A Case Of Adaptive Nonlinear System Identification With Third Order Tensors In TensorFlow

Georgios Drakopoulos
Department of Informatics
Ionian University
Kerkyra, Hellas
c16drak@ionio.gr

Phivos Mylonas
Department of Informatics
Ionian University
Kerkyra, Hellas
fmylonas@ionio.gr

Spyros Sioutas
CEID
University of Patras
Patra, Hellas
sioutas@ceid.upatras.gr

***Abstract*—Non-linear system identification is a challenging problem with a plethora of engineering applications including digital telecommunications, adaptive control of biological systems, assessing integrity of mechanical constructs, and geological surveys. Various approaches have been proposed in the scientific literature, including Volterra and multivariate Taylor series, fuzzy neural networks, state space models, and wavelets. This conference paper proposes a succinct model of a non-linear system with memory based on a third order tensor whose coefficients are trained in an LMS-like way. Moreover, two variants deriving from sign LMS and batch LMS algorithms respectively are also implemented in TensorFlow. The results of applying the three training algorithms to this system are compared in terms of the mean square error in validation phase, the convergence rate of the coefficients, and the convergence rate of the Euclidean norm of the local gradients of the system model.**

***Index Terms*—tensor algebra, higher order data, non-linear system identification, adaptive system identification, local gradient estimation, Volterra series, LMS training, sign LMS, batch LMS, TensorFlow**

I. Introduction

Non-linear system modeling plays a central role in a broad spectrum of engineering applications including but by no means limited to MIMO ground communication networks, satellite communications and underwater acoustic channels with non-linear channels, control of robotic arms, geological surveys for water, oil, or rare earths, and adaptive control of biological and medical systems. Because of both the plethora of applications and the challenging background involved in the analysis of non-linear systems, a number of ingenious approaches drawing inspiration from various branches of mathematics, physics, or other fields have been proposed in the scientific literature.

The primary research objective of this conference paper is twofold. First, a tensor model for a non-linear system is developed using a third order tensor, displaying the potential of tensor algebra. The coefficients of this model are trained using an LMS-like training process. Second, two alternatives of the training process based on the sign LMS and the batch LMS are also developed. All three algorithms were tested on a non-linear system with memory whose structure is similar to that of many systems found in applications such as satellite or underwater communications.

The remaining of this work is structured as follows. Section II briefly review current scientifc literature regarding tensor algebra and system identification theory. Certain tensor operations which will be the building blocks for the system modeling are described in III, where the model itself is devloped in IV. Section V containes some notes on the TensorFlow implementation and the results obtained by of the three algorithms. Future research directions are mentioned in section VI. Table I summarizes the notation of this conference paper. Finally, concerning notation tensors are represented with capital calligraphic letters, matrices with capital boldface, vectors with small boldface, and scalars with small letters.

TABLE I
Notation of this conference paper

Symbol	Meaning
$\triangleq$	Definition or equality by definition
$\{s_1, \ldots, s_n\}$	Set with elements $s_1, \ldots, s_n$
$(i_1, \ldots, i_p)$	Tuple with elements $i_1, \ldots, i_p$
$\langle t_k \rangle$	A sequence with elements t_k
$\lvert S \rvert$	Set, sequence, or tuple cardinality
$\times_k$	Tensor multiplication along the k-th dimension
$\lVert \mathcal{T} \rVert_F$	Frobenius norm of tensor $\mathcal{T}$
$\mathcal{T}^{(k;d)}$	k-th slice of tensor $\mathcal{T}$ along dimension d
$\mathbf{I}_n$	$n \times n$ identity matrix
$\mathbf{0}_{m \times n}$	$m \times n$ zero matrix
$\otimes$	Tensor Kronecker product
$\circ$	Tensor outer product
$\star$	Discrete linear convolution

II. Previous Work

System identification can take a number of forms depending on the context. State space models, namely white box models, for non-linear systems are explored in [1], whereas black box modeling in extensively treated in [2]. Concerning the modeling of non-linear systems with models other than state space ones, various approaches have been proposed in the literature including Volterra series with coefficients derived from an adaptive training process [3], fuzzy neural networks trained with a variation of the BEP algorithm [4] or with a modified fuzzy BEP algorithm [5], and wavelets [6]. The

978-1-7281-1862-8/19/$31.00 ©2019 IEEE

relationship between the Volterra and Wiener series is explored in [7]. Wiener and Hammerstein cascade models for non-linear biological systems are examined in [8].

Tensor algebra and well as its main properties, including the fundamental Tucker and Kruskal decompositions, have been introduced among others in [9] and in [10]. Moreover, the role of tensor factorizations in discovering structure inherently found in data is explored in [11]. Tensor applications are numerous. In [12] a graph resilience metric based on path lengths and the number of triangles is proposed. In social network analysis multiple Twitter functional analytics are combined in a multilayer graph in [13], while the digital influence of an account in Twitter can be expressed in higher order terms [14]. An extension of ontologies to multiple edges, each with a different semantic aspect, is proposed in [15]. For a genetic algorithm for clustering large third order tensors containing geolocation and linguistic data with an objective function based on linguistic variation models see [16].

