Self-adaptive Topology Neural Network for Online Incremental Learning

Beatriz Pérez-Sánchez, Oscar Fontenla-Romero and Bertha Guijarro-Berdiñas
Department of Computer Science, Faculty of Informatics, University of A Coruña,
Campus de Elviña s/n, 15071, A Coruña, Spain

Keywords: Incremental Learning, Sequential Learning, Forgetting Ability, Adaptive Topology, Vapnik-Chervonenkis Dimension.

Abstract: Many real problems in machine learning are of a dynamic nature. In those cases, the model used for the learning process should work in real time and have the ability to act and react by itself, adjusting its controlling parameters, even its structures, depending on the requirements of the process. In a previous work, the authors proposed an online learning method for two-layer feedforward neural networks that presents two main characteristics. Firstly, it is effective in dynamic environments as well as in stationary contexts. Secondly, it allows incorporating new hidden neurons during learning without losing the knowledge already acquired. In this paper, we extended this previous algorithm including a mechanism to automatically adapt the network topology in accordance with the needs of the learning process. This automatic estimation technique is based on the Vapnik-Chervonenkis dimension. The theoretical basis for the method is given and its performance is illustrated by means of its application to distint system identification problems. The results confirm that the proposed method is able to check whether new hidden units should be added depending on the requirements of the online learning process.

1 INTRODUCTION

In many applications, learning algorithms act in dynamic environments where data flows continuously. In those situations, the algorithms should be able to dynamically adjust to the underlying phenomenon when new knowledge arrives. There are many learning methods and variants for neural networks. Classical batch learning algorithms are not suitable for handling these types of situations. An appropriate approach would be an incremental learning technique that assumes that the information available at any given moment is incomplete and any learned theory is potentially susceptible to changes. These types of methods are able to adapt the previously induced concept model during training. When dealing with neural networks, this adaptation implies changing not only the weights and biases but also the network architecture. The adaptation of network topology depends on the needs of the learning process, as the size of the neural network should fit the number and complexity of the data analyzed. The challenge is to find the correct network size, i.e., the smallest network structure that allows reaching the desired performance specifications. If the network is too small it will not be able to learn the problem well, but if its size is too large it will lead to overfitting and poor generalization performance (Kwok and Yeung, 1997). Therefore, the network topology should be modified only if its capacities are insufficient to satisfy the needs of the process of learning.

Different studies have proposed approaches that modify the network topology as the learning process evolves. Specifically there are two general strategies to achieve it. The former trains a network that is larger than necessary until an acceptable solution is found and then removes hidden units and weights that are not needed. The methods which follow this approach are denominated pruning methods (Reed, 1993). In the second approach the search for the suitable network topology follows another direction, these are the constructive algorithms (Bishop, 1995). These methods start from a small network which later increases its size, adding hidden units and weights, until a satisfactory solution is found. Generally it is considered that the pruning technique presents several drawbacks with respect to the constructive approach (Parekh et al., 2000).

An important aspect to consider is what is the appropriate number of hidden units. This value is very difficult to estimate theoretically; however, different methods have been proposed to dynamically adapt the

Pérez-Sánchez B., Fontenla-Romero O. and Guijarro-Berdiñas B..
Self-adaptive Topology Neural Network for Online Incremental Learning.
DOI: 10.5220/0004811500940101
In *Proceedings of the 6th International Conference on Agents and Artificial Intelligence* (ICAART-2014), pages 94-101
ISBN: 978-989-758-015-4

network structure during the learning process based on several empirical measures. Among others, Ash (Ash, 1989) developed an algorithm for the dynamic creation of nodes where a new hidden neuron is generated when the training error rate is lower than a critical value.

Aylward and Anderson (Aylward and Anderson, 1991) proposed a set of rules based in the error rate, the convergence criterion and the distance to the target error. Other studies researched the application of evolutive algorithms to optimize the number of hidden neurons and the value of the weights (Yao, 1999; Fiesler, 1994). Murata (Murata, 1994) studied the problem of determinating the optimum number of parameters from a statistical point of view. In (Ma and Khorasani, 2003) new hidden units and layers are included incrementally only when they are needed based on monitoring the residual error that cannot be reduced any further by the already existing network. An approach referred to as error minimized extreme learning machine that can add random hidden nodes was included in (Islam et al., 2009). Despite the numerous studies, the authors are not aware of an efficient method for determining the optimal network architecture and nowadays this remains an open problem (Sharma and Chandra, 2010).

