Extended evaluation of the Volterra-Neural Network for model compression

Mariano Rubiolo

CONICET, CIDISI-UTN-FRSF, Lavaise 610, (3000) Santa Fe, Argentina
Tel.: +54-342-4601579/2390 (Int: 258)
mrubiolo@santafe-conicet.gov.ar

Abstract. When large models are used for a classification task, model compression is necessary because there are transmission, space, time or computing constraints that have to be fulfilled. Multilayer Perceptron (MLP) models are traditionally used as a classifier, but depending on the problem, they may need a large number of parameters (neuron functions, weights and bias) to obtain an acceptable performance. This work extends the evaluation of a technique to compress an array of MLPs, through the outputs of a Volterra-Neural Network (Volterra-NN), maintaining its classification performance. The obtained results show that these outputs can be used to build an array of (Volterra-NN) that needs significantly less parameters than the original array of MLPs, furthermore having the same high accuracy in most of the cases. The Volterra-NN compression capabilities have been tested by solving several kind of classification problems. Experimental results are presented on three well-known databases: Letter Recognition, Pen-Based Recognition of Handwritten Digits, and Face recognition databases.

1 Introduction

Model compression aims to find a fast and compact model to approximate a function learned by, for example, a classifier, preferably without significant loss in performance [1]. Complex and large classifiers are often needed to obtain the best performing models when supervised learning is used. In some situations, however, a classifier not only has to be highly accurate but also has to meet some requirements regarding on-line execution time, storage space and limited computational power [2].

Multilayer Perceptron (MLP) models are usually considered as a powerful classification model, but they easily become large models, which has a direct effect over the number of model weights. The increment of the model complexity can be caused by introducing more information to the model. That is to say, the introduction of more input variables in order to better learn the training data and to improve its classification ability, or the addition of hidden neuron in order to improve the learning process.

From the parameters of a trained Neural Network (NN) it is possible to extract a Volterra model [3], which is formed by Volterra kernels associated with these parameters, being useful for reproducing the nonlinear and dynamic behavior of new wireless communications devices [4], for instance. In [5] the Volterra kernels extraction procedure has been used to build a Volterra-Neural Network (V-NN) model, which is a compressed version of a trained MLP model over a simple classification task, maintaining the same recognition rate than the original model but with fewer parameters.

Often, better results can be reached by using arrays or ensembles of neural classifiers than single models [6][7]. For instance, an array of Multi Layer Perceptrons (*a*MLP) model was proposed in [8] for a complex classification task, consisting of one MLP for each class, with a final decision made over the network outputs of the complete array. The classification was performed by the maximum output calculation among all the networks outputs. This configuration has achieved improvements over the performance of a classic MLP. However, the use of this new neural configuration implies much more parameters, and therefore, a larger and more complex classification system.

Since it is possible to obtain a Volterra-NN model from a single trained MLP model, this work show that it is also feasible to compress an *a*MLP using Volterra-NNs. Thus, the same methodology applied to build a Volterra-NN model from a single MLP can be used to obtain an array of V-NN models (*a*V-NN) from an *a*MLP. Extending the evaluation of the method for compressing a classification model based on the application of the V-NN method to an array of MLP, it will be shown how an *a*MLP model that has learnt a classification problem with a certain (high) accuracy can be compressed into a more compact representation using an *a*V-NN model. This representation involves less parameters, maintaining however a high recognition accuracy. Three different databases have been used to show the effectiveness of the proposed method in classification problems of different level of complexity, number of classes and types of applications.

The paper is organized as follows. Section 2 explains in detail the proposed Volterra-NN model and its use as a classifier. The materials and methods used in the study are presented on Section 3. Results of the model evaluation through three public databases as well as a discussion of the experiments are shown in Section 4. Finally, Section 5 presents the conclusions and future work.

2 Model compression by using Volterra-Neural Network

The Volterra series and Volterra theorem was developed in 1887 by Vito Volterra. It is a model for representing nonlinear dynamic behavior frequently used in system identification [9]. The formulas for the extraction of Volterra weights, independently of the neural model topology, number of variables involved in the problem and nonlinearity of the system have been presented in [3]. Those equations are based on architectures having an hyperbolic tangent activation functions in the hidden nodes, trained with a classical backpropagation algorithm [10], for multi-input, multi-output systems. The MLP model is trained using the available training data and, after that, the Volterra weights are obtained from the trained network parameters.