Multilinear signal processing, namely signal processing based on tensor algebra, has been the subject of many research papers. In [17] a concise presentation of the subject from a higher dimensional viewpoint is given along with important concepts such as base tensors, efficient base change, dimensionality reduction, and the relationship between the different variants the SVD takes for more than two dimensions. Higher order statistics play a crucial role in source identification in MIMO systems since third and fourth order statistics can discover signal structure second order statistics are oblivious to [18] [19].

TensorFlow has been originally developed by Google in order to simulate complex brain circuits. Since 2015 it is an open source, low level framework designed specifically for tensor algebra based on the dataflow paradigm [20]. The latter is explained in detail along with other implementation aspects in [21]. A versatile graph structure which allows versioning and keeping the history of changes in symbolic operations stored as strings is proposed in [22]. An efficient generator of Gaussian processes for TensorFlow is described in detail in [23], while a tool for visualizing symbolic computation trees in TensorFlow is given in [24].

III. Background

Tensor algebra is the generalization of linear algebra to more than two dimensions. Additionally, it includes linear algebra as a special case. Formally, in the most general case a tensor is defined in a straightforward manner as [9]:

Definition 1 (Tensor): A p-th order tensor $\mathcal{T} \in \mathbb{V}_1 \times \ldots \times \mathbb{V}_p \rightarrow \mathbb{S}$ represents the simultaneous linear mapping between p not necessarily distinct vector spaces $\mathbb{V}_k$ to a vector space $\mathbb{S}$, $1 \leq k \leq p$.

For the purposes of this work we will use real valued tensors defined over real domains. Therefore, in the remaining of the text tensors are either $\mathcal{T} \in \mathbb{R}^{n_1 \times \ldots \times n_p}$ or $\mathcal{T} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$.

In equation (1) the exact structure of a $2 \times 2 \times 2$ tensor $\mathcal{T}$ is shown. It consists of eight real scalars, each indexed by a distinct triplet of integers (i_1, i_2, i_3) where $i_k \in \{1, 2\}$.

$$\mathcal{T}[i_1, i_2, 1] = \begin{bmatrix} \mathcal{T}[1,1,1] & \mathcal{T}[1,2,1] \\ \mathcal{T}[2,1,1] & \mathcal{T}[2,2,1] \end{bmatrix}$$

$$\mathcal{T}[i_1, i_2, 2] = \begin{bmatrix} \mathcal{T}[1,1,2] & \mathcal{T}[1,2,2] \\ \mathcal{T}[2,1,2] & \mathcal{T}[2,2,2] \end{bmatrix} \quad (1)$$

Definition 2 (Tensor slices): Slices of a p-th order tensor $\mathcal{T}$ along dimension d is a set of n_d tensors $\left\{\mathcal{T}^{(i_d;d)}\right\}$ of order $p-1$ which are derived by fixing the d-th index and using it as a parameter, whereas the remaining indices remain free variables.

In the specific case of a $2 \times 2 \times 2$ tensor $\mathcal{T}$ there are six possible slices in total, namely two slices for each of the three dimensions. The upper and lower subequations of equation (1) are $\mathcal{T}^{(1;3)}$ and $\mathcal{T}^{(2;3)}$ respectively. The other four slices are:

$$\mathcal{T}^{(1;1)} \triangleq \begin{bmatrix} \mathcal{T}[1,1,1] & \mathcal{T}[1,1,2] \\ \mathcal{T}[1,2,1] & \mathcal{T}[1,2,2] \end{bmatrix}$$
$$\mathcal{T}^{(2;1)} \triangleq \begin{bmatrix} \mathcal{T}[2,1,1] & \mathcal{T}[2,1,2] \\ \mathcal{T}[2,1,1] & \mathcal{T}[2,1,2] \end{bmatrix}$$

$$\mathcal{T}^{(1;2)} \triangleq \begin{bmatrix} \mathcal{T}[1,1,1] & \mathcal{T}[1,1,2] \\ \mathcal{T}[2,1,1] & \mathcal{T}[2,1,2] \end{bmatrix}$$
$$\mathcal{T}^{(2;2)} \triangleq \begin{bmatrix} \mathcal{T}[1,2,1] & \mathcal{T}[1,2,2] \\ \mathcal{T}[2,2,1] & \mathcal{T}[2,2,2] \end{bmatrix} \quad (2)$$

