ADAPTIVE RETRAINING ALGORITHM WITH SHAKEN INITIALIZATION

DUMITRU IULIAN NASTAC[1], IONEL BUJOREL PĂVĂLOIU[2], RODICA TUDUCE[2],
PAUL DAN CRISTEA[3]

Key words: Artificial neural networks, Retraining, Time series, Shaken parameter.

The paper presents new specific aspects that could improve the adaptive retraining procedure of artificial neural networks (ANNs) for time series predictions. Usually, a retraining step starts from proportionally reduced values of the parameters (weights) used in the previous version of the ANN model. This time, the initial configuration of the weights is randomly "shaken" in order to further improve the model. The present results are promising and show a better adaptation of the forecasting system in a nonstationary environment.

1. **INTRODUCTION**

The adaptive retraining technique, based on feedforward artificial neural networks (ANNs), was firstly proposed in 2004 [1] as an enhancement of the forecasting method developed by Iulian Nastac and Adrian Costea for the EUNITE Competition 2003 [2]. Then, it was used in various applications that concern a wide range of data [3–6], including nucleotide genomic signals [7, 8] for special predictions where there are spatial sequences instead of time series. In this paper, a new feature is added to the original model, by reconsidering the initial distribution of the weights before each retraining phase. In literature, other approaches concerning data prediction using artificial intelligence are in many fields such as load forecasting [9], speech signal [10] or technical applications [11]. Since the adaptive retraining technique already proved its worth even in an international competition on data forecasting [2], here we are not focused to compare it with other predictive models, but we want to see if a special modification could enhance

[1] "Politehnica" University of Bucharest, Dept. of Electronics, Telecom. and IT, O.P. 16, C.P. 77, 061112 Bucharest, Romania (corresponding author – phone: + 40-721-485426; e-mail: nastac@ieee.org).
[2] "Politehnica" University of Bucharest, Bio-Medical Engineering Centre, Splaiul Independenței 313, 060042 Bucharest, Romania.
[3] "Politehnica" University of Bucharest, Bio-Medical Engineering Centre, Splaiul Independenței 313, 060042 Bucharest, Romania (corresponding author – phone: +40-745-117062; e-mail: pcristea@dsp.pub.ro).

the model accuracy. The data that are used in this paper are time series from electrical load domain.

The paper is further organized as follows: Section 2 presents the structure of the prediction model and describes aspects of its implementation, Section 3 gives some of the experimental forecasting results, and Section 4 concludes the paper.

2. ALGORITHM DESCRIPTION

The retraining algorithm is well suited in forecasting applications, where there is a huge amount of data. Next, we will provide a brief description of the entire procedure and particularly on the issues that concern the weight initialization, which precede the effective retraining phases.

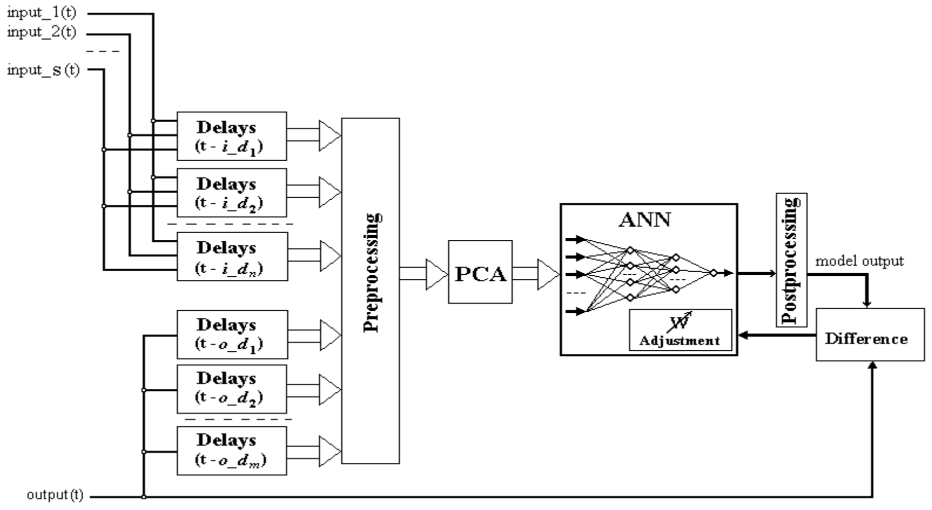

Fig. 1 – General architecture of the forecasting model used during the training process.

The data that are used in this paper refer to the implementation of a short term electric load forecasting system. We expect that time-delays could be encountered among the data in the input matrix. It is well known that feedback control in the presence of time delays can lead to difficulties since it places a limit on the time interval. Moreover, the electric load forecasting is inherently nonstationary, *i.e.*, the distribution of the time series changes over time. Furthermore, gradual changes in the dependency between the input and output variables may occur. Specifically, the recent data points could provide more important information than the distant data points. The model used in our approach is based on a feedforward ANN and it is represented in Fig. 1.

