A Constrained-Optimization Approach to Training Neural Networks for Smooth Function Approximation and System Identification

Gianluca Di Muro and Silvia Ferrari

***Abstract*— A constrained-backpropagation training technique is presented to suppress interference and preserve prior knowledge in sigmoidal neural networks, while new information is learned incrementally. The technique is based on constrained optimization, and minimizes an error function subject to a set of equality constraints derived via an algebraic training approach. As a result, sigmoidal neural networks with long term procedural memory (also known as implicit knowledge) can be obtained and trained repeatedly on line, without experiencing interference. The generality and effectiveness of this approach is demonstrated through three applications, namely, function approximation, solution of differential equations, and system identification. The results show that the long term memory is maintained virtually intact, and may lead to computational savings because the implicit knowledge provides a lasting performance baseline for the neural network.**

I. Introduction

SIGMOIDAL neural networks (NNs) are used in a variety of applications thanks to their ability to provide excellent universal function approximation for multivariate input/output spaces on a compact set. However, as was first pointed out by Cohen [1], Todorov [2], and Ratcliff [3], they also suffer from a serious limitation known as interference, which impairs their ability to learn multivariate mapping sequentially. Interference refers to the phenomenon by which the process of learning new patterns may suddenly and completely erase the previous knowledge of the network. Thus, it may seriously jeopardize the use of artificial neural networks in applications where new knowledge becomes available on-line or cannot be assimilated at once due to computational complexity issues. We define as *long-term memory* (LTM) the prior knowledge that must be preserved by an artificial neural network at all times. New information that is assimilated incrementally but needs not be consolidated into LTM is referred to as *short-term memory* (STM).

The ability to suppress interference and to accurately preserve LTMs is crucial to many engineering applications, including control, modeling of system dynamics, and system identification (ID). Consequently, various solutions have been proposed to address the problem. One approach presents some of the LTM and STM data together to suppress interference in supervised incremental training algorithms [4], [5]. While very effective for some applications, it is not suitable for those that require highly accurate preservation of LTM knowledge, or that have stringent computational requirements due, for example, to large input-output spaces, or to learning being performed online. Another approach consists of using extra hidden units to augment the approximation power of the neural network, and of partitioning the weights into two subsets, referred to as LTM and STM weights. The LTM weights are used to preserve LTM data by holding their values constant, while the STM weights are updated with the new STM data [6], [7]. The latter approach is more consistent with the memory-formation mechanisms observed in the nervous system [8]. However, due to nonlinearity and global support properties, modifying the STM weights in a sigmoidal neural network can unfortunately also significantly change its input-output mapping everywhere in its domain. Therefore, holding the LTM weights constant cannot always guarantee accurate LTM preservation, and is particularly ineffective when training sigmoidal neural networks.

Considerable success has also been met in the field of self-organizing networks and associative memories (e.g., [9], [10], [11], [12], [13]). In these networks, the neurons use competitive learning to recognize groups of similar input vectors and associate them with a particular output by allowing neurons that are physically near each other to respond to similar inputs. Although these networks are very important to pattern recognition and classification applications, they cannot substitute for nonlinear backpropagating NNs (e.g., multilayer sigmoidal perceptrons) in function approximation, system identification, and control tasks. Nonlinear backpropagating NNs can also greatly benefit from the ability to memorize patterns in the long term, and from the elimination of catastrophic forgetting. Moreover, most of the aforementioned research consists of preserving *declarative memories*, which refer to episodic and semantic forms of memories, such as facts and categories.

Recently, the authors presented a novel constrained-backpropagation (CPROP) approach to eliminate interference and preserve LTM in fully-connected sigmoidal neural networks [14]. 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 [15]. In [14], it 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. The previous results demonstrated the ability of CPROP to retain *procedural memories*, also known as implicit knowledge, which refer to memories of actions and motor sequences, such as as how to ride a bicycle.

In this paper, the CPROP approach is extended to other

This work was supported by National Science Foundation, under grant ECS CAREER 0300236

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

engineering applications, namely, function approximation, solution of ordinary differential equations (ODEs), and system identification. In the function approximation application, the LTM consists of prior knowledge of the function to be approximated over a bounded domain (Section IV-A). In the ODE solver, the LTM consists of boundary or initial conditions to be satisfied by the solution in functional form (Section IV-B). In the system identification application, the LTM consists of system dynamics at chosen equilibria points (Section IV-C). The results illustrate the CPROP approach is widely applicable and very effective at preserving accurate LTMs (Sections IV). Also, it is found that an added benefit of preserving the LTM by means of CPROP is that it can lead to significant computational savings in training the NN, because the prior knowledge provides a performance baseline that aids in the learning of new STMs, as is commonly observed in biological neural systems.

II. Background

A. Feedforward Neural Networks Algebraic Training

Suppose a feedforward neural network (NN) is used to approximate a multi-dimensional function $h : \mathbb{R}^q \to \mathbb{R}^n$ on a compact set. Let the adjustable parameters of the NN be $\mathbf{W}$, $\mathbf{d}$, and $\mathbf{V}$, and assume the output bias is set to zero, for simplicity. Assume h is to be approximated using a training set of input-output samples $\mathcal{T} = \{\mathbf{p}^k, \mathbf{y}^k\},\ k = 1, 2, ..., r$, where $\mathbf{p} \in \mathbb{R}^q$ and $\mathbf{y} \in \mathbb{R}^n$ denote the function's input and output, respectively. The NN contains one hidden layer with s nonlinear units, and sigmoidal transfer function

