Compressing a Neural Network Classifier using a Volterra-Neural Network model

M. Rubiolo, G. Stegmayer, *Member IEEE* and D. Milone, *Member IEEE*

Abstract— **Model compression is a required task when slow and large models are used, for example, for classification, but there are transmissions, space, time or computing capabilities constraints that have to be fulfilled. Multilayer Perceptron (MLP) models have been traditionally used as classifiers. Depending on the problem, they may need a large number of parameters (neuron functions, weights and bias) to obtain an acceptable performance. This work proposes a technique to compress an MLP model preserving, at the same time, its classification performance, through the kernels of a Volterra series model. The Volterra kernels can be used to represent the information that a Neural Network (NN) model has learnt with almost the same accuracy but compressed into less parameters. The Volterra-NN approach proposed in this work has two parts. First of all, it allows extracting the Volterra kernels from the NN parameters after training, which will contain the classifier knowledge. Second, it allows building different orders Volterra series model for the original problem using the Volterra kernels, significantly reducing the number of neural parameters involved to a very few Volterra-NN parameters (kernels). Experimental results are presented over the standard Iris classification problem, showing the good Volterra-NN model compression capabilities.**

I. Introduction

The purpose of model compression is to find a fast and compact model, in comparison to a slow and large model, to approximate a function learned by, for example, a classifier. Moreover, it is desirable to achieve this without significant loss in performance [1]. Often the best performing models that use supervised learning are combinations of complex and large classifiers. However, in some situations, it is not enough for a classifier to be highly accurate, it also has to meet some requirements regarding on-line execution time, storage space and limited computational power.

It is well-known that Multilayer Perceptron (MLP) models are usually considered as a powerful classification model. However, they easily became large models just by the addition of hidden neurons (this has a direct effect over the number of model weights) or more information to the model (more input variables), in order to better learn the training data and to improve its classification ability.

In fact, there are several proposals in literature for approximating a complex analytical model [2] [3], compressing an ensemble of classifiers [4] [5] and even extracting knowledge from trained neural network models [6] [7]. All of these cases use a neural model to learn another model. However, the authors are not aware of a technique that actually allows compressing the neural model itself.

It has been recently shown that a Volterra series model can be extracted from the parameters of a trained neural network (NN) based model. This has proven to be particularly useful for reproducing the nonlinear and dynamic behavior of new wireless communications devices [8]. The Volterra series and Volterra theorem was developed in 1887 by Vito Volterra. It is a model for representing nonlinear dynamic behavior, similar to the Taylor series, frequently used in system identification [9]. In [10][8] several formulas for the extraction of Volterra kernels, independently of the neural model topology, activation functions, number of variables involved in the problem and nonlinearity of the system, have been presented. Those proposals are based on architectures having any kind of nonlinear or polynomial activation functions in the hidden nodes, trained with a classical backpropagation algorithm [11], for multi-input, multi-output systems. The MLP model is trained using the available training data and, after that, the Volterra kernels are obtained from the trained network parameters.

This work presents a novel application of the Volterra kernels extraction method for model compression of a classical multilayer perceptron classifier. That is to say, we will show how a MLP model which has learnt a classification problem with a certain (high) accuracy, can be compressed into a more compact representation using a Volterra model and its parameters (named Volterra kernels). This new proposed representation involves less parameters, maintaining however a high classification accuracy. The proposed approach (named Volterra-NN) for neural model compression will be exemplified with the Iris database problem. It will be shown that several MLP topologies can be well-compressed into the first, second and third order Volterra kernels. The obtained results show also that these kernels can be used to build a Volterra model that needs less parameters than the MLP model and, however, has the same high accuracy than the trained neural model.

The organization of this paper is the following. Section II explains the proposed algorithm for neural model compression through the use of Volterra series approximations of different orders. Section III shows the case study and measures used for performance comparison. Section IV presents the analysis of results obtained and finally, the conclusions and future work can be found on Section V.

M. Rubiolo and G. Stegmayer are with Research and Development Center for Information Systems Engineering (CIDISI), Universidad Tecnológica Nacional, Facultad Regional Santa Fe, Lavaise 610, S3004EWB Santa Fe, Argentina (mrubiolo@santafe-conicet.gov.ar).

D. Milone is with Research Center for Signals, Systems and Computational Intelligence (SINC), Facultad de Ingeniería y Ciencias Hídricas - UNL, Ciudad Universitaria - CC 217, Ruta Nacional No 168 - Km 472.4, 3000 Santa Fe, Argentina (d.milone@ieee.org).