In a previous work (Pérez-Sánchez et al., 2013) the authors presented an online learning algorithm for two-layer feedforward neural networks, called OANN, that includes a factor that weights the errors committed in each of the samples. This method is effective in dynamic environments as well as in stationary contexts. This previous work also included the study to check and justify the viability of OANN to work with incremental data structures. In that study, the modification of the topology is manually forced every fixed number of iterations. In this paper, we extend the learning method by including an automatic mechanism to check whether new hidden units should be added depending on the requirements of the online learning process. As a result, in this paper a new learning algorithm denoted as automatic-OANN is presented. This algorithm learns online and it is incremental both with respect to its learning ability and its topology.

The paper is structured as follows. In Section 2 the algorithm is explained. In Section 3, its behavior is illustrated by its application to several time series in order to check its performance in different contexts. In Section 4 the results are discussed and some conclusions are given. Finally, in Section 5 we raise some future lines of research.

2 DESCRIPTION OF THE PROPOSED METHOD

To address the development of an automatic incremental topology two crucial problems have to be solved. First, there is a need to verify whether a learning algorithm can adapt the network topology by incorporating new hidden neurons while maintaining, as much as possible, the knowledge gained in previous stages of learning.This challenge was solved in a previous paper (Pérez-Sánchez et al., 2013) resulting in OANN, which is summarized in section 2.1. Secondly, we must developed a mechanism to know when it is appropriate to change the network topology. This is a new contribution that will be explained in section 2.2. As a result, the new algorithm, automatic-OANN, will be obtained.

2.1 How to Modify the Structure of the Network

Consider the two-layer feedforward neural network in Figure 1 where the inputs are represented as the column vector $\mathbf{x}(s)$, the bias has been included by adding a constant input $x_0 = 1$, and outputs are denoted as $\mathbf{y}(s)$, $s = 1, 2, \ldots, S$ where S is the number of training samples. J and K are the number of outputs and hidden neurons, respectively. Functions $g_1, \ldots, g_K$ and $f_1, \ldots, f_J$ are the nonlinear activation functions of the hidden and output layer, respectively.

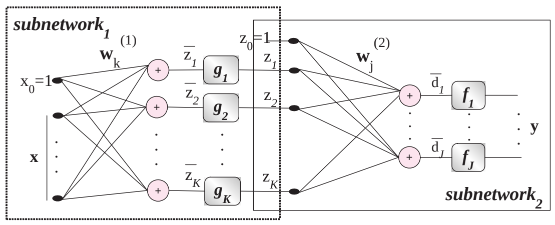

Figure 1: Two-layer feedforward neural network.

This network can be considered as the composition of two one-layer subnetworks. As the desired outputs for each hidden neuron k at the current learning epoch s, $z_k(s)$, are unknown, arbitrary values are employed. These values are obtained in base to a previous initialization of the weights using a standard method, for example Nguyen-Widrow (Nguyen and Widrow, 1990). The desired output of hidden nodes are not revised during the learning process and they are not influenced by the desired output of the whole network. Thus, the training of the first subnetwork is avoided in the learning task. Regarding the second subnetwork, as $d_j(s)$ is the desired output for the j output neuron, that it is always available in a supervised learning, we can use $\bar{d_j}(s) = f_j^{-1}(d_j(s))$ to define the objective function for the j output of subnetwork 2 as the sum of squared errors before the nonlinear activation function f_j,

$$Q_j^{(2)}(s) = \\ h_j(s)\left(f_j'(\bar{d_j}(s))\left({\mathbf{w}_j^{(2)}}^T(s)\mathbf{z}(s) - \bar{d_j}(s)\right)\right)^2 \quad (1)$$

where $j = 1,\ldots,J$, $\mathbf{w}_j^{(2)}(s)$ is the input vector of weights for output neuron j at the instant s and $f_j'(\bar{d_j}(s))$ is a scaling term which weighs the errors (Fontenla-Romero et al., 2010). Moreover, the term $h_j(s)$ is included as forgetting function and it determines the importance of the error at the s_{th} sample. This function establishes the form and the speed of the adaptation to the recent samples in a dynamic context (Martínez-Rego et al., 2011).