$$\sigma(r) = \frac{e^r - 1}{e^r + 1} \tag{1}$$

Following the algebraic training approach presented in [15], a so-called *input-to-node matrix* is defined from $\mathcal{T}$ as $\mathbf{N} \equiv [\mathbf{WP} + \mathbf{D}]^T$, where $\mathbf{P} \equiv [\mathbf{p}^1\mathbf{p}^2 \cdots \mathbf{p}^r]$ and $\mathbf{D} \equiv [\underbrace{\mathbf{dd} \cdots \mathbf{d}}_{r}]$, such that $\mathbf{P} \in \mathbb{R}^{q\times r}$, $\mathbf{D} \in \mathbb{R}^{s\times r}$ and $\mathbf{W} \in \mathbb{R}^{s\times q}$. Then, the NN output, $\hat{\mathbf{y}}$, may be written as:

$$\hat{\mathbf{y}} = \mathbf{V}\mathbf{S}_0^T \tag{2}$$

where $\mathbf{V} \in \mathbb{R}^{n\times s}$ contains the output weights and $\mathbf{S}_0 \equiv \{\sigma(n_{ij})\}$ is a matrix of sigmoidal functions, evaluated at each element of the matrix $\mathbf{N} = \{n_{ij}\}$. The function h may be approximated from $\mathcal{T}$ using supervised training, in which the error vectors are defined as

$$\boldsymbol{\epsilon}_k = \hat{\mathbf{y}}^k - \mathbf{y}^k, \qquad k = 1, 2, ..., r \tag{3}$$

Then, the error function to be minimized during training is:

$$V(\mathbf{N}, \mathbf{V}) = \frac{1}{r}\sum_{k=1}^{r} \boldsymbol{\epsilon}_k^T \boldsymbol{\epsilon}_k \tag{4}$$

In this paper, the Levenberg-Marquardt (LM) is implemented for training due to its excellent convergence and stability properties [16], [17]. The LM algorithm iteratively minimizes V with respect to $\mathbf{w}$, based on the NN *Jacobian*, $\mathbf{J} = \partial\boldsymbol{\epsilon}^k/\partial\mathbf{w}$, which may be computed analytically via classical backpropagation [18], [19].

III. Methodology

A. Problem Formulation for Supervised Training via Constrained Backpropagation (CPROP)

In this paper, training is formulated as a constrained optimization problem that preserves prior knowledge through a set of equality constraints, avoiding the need for representing LTM training sets repeatedly to the NN. For simplicity, consider a smooth single input scalar function $h : \mathbb{R} \to \mathbb{R}$ to be approximated using a feed-forward NN. The extension to multivariate functions is illustrated in section III-D. The long-term memory (LTM) of a NN is defined as *the knowledge of the function $h : p \to y$ to be approximated by (2) that must be preserved at all times, during one or more sequential-training session*. It is assumed that the LTM may comprise sampled output and derivative information, or information about the shape of h over a bounded subset $\mathcal{S} \subset \mathbb{R}$ of its domain. Let the short-term memory (STM) be defined as *the sequence of tasks or information that must be learned through one or more training functions $\{e_l(\mathbf{w})\}_{l=1,2,...}$, but may not be consolidated into LTM.*

Assuming that the long-term memory (LTM) can be expressed by a training set of input/output samples and derivatives information, an LTM training set is defined as $\mathcal{T}_{LTM} = \{\xi^k, \zeta^k, \chi^k\}_{k=1,...,r_{LTM}}$ where $\chi^k = dh(\xi^k)/d\xi$. Then, given a trained NN that has learned $\mathcal{T}_{LTM}$ within a desirable tolerance, the network connections may be partitioned into LTM synaptic connections and STM connections, with weights $\mathbf{w}_L$ and $\mathbf{w}_S$ respectively. The hidden nodes are also partitioned into LTM and STM nodes that are associated with LTM and STM input and output connections, respectively. Typically, the STM nodes and weights, $\mathbf{w}_S$, are added after the NN has been trained with $\mathcal{T}_{LTM}$ in order to augment its approximation power, and to allow it to acquire new knowledge while preserving previous information, as explained below.

In order to preserve prior knowledge, the LTM constraints are always enforced during training, through a constrained optimization technique. The conditions used to simplify the constrained training problem are that the LTM input-to-node values and the LTM input biases are held constant, and that there are as many LTM connections as the number of equations constituting the LTM constraints. However all of the remaining LTM and STM weights are adjusted iteratively during training. Then, the LTM output weights can be expressed as a function of the STM weights (as shown in [14]):

$$\mathbf{w}_L = \mathbf{g}(\mathbf{w}_S) \tag{5}$$

Based on supervised training, let (3) be the error function used to acquire STM knowledge. Then, CPROP training may be stated as,

$$\begin{aligned} &\text{minimize } V(\mathbf{w}_L, \mathbf{w}_S) \\ &\text{subject to } \mathbf{w}_L = \mathbf{g}(\mathbf{w}_S) \end{aligned} \tag{6}$$

The main advantage of this approach is its ability to formulate the LTM constraints using algebraic equations that can be easily inverted to obtain $\mathbf{g}(\cdot)$. Subsequently, the constraints (5) are taken into account by the Jacobian operator, and classical backpropagation techniques (such as the LM algorithm) can be directly adopted to update the STM weights. While this adaptation gives enough plasticity to the network to acquire the STM data, the preservation of the memory is guaranteed through the LTM output weights.

B. Explicit Computation of the Unconstrained Jacobian

For a scalar function, involving derivatives, the Jacobian may be written as,

$$\mathbf{J}^m = [\mathbf{J}^m_{\mathbf{w}} \mid \mathbf{J}^m_{\mathbf{d}} \mid \mathbf{J}^m_{\mathbf{v}}]$$

where,

$$\mathbf{J}^m_{\mathbf{w}} = m\ \mathbf{S}^m \mathbf{W}_d^{m-1} \mathbf{V}_d + \mathbf{P}\mathbf{S}^{m+1}\mathbf{W}_d^m \mathbf{V}_d \tag{7}$$

$$\mathbf{J}^m_{\mathbf{d}} = \mathbf{S}^{m+1}\mathbf{W}_d^m \mathbf{V}_d \tag{8}$$

and

$$\mathbf{J}^m_{\mathbf{v}} = \mathbf{S}^m \mathbf{W}_d^m \tag{9}$$

Where, $\mathbf{S}^m \equiv \{\sigma^m(n_{ij})\}$, $\sigma^m(\cdot)$ denotes the m^{th} derivative of the sigmoidal function, and every derivative is evaluated at each element of the matrix $\mathbf{N}$. And, $\mathbf{J}^m$ is the Jacobian corresponding to the m^{th} derivative with respect to the weights.