The output at a moment t is determined in terms of the inputs at a set of previous moments (t - i_d_1, …, t - i_d_n), as well as of the outputs at other set of moments (t - o_d_1, …, t - o_d_m). The prediction model is thus defined by two *delay vectors* that comprise the delays:

$$In\_Del = [i\_d_1, i\_d_2, ..., i\_d_n] \quad (1)$$

and

$$Out\_Del = [o\_d_1, o\_d_2, ..., o\_d_m], \quad (2)$$

where, usually, $n > m$ and i_d_n significantly exceeds o_d_m.

Typically, the model uses a huge number of inputs. Fortunately, this number can be reduced drastically by employing a step of principal component analysis (PCA) [12, 13] at the input of the ANN. The preprocessing block, from Fig. 1, normalizes all the input data before operating the PCA, while the postprocessing block executes a denormalization of the ANN output. After PCA we keep about 99.99% of initial information (which it means that the model usually preserves first eigenvectors that are able to reproduce the most valuable data). It is assumed that the PCA block acts as a filter for the outliers data. Finally, the recurrent relation performed by the general model that predicts the output is:

$$y_k(t+1) = F(\ X(t+1-In\_Del(i)),\quad y(t-Out\_Del(j))\,), \quad (3)$$

where X is the input vector, y – the output, with i = 1,.., n and j = 1,.., m.

Based on our previous experience, we have used the scale conjugate gradient (SCG) algorithm [14] as training algorithm. We have applied the early stopping method (validation stop) during the training process to avoid the overfitting phenomenon. The set of available data was split randomly in approximately 85% of the data for training and the rest for validation. In our approach, the validation set also acts as a test set, even if there is a separate test set containing T different moments (T<<V, where V is the initial set of time moments employed for training purpose). As mentioned, the SCG algorithm has been used for training (and then retraining) of the ANN, even if it is not the fastest one. Its great advantage is that it works very efficiently for networks with a large number of weights, does not require large computational memory, has a good convergence and it is very robust. Furthermore, as we always use the validation stop during the training, it is better to avoid algorithms that converge too rapidly, such as Levenberg-Marquardt (LM) [15]. The SCG is better suited for validation stop method. Nevertheless, it is quite easy to replace the SCG algorithm with another one, since the adaptive retraining technique is flexible and independent of the training algorithm.

An iterative procedure was used to find the best architecture of the ANN. The proper number of hidden neurons for each hidden layer (N_{h1} and N_{h2}) has been found by testing several pyramidal ANN architectures (following the rules described in [1]). We have chosen the best model as the one that gave the smallest error between the desired and the simulated output. This error (E_{tot}) was calculated for V data, which included both the training and validation sets.

The process of searching for the best ANN architecture is quite long since it implies imbricated loops, which change: the numbers of neurons for each layer, the starting points for the initial distribution of the weights, and the randomly established training and validation data sets. When the searching is completed, the ANN architecture remains unchanged, and only the network weights are recomputed during each retraining processes.

The novelty introduced by this paper concerns an aspect of the retraining procedure, which allows a fast recalibration of the ANN for the newest acquired data in a nonstationary environment. In our previous approaches, the reference network weights were reduced with a scaling factor γ ($0 < \gamma < 1$) and further used as the initial weights of a new training sequence, with the expectation of a better accuracy. Here we have to use the ANN weights that resulted at the end of the previous step. Usually, we applied successively this technique for nine discrete values of γ (γ = 0.1, 0.2, …, 0.9), keeping the ANN weight distribution that achieved the minimum error as the reference network. We repeated this step several times (by using the parameter N_{rep}), and we randomly reconstructed the training and validation sets for each retraining step. For statistical reason, the value of the parameter γ, which achieve the best result, can be stored at each step. Now we want to enhance this approach by using a shaking parameter (Q_{shake}) of the initial weights. There will be a double loop that will control this process:

```
for $i = 1: N_{rep}$
   for $j = 0.1:0.1:0.9$
            $Net_{new\_weights} = Net_{previous\_weights} \cdot (j + Q_{shake} \cdot (\text{rand}(1) – 0.5));$      (4)
….
```

where Q_{shake} has a small value ($Q_{shake} << j$) in order to not disturb too much the values of the weights after theirs rescaling. Notice that the shaking is made with small positive and negative values around zero.

Furthermore, the shaking of weight positions could be gradually made, from no shaking at $i = 1$ to the maximum shaking for $i = N_{rep}$, as following:

```
for $i = 1: N_{rep}$
   for $j = 0.1:0.1:0.9$
        $Net_{new\_weights} = Net_{previous\_weights} \cdot (j + Q_{shake} \cdot (i-1) \cdot (\text{rand}(1) – 0.5));$      (5)
….
```

Afterwards, we can apply the retraining technique for a shifted interval of data. The parameter *Shift* represents the number of time steps used to shift the entire interval of data from which we establish both the training and validation sets.