II. Volterra-NN model algorithm: compressing NN model through Volterra kernels

This section presents a novel algorithm for NN model compression, based on the extraction of the Volterra kernels from a trained NN model. It will be shown that different order Volterra-NN outputs can be obtained, according to the number and order of kernels considered. These facts will influence on the model compression rate obtained. However, we will also show that the classification rate can be maintained, even when high compression rates are achieved.

The following subsections explain some basics concepts on Volterra models necessary to understand the proposed approach; after that, the Volterra kernels extraction method is explained. Finally, the Volterra-NN model and its use for a classification problem are presented.

A. Algorithm basics

For representing a multiple-input single-output nonlinear time-invariant system, a Volterra series model can be used. For example, if a third order approximation is considered as enough to describe exactly the problem under study, a third-order discrete-time Volterra model relates the system inputs $x(t)$ and output $y(t)$ as follows

$$\begin{aligned} y(t) = h_0 + & \\ \sum_{k_1=0}^{\infty} h_1(k_1)x(t-k_1) + & \\ \sum_{k_1=0}^{\infty}\sum_{k_2=0}^{\infty} h_2(k_1,k_2)x(t-k_1)x(t-k_2) + & \\ \sum_{k_1=0}^{\infty}\sum_{k_2=0}^{\infty}\sum_{k_3=0}^{\infty} h_3(k_1,k_2,k_3)x(t-k_1)x(t-k_2)x(t-k_3). & \end{aligned} \tag{1}$$

where h_0, $h_1(.)$, $h_2(.)$ and $h_3(.)$ are known as the *Volterra kernels*. As can be seen, the Volterra kernels can be combined with the system inputs to obtain the system output. It is generally assumed that the Volterra kernels are symmetrical, that is to say, for any order $p \geq 2$

$$h_p(k_i, \cdots, k_p) = h_p(\pi(k_i, \cdots, k_p)) \tag{2}$$

where $\pi(.)$ is any permutation of $(k_1, \cdots, k_p)$ [12].

B. Volterra kernels extraction

In a previous work [8] it has been shown that the Volterra kernels can be extracted from the parameters of a MLP model trained with data D that represents the behavior of the system (or problem) $y(t) = f(x_1(t), x_2(t))$. For example, let us consider a MLP model that has two inputs ($N_I = 2$), one linear output neuron ($N_O = 1$) with a corresponding output neuron bias b_o, any number of hidden neurons (N_H) having any nonlinear function (ϕ) and an associated bias (b_h). For simplifying the notation, time will not be considered in what follows. The MLP model equation is the following

$$y = b_o + \sum_{h=1}^{N_H} w_h^2 \, \phi\left(b_h + \sum_{i=1}^{N_I} w_{h,i}^1 \, x_i\right). \tag{3}$$

Once this model has been trained on a problem, using sigmoidal hidden functions $\left(\phi(b_h) = \frac{1}{1+e^{-b_h}}\right)$, the following formulas allow the calculus of zero, first, second and third order Volterra kernels using the parameters of the trained MLP model,

$$h_0 = b_o + \sum_{h=1}^{N_H} w_h^2 \frac{1}{(1+e^{-b_h})} \tag{4}$$

$$h_1(.) = \sum_{h=1}^{N_H} w_h^2 w_{h,i}^1 \frac{e^{-b_h}}{(1+e^{-b_h})^2} \tag{5}$$

$$h_2(.) = \sum_{h=1}^{N_H} w_h^2 w_{h,i}^1 w_{h,j}^1 \frac{\frac{e^{-b_h}(e^{-b_h}-1)}{(1+e^{-b_h})^3}}{2!} \tag{6}$$

$$h_3(.) = \sum_{h=1}^{N_H} w_h^2 w_{h,i}^1 w_{h,j}^1 w_{h,k}^1 \frac{\frac{-e^{-b_h}(-e^{-2b_h}+4e^{-1}-1)}{(1+e^{-b_h})^4}}{3!} \tag{7}$$

for $i, j, k = [1, \cdots, N_I]$.

This work proposes a novel application of this Volterra kernels extraction method for neural model compression through a Volterra-NN model. This model is built using Volterra weights, that we define as follows: $V^{(0)} = h_0$, $v_i^{(1)} = h_1(k_i)$, $v_{i,j}^{(2)} = h_2(k_i, k_j)$ and $v_{i,j,k}^{(3)} = h_3(k_i, k_j, k_k)$. We also define the following Volterra-NN parameters: $\mathbf{V}^{(1)}$ is a vector that contains all the first-order Volterra weights $v_i^{(1)}$; $\mathbf{V}^{(2)}$ is a matrix that has the second-order Volterra weights $v_{i,j}^{(2)}$; and finally $\mathbf{V}^{(3)}$ is a matrix having the third-order Volterra weights $v_{i,j,k}^{(3)}$.