This section presents the *a*V-NN model for an *a*MLP model compression, extending the simple algorithm for classification proposed in [5]. The following subsection presents the neural model used; after that, some basics concepts on Volterra models necessary to understand the proposed approach are explained. Finally, it is shown how the *a*V-NN can perform as a classifier when an *a*MLP is used.

2.1 Neural model for classification

It has been proved that arrays and ensembles of single multilayer perceptron networks can reach better results in classification problems than single models [11][12][8]. The

Algorithm 1: Volterra weights extraction, based on formulas presented in [3], in which N_H is the number of hidden neurons, N_I is the number of input neurons, w_h and b_h are the weight and the bias associated with a hidden sigmoidal neuron.

```
Data:
    $D$: training data
Results:
    $v^{(0)}$: 0-order Volterra weight
    $\boldsymbol{v}^{(1)}$: $1^{st}$-order Volterra weigths
    $\boldsymbol{v}^{(2)}$: $2^{nd}$-order Volterra weigths
    $\boldsymbol{v}^{(3)}$: $3^{rd}$-order Volterra weigths
begin
    $M \leftarrow$ train MLP model with $D$
    $v^{(0)} \leftarrow$ calculate zero-order Volterra weight from $M$ using the formula: $h_0 = b_o + \sum_{h=1}^{N_H} w_h^2 \frac{1}{(1+e^{-b_h})}$
    for $1 \leq i \leq N_I$ do
        $v_i^{(1)} \leftarrow$ calculate first-order Volterra weight from $M$ using the formula: $h_1(\cdot) = \sum_{h=1}^{N_H} w_h^2 w_{h,i}^1 \frac{e^{-b_h}}{(1+e^{-b_h})^2}$
    Assign each $v_i^{(1)}$ extracted to $\boldsymbol{v}^{(1)}$
    for $1 \leq i \leq N_I$ do
        for $1 \leq j \leq N_I$ do
            $v_{i,j}^{(2)} \leftarrow$ calculate second-order Volterra weight from $M$ using the formula:
            $h_2(\cdot) = \sum_{h=1}^{N_H} w_h^2 w_{h,i}^1 w_{h,j}^1 \frac{\frac{e^{-b_h}(e^{-b_h}-1)}{(1+e^{-b_h})^3}}{2!}$
    Assign each $v_{i,j}^{(2)}$ extracted to $\boldsymbol{v}^{(2)}$
    for $1 \leq i \leq N_I$ do
        for $1 \leq j \leq N_I$ do
            for $1 \leq k \leq N_I$ do
                $v_{i,j,k}^{(3)} \leftarrow$ calculate third-order Volterra weight from $M$ using the formula:
                $h_3(\cdot) = \sum_{h=1}^{N_H} w_h^2 w_{h,i}^1 w_{h,j}^1 w_{h,k}^1 \frac{\frac{-e^{-b_h}(-e^{-2b_h}+4e^{-1}-1)}{(1+e^{-b_h})^4}}{3!}$
    Assign each $v_{i,j,k}^{(3)}$ extracted to $\boldsymbol{v}^{(3)}$
end
```

final classifier model is an array of MLPs where there is one MLP model for each class *k* to be identified, with *k*=1..*K*, being *K* the total number of classes. Therefore the *a*MLP model is formed by *K* networks, whose outputs takes a value of 1 if the class is identified or 0 otherwise. The first layer of each MLP in the *a*MLP is a set of N_I input neurons and there are N_H hidden neurons. When a class has to be determined, its input is presented to all the *k* networks defined for the *a*MLP model, and the maximum output obtained among all network outputs is assigned as class label. If a pattern of the *k*th-class has been presented to the model, a value of (near) 1 is expected at the *k*th network.

In [5] was presented the application of the Volterra weights extraction method for model compression of a classical multilayer perceptron classifier. It was shown 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 weights. Several MLP topologies can be well-compressed into the first, second and third order Volterra weights, which can be used to build a Volterra model that needs less parameters than the original MLP model and, at the same time, has a similar high accuracy. This work proposes to apply and extend the compression procedure based on Volterra models to an array of MLPs to solve many kind of different classification problems. The next subsection presents the details of the