The objective function presented in equation 1 is a convex function, whose global optimum can be easily obtained deriving it with respect to the parameters of the network and setting the derivative to zero (Fontenla-Romero et al., 2010). Therefore, we obtain the following system of linear equations,

$$\sum_{k=0}^{K} A_{qk}^{(2)}(s) w_{jk}^{(2)}(s) = b_{qj}^{(2)}(s), \\ q = 0,1,\ldots,K; j = 1,\ldots,J, \quad (2)$$

where

$$A_{qk}^{(2)}(s) = A_{qk}^{(2)}(s-1) + h_j(s) z_k(s) z_q(s) f_j'^2(\bar{d_j}(s)) \quad (3)$$

$$b_{qj}^{(2)}(s) = b_{qj}^{(2)}(s-1) + h_j(s) \bar{d_j}(s) z_q(s) f_j'^2(\bar{d_j}(s)) \quad (4)$$

$A^{(2)}(s-1)$ and $b^{(2)}(s-1)$ being, respectively, the matrices and the vectors that store the coefficients of the system of linear equations employed to calculate the values of the weights of the second layer in previous learning stage. In other words, the coefficients employed to calculate the weights in the actual stage are used further to obtain the weights in the following one. Therefore, this permits handling the earlier knowledge and using it to incrementally approach the optimum value of the weights. Equation 2 can be rewritten using matrix notation as,

$$\mathbf{A}_j^{(2)}(s)\mathbf{w}_j^{(2)}(s) = \mathbf{b}_j^{(2)}(s), \quad j = 1,\ldots,J, \quad (5)$$

where

$$\mathbf{A}_j^{(2)}(s) = \mathbf{A}_j^{(2)}(s-1) + h_j(s)\mathbf{z}(s)\mathbf{z}^T(s) f_j'^2(\bar{d_j}(s))$$

$$\mathbf{b}_j^{(2)}(s) = \mathbf{b}_j^{(2)}(s-1) + h_j(s) f_j^{-1}(d_j(s))\mathbf{z}(s) f_j'^2(\bar{d_j}(s))$$

Finally, from equation 3 the optimal weights for the second subnetwork can be obtained as:

$$\mathbf{w}_j^{(2)}(s) = {\mathbf{A}_j^{(2)}}^{-1}(s)\mathbf{b}_j^{(2)}(s), \forall j \quad (6)$$

As regards the incremental property of the learning algorithm, the network structure can be adapted depending on the requirements of the learning process. Several modifications have to be carried out in order to adapt the current topology to a new one. As can be observed in Figure 2 the increment of hidden neurons affects not only the first subnetwork (its number of output units increases) but also the second subnetwork (the number of its inputs also grows). As mentioned previously, the training of the first subnetwork is avoided in the training task, therefore we only comment on the modifications corresponding to the second subnetwork. Thus, in Figure 2 it can be observed that as the number of hidden neurons grows and consequently, all matrices $\mathbf{A}_j^{(2)}$ and the vectors $\mathbf{b}_j^{(2)}$ $(j = 1,\ldots,J)$, computed previously modify their size. Therefore in order to adapt them, each matrix $\mathbf{A}_j^{(2)}$ is enlarged by including a new row and a new column of zero values, in order to continue the learning process from this point (zero is the null element for the addition). At the same time, each vector $\mathbf{b}_j^{(2)}$ incorporates a new element of zero value. The rest of elements of the matrices and vectors are maintained without variation, this fact allows us to preserve in some way the knowledge acquired previously with the earlier topology. After these modifications, the latter described matrices and vectors of coefficients allow us to obtain the new set of weights for the current topology of the network.

2.2 When to Change the Structure of the Network

In statistical learning theory the Vapnik-Chervonenkis dimension (Vapnik, 1998) is a measure of the capacity of a statistical classification algorithm and it is defined as the cardinality of the largest set of points that the algorithm can shatter. Therefore, this term allows us to predict a probabilistic upper bound on the test error of a classification model based in its complexity. This is an ideal bound that can be calculated according to the training error and the network topology. If the model generalizes properly, the bound indicates the worst test error that the model can obtain. Thus, the bound value establishes the margin to test error and therefore it is possible to use this value to determine whether the current topology is sufficient. In this way, it can be said that an adequate number of hidden units is the one giving the lowest expected risk. Taking into account all these considerations it is established that, with a probability of $1-\eta$ the smallest test error (R, expected risk) is delimited by the following inequality (Vapnik, 1998),