Multiplying a p-th order tensor $\mathcal{T} \in \mathbb{R}^{n_1 \times \ldots \times n_p}$ with a vector $\mathbf{x} \in \mathbb{R}^{n_k}$ along the k-th dimension always results in a $(p-1)$-th order tensor $\mathcal{G}$ whose elements are defined as:

$$\mathcal{G}[i_1, i_2, \ldots, i_{k-1}, i_{k+1}, \ldots, i_p] \triangleq$$
$$(\mathcal{T} \times_k \mathbf{x})[i_1, i_2, \ldots, i_{k-1}, i_{k+1}, \ldots, i_p] =$$
$$\sum_{i_k=1}^{n_k} \mathcal{T}[i_1, \ldots, i_{k-1}, i_k, i_{k+1}, \ldots, i_p]\mathbf{x}[i_k] \quad (3)$$

Definition 3 (Frobenius tensor norm): The Frobenius tensor norm is defined as:

$$\|\mathcal{T}\|_F \triangleq \left(\sum_{i_1=1}^{n_1} \ldots \sum_{i_p=1}^{n_p} \mathcal{T}^2[i_1, \ldots, i_p] \right)^{\frac{1}{2}} \quad (4)$$

Frobenius norm is especially useful for evaluating the distance between a sequence of tensors $\langle \mathcal{G}_k \rangle$ generated successively by an iterative process. A common termination criterion in these cases is the comparison of the Frobenius norm with a prespecified threshold ρ_0 as follows:

$$\|\mathcal{G}_{k+1} - \mathcal{G}_k\| \leq \rho_0 \quad (5)$$

The last equation converges to a global minimum or maximum if $\langle \mathcal{G}_k \rangle$ is a Cauchy sequence, otherwise the convergence may be local.

IV. Tensor System Model

A. Formulation

Let $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$ be three independent real valued data vectors:

$$\mathbf{x} \triangleq \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \qquad \mathbf{y} \triangleq \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \qquad \mathbf{w} \triangleq \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \tag{6}$$

Then, the proposed system model can be constructed by multiplying a third order tensor with all three input vectors. Each dimension of the tensor is multiplied by one input vector:

$$\begin{aligned} q(\mathbf{x},\mathbf{y},\mathbf{w};\mathcal{T}) &\triangleq \mathcal{T} \times_1 \mathbf{x} \times_2 \mathbf{y} \times_3 \mathbf{w} \\ &= (\mathcal{T} \times_3 \mathbf{w}) \times_1 \mathbf{x} \times_2 \mathbf{y} \\ &= \mathbf{M} \times_1 \mathbf{x} \times_2 \mathbf{y} \\ &= \mathbf{y}^T \mathbf{M} \mathbf{x} \\ &= \sum_{i_1=1}^{2}\sum_{i_2=1}^{2}\sum_{i_3=1}^{2} \mathcal{T}[i_1,i_2,i_3] x_{i_1} y_{i_2} w_{i_3} \end{aligned} \tag{7}$$

In equation (7) matrix $\mathbf{M}$ can be written as:

$$\mathbf{M} = \begin{bmatrix} \mathcal{T}^{(1;2)}\mathbf{w} & \mathcal{T}^{(2;2)}\mathbf{w} \end{bmatrix} \tag{8}$$

An alternative form for equation (8) is:

$$\mathbf{M} = \begin{bmatrix} \mathcal{T}^{(1;2)} & \mathcal{T}^{(2;2)} \end{bmatrix} \begin{bmatrix} \mathbf{w} & \mathbf{0}_{2\times 1} \\ \mathbf{0}_{2\times 1} & \mathbf{w} \end{bmatrix} \tag{9}$$

The last equation can be compactly rewritten using Kronecker tensor products as:

$$\mathbf{M} = \begin{bmatrix} \mathcal{T}^{(1;2)} & \mathcal{T}^{(2;2)} \end{bmatrix} (\mathbf{I}_2 \otimes \mathbf{w}) \tag{10}$$

B. Adaptive Training

The general formula for the update of the tensor elements at the k-th step is a delta rule [25] [26]:

$$\mathcal{T}^{[k]}[i_1,i_2,i_3] = \mathcal{T}^{[k-1]}[i_1,i_2,i_3] + \Delta\mathcal{T}^{[k]}[i_1,i_2,i_3] \tag{11}$$

Notice that in equation (11) the superscripted bracket index refers to the k-th step of the training process, whereas ordinary bracket indexing refers to the elements of the tensor. The convention for representing the step index will be used throughtout the text.