Algorithm 2: 3^{rd}-order Volterra-NN outputs forward computation.

```
    Data:
        $\boldsymbol{x}$: input point to classify
        $K$: number of elements in the array
        $\boldsymbol{v}^{(0)}$: 0-order Volterra weights for the array
        $\boldsymbol{v}^{(1)}$: array $1^{st}$-order Volterra weights
        $\boldsymbol{v}^{(2)}$: array $2^{nd}$-order Volterra weights
        $\boldsymbol{v}^{(3)}$: array $3^{rd}$-order Volterra weights
    Results:
        $\boldsymbol{s}^{(1)}$: $V^{(1)} - NN$ outputs for the array
        $\boldsymbol{s}^{(2)}$: $V^{(2)} - NN$ outputs for the array
        $\boldsymbol{s}^{(3)}$: $V^{(3)} - NN$ outputs for the array
1   begin
2       for each element in the array do
3           $s^{(1)} \leftarrow v^{(0)}$
4           for $1 \leq i \leq N_I$ do
5               $s^{(1)} = s^{(1)} + v_i^{(1)} x_i$
6           $s^{(2)} \leftarrow s^{(1)}$
7           for $1 \leq i \leq N_I$ do
8               for $1 \leq j \leq N_I$ do
9                   $s^{(2)} = s^{(2)} + v_{i,j}^{(2)} x_i x_j$
10          $s^{(3)} \leftarrow s^{(2)}$
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                      $s^{(3)} = s^{(3)} + v_{i,j,k}^{(3)} x_i x_j x_k$
15          Assign each $s^{(1)}$ calculated to $\boldsymbol{s}^{(1)}$
16          Assign each $s^{(2)}$ calculated to $\boldsymbol{s}^{(2)}$
17          Assign each $s^{(3)}$ calculated to $\boldsymbol{s}^{(3)}$
18      Determine upper and lower thresholds for each array element (class)
19  end
```

algorithm proposed for the procedure of obtaining different order Volterra-NN outputs from an array of MLPs used for classification.

2.2 Volterra-NN model forward computation

Volterra weights extraction procedure is shown in detail in Algorithm 1. Training data are the input and Volterra weights are the outputs. 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: the zero-order Volterra weight is obtained in line 3, as well as the first (line 5), second (line 9) and third-order (line 14) Volterra weights. The formulas that allow to compute the zero, first, second and third order Volterra weights were defined in [3] to calculate the zero, first, second and third-order Volterra kernel from a trained MLP having sigmoidal hidden units.

In the same way, higher order V-NN outputs can be calculated. Each high order Volterra model includes its own parameters (Volterra weights) plus the lower order ones. The new algorithm for an array of V-NN models forward computation is presented in detail in Algorithm 2. It receives a point from where the number of input variables N_I is determined, the number of elements in the array and the Volterra weights for each element in the array, obtained by using Algorithm 1. The output of this algorithm is an array of third-order $(V^{(3)} - NN)$ Volterra outputs, but it is also possible to obtain only the first-order $(V^{(1)} - NN)$ (lines 3 to 5) and second-order $(V^{(2)} - NN)$ outputs (lines

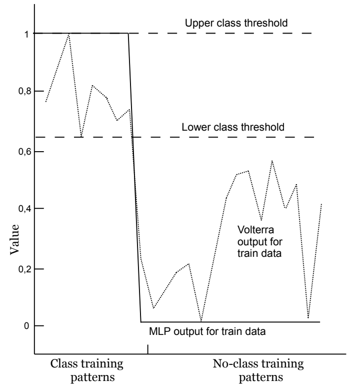

Fig. 1. Model output signal analysis for classification. Full and dotted lines: output signal corresponding to the MLP and V-NN models, respectively, for training data. Dashed lines: class thresholds.

7 to 10), thus providing different compressed versions of the original *a*MLP classifier. The next subsection shows how the compressed model can be used as a classifier.

2.3 Classifying by using Volterra-NN.

Different classes can be recognized from the output signal by applying the Volterra-NN model. After the compression of an array of MLP models by using V-NN, each V-NN model is specialized on a class and the output signal analysis determines if the pattern is part of the class. That is, it is possible to identify only a class from each of the output signals. During the training phase, those patterns beloging to the class associated to the model are shown first. In this way, the model output can be considered as a signal having two levels, with a bound between levels corresponding to the class limit [13]. For instance, if we have a three classes problem, the first model is trained with data where the patterns corresponding to the first-class are activated (they have a 1 as target) and other patterns, which do not represent the first class, are not activated (they have a 0 as target). Similarly, training data for learning the second and third classes are presented to their corresponding models.