Algorithm 1 shows in detail the Volterra weights extraction procedure. The input is the training data and the outputs are the Volterra weights. The first step consists in training a MLP classifier model with the training data D as it is possible to see in line 2. From the trained neural model M the Volterra weights can be calculated. According to (4), the zero-order Volterra weight is obtained (line 3). Similarly, by applying (5), (6) and (7), it is possible to compute the first (lines 4 to 5), second (lines 6 to 8) and third-order Volterra weights (lines 9 to 12) respectively.

C. Volterra-NN model forward computation

The proposed Volterra-NN model for NN compression is depicted graphically in Figure 1. The boxes represent the input variables (x_1 and x_2) and the arrows that join them symbolize their product with their corresponding Volterra weight. The white circles act as summarizing all the products between input variables and Volterra weigths, plus the lower N^{th}-order Volterra series model (S_{N-1}). As a result, a new N^{th}-order Volterra approximation (S_N) is obtained. It is important to highlight that the different cross-products combinations between the input variables are shown in the figure

Algorithm 1: Volterra weights extraction.

```
Data:
    $D$: training data
Results:
    $V^{(0)}$: 0-order Volterra weight
    $\mathbf{V}^{(1)}$: $1^{st}$-order Volterra weigths
    $\mathbf{V}^{(2)}$: $2^{nd}$-order Volterra weigths
    $\mathbf{V}^{(3)}$: $3^{rd}$-order Volterra weigths
1  begin
2      $M \leftarrow$ train MLP model with $D$
3      $V^{(0)} \leftarrow$ calculate zero-order Volterra weight from
       $M$ using (4)
4      for $1 \leq i \leq N_I$ do
5          $v_i^{(1)} \leftarrow$ calculate first-order Volterra weight
           from $M$ using (5)
6      $\mathbf{V}^{(1)} \leftarrow v_i^{(1)}$
7      for $1 \leq i \leq N_I$ do
8          for $1 \leq j \leq N_I$ do
9              $v_{i,j}^{(2)} \leftarrow$ calculate second-order Volterra
               weight from $M$ using (6)
10     $\mathbf{V}^{(2)} \leftarrow v_{i,j}^{(2)}$
11     for $1 \leq i \leq N_I$ do
12         for $1 \leq j \leq N_I$ do
13             for $1 \leq k \leq N_I$ do
14                 $v_{i,j,k}^{(3)} \leftarrow$ calculate third-order Volterra
                   weight from $M$ using (7)
15     $\mathbf{V}^{(3)} \leftarrow v_{i,j,k}^{(3)}$
16 end
```

inside a rectangular box, in an effort to clarify the process for obtaining the different order Volterra outputs. For instance, the $1st$-order Volterra output S_1 can be obtained by adding the 0-order Volterra weight $V^{(0)}$ to the product between each input variable (x_1 and x_2) and their corresponding 1^{st}-order Volterra weight

$$S_1 = V^{(0)} + v_1^{(1)}\ x_1 + v_2^{(1)}\ x_2. \tag{8}$$

The second order Volterra output S_2 is obtained by adding S_1 together with the products among x_1^2, x_2^2, the cross-product $x_1 x_2$ and their corresponding 2^{nd}-order Volterra weights, resulting in:

$$S_2 = S_1 + v_{1,1}^{(2)}\ x_1^2 + v_{2,2}^{(2)}\ x_2^2 + v_{1,2}^{(2)}\ x_1 x_2. \tag{9}$$

The third order output S_3 is obtained similarly:

$$S_3 = S_2 + v_{1,1,1}^{(3)}\ x_1^3 + v_{2,2,2}^{(3)}\ x_2^3 + v_{1,1,2}^{(3)}\ x_1^2 x_2 + v_{1,2,2}^{(3)}\ x_1 x_2^2. \tag{10}$$

As can be seen, each high order Volterra output includes its own parameters (Volterra weights) plus the lower order ones.

The Volterra-NN forward computation is presented in detail in Algorithm 2. It receives the classification problem data D, from where the number of input variables N_I is determined, and the Volterra weights matrixes $V^{(0)}$, $\mathbf{V}^{(1)}$, $\mathbf{V}^{(2)}$ and $\mathbf{V}^{(3)}$), which have been obtained by using Algorithm 1. The Volterra-NN model outputs are the first-order (S_1), second-order (S_2) and third-order (S_3) Volterra outputs for the classification problem, that provide different compressed versions of the original MLP classifier.