The elementwise correction term at the k-th step is LMS-like [27] [28] [29]. Therefore, the correction term is:

$$\begin{aligned} \Delta\mathcal{T}^{[k+1]}[i_1,i_2,i_3] &\triangleq \mu^{[k]} e^{[k]} u^{[k]}[i_1,i_2,i_3] \\ &= \mu^{[k]} \left(d^{[k]} - s^{[k]} \right) u^{[k]}[i_1,i_2,i_3] \end{aligned} \tag{12}$$

In equation (12) the last term is:

$$u^{[k]}[i_1,i_2,i_3] \triangleq \mathbf{x}^{[k]}[i_1]\mathbf{y}^{[k]}[i_2]\mathbf{w}^{[k]}[i_3] \tag{13}$$

A vectorized form of equation (12) using the tensor outer product definition of [9] is:

$$\mathcal{T}^{[k+1]} = \mathcal{T}^{[k]} + \mu^{[k]} e^{[k]} \mathbf{x}^{[k]} \circ \mathbf{y}^{[k]} \circ \mathbf{w}^{[k]} \tag{14}$$

C. Convergence Metrics

Besides the Frobenius norm of the difference between two successive iterations as in equation (5), the Euclidean norm of the local gradients $\nabla_{\mathbf{x}} g$, $\nabla_{\mathbf{y}} g$, $\nabla_{\mathbf{w}} g$ in each iteration will be used.

The partial derivatives of q with respect to the two elements of $\mathbf{x}$ can be directly computed in terms of a tensor slice as:

$$\begin{aligned} \frac{\partial q}{\partial x_1} &= \mathcal{T}[1,1,1]y_1 w_1 + \mathcal{T}[1,1,2]y_1 w_2 \\ &+ \mathcal{T}[1,2,1]y_2 w_1 + \mathcal{T}[1,2,2]y_2 w_2 \\ &= \mathbf{y}^T \mathcal{T}^{(1;1)} \mathbf{w} \end{aligned} \tag{15}$$

Similarly, the partial derivative of q with respect to the second element x_2 has a similar closed form involving a different tensor slice:

$$\begin{aligned} \frac{\partial q}{\partial x_2} &= \mathcal{T}[2,1,1]y_1 w_1 + \mathcal{T}[2,1,2]y_1 w_2 \\ &+ \mathcal{T}[2,2,1]y_2 w_1 + \mathcal{T}[2,2,2]y_2 w_2 \\ &= \mathbf{y}^T \mathcal{T}^{(2;1)} \mathbf{w} \end{aligned} \tag{16}$$

Stacking the two derivatives to a column vector yields:

$$\nabla_{\mathbf{x}} q = \begin{bmatrix} \frac{\partial q}{\partial x_1} & \frac{\partial q}{\partial x_2} \end{bmatrix}^T = \begin{bmatrix} \mathbf{y}^T \mathcal{T}^{(1;1)} \mathbf{w} \\ \mathbf{y}^T \mathcal{T}^{(2;1)} \mathbf{w} \end{bmatrix} \tag{17}$$

The last equation can be rewritren in matrix-vector form:

$$\nabla_{\mathbf{x}} q = \begin{bmatrix} \mathbf{y}^T & \mathbf{0}_{1\times n} \\ \mathbf{0}_{1\times n} & \mathbf{y}^T \end{bmatrix} \begin{bmatrix} \mathcal{T}^{(1;1)} \\ \mathcal{T}^{(2;1)} \end{bmatrix} \mathbf{w} \tag{18}$$

Another alternative form of equation (17) using the Kronecker tensor mutliplication is equation (19).

$$\nabla_{\mathbf{x}} q = \left(\mathbf{I}_2 \otimes \mathbf{y}^T \right) \begin{bmatrix} \mathcal{T}^{(1;1)} \\ \mathcal{T}^{(2;1)} \end{bmatrix} \mathbf{w} \tag{19}$$

In a similar manner, the vector partial derivative with respect to input vector $\mathbf{y}$ has the following high level structure:

$$\nabla_{\mathbf{y}} q = \left(\mathbf{I}_2 \otimes \mathbf{x}^T \right) \begin{bmatrix} \mathcal{T}^{(1;2)} \\ \mathcal{T}^{(2;2)} \end{bmatrix} \mathbf{w} \tag{20}$$

And the vector partial derivative with respect to the last input $\mathbf{w}$ has the following high level structure:

$$\nabla_{\mathbf{w}} q = \left(\mathbf{I}_2 \otimes \mathbf{x}^T \right) \begin{bmatrix} \mathcal{T}^{(1;3)} \\ \mathcal{T}^{(2;3)} \end{bmatrix} \mathbf{y} \tag{21}$$

D. Variations

LMS is an approximation to the stochastic gradient descent. Therefore, one might only be interested in the direction of the instantaneous error $e^{[k]}$:

$$\mathcal{T}^{[k+1]} = \mathcal{T}^{[k]} + \mu^{[k]} \frac{e^{[k]}}{\left| e^{[k]} \right|} \mathbf{x}^{[k]} \circ \mathbf{y}^{[k]} \circ \mathbf{w}^{[k]} \tag{22}$$