Output signal analysis that has to be performed for determining upper and lower thresholds for each class is shown in Figure 1. It is possible to see the output signal corresponding to the MLP (full line) and V-NN (dotted line) models, both for the training data. The lower and the upper threshold (dashed lines) of the class are measured in order to determine the output-signal level change for the V-NN model, considering the values obtained for the training set. These changes (thresholds) will allow us to identify a new data point received for classification as belonging (or not) to the class. The membership of a new pattern (test point) to the class is identified by analyzing if the output associated to this pattern has a value between the lower and upper class thresholds.

Since we are analyzing the output of an *a*MLP model, *a* different signals, each one corresponding to a class, must be considered in order to identify which class is activated

Table 1. Datasets.

Dataset	k	p	train	test
Pendigits	10	16	8400	2100
Letters	26	16	15184	3796
Feret	40	37	24	66

as a result. First of all, each individual signal must be evaluated as it was presented in the above paragraph. Secondly, it is necessary to discover if more than one class model was activated for each pattern. If this situation occurs, a *max* criteria is applied. That is to say, the higher value of the activated classes for each pattern will be determined as the winner class.

3 Materials and Methods

3.1 Databases and *a*MLP architecture

Three independent classification databases have been used in this study: Letter Recognition[1] (Letters), Pen-Based Recognition of Handwritten Digits[2] (Pendigits), and Face recognition database FERET [3]). These well known databases provide typical experimental setups for classification problems.

Table 1 presents the three databases mentioned before in detail. The name of the databases are shown in the first column, meanwhile in the second and third column the number of classes (k) and the number of variables (p) of the databases are presented, respectively. The number of patterns in the training dataset are shown in the fourth column (train), as well as the number of patterns in testing dataset is presented in the fifth column.

To avoid overfitting, a k-fold cross-validation procedure [14] has been used, using the standard setup of splitting the available patterns into 80% of each class for training and 20% for testing. The complete dataset has been randomly split into $k = 3$ 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 test datasets. In each experiment and repetition, MLP and V-NN models having less than 95% classification rate for each class of the training dataset have not been considered in this study. This restriction was imposed to the classifier performance in order to be able to measure precisely any loss of accuracy originated by the proposed Volterra-NN compression method.

Figure 2 shows the neural network topology used in this study, an array of MLP models, each one associated with a class to recognize. Several MLP topologies have been evaluated from where the corresponding Volterra weights have been extracted after training, in order to build the V-NN models. For each input signal arriving at the array,

[1] http://archive.ics.uci.edu/ml/datasets/Letter+Recognition

[2] http://archive.ics.uci.edu/ml/datasets/Pen-Based+Recognition+of+Handwritten+Digits

[3] http://www.itl.nist.gov/iad/humanid/feret

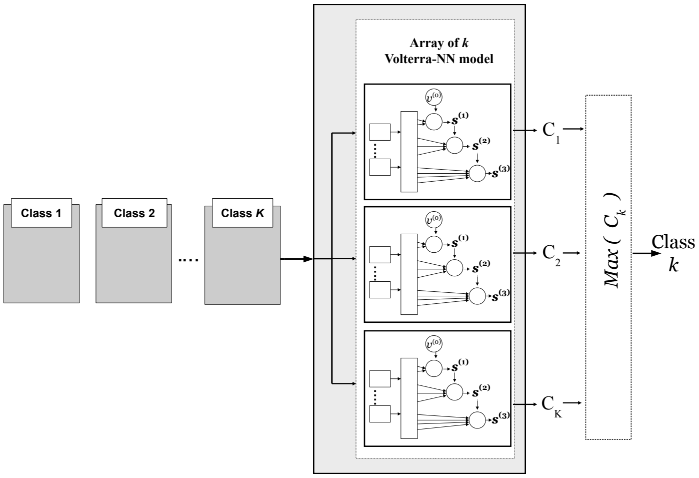

Fig. 2. Array of V-NN for classification problems.

a V-NN output of a certain order can be obtained. For example, to obtain a first order $V^{(1)} - NN$ output, the maximum $s^{(1)}$ from the array is selected. The model parameters (weights and biases) are initialized with random values uniformly distributed between 0 and 1. Neurons in the hidden and output layers have sigmoid activation functions. All the MLPs are trained with the Levenberg-Marquardt algorithm [15] in order to guarantee a fast convergence.

3.2 Performance measures

For comparing each output from the proposed V-NN models against the corresponding original MLP classifier, two performance measures are used: recognition rate (RR) and data space savings (SS), adapted here for an *a*MLP classifier. In classification problems, the primary source of performance measurements is the overall accuracy of a classifier estimated through the classification or recognition rate [16]. For measuring compression, data compression ratio can be used to quantify the reduction in data representation size produced by a compression algorithm. However, the space saving measure is given here instead, defined as the reduction in size relative to the uncompressed space [17], often reported as a percentage, which gives a better idea of compression power. For both cases, the greater the rate, the better the result.