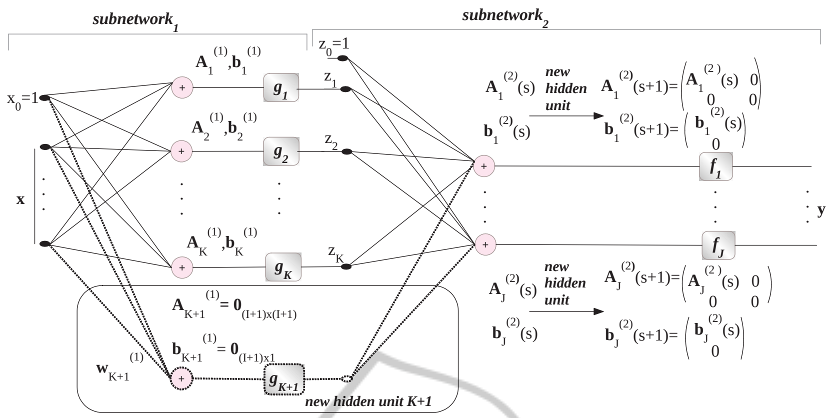

Figure 2: Incremental Topology.

$$R(d_{VC},\theta) \leq \frac{\frac{1}{n}\sum_{i=1}^{n}\left(z_i - \hat{z}_i\left(d_{VC},\theta\right)\right)^2}{\left[1-\sqrt{\frac{d_{VC}\left(ln\left(\frac{n}{d_{VC}}\right)+1\right)-ln\left(\frac{\eta}{4}\right)}{n}}\right]_+} \quad (7)$$

d_{VC} represents the Vapnik-Chervonenkis dimension, θ the ajustable parameters of the learning system, n the number of training patterns, z target outputs, $\hat{z}$ the outputs obtained by the learning system and the notation $[.]_+$ indicates the maximum value between 0 and the term between square brackets. It can be observed that the numerator of equation 7 is the training error and the denominator corresponds with a correction function.

In an informal way, we can say that the generalization ability of a classification model is related with the complexity of the structure. Specifically the greater complexity, the greater value of the Vapnik-Chervonenkis dimension and the lesser generalization ability. In (Baum and Haussler, 1989), authors established certain bounds to a particular network architecture. They indicated an upper bound for a feedforward neural network with M units ($M \geq 2$) and W weights (biases included) that allows to calculate the d_{VC} variable as

$$d_{VC} \leq 2W log_2(eM), \quad (8)$$

e being the basis of the natural numbers. Although equation 8 established an upper bound for the value of the Vapnik-Chervonenkis dimension, this is a valid approach as it allows calculating the error bound in the worst case.

Taking into account all these considerations, an automatic technique is developed to control the neural network growth in the online learning algorithm. The estimation method is based in the ideal bound for the test error calculated thanks to Vapnik-Chervonenkis dimension of the neural network. We obtained a constructive algorithm which adds a new unit to the hidden layer of the network when the obtained error test is higher than the limit established by the ideal bound (equation 8). Finally the new proposed online learning algorithm with incremental topology is detailed in Algorithm 1.

3 EXPERIMENTAL RESULTS

In (Pérez-Sánchez et al., 2013), the OANN was compared to other online algorithms and proved its suitability to work with adaptive structures without significantly degrading its performance. However, the modification of the topology is manually forced every fixed number of iterations. In this paper, we complete the method with an automatic mechanism to control whether new hidden units should be added depending on the requirements of learning process. In this experimental study we want to demonstrate the viability of the automatic adjustment mechanism of the structure. Therefore, taking these results into account, automatic-OANN will only be compared to the previous OANN version (fixed topology) with the following objectives:

- To check if automatic-OANN is able to incorporate hidden units without significant performance degradation with respect to the version without adaptation of the topology.
- To prove that the performance of the automatic-OANN at the end of the learning process, (when it reaches a final network topology), is similar to the performance that would be obtained by employing this final topology from the beginning to the end of learning process.

For this experimental study, we employed distinct system identification problems. Moreover, the behav-

