint Conference on Neural Networks*, Hong Kong, 2008, pp. 2354–2360.

[11] K. B. Petersen and M. S. Pedersen, *The Matrix CookBook*, February 2007.

[12] J. M. De Freitas, *POLAR3D: A 3-Dimensional Polar Plot Function in Matlab®*, QinetiQ Ltd, Winfrith Technology Centre, Winfrith, Dorchester DT2 8DXJ. UK, June 2005.