To estimate how much is the compression level obtained, SS is calculated as the relation between the number of parameters needed for a Volterra-NN output (the Volterra weights) and the number of parameters of each corresponding MLP architecture (the weights and biases). It is well-known that for a MLP model, the number of model parameters can be calculated as $P_{MLP} = N_I \times N_H + N_{BH} + N_H \times N_O + N_{BO}$, 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. In an array of MLP models, these measures are affected by the array length a. Hence, the number of model parameters for an aMLP can be calculated as $P_{aMLP} = a \times P_{MLP}$.

From each MLP that is part of the array of MLPs, a Volterra-NN output (of a particular order) can be extracted: $s^{(1)}$ includes only the zero and first-order Volterra weights; $s^{(2)}$ includes up to the second-order Volterra weights; and $s^{(3)}$ corresponds to a third-order model output. The following equations show the number of parameters necessary to build each of these outputs, for a given training set.

For the $V^{(1)} - NN$ model output we have $P_{s^{(1)}} = N_I$. In this case, the number of parameters necessary to build $s^{(1)}$ are each first-order Volterra weight that multiplies each input variable.

For the second-order model $V^{(2)} - NN$ we have the following output $P_{s^{(2)}} = P_{s^{(1)}} + N_I^2 - \frac{N_I^2 - N_I}{2}$. 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, since the symmetrical weights are equivalent, they are considered only once. That is why half of the cross-product weights are counted.

The number of parameters necessary to build $V^{(3)} - NN$ are $P_{s^{(3)}} = P_{s^{(2)}} + N_I^2$, that is to say, the number of weights associated with $s^{(2)}$ plus the product of each third-order weight corresponding to each input variable, counting only once the symmetrical weights.

As Volterra-NN can be applied to the compression of an aMLP model, the output now will be an array of V-NN outputs of different order. That is to say, for example, for each MLP inside the array three different order V-NN outputs can be obtained. In this case, the number of V-NN models parameters has to be redefined as $P_{as^{(i)}} = a \times P_{s^{(i)}}$, where i = 1,2,3. Therefore, the space saving measure SS is calculated as

$$SS = 1 - \frac{P_{as^{(i)}}}{P_{aMLP}}. \tag{1}$$

4 Results and discussion

The experimental results obtained on three well-known classification problems are shown in this section. For simplicity in the analysis of the results, in particular which respects the performance of the V-NN compression capabilities, the tables show only global RR of each database (the average RR value over all classes) and the space savings rate for the models discussed in this paper.

From each array of MLPs model, with I number of inputs neurons, H number of hidden neurons, and O outpus neurons ($aMLP_{I,H,O}$) considered in this study, their corresponding zero-order, first-order, second-order and third-order Volterra weights have been extracted according to Algorithm 1 and their corresponding array of Volterra-NN outputs $s^{(1)}$, $s^{(2)}$ and $s^{(3)}$ have been obtained using Algorithm 2. Table 2 presents the results to the global recognition rate (RR) and the space savings rate (SS), calculated over the Pen-Based Recognition of Handwritten Digits test dataset; whereas Table 3 and 4 show similar results but calculated over the Letter Recognition and FERET database, respectively.

Table 2. Global recognition rate (RR) and space saving(SS) comparison for three *a*V-NN (*a*=10) classifiers and their corresponding first-order $s^{(1)}$, second-order $s^{(2)}$ and third-order $s^{(3)}$ outputs, for the Pen-Based Recognition of Handwritten Digits Database.

		10MLP$_{16,16,1}$		*10MLP*$_{16,32,1}$		*10MLP*$_{16,48,1}$	
$P_{aMLP_{I,H,O}}$ →		3040		6080		9120	
RR[%] →		98.47		98.61		98.58	
*10*V-NN	$P_{a_{s^{(i)}}}$	RR[%]	SS[%]	RR[%]	SS[%]	RR[%]	SS[%]
*10*V-NN$_{s^{(1)}}$	160	84.20	94.74	83.92	97.37	82.46	98.25
*10*V-NN$_{s^{(2)}}$	1520	83.18	50.00	84.01	75.00	81.73	83.33
*10*V-NN$_{s^{(3)}}$	4080	84.14	−34.21	84.56	32.89	84.35	55.26