Algorithm 1: Automatic-OANN algorithm with adaptive topology for two-layer feedforward neural networks.

```
Inputs: $\mathbf{x}_s = (x_{1s}, x_{2s}, \ldots x_{Is}); \mathbf{d}_s = (d_{1s}, d_{2s}, \ldots d_{Js});\ s = 1, \ldots, S.$
Initialization Phase
    K = 2, initially the network has two units in its hidden layer
    $\mathbf{A}_j^{(2)}(0) = \mathbf{0}_{(K+1)\times(K+1)}, \quad \mathbf{b}_j^{(2)}(0) = \mathbf{0}_{(K+1)}, \quad \forall j = 1, \ldots, J.$
    Calculate the initial weights, $\mathbf{w}_k^{(1)}(0)$, by means of an initialization method.
For every new sample $s$ ($s = 1, 2, \ldots, S$) and $\forall k = 1, \ldots, K$
    $z_k(s) = g(\mathbf{w}_k^{(1)}(0), \mathbf{x}(s))$
    For each output $j$ of the subnetwork 2 ($j = 1, \ldots, J$),
        $\mathbf{A}_j^{(2)}(s) = \mathbf{A}_j^{(2)}(s-1) + h_j(s)\mathbf{z}(s)\mathbf{z}^T(s) f_j'^2(\bar{d_j}(s)),$
        $\mathbf{b}_j^{(2)}(s) = \mathbf{b}_j^{(2)}(s-1) + h_j(s) f_j^{-1}(d_j(s))\mathbf{z}(s) f_j'^2(\bar{d_j}(s)),$
        Calculate $\mathbf{w}_j^{(2)}(s)$ according to $\mathbf{w}_j^{(2)}(s) = {\mathbf{A}_j^{(2)}}^{-1}(s)\mathbf{b}_j^{(2)}(s)$
    end of For
    Calculate Vapnik-Chervonenkis dimension, $d_{VC}$, using equation 8
    Calculate the error test, $MSE_{Test}$
    Calculate the test error *bound* using equation 7
    If $MSE_{Test} > bound$ then a new hidden unit $K$+1 is added
        $$\mathbf{A}_j^{(2)}(s) =
        \begin{array}{c c}
         & \begin{array}{ccccc} \phantom{\mathbf{A}_j^{(2)}(s-1)} & & & (K+1) & \end{array} \\
        \begin{array}{c} \\ \\ \\ (K+1) \\ \end{array} &
        \left(\begin{array}{ccccc}
        \mathbf{A}_j^{(2)}(s-1) & 0 & \ldots & 0 & 0 \\
        0 & 0 & \ldots & 0 & 0 \\
        \vdots & \vdots & \vdots & \vdots & \vdots \\
        0 & 0 & \ldots & 0 & 0 \\
        0 & 0 & \ldots & 0 & 0
        \end{array}\right),
        \end{array}$$
        $$\mathbf{b}_j^{(2)}(s) =
        \begin{array}{c c}
        \begin{array}{c} \\ \\ (K+1) \\ \end{array} &
        \left(\begin{array}{c}
        \mathbf{b}_j^{(2)}(s-1) \\ 0 \\ \vdots \\ 0
        \end{array}\right),
        \end{array}$$
        For each new connection,
            calculate the weights, $\mathbf{w}_{K+1}^{(1)}(0)$, by means of some initialization method
        end of For
        $K = K + 1$
    end of If,
end of For
```

ior of the learning algorithm will be checked when it operates in both stationary and dynamic environments. All experiments shared the following conditions:

- In all cases, the logistic sigmoid function was employed for hidden neurons, while for output units a linear function was applied as recommended for regression problems (Bishop, 1995).
- The input data set was normalized, with mean=0 and standard deviation=1.
- In order to obtain significant results, five simulations were carried out. Therefore, mean results will be presented in this section. Moreover, in the case of stationary context 10-fold cross validation was applied.
- In all cases an exponential forgetting function, defined as,

  $$h(s) = e^{\mu s}, s = 1, \ldots, S, \tag{9}$$

  was employed, where μ is a positive real parameter that controls the growth of the function and thus the response of the network to changes in the environment. When $\mu = 0$ we obtain a constant function, and therefore all errors have the same weight and the forgetting function has no effect. The value of the μ factor for the forgetting ability was set to 0.01.

3.1 Stationary Contexts

In this section, we consider two stationary time series, Hénon and Mackey-Glass. The goal is to predict the actual sample based in seven previous patterns, in the case of Hénon series and according to the eight previous pattern for Mackey-Glass. A brief explanation of both time series is given as follow.

- *Hénon*. The Hénon map is a dynamic system that presents a chaotic behavior (Hénon, 1976). The map takes a point (x_n, y_n) in the plane and transforms it according to,

  $$x(t+1) = y(t+1) - \alpha x(t)^2$$
  $$y(t+1) = \gamma x(t)$$

  $x(t)$ being the series value at instant t. In this case the parameters values are established as $\alpha = 1.4$ and $\gamma = 0.3$. A total number of 4,000 patterns are generated, 3,000 of them are used as training data set and 1,000 patterns are employed to validate the model.