C. Jacobian computation, in the presence of constraints

The *adjoined* Jacobian is inspired by the adjoined gradient presented in [14] and is derived analytically from the unconstrained Jacobian (Section III-B), using the tensorial approach. Emphasizing the network partition into LTM and STM nodes and connections, the NN output (2) is re-written as:

$$\hat{\mathbf{y}} = \mathbf{S}\mathbf{v} + \mathbf{\Sigma}\boldsymbol{\nu} \tag{10}$$

Then, provided the assumptions in Section III-A are satisfied, the *explicit memory equation* can always be written as:

$$\boldsymbol{\nu} = \mathbf{\Phi}^{-1}\left[\boldsymbol{\lambda} - \mathbf{\Psi}\mathbf{v}\right] \tag{11}$$

Where, throughout the paper, Greek letters are used to denote quantities associated with the LTM, and Latin letters are used to denote quantities associated with the STM connections, with the exception of $\mathbf{w}_L$ and $\mathbf{w}_S$.

D. Extension to vectorial functions

Suppose the NN approximates a vector function $\mathbf{f}$: $\mathbb{R}^q \rightarrow \mathbb{R}^n$ and contains s_{LTM} LTM nodes. Then, the NN output equivalent to eq. (2) is $\hat{\mathbf{Y}} \in \mathbb{R}^{n\times s_{LTM}}$, and may be expressed as:

$$\hat{\mathbf{Y}} = [\mathbf{B} - \mathbf{V}\mathbf{\Psi}^T]\mathbf{\Phi}^{-T} \tag{12}$$

Where $\mathbf{B}$ is a matrix containing LTM output information and, possibly, derivative information obtained from $\mathcal{T}_{LTM}$:

$$\mathbf{B} = \left[\begin{array}{ccccc} \mathbf{0}_{n\times r_{LTM}} & \frac{\partial \mathbf{f}}{\partial x_1} & \frac{\partial \mathbf{f}}{\partial x_2} & \cdots & \frac{\partial \mathbf{f}}{\partial x_q}\end{array}\right] \tag{13}$$

x_i denotes the i^{th} input of the vector function for which derivatives are known. The first block of the matrix is set to zero, provided that the output of the NN has to be the the zero vector, when fed with the $\mathcal{T}_{LTM}$, since we assume that we equilibria points form LTM memory information.

The Jacobian is again decomposed into an unconstrained and a constrained contribution. For convenience, the error vector and Jacobian matrix are re-defined as,

$$\mathbf{J} \equiv \left[\frac{\partial e_1^1}{\partial \mathbf{w}}\frac{\partial e_1^2}{\partial \mathbf{w}}\cdots\frac{\partial e_1^{r_{STM}}}{\partial \mathbf{w}} \quad \frac{\partial e_2^1}{\partial \mathbf{w}}\frac{\partial e_2^2}{\partial \mathbf{w}}\cdots\frac{\partial e_2^{r_{STM}}}{\partial \mathbf{w}}\right]^{\mathbf{T}} \tag{14}$$

where, r_{STM} is the number of STM training pairs, such that the methodology in Section III still holds and for simplicity and without the loss of generality (14) is specialized in case of $n = q = 2$. In this case, the Jacobian matrices defined with respect to the individual weights are:

$$\begin{aligned}
\mathbf{J_w} &= \left[\begin{array}{cc} \mathbf{P_1}\mathbf{S}'(\mathbf{p})\mathbf{V_{d1}} & \mathbf{P_2}\mathbf{S}'(\mathbf{p})\mathbf{V_{d1}} \\ \mathbf{P_1}\mathbf{S}'(\mathbf{p})\mathbf{V_{d2}} & \mathbf{P_2}\mathbf{S}'(\mathbf{p})\mathbf{V_{d2}} \end{array}\right] \\
\mathbf{J_d} &= \left[\begin{array}{c} \mathbf{S}'(\mathbf{p})\mathbf{V_{d1}} \\ \mathbf{P_1}\mathbf{S}'(\mathbf{p})\mathbf{V_{d2}} \end{array}\right] \\
\mathbf{J_v} &= \left[\begin{array}{cc} \mathbf{S}(\mathbf{p}) & \mathbf{0} \\ \mathbf{0} & \mathbf{S}(\mathbf{p}) \end{array}\right]
\end{aligned} \tag{15}$$

IV. CPROP Applications and results

A. Function approximation

The first application considers the approximation of a smooth scalar function for which a closed-form analytic representation is unavailable. Suppose the function has to be learned sequentially by the NN using sampled information. First, the NN must be trained using LTM training sets, $\mathcal{T}^1_{LTM} = \{\xi^j, \zeta^j\}_{j=1,\ldots,r_{LTM}}$ and $\mathcal{T}^2_{LTM} = \{\upsilon^l, \chi^l\}_{l=1,\ldots,p_{LTM}}$, which contain input/output and derivative samples, respectively, and then it must be re-trained using an STM training set, $\mathcal{T}_{STM} = \{p^k, y^k\}_{k=1,\ldots,r_{STM}}$. This situation may come about if $\mathcal{T}_{STM}$ becomes available at a later time, and the NN must be updated accordingly, or if the sets are too large to be learned in batch mode. Then, the neural network may forget the LTM while learning the STM data due to interference.