Table 3. Global recognition rate (RR) and space saving(SS) comparison for three *a*V-NN (*a*=26) classifiers and their corresponding first-order $s^{(1)}$, second-order $s^{(2)}$ and third-order $s^{(3)}$ outputs, for the Letter Recognition Database

		26MLP$_{16,16,1}$		*26MLP*$_{16,32,1}$		*26MLP*$_{16,48,1}$	
$P_{aMLP_{I,H,O}}$ →		7904		15808		23712	
RR[%] →		97.41		97.70		97.67	
*26*V-NN	$P_{a_{s^{(i)}}}$	RR[%]	SS[%]	RR[%]	SS[%]	RR[%]	SS[%]
*26*V-NN$_{s^{(1)}}$	416	92.37	94.74	92.93	97.37	92.78	98.25
*26*V-NN$_{s^{(2)}}$	3952	92.81	50.00	93.17	75.00	93.25	83.33
*26*V-NN$_{s^{(3)}}$	10608	92.83	−34.21	93.24	32.89	92.70	55.26

Table 4. Global recognition rate (RR) and space saving(SS) comparison for three *a*V-NN (*a*=40) classifiers and their corresponding first-order $s^{(1)}$, second-order $s^{(2)}$ and third-order $s^{(3)}$ outputs, for the Facial Recognition Technology (FERET) Database.

		40MLP$_{37,37,1}$		*40MLP*$_{37,74,1}$		*40MLP*$_{37,111,1}$	
$P_{aMLP_{I,H,O}}$ →		59200		118400		177600	
RR[%] →		97.16		96.76		96.23	
*40*V-NN	$P_{a_{s^{(i)}}}$	RR[%]	SS[%]	RR[%]	SS[%]	RR[%]	SS[%]
*40*V-NN$_{s^{(1)}}$	1480	96.69	97.50	95.90	98.75	96.51	99.17
*40*V-NN$_{s^{(2)}}$	29600	96.34	50.00	96.49	75.00	96.35	83.33
*40*V-NN$_{s^{(3)}}$	84360	96.19	−42.50	96.60	28.75	96.29	52.50

The upper part of each table shows the number of parameters and global RR measure for the three *a*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 neural classifier architecture (P_{aMLP}) involved in this study is shown at the first row, while the number of Volterra weights needed for each Volterra-NN model ($P_{a_{s^{(i)}}}$) is shown in the first column. The values which fill the RR and SS columns are the average recognition rate and space saving capabilities, respectively, for each combination between a MLP classifier and the corresponding Volterra-NN output for all the 3 folds used.

As stated before (see subsection 3.2), the number of parameters required for the neural classifier depends not only on N_I but also on the number of neurons at the hidden layer N_H, as well as in the number of elements in the array. Therefore, considering the pendigits database the number of parameters for *10*$\mathrm{MLP}_{16,16,1}$ is 3040, for *10*$\mathrm{MLP}_{16,32,1}$ it is 6080, and the *10*$\mathrm{MLP}_{16,48,1}$ model has a total of 9120 parameters (see Table 2). Regarding the letters database, the number of parameters for *26*$\mathrm{MLP}_{16,16,1}$ is 7904, for *26*$\mathrm{MLP}_{16,32,1}$ it is 15808, and the *26*$\mathrm{MLP}_{16,48,1}$ model has a total of 23712 parameters (see Table 3). Finally, taking into account the FERET database, the number of parameters are 59200 for the *40*$\mathrm{MLP}_{37,37,1}$ model, 118400 for *40*$\mathrm{MLP}_{37,74,1}$, and 177600 for *40*$\mathrm{MLP}_{37,111,1}$ (see Table 4).

In the case of the different order Volterra-NN outputs, the number of parameters only depends on the number of inputs, as it is possible to see in subsection 3.2. However, for an array, this number must be multiplied by the array size. Therefore, the number of parameters for *10*$\text{V-NN}_{s(1)}$ is 160 in the Pendigits database model, and the number of parameters needed for *10*$\text{V-NN}_{s(2)}$ and *10*$\text{V-NN}_{s(3)}$ are 1520 and 4080, respectively. Regarding Letters database, the number of parameters are 416, 3952 and 10608 for the *26*$\text{V-NN}_{s(1)}$, *26*$\text{V-NN}_{s(2)}$ and *26*$\text{V-NN}_{s(3)}$, respectively. In the FERET database, the number of parameters for *40*$\text{V-NN}_{s(1)}$ is 1480, for *40*$\text{V-NN}_{s(2)}$ is 29600 and for *40*$\text{V-NN}_{s(3)}$ is 84360.