Another variation is batch LMS which approximates the stochastic gradient algorithm in a statistically better way. As its name suggests batch LMS relies on replacing the instantaneous error with a smoothed one over a window of length L:

$$\mathcal{T}^{[k+L]} = \mathcal{T}^{[k]} + \mu^{[k+L]} \hat{e}^{[k+L]} \mathbf{x}^{[k+L]} \circ \mathbf{y}^{[k+L]} \circ \mathbf{w}^{[k+L]} \tag{23}$$

The smoothed error in equation (23) is the sample mean, implying that the underlying process is ergodic.

$$\hat{e}^{[k+L]} \triangleq \frac{1}{L}\sum_{p=0}^{L-1} e^{[k+L-p]} \tag{24}$$

Combining the above, the framework for executing the LMS training requires the following parameters.

- The type of LMS which will be used, namely one of the equations (14), (22), or (23).
- Whether the training I/O pairs appear in the same order every epoch in a random order.
- The decay rate, if any, of the learning rate $\mu^{[k]}$. Although many options have been proposed in the scientific literature, three appear to be the most popular ones and have, thus, been implemented. Specficially, these options are:
  - **Cosine decay:** In this case, the learning rate is a scaled one fourth of a cosine period, namely when the argument of the cosine ranges in $[0, \pi/2)$. In this case the formula for the learing rate is:

$$\mu^{[k]} \triangleq \alpha_0 \cos\left(\frac{\pi k}{2\beta_0}\right), \quad 0 \le k \le \beta_0 - 1 \tag{25}$$

  - **Inverse linear decay:** This option provides a smooth decay rate, especially during the later iterations. It is an easily understood learning rate decay and can be implemented easily in software without calls to libraries. The formula for the learning rate is:

$$\mu^{[k]} \triangleq \alpha_0(1 + \beta_0 k), \quad 0 \le k \le 9/\beta_0 \tag{26}$$

  - **Exponential decay:** Finally, the exponential decay has a steep descent rate during all iterations and also leaves the less room for iterations for the same value of β_0. In this case the learning rate drops as follows:

$$\mu^{[k]} \triangleq \alpha_0 e^{-\beta_0 k}, \quad 0 \le k \le 5/\beta_0 \tag{27}$$

- The number of epochs, namely how many times can the training vectors be used in total.
- The training dataset, including the training, testing, and validation sets.
- The tensor initalization scheme. Tensors can be initialized with either random or zero values.
- The termination criterion. Common options include:
  - **T1**: The Frobenius norm of the difference of two successive tensors drops below a given threshold ρ_F as follows:

$$\left\| \mathcal{T}^{[k+1]} - \mathcal{T}^{[k]} \right\|_F \le \rho_F \tag{28}$$

  - **T2**: The sum of the Euclidean norms of the local gradient vectors drop below a given threshold ρ_E as follows:

$$\|\nabla_{\mathbf{x}} q\|_2 + \|\nabla_{\mathbf{y}} q\|_2 + \|\nabla_{\mathbf{w}} q\|_2 \le \rho_E \tag{29}$$

V. Results

As a concrete example, the adaptive training methohologies of section IV are applied to the non-linear system with memory of equation (30).

$$s[n] = g[n]g[n-1] + \frac{1}{2}g[n-2] + \eta[n] \tag{30}$$

Observe that $s[n]$ can be written as the sum of a strictly non-linear system with memory with a purely linear system also with memory. Finally, $\eta[n]$ is a discrete zero mean AWGN process with controlled variance σ_η^2, which in turn controls the SNR. The true spectrum of $s[n]$ is:

$$S\big(e^{j\omega}\big) = e^{-j\omega} G\big(e^{j\omega}\big) \star G\big(e^{j\omega}\big) + \frac{1}{2} e^{-j2\omega} G\big(e^{j\omega}\big) \tag{31}$$

The system (30) was driven with a long sequence of 2048 PAM symbols, where each such symbol can take one of four values $\{\pm 1, \pm 3\}$. Note that the indexing of the input vectors refer to training iterations, while the indexing of system input $g[k]$ refers to system time. In our experiments both time scales are equal.

$$\mathbf{x}^{[k]} \triangleq \begin{bmatrix} g[k] \\ 1 \end{bmatrix} \quad \mathbf{y}^{[k]} \triangleq \begin{bmatrix} g[k-1] \\ 1 \end{bmatrix} \quad \mathbf{w}^{[k]} \triangleq \begin{bmatrix} g[k-2] \\ 1 \end{bmatrix} \tag{32}$$

A total of 256 training pairs have been selected at random and have been partitioned to the following disjoint categories:

- **Training set**: It consists of 64 training pairs and it is used to perform the actual tensor coefficient corrections.
- **Testing set**: It is made up of 64 training pairs and its role is to evaluate the mean square error (MSE) of the model at the end of each epoch. During the testing phase the model coefficients are not updated. The MSE J for a model with memory length Q and parameters $\{\vartheta_j\}$ is:

$$J(\{\vartheta_j\}) \triangleq \frac{1}{P}\sum_{k=1}^{P} \Big(\tilde{d}[k] - \tilde{s}[k]\Big)^2 \tag{33}$$

  In equation (33) $\tilde{d}[k]$ and $\tilde{s}[k]$ stand for the expected system output and the actual system response respectively given that its input during the past Q steps was the very same sequence of vectors as denoted in equation and that the system parameters were $\{\vartheta_j\}$.

$$\begin{aligned} \tilde{d}[k] &\triangleq d[k]|v[k]\ldots v[k-Q+1] \\ \tilde{s}[k] &\triangleq s[k]|v[k]\ldots v[k-Q+1];\{\vartheta_j\} \end{aligned} \tag{34}$$

  Moreover, P is the total available points in the training or in the validation sets and the index j ranges over the samples of the training or the validation sets.
- **Validation set**: It comprises the other half of the total dataset. Once each model has been trained, the validation set is used only once in order to compute the MSE of each model.

In total nine training algorithms were used for the model (7) using SNR values from 2dB to 20dB with an increment of 2dB. Each of the three algorithms were coupled with three instances of the learning rate decay. Specifically, these instances were:

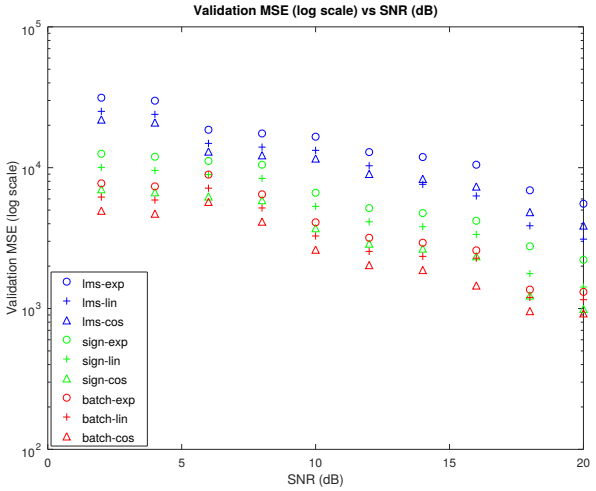

Fig. 1. Mean square error for validation phase vs SNR (dB).

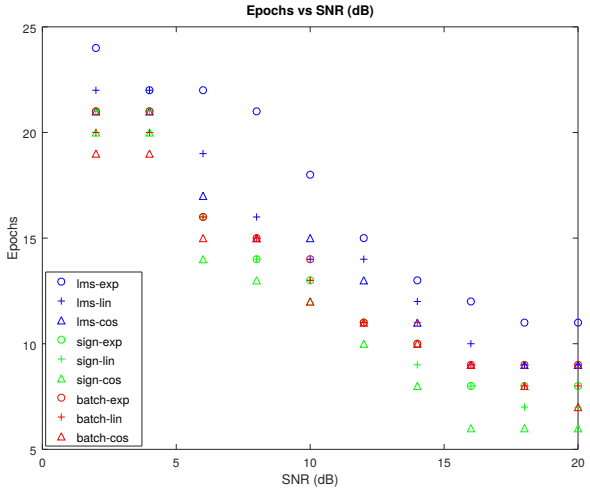

Fig. 2. Number of epochs vs SNR (dB).

- Cosine decay with $\alpha_0 = 0.01$ and $\beta_0 = 65$.
- Inverse linear decay with $\alpha = 0.01$ and $\beta_0 = 9/64$.
- Exponential decay with $\alpha = 0.1$ and $\beta_0 = 5/64$.

In figure 1 the validation MSE is shown for each of the nine combinations. It can be inferred that the batch LMS with the cosine decay has clearly the lowest one, followed closely by the sign LMS with cosine decay and the batch LMS with inverse linear decay. This can be attributed to the way both variants handle the approximation of the stochastic expected value of the error between the desired and the actual system response. The effect of SNR is also visible, as lower SNR values create a very strong trend for MSE values to increase.

In figure 2 is depicted the total number of epochs required for each combination. An epoch is defined as the number of iterations necessary to drive the model with every training pair, thus an epoch is a batch of iterations. Notice that batch LMS does not update model coefficients in every iteration as the other two schemes do. However, since the window length is small compared to the total number of training pairs, even this fractional update suffices to give batch LMS a very good performance in terms of epochs as well. In any case, sign LMS is the clear winner concerning the number of epochs, since two of its combinations consistently need the lowest number of epochs. Once again the effect of SNR can be seen, since the higher the SNR the less epochs are necessary for the training process to converge.