Algorithm 2: Volterra-NN forward computation.

```
Data:
    $\mathbf{x}$: input point to classify
    $V^{(0)}$: 0-order Volterra weight
    $\mathbf{V}^{(1)}$: $1^{st}$-order Volterra weights
    $\mathbf{V}^{(2)}$: $2^{nd}$-order Volterra weights
    $\mathbf{V}^{(3)}$: $3^{rd}$-order Volterra weights
Results:
    $S_1$: $1^{st}$ order Volterra series output
    $S_2$: $2^{nd}$ order Volterra series output
    $S_3$: $3^{rd}$ order Volterra series output
1  begin
2      $S_1 \leftarrow V^{(0)}$
3      for $1 \leq i \leq N_I$ do
4          $S_1 = S_1 + v_i^{(1)}\ x_i$
5      $S_2 \leftarrow S_1$
6      for $1 \leq i \leq N_I$ do
7          for $1 \leq j \leq N_I$ do
8              $S_2 = S_2 + v_{i,j}^{(2)}\ x_i x_j$
9      $S_3 \leftarrow S_2$
10     for $1 \leq i \leq N_I$ do
11         for $1 \leq j \leq N_I$ do
12             for $1 \leq k \leq N_I$ do
13                 $S_3 = S_3 + v_{i,j,k}^{(3)}\ x_i x_j x_k$
14 end
```

D. Volterra-NN model for classification

This subsection explains how Volterra-NN can be applied to a classification problem. The class-limits have to be calculated over the training data and are applied testing data, for the classification rate calculus. This allows us to identify the different classes when unknown data is be presented to each model (MLP and/or Volterra-NN model).

Training data is shown in an ordered way, from the first to the third class samples. That is to say, first of all the data corresponding to the first class is introduced to the model; then the second-class data and the third-class data at last. In this way, the model output can be considered as a signal having three different levels, with bounds between levels corresponding to class limits [13].

In order to calculate them, the minimum and the maximum values of each classes are measured in order to determine the mean point between the level output-signal changes. These changes (thresholds) will allow us to identify to which class

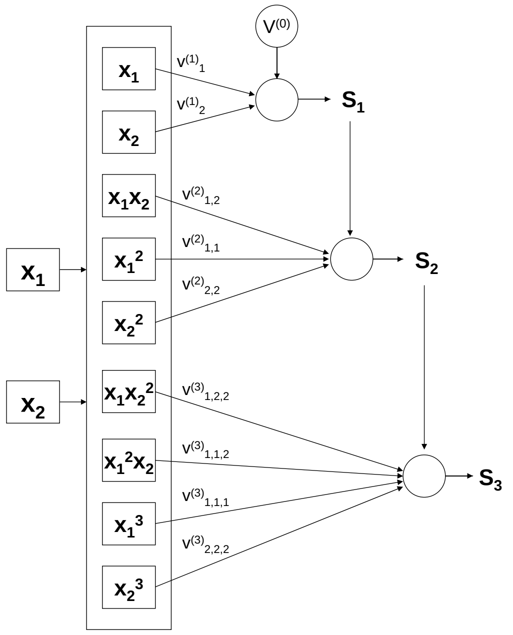

Fig. 1. Volterra-NN model.

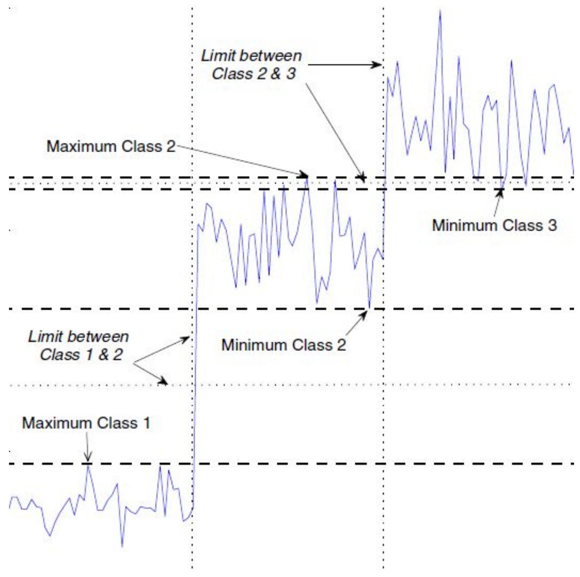

Fig. 2. Class limits calculation.

belongs a new data point when received for classification. For instance, a dataset can have three different levels representing the three possible classes to identify, as it is possible to see in Figure 2 where vertical dot lines set the class limit. In order to establish horizontal class limits, the minimum and maximum values for each neighboring classes are detected (horizontal dash lines show these value level). The new class limit will be located at the mean level between the maximum and minimum value, what can be seen as the horizontal dot lines. Therefore, depending on the level where the testing output value will be set down, it will be the class where this point will be classified.