Focusing on Table 2, it is possible to see that when using *10*$\text{V-NN}_{s(1)}$ instead of the *10*$\mathrm{MLP}_{16,16,1}$ classifier, a global recognition rate of 84.20% can be obtained and a compression or space saving rate of 94.74% can be achieved. Almost half of this space saving rate is obtained if the *10*$\text{V-NN}_{s(2)}$ output is used, and there is no compression when using *10*$\text{V-NN}_{s(3)}$. For the *10*$\mathrm{MLP}_{16,32,1}$ model, the compression achieved by the Volterra-NN models is higher because of the amount of parameters associated with this model; in this case, the SS value achieves a maximum of 97.37% when the smallest possible number of parameters is considered (a first-order Volterra-NN output). A similar case is related to *10*$\mathrm{MLP}_{16,48,1}$ model, where this trend is also verified.

From Table 3, it is possible to see that a global recognition rate of 92.37% can be obtained and a compression or space saving rate of 94.74% can be achieved when the *26*$\text{V-NN}_{s(1)}$ is used instead of the *26*$\mathrm{MLP}_{16,16,1}$ classifier in the Letter Recognition case (see Table 3). Considering the *26*$\mathrm{MLP}_{16,32,1}$ case, the best RR value is associated with the *26*$\text{V-NN}_{s(2)}$ output, which has only a SS of 75%. But if a better compression is necessary for this topology, it is possible to choose the *26*$\text{V-NN}_{s(1)}$ which has a SS of 97.37% and preserves a very high RR value of 92.93%. The best RR value for the *26*$\mathrm{MLP}_{16,48,1}$ model is associated with the *26*$\text{V-NN}_{s(2)}$ Volterra-NN model and it is possible to achieve a SS of 83.33%. But, if *26*$\text{V-NN}_{s(2)}$ Volterra-NN model is chosen, the value of RR worsens just a few tenths, and a very high SS value of 98.25% is obtained. This is a very interesting result, because with approximately less than 2% of the parameters of the original *a*MLP classifier, the classification capacity of *26*$\text{V-NN}_{s(1)}$ is very high (92.78%) and very close to the *a*MLP model. This particular case can be considered as the best overall result obtained in this study, where the *26*$\mathrm{MLP}_{16,48,1}$ classifier can be compressed in a 98% using the corresponding *26*$\text{V-NN}_{s(1)}$ Volterra-NN output without significantly loosing recognition capability

Taking into account Table 4, when the *40*$\mathrm{MLP}_{37,37,1}$ is replaced by the *40*$\text{V-NN}_{s(1)}$ classifier, a RR of 96.69% and a SS as high as 97.50% are achieved. If it is considered the model *40*$\mathrm{MLP}_{37,74,1}$, the best RR (96.60%) is obtained if the *40*$\text{V-NN}_{s(3)}$ is chosen, but having the worst SS value (28.75%). But if the best SS value of 98.75% is selected, corresponding to the *40*V-NN$_{s(1)}$, a very high recognition of 95.90% is preserved. As regards the *40*MLP$_{37,111,1}$ classifier, it is possible to state that the best overall result obtained in this case is related to the *40*V-NN$_{s(1)}$ Volterra-NN output, in which the classifier can be compressed in a 99.17%, maintaining almost a 96.51% of RR.

From the three tables analyzed above, it is possible to conclude that an array of MLPs for classification, in several different tasks, can be well-compressed by using an array of Volterra-NN models; and this compression rate can be, in some cases, even higher than 90%. Moreover, in most cases, very high recognition rates are achieved. In fact, the *a*MLP model architecture can be 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.

As can be generally expected, in the case of the Pendigits database, high space savings value are achieved but paying the cost of lower recognition rates. For example, for the three models shown in Table 2, SS values between 95% and 98% aproximately, are related to RR values between 82.46% and 84.20%. Despite these results, it is possible to see that the values of RR are still high. It is important to highlight that, in most of the cases studied, these high compression rates have not to be payed by lower model classification rates. For example, for the *26*MLP$_{16,16,1}$ model in the Letters database, the best RR value is related to the *26*V-NN$_{s(2)}$ output, achieving 92.81%, and having a space saving rate of 50%. But, if the best SS result (94.74%) is chosen in the same model, a RR value of more than 92% is obtained (see Table 3). In the case of the FERET Database, in Table 4, the *40*MLP$_{37,37,1}$ model acchieves an the best RR value of 97.16%, having a space saving rate of 97.50% by selecting the *40*V-NN$_{s(1)}$, which has an RR of 97% approximately.