Figure 3 shows the Frobenius norm of the difference between two consecutive tensors. This figure has been generated using only the cosine decay rate for each of the three schemes and for the best and the worst SNR conditions. The number of epochs for each combination is repeated for clarity in table II. Notice that, since differences are shown, the epoch index starts from two and not from one.

Along a similar line of reasoning, in figure 4 is shown the sum of the Euclidean norms of the three local gradient vectors for the same combinations used in the previous figure. The same observations as with the previous figure can be made,

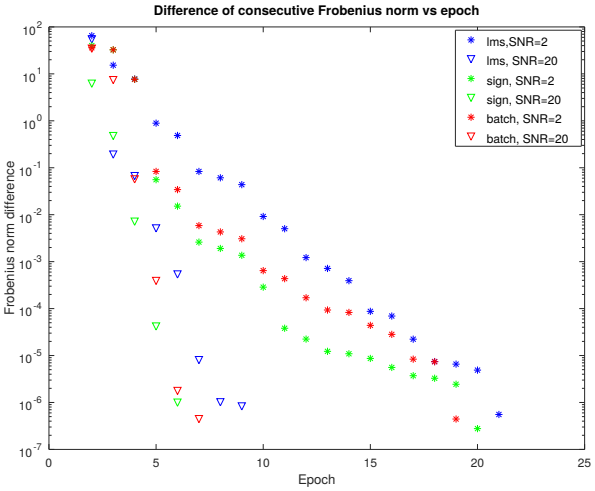

Fig. 3. Frobenius norm vs epoch.

verifying that convergence has been achieved.

VI. Conclusions And Future Work

This conference paper presented a model for certain non-linear sytems based on a third order tensor as well as an adaptive LMS-based training process for computing the coefficients of this model. Moreover, two variants of the training process, one based on sign LMS and one on batch LMS are also proposed. A total of nine combinations of training processes, three algorithms with three learning rate formulas, have been used to approximate a given non-linear system. Experiments indicate that the batch LMS with a window of length five and the sign LMS when combined with cosine decay yield the lowest and the second lowest mean square error during validation, even for relatively low SNR.

Both the LMS-based method and its two variants are based on an I/O description of the system under study. This implies that a large number of training pair patterns need to be provided to them, leading to a downtime which may not be

TABLE II
NUMBER OF EPOCHS FOR EACH COMBINATION (COSINE DECAY RATE).

Combination	LMS (SNR:2dB)	LMS (SNR:20dB)	Sign (SNR:2dB)	Sign (SNR:20dB)	Batch (SNR:2dB)	Batch (SNR:20dB)
Epochs	21	9	20	6	19	7

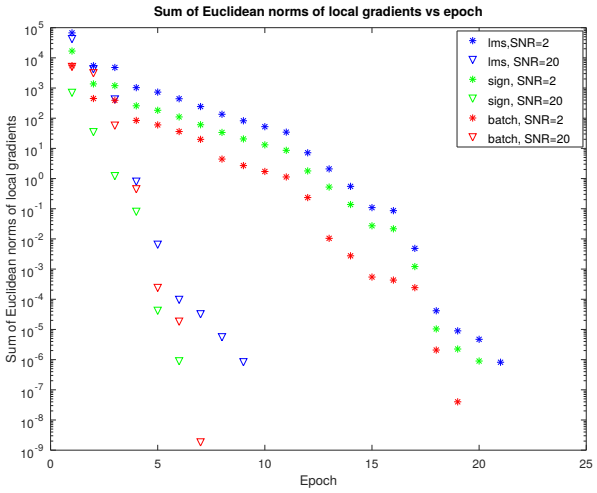

Fig. 4. Sum of Euclidean norms of local gradients vs epoch.

acceptable. However, this can be remedied by using online training. Moreover, the extension of the proposed model to longer input vectors so that more interactions can be represented or even the addition of more dimensions to the model are points worth investigating.

ACKNOWLEDGMENT

This research has been co-financed by the European Union and Greek national funds through the Competitiveness, Entrepreneurship and Innovation Operational Programme, under the Call "Research - Create - Innovate", project title: "Development of technologies and methods for cultural inventory data interoperability", project code: T1EDK-01728, MIS code: 5030954.

Moreover, this conference paper is part of Project 451, a long term research initiative for developing novel, scalable, numerically stable, and interpretable tensor analytics.

Finally, the authors gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research.

REFERENCES

[1] T. B. Schön, A. Wills, and B. Ninness, "System identification of nonlinear state-space models," *Automatica*, vol. 47, no. 1, pp. 39–49, 2011.

[2] A. Juditsky, H. Hjalmarsson, A. Benveniste, B. Delyon, L. Ljung, J. Sjöberg, and Q. Zhang, "Nonlinear black-box models in system identification: Mathematical foundations," *Automatica*, vol. 31, no. 12, pp. 1725–1750, 1995.