III. Case Study and Performance Measures

This section presents the case study used to test the Volterra-NN compression capabilities proposed in this paper. First, the data set used for training a neural classifier is presented, as well as the MLP classifier architectures evaluated. After that, the performance measures used to evaluate the proposed method performance are explained.

A. Classification dataset and neural topologies

The Iris dataset[1] has been used for training and testing the proposed algorithm. It is perhaps the best known database to be found in the pattern recognition literature [14]. The dataset contains 3 classes of 50 instances each, where each class refers to a type of Northern American species of Iris plant (*Iris setosa*, *Iris versicolor* and *Iris virginica*). The first class is linearly separable from the other two; the Iris versicolor is not linearly separable from the Iris virginica. Each pattern has four numeric attributes for each flower: length and width, of sepal and petal.

We have used a k-fold cross-validation procedure [15], with 80% of each class data for training and 20% for testing. The complete dataset has been randomly split into $k = 5$ mutually exclusive subsets of equal size, repeating the experiments three times in each fold. The cross validation estimate of the overall accuracy of a model has been calculated by simply averaging the accuracy measures over the five test datasets.

Several MLP topologies have been evaluated, from where the corresponding Volterra kernels have been extracted after training. Each tested MLP model has four input variables ($N_I = 4$), corresponding to class attributes; and one output neuron indicating the class label ($N_O = 1$). Different number of neurons has been used for the hidden layer ($N_H = [4, 8, 12]$). For all cases, the hidden units have a sigmoidal activation function. The initial weights and bias values for the model have been set using a random function that throws a scalar value drawn from a uniform distribution on the unit interval [0;1].

In each experiment and repetition, MLP models having less than 90% classification rate for each class have not been considered for the analysis and Volterra kernels extraction. We have imposed this restriction for the classifier performance because we need to be able to measure precisely any loss of accuracy originated by the proposed Volterra-NN compression method.

B. Performance measures

From each MLP classifier trained over the Iris problem, three Volterra-NN outputs have been extracted: S_1, that only includes the zeror and first-order Volterra weights; S_2,

[1]http://archive.ics.uci.edu/ml/datasets/Iris

including up to the second-order Volterra weights; and finally S_3 that corresponds to a third-order output.

For comparing the proposed Volterra-NN against a classical MLP classifier, two performance measures are used: recognition rate (R) and data space savings (SS). In classification problems, the primary source of performance measurements is the overall accuracy of a classifier estimated through the *classification or recognition rate (R)*.

For measuring compression power, data compression ratio is used to quantify the reduction in data representation size produced by a compression algorithm. However, sometimes the *space savings (SS)* measure is given instead, defined as the reduction in size relative to the uncompressed size. To estimate how much is the compression level obtained by the method proposed in this paper, we define SS as the relation between the number of parameters of a MLP architecture (the weights and biases) and the number of parameters needed for each corresponding Volterra-NN output (the Volterra weights).

It is well-known that for a MLP model, the number of model parameters can be calculated as follows:

$$P_{MLP} = N_I \times N_H + N_{BH} + N_H \times N_O + N_{BO} \qquad (11)$$

where N_I, N_H and N_O are the number of neurons at input, hidden and output layers, respectively; and N_{BH} and N_{BO} are the number of bias for each neuron in the hidden and output layers, respectively.

The different Volterra-NN outputs are built according to the order of the Volterra weights involved in their construction. The following equations show the number of parameters necessary to build a first-order, second-order and third-order Volterra-NN outputs for a given training set.

For the first-order output S_1 we have

$$P_{S_1} = 1 + N_I. \qquad (12)$$

In this case, the number of parameters necessary to build S_1 are the zero-order weight ($V^{(0)}$ that is always a constant number), plus each first-order Volterra weight that multiplies each input variable.

For the second-order output S_2 we have

$$P_{S_2} = P_{S_1} + N_I^2 - \frac{N_I^2 - N_I}{2}. \qquad (13)$$

In this case, the number of parameters necessary to build S_1 are summed up together with the number of parameters necessary for S_2 (the second-order Volterra weights). There is a second-order weight for each input variable squared, plus the cross-products between each input variable. With respect to these last ones, and as stated before, since the symmetrical kernels are equivalent, they are considered only once. That is why half of the cross-product kernels are counted.

The number of parameters necessary to build S_3 are

$$P_{S_3} = P_{S_2} + N_I^2, \qquad (14)$$

that is to say, the number of kernels associated with S_2 plus the product of each third-order weight corresponding to each input variable, counting only once the symmetrical kernels.