It is important to clarify that there are same SS values for all of the cases in Tables 2 and 3. As was stated in Subsection 3.2, the SS value is obtained from the number of parameters of each V-NN model, which is based on the number of input parameters for the MLP model. As it is possible to see in Table 1, for both Pendigits and Letters databases, the number of parameters (p) is 16. Thus, independently of the number of classes present in both databases, the SS value is exactly the same when the number of input parameters is the same.

After the analysis of the results, it is possible to think that the best overall results are always obtained in models with high, and possibly innecesary, amount of hidden neurons. But undoubtedly, in the three classification problems here studied, if the model *a*MLP with the same number of hidden neurons than the inputs is replaced with the *a*V-NN$_{s(1)}$ output, the RR values are closed to the original one, and the space saving rates are definetly high.

5 Conclusions and future work

This paper has presented an extended evaluation of a method to obtain a compact representation of an array of MLPs using different-order *a*V-NN model outputs. Three different and well known databases were used to test the *a*V-NN model, proving that it is possible to apply this method to classification problems of increasingly complexity, and covering a wide area of application.

Experimental results demonstrate the capabilities of the proposed *a*V-NN model to compress a solution for different classification problems with very high recognition and space savings rates. It has been proved that this model is capable of solving several classification problems, obtaining almost the same accuracy than three different configurations of *a*MLP classifiers. Furthermore, these *a*MLP classifiers have been significantly compressed into simpler models, allowing to use less parameters to obtain similar results.

Future work involves further application of the proposed method to more complex problems, involving data having an evolution on time. Besides, the V-NN compression capabilities could be tested on ensembles of different neural network models. These different NN classifier topologies should be further studied in order to establish their impact on the compression capabilities offered by the proposed V-NN model approach.

References

1. Buciluǎ, 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, ACM (2006) 535–541
2. Zhang, D., Wangmeng, Z.: Computational intelligence-based biometric technologies. IEEE Computational Intelligence Magazine **2**(2) (May 2007) 26–36
3. Stegmayer, G., Chiotti, O.: Volterra NN-based behavioral model for new wireless communications devices. Neural Computing and Applications **18** (2009) 283–291
4. 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
5. Rubiolo, M., Stegmayer, G., Milone, D.: Compressing a neural network classifier using a volterra-neural network model. In: IEEE International Joint Conference on Neural Networks (IJCNN), Barcelona, Spain (Jul 2010) 1 –7
6. Dzeroski, S., Zenko, B.: Is combining classifiers with stacking better than selecting the best one? Machine Learning **54** (2004) 255–273
7. Rahman, A., Verma, B.: Novel layered clustering-based approach for generating ensemble of classifiers. IEEE Transactions on Neural Networks **22**(5) (2011) 781–792
8. Capello, D., Martinez, C., Milone, D., Stegmayer, G.: Array of multilayer perceptrons with no-class resampling training fr face recognition. Revista Iberoamericana de Inteligencia Artificial **13**(44) (2009) 5–13
9. Volterra, V.: Theory of Functionals and Integral and Integro-Differential Equations. Dover (1959)
10. Marquardt, D.: An algorithm for least-squares estimation of nonlinear parameters. SIAM Journal on Applied Mathematics **11**(2) (1963) 431–441
11. Aitkenhead, M.J., McDonald, A.J.S.: A neural network face recognition system. Engineering Applications of Artificial Intelligence **16**(3) (2003) 167 – 176
12. Bianchini, M., Maggini, M., Sarti, L., Scarselli, F.: Recursive neural networks learn to localize faces. Pattern Recognition Letters **26**(12) (2005) 1885 – 1895
13. Korenberg, M., David, R., Hunter, I., Solomon, J.: Parallel cascade identification and its application to protein family prediction. Journal of Biotechnology **91** (2001) 35–47
14. Haykin, S.: Neural Networks: A Comprehensive Foundation. Prentice-Hall (1999)

15. Madsen, K., Nielsen, H.B., Tingleff, O., Modelling, M.: Imm methods for non-linear least squares problems (2004)
16. Duda, R., Hart, P.: Pattern Classification and Scene Analysis. Wiley (2003)
17. Salomon, D.: Data Compression: The Complete Reference. Springer (2007)