The CPROP approach can be used to suppress interference by writing eq. (2) as,

$$\hat{\mathbf{y}} = \mathbf{S}(\mathbf{p})\mathbf{v} + \mathbf{\Sigma}(\mathbf{p})\boldsymbol{\nu} \tag{16}$$

where $\Sigma(\cdot)$ is the sigmoidal matrix of the LTM nodes, $\mathbf{v}$ are the STM output weights, and $\boldsymbol{\nu}$ are the LTM output weights. The LTM constraints are derived from $\mathcal{T}^1_{LTM}$ and $\mathcal{T}^2_{LTM}$ using the algebraic training approach (Section III-A),

$$\mathbf{S}(\boldsymbol{\xi})\mathbf{v} + \mathbf{\Sigma}(\boldsymbol{\xi})\boldsymbol{\nu} = \boldsymbol{\zeta}$$

$$\mathbf{S}'(\boldsymbol{\xi})\mathbf{W_d}\mathbf{v} + \mathbf{\Sigma}'(\boldsymbol{\xi})\mathbf{\Omega_d}\boldsymbol{\nu} = \boldsymbol{\chi}$$

where,

$$\left[\begin{array}{c} \mathbf{S}(\boldsymbol{\xi}) \\ \mathbf{S}'(\boldsymbol{\xi})\mathbf{W_d} \end{array}\right]\mathbf{v} + \left[\begin{array}{c} \mathbf{\Sigma}(\boldsymbol{\xi}) \\ \mathbf{\Sigma}'(\boldsymbol{\xi})\mathbf{\Omega_d} \end{array}\right]\boldsymbol{\nu} = \left[\begin{array}{c} \boldsymbol{\zeta} \\ \boldsymbol{\chi} \end{array}\right]$$

and $\boldsymbol{\xi}$ is a vector with elements $\{\xi^1, \ldots, \xi^{r_{LTM}}\}$. Now, let,

$$\boldsymbol{\Psi} \equiv \begin{bmatrix} \mathbf{S}(\boldsymbol{\xi}) \\ \mathbf{S}'(\boldsymbol{\xi})\mathbf{W_d} \end{bmatrix} \quad \text{and} \quad \boldsymbol{\Phi} \equiv \begin{bmatrix} \boldsymbol{\Sigma}(\boldsymbol{\xi}) \\ \boldsymbol{\Sigma}'(\boldsymbol{\xi})\boldsymbol{\Omega}_\mathbf{d} \end{bmatrix} \tag{17}$$

such that eq. (16) can be written as,

$$\boldsymbol{\Psi}\ \mathbf{v} + \boldsymbol{\Phi}\ \boldsymbol{\nu} = \boldsymbol{\lambda} \tag{18}$$

with $\boldsymbol{\lambda} \equiv [\boldsymbol{\zeta}^T \quad \boldsymbol{\chi}^T]^T$. In order to invert (18), $\boldsymbol{\Phi}$ must be a square non-singular matrix, which can be accomplished by choosing the number of LTM connections to be,

$$s_{LTM} = r_{LTM} + p_{LTM}$$

such that $\boldsymbol{\Phi} \in \mathbb{R}^{s_{LTM} \times s_{LTM}}$, and by verifying that its determinant is non-zero (otherwise, $\boldsymbol{\Phi}$ is re-designed).

Equation (11) constitutes the *memory equation*, which, when substituted into (16) leads to:

$$\mathbf{y} = \mathbf{S}(\mathbf{p})\mathbf{v} + \boldsymbol{\Sigma}(\mathbf{p})\boldsymbol{\Phi}^{-1}\left[\boldsymbol{\lambda} - \boldsymbol{\Psi}\mathbf{v}\right] \tag{19}$$

Letting $\mathbf{e} \equiv \mathbf{y} - \mathbf{t}$ denote the STM error, with constant output target $\mathbf{t}$ formed from the set $\{y^1, \ldots, y^{r_{STM}}\}$. From eq. (19) and the STM training set $\mathcal{T}_{STM}$ it follows that the Jacobian needed to acquire STM subject to the LTM constraint eq. (11) is,

$$\mathbf{J} = \frac{\partial \mathbf{e}}{\partial \mathbf{w}} = \frac{\partial \mathbf{y}}{\partial \mathbf{w}} = \frac{\partial}{\partial \mathbf{w}}\left[\mathbf{S}(\mathbf{p})\mathbf{v} + \boldsymbol{\Sigma}(\mathbf{p})\boldsymbol{\Phi}^{-1}(\boldsymbol{\lambda} - \boldsymbol{\Psi}\mathbf{v})\right] \tag{20}$$

by the chain rule. The first term is the unconstrained Jacobian, $\mathbf{J}_{unc}$, derived in Section III-B. Thus, the final Jacobian is given by the contribution of $\mathbf{J}_{unc}$ and a memory term, $\mathbf{J}_{mem}$,

$$\mathbf{J} = \mathbf{J}_{unc} + \mathbf{J}_{mem} = \mathbf{J}_{unc} + \boldsymbol{\Pi}\frac{\partial}{\partial \mathbf{w}}(\boldsymbol{\Psi}\mathbf{v}) \tag{21}$$

where,

$$\boldsymbol{\Pi} \equiv -\boldsymbol{\Sigma}(\mathbf{p})\boldsymbol{\Phi}^{-1} \tag{22}$$

since $\boldsymbol{\lambda}$ and $\boldsymbol{\Pi}$ do not depend on $\mathbf{w}$, and,

$$\frac{\partial}{\partial \mathbf{w}}(\boldsymbol{\Psi}\mathbf{v}) = [\hat{\mathbf{J}}_\mathbf{w} \mid \hat{\mathbf{J}}_\mathbf{d} \mid \hat{\mathbf{J}}_\mathbf{v}] \tag{23}$$

where:

$$\begin{aligned} \hat{\mathbf{J}}_\mathbf{w} &= \begin{bmatrix} \mathbf{P_{LTM}}\mathbf{S}'(\boldsymbol{\xi})\mathbf{V_d} \\ \mathbf{S}'(\boldsymbol{\xi})\mathbf{V_d} + \mathbf{P_{LTM}}\mathbf{S}''(\boldsymbol{\xi})\mathbf{W_d}\mathbf{V_d} \end{bmatrix} \\ \hat{\mathbf{J}}_\mathbf{d} &= \begin{bmatrix} \mathbf{S}'(\boldsymbol{\xi})\mathbf{V_d} \\ \mathbf{S}''(\boldsymbol{\xi})\mathbf{W_d}\mathbf{V_d} \end{bmatrix} \\ \hat{\mathbf{J}}_\mathbf{v} &= \begin{bmatrix} \mathbf{S}(\boldsymbol{\xi}) \\ \mathbf{S}'(\boldsymbol{\xi})\mathbf{W_d} \end{bmatrix} \end{aligned} \tag{24}$$