Therefore, we define the space saving measure SS as follows:

$$SS = 1 - \frac{P_{S_i}}{P_{MLP}} \qquad (15)$$

where P_{S_i} is the number of parameters (Volterra weights or kernels) necessary to build the i^{th}-order Volterra-NN output, computed from (12), (13) or (14); and P_{MLP} is the number of parameters necessary to obtain the neural approximation, calculated using (11).

IV. Experimental results and discussion

Experimental results obtained for the case study presented above are shown in this section. Table I presents the results obtained for the models classification rate (R_i) for each class $i = 1, 2, 3$ over the Iris dataset problem. From each MLP model considered in this study ($MLP_{4,4,1}, MLP_{4,8,1}$ and $MLP_{4,12,1}$), their corresponding zero-order, first-order, second-order and third-order Volterra weights have been extracted according to Algorithm 1 and their corresponding Volterra-NN outputs S_1, S_2 and S_3 have been obtained using Algorithm 2.

The three MLP models show similar high classification rates, between 95% and 100% for each class, with a global average value over all classes of approximately 98%. When the MLP classifier has 4 hidden neurons ($MLP_{4,4,1}$), the *R* values regarding class 1 are 100% for all vNN-S_1, vNN-S_2 and vNN-S_3; for the second class, these values vary from 94% to 96%; they are between 88% and approximately 93% for the third class. From Table II it is possible to see that mean values for these Volterra-NN outputs vary between 94% and 95%. This means that, for this MLP architecture, a high average classifier performance can be maintained if it is replaced with a Volterra-NN output of order 1, 2 or 3. It can be seen that a worse *R* rate is obtained for the classification problem if vNN-S_1 is used instead of the original MLP, specially for class 3. This lower R is, still, of almost 90%. Furthermore, as will be shown in Table II, the vNN-S_1 Volterra-NN model requires a significant lower number of parameters than the original $MLP_{4,4,1}$ classifier (a relation of 1 : 5). Another important fact to take into account is that although the *R* values for vNN-S_3 are the highest, there is no compression in this case because the number of parameters necessaries to build it are more than the MLP ones.

With respect to the $MLP_{4,8,1}$ classifier, the vNN-S_1 Volterra-NN model has a low classification rate in comparison to the original neural architecture considering all the classes. A clear trend can be noticed with respect to the fact that, as more Volterra weights are included into the Volterra-NN output (the output-order is increased), the recognition rate improves, almost reaching the original classifier rate. This is achieved, however, at the cost of including more parameters in the output and therefore obtaining less compression. This trend can be also seen at the 4-hidden-neurons model, in spite of the very low differences between *R* values for the Volterra-NN outputs. Similar conclusions can be stated for the $MLP_{4,12,1}$ classifier. However, for this MLP model and their corresponding Volterra-NN results, the

TABLE I

RECOGNITION RATES (R) FOR THREE MLP CLASSIFIERS AND THEIR CORRESPONDING FIRST-ORDER, SECOND-ORDER AND THIRD-ORDER VOLTERRA-NN OUTPUTS.

	$MLP_{4,4,1}$			$MLP_{4,8,1}$			$MLP_{4,12,1}$		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
MLP	100.00%	96.00%	98.00%	100.00%	96.00%	95.33%	100.00%	97.33%	96.00%
vNN-S_1	100.00%	94.00%	88.00%	80.00%	66.00%	76.00%	50.00%	36.00%	52.67%
vNN-S_2	100.00%	94.00%	88.67%	94.67%	88.67%	95.33%	66.67%	46.00%	67.33%
vNN-S_3	100.00%	94.67%	92.67%	99.33%	94.67%	96.00%	99.33%	90.00%	93.33%

R obtained are very low for low-order Volterra-NN outputs. This means that this particular classifier topology cannot be efficiently reproduced by low-order Volterra-NN outputs, and therefore, by using such a small number of parameters. In spite of this fact, it can be said that, in general, as more parameters are used to build the Volterra-NN outputs, the recognition rates become higher, getting even very close to the original MLP model performance.

But if we consider that the three $MLP_{4,H,1}$-model topologies ($H = 4, 8, 12$) are proposed as solutions to the same problem, actually it is not necessary to consider $MLP_{4,12,1}$ topology because the best solution is given by $MLP_{4,4,1}$ topology. This means that there are a number of unnecessary weights at $MLP_{4,12,1}$ topology which are contributing to the overtraining of the model causing a recognition rate decline on the testing dataset. This weights can not be compressed efficiently with Volterra-NN model, throwing low recongition rates for this model solutions.