[3] T. Koh and E. Powers, "Second-order Volterra filtering and its application to nonlinear system identification," *TASSP*, vol. 33, no. 6, pp. 1445–1455, 1985.

[4] S. Chen and S. Billings, "Neural networks for nonlinear dynamic system modelling and identification," *International journal of control*, vol. 56, no. 2, pp. 319–346, 1992.

[5] R. Babuška and H. Verbruggen, "Neuro-fuzzy methods for nonlinear system identification," *Annual reviews in control*, vol. 27, no. 1, pp. 73–85, 2003.

[6] S. A. Billings and H.-L. Wei, "A new class of wavelet networks for nonlinear system identification," *TNN*, vol. 16, no. 4, pp. 862–874, 2005.

[7] T. Ogunfunmi, *Adaptive nonlinear system identification: The Volterra and Wiener model approaches*. Springer science and business media, 2007.

[8] I. W. Hunter and M. J. Korenberg, "The identification of nonlinear biological systems: Wiener and Hammerstein cascade models," *Biological cybernetics*, vol. 55, no. 2-3, pp. 135–144, 1986.

[9] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, 2009.

[10] B. W. Bader and T. G. Kolda, "Efficient MATLAB computations with sparse and factored tensors," *SIAM Journal on Scientific Computing*, vol. 30, no. 1, pp. 205–231, 2007.

[11] D. M. Dunlavy, T. G. Kolda, and E. Acar, "Temporal link prediction using matrix and tensor factorizations," *TKDD*, vol. 5, no. 2, p. 10, 2011.

[12] G. Drakopoulos, X. Liapakis, G. Tzimas, and P. Mylonas, "A graph resilience metric based on paths: Higher order analytics with GPU," in *ICTAI*. IEEE, November 2018.

[13] G. Drakopoulos, "Tensor fusion of social structural and functional analytics over Neo4j," in *IISA*. IEEE, July 2016.

[14] G. Drakopoulos, A. Kanavos, P. Mylonas, and S. Sioutas, "Defining and evaluating Twitter influence metrics: A higher order approach in Neo4j," *SNAM*, vol. 71, no. 1, 2017.

[15] G. Drakopoulos, A. Kanavos, D. Tsolis, P. Mylonas, and S. Sioutas, "Towards a framework for tensor ontologies over Neo4j: Representations and operations," in *IISA*, August 2017.

[16] G. Drakopoulos, F. Stathopoulou, A. Kanavos, M. Paraskevas, G. Tzimas, P. Mylonas, and L. Iliadis, "A genetic algorithm for spatiosocial tensor clustering: Exploiting TensorFlow potential," *Evolving Systems*, January 2019.

[17] L. De Lathauwer and J. Vandewalle, "Dimensionality reduction in higher-order signal processing and rank-$(r_1, r_2, \ldots, r_n)$ reduction in multilinear algebra," *LAA*, vol. 391, pp. 31–55, 2004.

[18] D. Nion and N. D. Sidiropoulos, "Tensor algebra and multidimensional harmonic retrieval in signal processing for MIMO radar," *IEEE Transactions on Signal Processing*, vol. 58, no. 11, pp. 5693–5705, 2010.

[19] J.-F. Cardoso, "Eigen-structure of the fourth-order cumulant tensor with application to the blind source separation problem," in *ICASSP-90*. IEEE, 1990, pp. 2655–2658.

[20] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *OSDI*, vol. 16, 2016, pp. 265–283.

[21] M. Abadi, "TensorFlow: Learning functions at scale," *ACM SIGPLAN Notices*, vol. 51, no. 9, pp. 1–1, 2016.

[22] S. Kontopoulos and G. Drakopoulos, "A space efficient scheme for graph representation," in *ICTAI*. IEEE, November 2014, pp. 299–303.

[23] D. G. Matthews *et al.*, "GPflow: A Gaussian process library using TensorFlow," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 1299–1304, 2017.

[24] K. Wongsuphasawat *et al.*, "Visualizing dataflow graphs of deep learning models in TensorFlow," *Transactions on visualization and computer graphics*, vol. 24, no. 1, pp. 1–12, 2018.

[25] S. Haykin, *Neural networks*. Prentice-Hall New York, 1994, vol. 2.

[26] C. M. Bishop, *Pattern recognition and machine learning*. Springer, 2006.

[27] B. Widrow, J. McCool, M. G. Larimore, and C. R. Johnson, "Stationary and nonstationary learning characteristics of the LMS adaptive filter," *Aspects of signal processing*, pp. 355–393, 1977.

[28] E. Ferrara, "Fast implementations of LMS adaptive filters," *TASSP*, vol. 28, no. 4, pp. 474–475, 1980.

[29] S. C. Douglas and W. Pan, "Exact expectation analysis of the LMS adaptive filter," *TSP*, vol. 43, no. 12, pp. 2863–2871, 1995.