As an example, consider the nonlinear function plotted by a dashed line in Fig. 1 over a domain $\mathcal{D} = [0,\ 3\pi] \subset \mathbb{R}$. Suppose the shape of the function over a bounded subset $\mathcal{S} = [0,\ \pi] \subset \mathcal{D}$ is known a priori to be a sine function, and $\mathcal{T}_{LTM}$ is formed using the LTM samples shown in Fig. 1. A sigmoidal NN with 15 hidden nodes is trained to approximate $\mathcal{T}_{LTM}$ using LM [20], and later, when 18 new STM samples become available (Fig. 1), the NN is re-trained by the same LM algorithm using $\mathcal{T}_{STM}$. If training is conducted sequentially, without re-using the LTM data, the NN starts out with proper LTM (dotted line in Fig. 1), but then experiences catastrophic interference and, although it learns STM well, it forgets the LTM entirely in the process (dashed-dotted line in Fig. 1). Figure 1 also shows the performance of a NN that is trained with the incremental training method proposed by Mandziuk [6]. That is, LTM weights ($\mathbf{w}_L$) are held constant while learning from $\mathcal{T}_{STM}$. As shown by the dashed line in Fig. 1, when the NN is trained by this method, it still experiences interference and forgets the LTM. Instead, when the LM algorithm is constrained by implementing the above Jacobian and memory constraint, the LTM is preserved at every epoch, and the NN learns the STM without forgetting the LTM, as shown by Fig. 2.

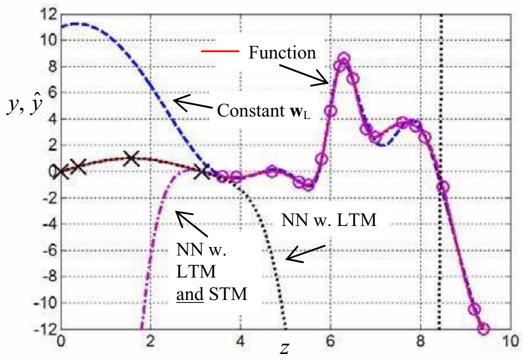

Fig. 1. Existing algorithms, including [6] which holds $\mathbf{w}_L$ constant to preserve LTM, all experience interference when the neural network is trained sequentially with LTM and, then, STM samples.

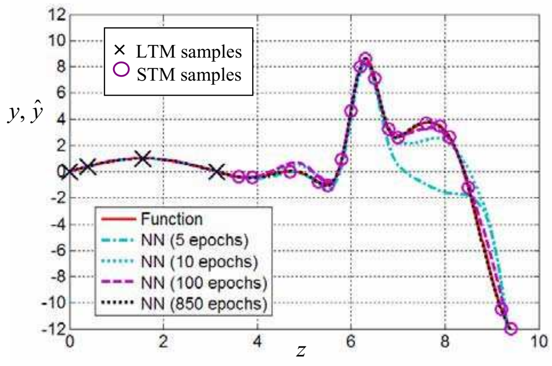

Fig. 2. CPROP preserves LTM accurately at every epoch, until it properly learns the STM at approximately 850 epochs.

B. Solution of Ordinary Differential Equations

An important application of artificial NNs is the solution of differential equations. Although there exist many numerical methods for solving differential equations (e.g., finite elements, Runge-Kutta, and B-splines), NN solutions present several advantages. For example, NNs provide closed-form, differentiable solutions that are amenable to analysis and control, and can be combined with sampled data by sequential training. Thus, NN solutions are very promising for solving data-assimilation problems in which prior knowledge is available from physical models comprised of differential equations, and additional sampled data becomes available incrementally over time from actual measurements.

A promising approach to solving initial and boundary value problems using NNs has been proposed by Lagaris in [21]. Using the collocation method, Lagaris showed that the differential equation problem can be written as a system of algebraic equations that are subject to a set of constraints imposed by the boundary conditions (BCs). Then, by defining a training error function $\mathbf{e}(\mathbf{w})$ as the squared sum of the algebraic systems' errors, $\mathbf{w}$ can be determined by minimizing $\mathbf{e}(\mathbf{w})$ using classical backpropagation. In order to address the presence of the BCs constraints, Lagaris expressed the differential equation solution as the sum of an analytical function (with no adjustable parameters) and a NN, and used the analytical function to satisfy the BCs constraints. The drawbacks of this approach are that the NN only approximates an additive component of the solution, and the analytical function must be custom-designed for every given BC. Also, the resulting solutions could not be easily adapted with new sampled data or new BCs due to the interference phenomenon and to the presence of the fixed analytic function, respectively.