Table II shows the space savings rate (SS) for the models discussed in this paper and the mean values of the recognition rate (R) for these models. The upper part of the table shows the number of parameters and global R measure for the three MLP topologies. The second part of the table shows parameters and performance measure values related to the Volterra-NN outputs. First of all, the number of parameters needed for each the neural classifier architecture (P_{MLP}) involved in this study is shown at the first row, while the number of Volterra weights needed for each Volterra-NN output P_{S_i} is shown in the first column. The values which fill the R and SS columns are the recognition rate and space saving capabilities for each combination between a MLP classifier and a corresponding Volterra-NN output.

As stated in 11, the number of parameters necessary for each neural classifier depends not only on N_I but also on the number of neurons at the hidden layer (N_H). The number of parameters for $MLP_{4,4,1}$ is 25; for $MLP_{4,8,1}$ it is 49; and the $MLP_{4,12,1}$ model has 73 parameters. In the case of the different order Volterra-NN outputs, the number of parameters only depends of the number of inputs. The number of parameters for S_1 is 5 according to 12. By applying 13 and 14 the number of parameters needed for the Volterra-NN outputs S_2 and S_3 is 15 and 31, respectively.

It is possible to see that when using the 1^{st}-order Volterra-NN model vNN-S_1 instead of the $MLP_{4,4,1}$ classifier, a global recognition rate for the Iris problem of 94% can be obtained and a compression or space saving rate of 80% can be achieved. Half of this space saving rate is obtained if the vNN-S_2 model is used, and there is no compression by using vNN-S_3. This lack of compression is due to the fact that the number of parameters needed for vNN-S_3 is higher than the number of parameters necessaries for the $MLP_{4,4,1}$ model. For the $MLP_{4,8,1}$ model, the compression achieved by the Volterra-NN outputs is higher because of the amount of parameters associated with this model; in this case, the *SS* value achieves a maximum close to 90% when the smallest number of parameters is considered (a first-order Volterra-NN). A similar case is related to $MLP_{4,12,1}$ model, where this trend is also verified.

From Table II it is possible to conclude that a MLP classifier can be well-compressed using a Volterra-NN model; and this compression can be, in some cases, even higher than 90%. In fact, the MLP model architecture can be even more and more complex, can have many more hidden units, but the number of weights needed to build each Volterra-NN output will remain the same while the number of input variables of the problems are the same, and this will certainly be reflected in even higher space saving rates values.

However, in some cases, these high compression rates have to be payed by lower model classification rates. Analyzing this table, the direct influence of the compression rate over the classification task can be noticed. For the $MLP_{4,4,1}$ model in this experience, the best *R* value is related to the vNN-S_3 Volterra-NN model, achieving 95.11% but, as explained before, there is no compression in this case. Therefore, although vNN-S_2 can be considered as a better option with its corresponding *SS* of 40%, it is better to select the vNN-S_1 Volterra-NN model that has almost the same *R* value but a higher *SS* value: 80%. Considering the $MLP_{4,8,1}$ case, the best *R* value is associated with the vNN-S_3 model, which has only a *SS* of $36, 73\%$. But if a better compression is necessary for this topology, it is possible to choose the vNN-S_2 model which has a *SS* of 69.39% and preserves a high *R* value of 92.89%.

For the $MLP_{4,12,1}$ model, the best *R* value is associated with the vNN-S_3 Volterra-NN model and it is possible to achieve a *SS* of 57.53%. This is a very interesting result, however, because with approximately half the number of parameters than the original MLP classifier, the classification capacity of the vNN-S_3 model is high (94.22%) and very close to the MLP model. This particular case can be consid-

TABLE II

GLOBAL RECOGNITION RATE (R) AND SPACE SAVING(SS) COMPARISON FOR THREE MLP CLASSIFIERS AND THEIR CORRESPONDING FIRST-ORDER, SECOND-ORDER AND THIRD-ORDER VOLTERRA-NN OUTPUTS.

		$MLP_{4,4,1}$		$MLP_{4,8,1}$		$MLP_{4,12,1}$	
$P_{MLP} \rightarrow$		25		49		73	
$R \rightarrow$		98.00%		97.11%		97.78%	
	P_{S_i}	R	SS	R	SS	R	SS
vNN-S_1	5	94.00%	80.00%	74.00%	89.90%	46.22%	93.15%
vNN-S_2	15	94.22%	40.00%	92.89%	69.39%	60.00%	79.45%
vNN-S_3	31	95.11%	-24.00%	96.67%	36.73%	94.22%	57.53%

ered as one of the best overall results obtained in this study, where the $MLP_{4,12,1}$ classifier can be compressed almost a 60% using the corresponding vNN-S_3 Volterra-NN model without significantly loosing recognition capability. But, as was previously concluded, this topology ($MLP_{4,12,1}$) can not be considered as the best MLP solution for this problem because it has too many parameters, causing a probable overtraining problem and affecting the recognition rate of the associated Volterra-NN model.