- *Mackey-Glass*. The second example is the time series of Mackey-Glass, which is generated due to a system with a time delay difference as follows (Mackey and Glass, 1977),

  $$\frac{dx}{dt} = \beta x(t) + \frac{\alpha x(t-\gamma)}{1 + x(t-\gamma)^{10}}$$

$x(t)$ being the value of the time series at the instant t. The system is chaotic to gamma values $\gamma > 16.8$. In this case, the series is generated with the following parameters values $\alpha = 0.2, \beta = -0.1, \gamma = 17$ and it is scaled between [-1,1]. 3,000 samples are generated, 2,250 of them are selected by training and 750 patterns are employed to evaluate the model.

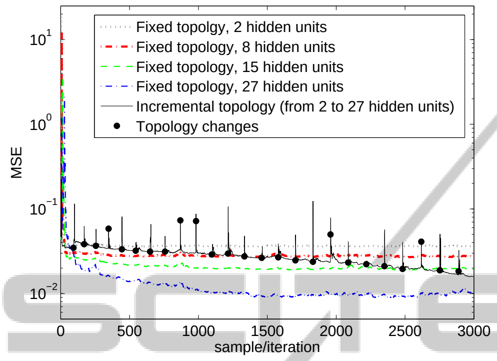

Figure 3: Test error curves for the Hénon time series.

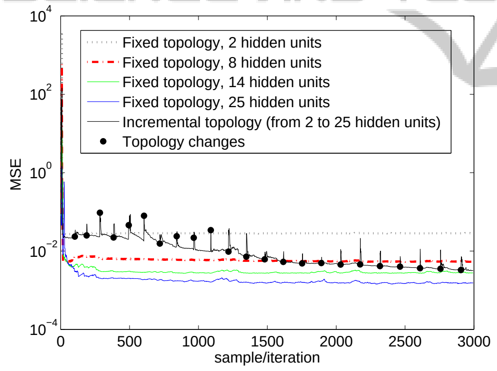

Figure 4: Test error curves for the Mackey-Glass time series.

Figures 3 and 4 show the test MSE curves. In the case of fixed topologies, curves are included for different number of hidden neurons. For readability only the most significant results are shown. As can be observed, the incremental topology algorithm achieves results close to those obtained when the final fixed topology is used from the initial epoch. The results are not exactly the same but it should be considered that the proposed method constructs the topology in an online learning scenario and then only a few samples are available to train the last topology. The points over the curves of incremental topology allow us to know when the error obtained by the network exceeds the bound established for the test error. At that moment, the structure is modified in order to response to needs of the learning process.

3.2 Dynamic Environments

In dynamic environments, the distribution of the data could change over time. Moreover, in these types of environments, changes between contexts can be *abrupt* when the distribution changes quickly or *gradual* if there is a smooth transition between distributions (Gama et al., 2004). In order to check the performance of the proposed method in different situations we have considered both types of changing environments. The aim of these experiments is to check that the proposed method is able to obtain appropriate behavior when it works in dynamic environments.

3.2.1 Artificial Data Set 1

The first data set is formed by 2,400 samples of 4 input random variables which contain values drawn from a normal distribution with zero mean and standard deviation equal to 0.1. To obtain the desired output, first, every input is transformed by a nonlinear function. Specifically, hyperbolic tangent sigmoid, exponential, sine and logarithmic sigmoid functions were applied, respectively, to the first, second, third and fourth output. Finally, the desired output is obtained by a linear mixture of the transformed inputs.

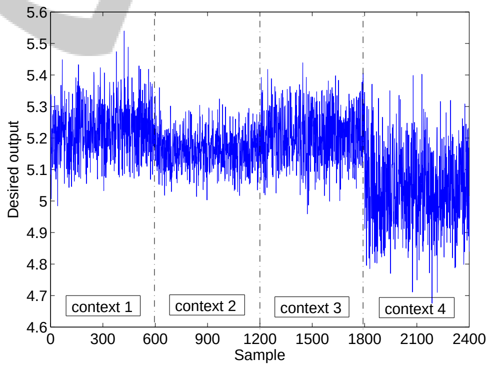

Figure 5: Artificial Data Set 1. Example of the desired output for the training set.

Figure 5 contains the signal employed as desired target during the training process. As can be seen, the signal evolves over time and 4 context changes are generated. Also we created different test sets for each context, so every training sample has associated the test set that represents the context to which it belongs. Thus, for this case we obtain 4 test sets, one for each of the changes in the linear mixture of the process.