These and other limitations of the method presented in [21] can are overcome by the CPROP approach. Using CPROP, the differential equation solution can be expressed solely by a NN, without requiring the use of any fixed analytic functions. The same training function $\mathbf{e}(\mathbf{w})$ proposed in [21] is minimized subject to the BCs constraints, which in this case constitute the memory equation in (5). This CPROP methodology is illustrated here by solving an initial-value problem (IVPs) involving a k^{th}-order ODE,

$$\sum_{k=1}^{K} \alpha_k \frac{d^k y}{dz^k} = \mathcal{L}(y) = g(z) \tag{25}$$

where $\mathcal{L}$ is a linear operator, and the following initial conditions are given:

$$\left.\frac{d^j y}{dz^j}\right|_{z=z_j} = h_j, \qquad j = 1, 2, ..., K-1 \tag{26}$$

From Cauchy's theorem on uniqueness of solution, there exists only one function satisfying an IVP such as (25) and (26). Thus, suppose the NN solution (output) is denoted by $\hat{y}$, and the exact (unknown) analytic solution is denoted by y. The initial conditions (26) constitute the NN LTM and the STM is obtained by sampling eq. (25). The ODE is sampled by specifying a grid on the domain $z \in \mathcal{I} \subset \mathbb{R}$, containing r_{STM} values of the independent variable z. Then, the following STM training set can be generated from eq. (25), for every value of z,

$$\mathcal{T}_{STM} = \{z^l, g(z^l)\}_{l=1,2,...,r_{STM}} \tag{27}$$

where, z^l denotes the l^{th} value of z on the grid. For every sample l, the training function can be evaluated as,

$$e^l(\mathbf{w}) \equiv \mathcal{L}(\hat{y}^l) - g(z^l), \qquad l = 1, 2, ..., r_{STM} \tag{28}$$

where $\hat{y}^j$ is the NN solution (output) given input z^l. The elements $\{e^1, \ldots, e^{r_{STM}}\}$ and $\{\hat{y}^1, \ldots, \hat{y}^{r_{STM}}\}$ are organized into vectors $\mathbf{e}$ and $\hat{\mathbf{y}}$, respectively. Then, by applying the linear differential operator to eq. (16) it follows that,

$$\mathcal{L}(\hat{\mathbf{y}}) = \mathcal{L}[\mathbf{S}(\mathbf{p})]\mathbf{v} + \mathcal{L}[\mathbf{\Sigma}(\mathbf{p})]\boldsymbol{\nu} \tag{29}$$

which can be used to derive the Jacobian, and operators $\mathbf{\Phi}$, $\mathbf{\Psi}$ and $\mathbf{\Pi}$ for the IVP eqs. (25)-(26).

Consider the following illustrative IVP example,

$$\begin{cases} \frac{dy}{dz} + \frac{1}{5}y &= e^{-\frac{z}{5}}\cos z \\ y(0) &= 0 \end{cases} \tag{30}$$

for which a NN approximation of the solution $y(z)$ is sought on the interval $\mathcal{I} = [0,\ 2]$. The *memory equation* takes the form,

$$\mathbf{S}(0)\mathbf{v} + \mathbf{\Sigma}(0)\boldsymbol{\nu} \ = 0 \tag{31}$$

and, thus, it follows that $\mathbf{S}(0) \equiv \mathbf{\Phi}, \mathbf{\Sigma}(0) \equiv \mathbf{\Psi}$ and $\lambda \equiv 0$. The terms $\mathcal{L}[\mathbf{S}(\mathbf{p})]$ and $\mathcal{L}[\mathbf{\Sigma}(\mathbf{p})]$ must be derived in order to compute the training function, eq. (28). For the differential operator defined by eq. (30),

$$\mathcal{L}[\mathbf{S}(\mathbf{p})] = \mathbf{S}'(\mathbf{p})\mathbf{W_d} + \frac{1}{5}\mathbf{S}(\mathbf{p}) \tag{32}$$

and,

$$\mathcal{L}[\mathbf{\Sigma}(\mathbf{p})] = \mathbf{\Sigma}'(\mathbf{p})\mathbf{\Omega_d} + \frac{1}{5}\mathbf{\Sigma}(\mathbf{p}) \tag{33}$$

Thus, the operator $\mathbf{\Pi}$ is given be the expression,

$$\mathbf{\Pi} = -[\mathbf{\Sigma}'(\mathbf{p})\mathbf{\Omega_d} + \frac{\mathbf{1}}{\mathbf{5}}\mathbf{\Sigma}(\mathbf{p})]\mathbf{\Phi^{-1}} \tag{34}$$

which is obtained from the Jacobian of eq. (28):

$$\mathbf{J} = \frac{\partial \mathbf{e}}{\partial \mathbf{w}} = \frac{\partial \mathcal{L}[\hat{\mathbf{y}}] - g(\mathbf{z})}{\partial \mathbf{w}} = \frac{\partial \mathcal{L}[\hat{\mathbf{y}}]}{\partial \mathbf{w}} \tag{35}$$

Making use of equations (32) and (33), the unconstrained Jacobian contribution is,

$$\begin{aligned} \mathbf{J}_{unc} &= \frac{\partial}{\partial \mathbf{w}}[\mathbf{S}'(\mathbf{p})\mathbf{W_d} + \tfrac{1}{5}\mathbf{S}(\mathbf{p})] \\ &= [\mathbf{J_w} \mid \mathbf{J_d} \mid \mathbf{J_v}] \end{aligned} \tag{36}$$

where:

$$\begin{aligned} &\mathbf{J_w} = [(\mathbf{I} + \tfrac{1}{5}\mathbf{P})\mathbf{S}'(\mathbf{p}) + \mathbf{P}\mathbf{S}''(\mathbf{p})\mathbf{W_d}]\mathbf{V_d} \\ &\mathbf{J_d} = [\tfrac{1}{5}\mathbf{P}\mathbf{S}'(\mathbf{p}) + \mathbf{S}''(\mathbf{p})\mathbf{W_d}]\mathbf{V_d} \\ &\mathbf{J_v} = \tfrac{1}{5}\mathbf{S}(\mathbf{p}) + \mathbf{S}'(\mathbf{p})\mathbf{W_d} \end{aligned} \tag{37}$$

And, finally, the memory term is derived from eqs. (28)-(34),

$$\mathbf{J}_{mem} = \mathbf{\Pi}\ [\mathbf{P_{LTM}}\mathbf{S}'(\xi)\mathbf{V_d} \mid \mathbf{S}'(\xi)\mathbf{V_d} \mid \mathbf{S}(\xi)] \tag{38}$$

where, ξ is obtained from the initial conditions. As before, $\mathbf{J} = \mathbf{J}_{unc} + \mathbf{J}_{mem}$ can be used in the CPROP algorithm to learn STM, while preserving the LTM.