The best compromise is reached by vNN-S_1 when the topology of the MLP model is $MLP_{4,4,1}$, which has a recognition rate almost equal (94%) to vNN-S_3 model in $MLP_{4,12,1}$ topology case, but with a space savings rate of 80%. This can also be seen by comparing these two cases in terms of absolute number of parameters. That is, while for vNN-S_1 model only 5 parameters are required in $MLP_{4,4,1}$ case, 31 parameters are necessaries for vNN-S_3 model when $MLP_{4,12,1}$ topology is considered.

V. CONCLUSIONS AND FUTURE WORK

In this work we have presented a new approach to neural network compression through the use of Volterra kernels. We have shown a method to obtain a compact representation of a NN model using the proposed Volterra-NN model and their corresponding different-order outputs. The Volterra-NN model has been tested on a pattern recognition task, obtaining almost the same accuracy than three different MLP classifiers that have been significantly compressed into less parameters.

Two algorithms that implement the proposed approach have been presented. Algorithm 1 allowed extracting the Volterra kernels from the NN parameters after training, which will contain the classifier knowledge and are named Volterra weights. Algorithm 2 allowed obtaining different Volterra-NN outputs using the Volterra weights when a new data point to be classified is received.

The experimental results, performed over the Iris database classification problem, have highlighted the Volterra-NN model compression capabilities, allowing to maintain in some cases the same recognition rate as the MLP model achieving, at the same time, very high compression rates.

Future work involves further application of the proposed method to other kind of problems that may involve a MLP model, not only classification. Additionally, different experiments may be done considering another NN architectures such us a MLP model having 3 neurons at the output layer allowing to specialize each output neuron with each class behavior. Furthermore, an array of three NN models may be used in order to better learn each class behaviour. These different MLP classifier topologies should be further studied in order to establish their impact on the compression capabilities offered by the proposed Volterra-NN model approach.

REFERENCES

[1] Buciluă, C., Caruana, R., Niculescu-Mizil, A.: Model compression. In: KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, New York, NY, USA, ACM (2006) 535–541

[2] Mhaskar, H.N.: Neural networks for optimal approximation of smooth and analytic functions. Neural Comput. **8**(1) (1996) 164–177

[3] Kim, T., Adali, T.: Approximation by fully complex multilayer perceptrons. Neural Comput. **15**(7) (2003) 1641–1666

[4] Hansen, L., Salamon, P.: Neural network ensembles. IEEE Transactions on Pattern Analysis and Machine Intelligence **12** (1990) 993–1001

[5] Kim, M.J., Kang, D.K.: Ensemble with neural networks for bankruptcy prediction. Expert Syst. Appl. **37**(4) (2010) 3373–3379

[6] Livingstone, D.J.: Artificial Neural Networks: Methods and Applications. Hardcover (2009)

[7] Sudheer, K.P.: Knowledge extraction from trained neural network river flow models. Journal of Hydrologic Engineering **10**(4) (2005) 264–269

[8] Stegmayer, G., Chiotti, O.: Volterra NN-based behavioral model for new wireless communications devices. Neural Computing and Applications **18** (2009) 283–291

[9] Volterra, V.: Theory of Functionals and Integral and Integro-Differential Equations. Dover (1959)

[10] Orengo, G., Colantonio, P., Serino, A., F., Stegmayer, G., Pirola, M., Ghione, G.: Neural networks and Volterra-series for time-domain pa behavioral models. International Journal of RF and Microwave CAD Engineering **17**(2) (2007) 160–168

[11] Marquardt, D.W.: An algorithm for least-squares estimation of nonlinear parameters. SIAM Journal on Applied Mathematics **11**(2) (1963) 431–441

[12] Kibangou, A., Favier, G.: Identification of fifth-order Volterra systems using i.i.d. inputs. IET Signal Processing **4** (2009) 30–44

[13] Korenberg M. J., Davidb R., I.W.H., Solomonc, J.E.: Parallel cascade identification and its application to protein family prediction. Journal of Biotechnology **91** (2001) 35–47

[14] Duda, R., Hart, P.: Pattern Classification and Scene Analysis. Wiley (2003)

[15] Haykin, S.: Neural Networks: A Comprehensive Foundation. (1999)