Figure 6 shows the test error curves obtained by the method with fixed and automatic incremental topologies. The error shown for each point of the signal is the mean value obtained over test set associated to the current training sample. It can be seen that the

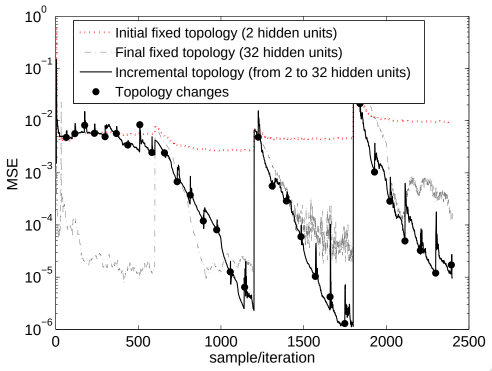

Figure 6: Artificial Data Set 1: Test error curves.

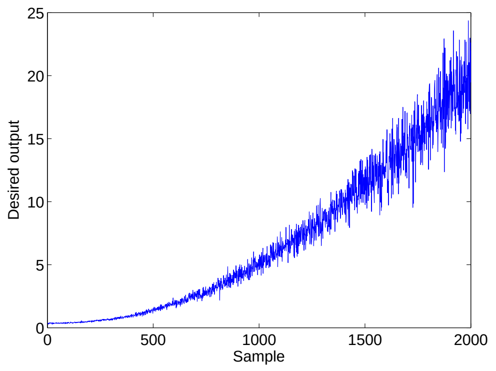

Figure 7: Artificial Data Set 2. Example of the desired output for the training set.

initial fixed topology (two hidden neurons) commits a high error because the structure is not sufficient to make a suitable learning. Regarding the incremental topology, although the results at the beginning of the process are not appropriate (the topology is still small), it can be observed as it performs better than the final fixed topology when there are changes of context.

3.2.2 Artificial Data Set 2

In this case, we generated another artificial data set that presents a *gradual* evolution in each sample of training. The data set is formed by 4 input variables (generated by a normal distribution with zero mean and standard deviation equal to 0.1) and the output is obtained by means of a linear mixture of these. In order to construct the time series we fixed the initial coefficients of the linear mixture vector as

$$a(0) = \begin{bmatrix} 0.5 & 0.2 & 0.7 & 0.8 \end{bmatrix}, \qquad (10)$$

and subsequently these values evolve over time according to the following equation,

$$a_j(s) = a_j(s-1) + \frac{s}{10^{4.7}} logsig(a_j(s-1)), s = 1, \ldots, S$$

where j indicates the component of the mixture vector. Finally, we obtained a set with S equals to 500 training samples. The signal employed as desired output during the training process can be observed in Figure 7. As the distribution presents constant changes we have a different context for each sample thus we have a different test data set for each one.

Figure 8 shows the test error curves obtained by the method with fixed and incremental topologies. As the input signal suffers continuous gradual changes, the error grows constantly. In spite of this, we can observe that the automatic- OANN overcomes the results obtained by the fixed topologies. Moreover, it is worth pointing out that the incorporation of a new unit implies a punctual performance improvement due to

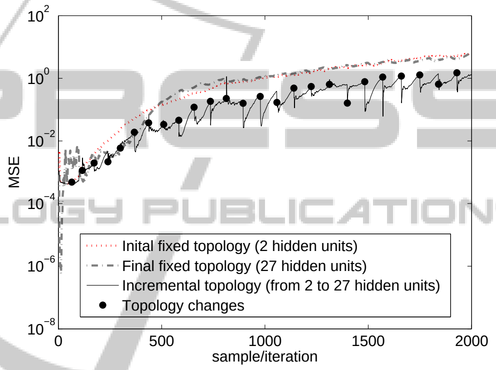

Figure 8: Artificial Data Set 2: Test error curves.

the perturbations included in the matrices that store the earlier information. This is not very relevant in this case as the process changes continuously. Again, the automatic technique allows the initial network structure (two hidden units) to evolve in time adapting its answer in function of the needs of the process.

4 DISCUSSION

In view of the experiments made and the results presented in Section 3 for stationary time series as well as for dynamic sets, we can say that the automatic technique based in the Vapnik-Chervonenkis dimension of a neural network allows us to obtain an estimation of the appropriate size of the network. It is worth mentioning that the proposed method is especially suitable in dynamic environments where the process to learn may change along the time. Taking as a reference the results obtained when a fixed topology is used during the whole learning process, we can check how the developed incremental approach obtains a similar performance without the need to estimate previously the suitable topology to solve the problem. Moreover, the proposed method ensures an appropriate size for the network during the learning process maximizing the available computational resources.