By implementing the above Jacobian in the LM algorithm, a 5-hidden-node NN trained via CPROP is found to perform as well as the 10-hidden-node NN (plus the analytical function) trained in [21] to solve the same IVP (25)-(26). As shown by the dashed line in Fig. 3, the NN matches the analytical solution of (25)-(26) exactly, preserving the LTM (cross) and extrapolating well outside the STM grid (circles). Figure 3 also shows that an otherwise equivalent but unconstrained NN performs poorly everywhere in $\mathcal{I}$.

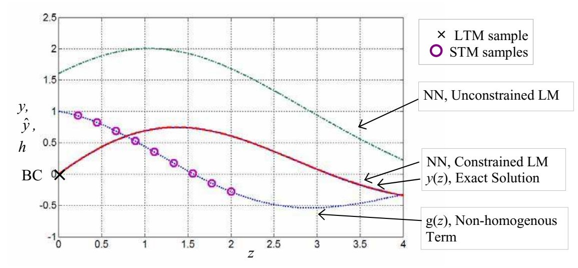

Fig. 3. The CPROP NN solution of the linear ODE IVP (25)-(26) identically overlaps the analytical solution, $y(z)$, given STM samples of the non-homogenous term, and LTM of the BC.

C. System Identification (ID)

In many applications, *a priori* knowledge of local dynamics may be available over one or more partitions of the phase space. Often, this knowledge may be complemented and improved by time series data that becomes available from the operating system in real time. However, due to catastrophic interference, a NN model that is trained sequentially with new global models, may forget prior local models and perform poorly in certain regions of its phase space. As an illustrative example, consider the nonlinear dynamics of a pendulum with friction,

$$\begin{cases} \dot{\theta} &= \omega \\ \dot{\omega} &= -a\sin\theta - b\ \omega \end{cases} \qquad a, b > 0 \tag{39}$$

where θ is the angular displacement of the pendulum, and ω is its angular velocity.

A 15-node sigmoidal NN is used to model (39) over the phase space $\Omega : [-8,\ 8] \times [-8,\ 8]$ by letting the NN output approximate the state-rate vector $\hat{\mathbf{y}} = [\dot{\theta}\ \dot{\omega}]^T$, given the input $\mathbf{p} = [\theta\ \omega]^T$. As shown by its phase portrait (solid line in Fig. 5), the pendulum has five equilibria in Ω: three stable foci (Q_1, Q_3, Q_5), and two saddles (Q_2, Q_4), representing multiple sways. Near the equilibria, the local dynamics can be approximated by five local models obtained by linearizing eqs. (39). In order to simulate on-line system ID, a NN is trained using an LTM training set $\mathcal{T}_{LTM}$ (plotted by crosses in Fig. 5) that is obtained from these local models. Subsequently, the same NN is re-trained sequentially with time-series STM data $\mathcal{T}_{STM}$ (plotted by circles in Fig. 5) that is obtained away from the equilibria in Ω . This can be viewed as a hybrid approach that integrates global models (time-series learning) with the local models (linearizations) developed by Principe et al. [22].

As shown in Fig. 4, when the NN is trained using unconstrained LM, it experiences catastrophic interference

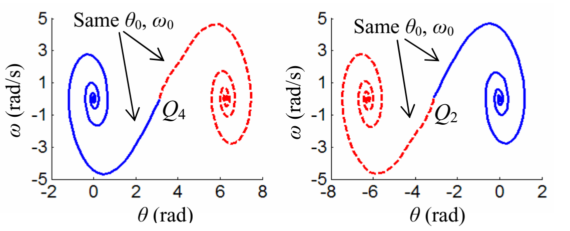

Fig. 4. The NN trained with an unconstrained LM algorithm (dashed line) exhibits catastrophic interference and completely forgets the LTM local models of the pendulum (solid line) that were initially used to train it near its equilibria.

near the unstable equilibria Q_2 and Q_4 (Figs. 4), which means prior knowledge is completely erased from the NN due to further training with new data. In fact, as shown by the dashed line in Fig. 4), the NN trajectory for the initial condition (θ_0, ω_0) = ($-\pi$ + 1/100 rad, 0 rad/s) is completely different from that of the actual pendulum (solid line), and is physically impossible. When the same NN is trained using CPROP, it learns the STM very well, displaying an STM mean-square-error (MSE) of $O(10^{-3})$. At the same time, CPROP preserves its LTM virtually intact, displaying an LTM-MSE of $O(10^{-11})$. As a result, the NN performs excellent system ID everywhere in Ω, as demonstrated by the overlapping phase portraits of the CPROP NN (dashed line) and the actual pendulum (solid line) that are both plotted in Figs. 5-6.

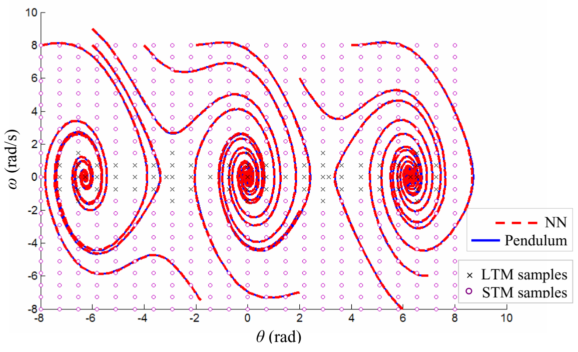

Fig. 5. The phase portraits of the pendulum and of the NN trained via CPROP identically overlap everywhere in Ω.

Moreover, if we represented also the equilibria points, during subsequent training, for the unconstrained case, we had to use a fitter grid, in order to avoid catastrophic interference. Besides, starting from the same initial values of the weights, the unconstrained training converged, with reasonable precision ($O(10^{-3})$ after 802 iterations, in 188.81 secs, while CPROP converged after 300 iterations, in 40.27 secs. It is worth observing that, since CPROP is dealing with a less dens grid, we need to invert smaller matrices, feature which is crucial, for bigger multi-dimensional problems; in fact the time-per-iteration required is .1342 secs, whereas a

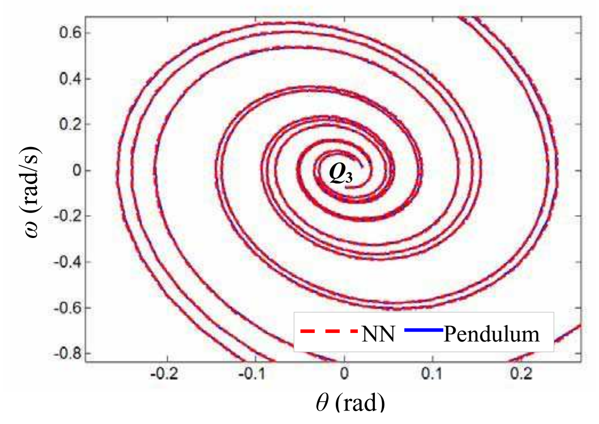

Fig. 6. The accuracy of the LTM preserved by CPROP is illustrated by zooming in the phase portraits about Q_3 illustrating that the NN and pendulum phase portraits overlap with very high precision.

classical unconstrained approach requires .2354 secs.

V. Conclusions

A constrained-backpropagation training technique is presented to suppress interference and preserve prior knowledge in sigmoidal neural networks, while new information is learned incrementally. The technique is based on constrained optimization, and minimizes an error function subject to a set of equality constraints derived via an algebraic training approach. The numerical results show excellent generalization and extrapolation properties, and demonstrate the wide range of neural network problems that can be treated by this novel technique. The generality of the problem formulation makes the method flexible and promising both for prediction and data assimilation. Since the memory equations are derived analytically, it is possible to represent the long term memory exactly, up to machine precision. This feature is crucial to neural network control and system identification applications, especially in the presence of highly nonlinear and possibly unstable dynamics, as in the pendulum example where a classic neural network is shown to experience catastrophic interference. A possible future work is to use this approach to solve PDE and non-linear PDE, and to challenge the method through the assimilation of real data, possibly noisy.

VI. Acknowledgments

This work was supported by the National Science Foundation, under grant ECS CAREER 0300236

References

[1] M. McCloskey and N. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," *The Psychology of Learning and Motivation*, vol. 24, no. 109-164, 1989.

[2] T. Brashers-Krug, R. Shadmehr, and E. Todorov, "Catastrophic interference in human motor learning," *Advances in Neural Information Processing*, 1995.

[3] Ratcliff, "Connectionist models of recognition memory: Constraints imposed by learning and forgetting functions," *Psychological Review*, pp. 285–308, 1990.

[4] K. Yamauchi, N. Yamaguchi, and N. Ishii, "Incremental learning methods with retrieving of interfered patterns," *IEEE Trans. Neural Networks*, vol. 10, no. 6, 1999.

[5] M. Kobayashi, A. Zamani, S. Ozawa, and S. Abe, "Reducing computations in incremental learning for feedforward neural network with longterm memory," in *Proc. Int. Joint Conf. Neural Networks*, 1989.

[6] J. Mándziuk and L. Shastri, "Incremental class learning - an approach to longlife and scalable learning," in *Proc. Int. Joint Conf. Neural Networks*, 1999.

[7] M. Kotani, K. Akazawa, S. Ozawa, and H. Matsumoto, "Detection of leakage sound by using modular neural networks," in *Proc. Sixteenth Congress of the Int. Measurement Confederation*, 2000.

[8] R. Lamprecht and J. LeDoux, "Structural plasticity and memory," *Nature Reviews - Neuroscience*, vol. 5, 2004.

[9] T. Kohonen, *Self-Organization and Associative Memory*. Berlin, Germany: Springer-Verlag, 1987.

[10] I. Kanter and H. Sompolinsky, "Associative recall of memory without errors," *Physical Review A*, vol. 35, no. 1, 1987.

[11] A. Zhao, "Global stability of bidirectional associative memory neural networks with distributed delays source," *Physics Letters*, vol. 297, no. 3-4, 2002.

[12] G. A. Carpenter, "Neural network models for pattern recognition and associative memory," *Neural Networks*, vol. 2, no. 4, 1989.

[13] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proc. of the National Academy of Sciences USA*, vol. 79, 1982.

[14] S. Ferrari and M. Jensenius, "A constrained optimization approach to preserving prior knowledge during incremental training," *IEEE Trans. On Neural Networks*, to appear April 2008, available upon request.

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

[16] D. W. Marquardt, "An algorithm for least-squares estimation of non-linear parameters," *Journal of the Society of Industrial and Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963.

[17] K. Levenberg, "A method for the solution of certain non-linear problems in least squares," *Quarterly Journal of Applied Mathmatics*, vol. II, no. 2, pp. 164–168, 1944.

[18] Y. LeCun, "A theoretical framework for backpropagation," in *Proceedings of the 1988 Connectionist Models Summer School*, D. Touretzky, G. Hinton, and T. Sejnowski, Eds. CMU, Pittsburgh, Pa: Morgan Kaufmann, 1988, pp. 21–28.

[19] P. J. Werbos, *The roots of backpropagation: from ordered derivatives to neural networks and political forecasting*. New York, NY, USA: Wiley-Interscience, 1994.

[20] *MATLAB Neural Network Toolbox, User's Guide*. The MathWorks, 2005.

[21] 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.

[22] J. Principe, L. Wang, and M. Motter, "Local dynamic modeling with self-organizing maps and applications to nonlinear system identification and control," *Proc. of the IEEE*, vol. 86, no. 11, 1998.