5 CONCLUSIONS AND FUTURE WORK

In this work we have presented an adaptation of the OANN online learning algorithm (Pérez-Sánchez et al., 2013) to modify the network topology according to the needs of the learning process. The network structure begins with the minimum number of hidden neurons and a new unit is added whenever the current topology was not appropriate to satisfy the needs of the process. Moreover, the method allows saving both temporal and spatial resources, an important characteristic when it is necessary to handle a large number of data for training or when the problem is complex and requires a network with a high number of nodes for its resolution.

In spite of these favorable characteristics, there are some aspects that need an in-depth study and will be addressed as future work. A new line of research seems adequate in order to limit the addition of hidden units employing different measures, as for example, the increasing tendency of the errors committed. Also it could be proposed as an modification of the method so as to include some pruning technique to allow, not only the addition, but also the removal of unnecessary hidden units according to the complexity of learning.

ACKNOWLEDGEMENTS

The authors would like to acknowledge support for this work from the Xunta de Galicia (Grant codes CN2011/007 and CN2012/211), the Secretaría de Estado de Investigación of the Spanish Government (Grant code TIN2012-37954), all partially supported by the European Union ERDF.

REFERENCES

Ash, T. (1989). Dynamic node creation in backpropagation networks. *Connection Science*, 1(4):365–375.

Aylward, S. and Anderson, R. (1991). An algorithm for neural network architecture generation. In *AIAA Computing in Aerospace Conference VIII*.

Baum, E. B. and Haussler, D. (1989). What size net gives valid generalization? *Neural Computation*, 1(1):151–160.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press, New York.

Fiesler, E. (1994). Comparative Bibliography of Ontogenic Neural Networks. In *Proccedings of the International Conference on Artificial Neural Networks (ICANN 1994)*, pages 793–796.

Fontenla-Romero, O., Guijarro-Berdiñas, B., Pérez-Sánchez, B., and Alonso-Betanzos, A. (2010). A new convex objective function for the supervised learning of single-layer neural networks. *Pattern Recognition*, 43(5):1984–1992.

Gama, J., Medas, P., Castillo, G., and Rodrigues, P. (2004). Learning with drift detection. *Intelligent Data Analysis*, 8:213–237.

Hénon, M. (1976). A two-dimensional mapping with a strange attractor. *Communications in Mathematical Physics*, 50(1):69–77.

Islam, M., Sattar, A., Amin, F., Yao, X., and Murase, K. (2009). A new adaptive merging and growing algorithm for designing artificial neural networks. *IEEE Transactions on Neural Networks*, 20:1352–1357.

Kwok, T.-Y. and Yeung, D.-Y. (1997). Constructive Algorithms for Structure Learning in FeedForward Neural Networks for Regression Problems. *IEEE Transactions on Neural Networks*, 8(3):630–645.

Ma, L. and Khorasani, K. (2003). A new strategy for adaptively constructing multilayer feedforward neural networks. *Neurocomputing*, 51:361–385.

Mackey, M. and Glass, L. (1977). Oscillation and chaos in physiological control sytems. *Science*, 197(4300):287–289.

Martínez-Rego, D., Pérez-Sánchez, B., Fontenla-Romero, O., and Alonso-Betanzos, A. (2011). A robust incremental learning method for non-stationary environments. *NeuroComputing*, 74(11):1800–1808.

Murata, N. (1994). Network Information Criterion-Determining the number of hidden units for an Artificial Neural Network Model. *IEEE Transactions on Neural Networks*, 5(6):865–872.

Nguyen, D. and Widrow, B. (1990). Improving the learning speed of 2-layer neural networks choosing initial values of the adaptive weights. In *Proccedings of the International Joint Conference on Neural Networks, (IJCNN 1990)*, volume 3, pages 21–26.

Parekh, R., Yang, J., and Honavar, V. (2000). Constructive Neural-Network Learning Algorithms for Pattern Classification.

Pérez-Sánchez, B., Fontenla-Romero, O., Guijarro-Berdiñas, B., and Martínez-Rego, D. (2013). An online learning algorithm for adaptable topologies of neural networks. *Expert Systems with Applications*, 40(18):7294–7304.

Reed, R. (1993). Pruning Algorithms: A Survey. *IEEE Transactions on Neural Networks*, 4:740–747.

Sharma, S. K. and Chandra, P. (2010). Constructive neural networks: A review. *International Journal of Engineering Science and Technology*, 2(12):7847–7855.

Vapnik, V. (1998). *Statistical Learning Theory*. John Wiley & Sons, Inc. New York.

Yao, X. (1999). Evolving Artificial Neural Networks. In *Proceedings of the IEEE*, volume 87, pages 1423–1447.