At the end of each complete retraining phase, we predict T values of the outputs, in a sequential mode. To estimate the efficiency of the forecasting model, we have computed the error ERR [1, 2], which represents the accuracy of the approximation of the output data within the forecasting horizon of T steps:

$$\mathrm{ERR} = \frac{100}{T}\sum_{p=1}^{T}\frac{\left|O_{Rp} - O_{Fp}\right|}{\left|O_{Rp}\right|}\cdot\frac{T}{T+p}, \tag{6}$$

where T is the number of time steps, O_{Rp} – the real output at step p, and O_{Fp} – the forecasted output at step p. In this formula, the more recent errors in the individual estimations have a larger contribution to the global prediction error (*ERR*) than the ones which are farther distanced in future.

Next, we will see if these modifications could affect the achieved results.

3. **EXPERIMENTAL RESULTS**

The data used in our experiments consist of a set of time series that describe the hourly evolution, during several years, of the power consumption in Romania and also the actual production for different energy sources such as coal, hydro, oil, nuclear, wind, and solar. We performed the steps described in Section 2 for the following combination of the delay vectors: *In_Del* = [1 2 4 7] and *Out_Del* = [0 1 3]. This way by keeping 99.99% of initial information, the PCA block reduced the number of the inputs, before the ANN, from initial 43 lines to only 25. Our goal is to predict the energy consumption in advance with one hour, to allow the system dispatcher to prepare an optimum supply. The inputs of the system are information concerning data from different suppliers of electricity together with time marks (hour, month calendar, year). There are three cases that we want to analyze in this paper: classic retraining (without shaking), retraining with shaking by using the reinitialization from (4), and, finally, retraining with shaking when employing (5).

We carried out the simulations under the following assumptions: V = 8805 timesteps (more than one year) are enough for both training and then for each retraining phase; $T = 24\times7 = 168$ hours (one week) represent the prediction interval; and *Shift* = 168 is the shifting time for the next retraining. It is worth to mention that the values of these parameters can be easily changed, if necessary. Choosing the number of samples for training is still an open issue: it should be not too small, to have enough data (more than five times the number of samples versus the number of weights), but not too large, especially in a nonstationary environment.

The first step, when we choose the ANN architecture, needs about one day of intensive computation. We used a series of multiple imbricate loops, where we vary the number of neurons of each layer. If we extend the searching possibilities, then the computational effort will increase accordingly, but with a greater expectation for a better solution. The most inner loop of this searching algorithm concerns the starting of the training from different uniformly distributed weights of the ANN architecture. We choose to restart five times the last loop for each ANN architecture, and finally select the model that provides the minimum error, using the early stop method. The obtained ANN, with 19 and 14 neurons on its hidden layers, is then used to predict the time horizons in all three cases. This architecture remains unchanged (as numbers of neurons), but it is successively retrained with a one week shifted database. A complete retraining phase lasts about ten to thirty minutes, depending on the available computational resources and their performance.

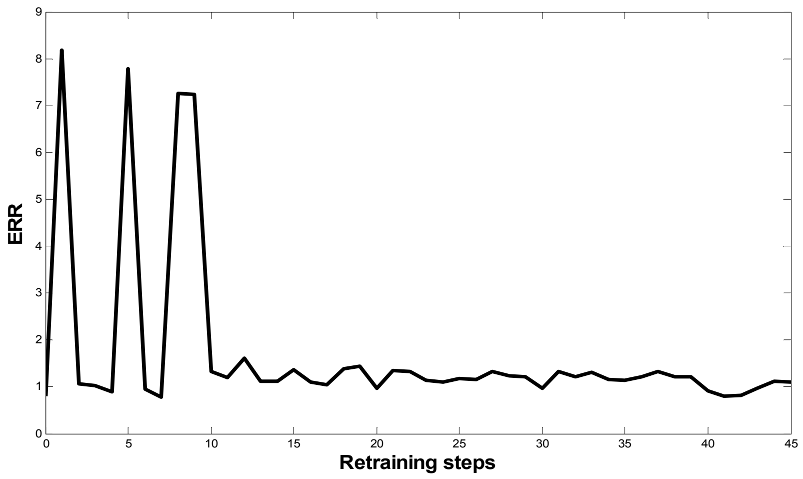

Fig. 2 – ERR trend (Case I) of test sets for the first training and $L = 45$ successive retraining phases.

In Fig. 2 we can see the evolution of the error ERR during $L = 45$ successive retraining phases. One may note that the abscissa represents the numbers of the successive retraining phases and the value 0 is associated with the first training. A descending trend of ERR could be observed when we successively calculated the mean values of the error (see Table I) for the whole interval (of 46 values), then for second half, and, finally, for the last quarter of this interval. Here we obtained the following values: 1.7061, 1.1456 and 1.0831. The descending trend is quite attenuated at the end of the ERR graph.

Behind each point in the graph from Fig. 2 there is another graph (like in Fig. 3), which it is associated with the corresponding forecasting time horizon, in which ERR was computed. The quality of the predictions can be analyzed graphically, by enforcing a tube around the real outputs, given by a function like the following one:

$$f(n) = A + n \cdot q \, . \tag{7}$$

Here, A is an acceptable prediction error, q is an increasing factor and n is the number of predicted timesteps. The predicted output values should lay in the interval $output(n)$ +/– $f(n)$, represented with dotted lines in Fig. 3 that shows the graphs of the energy consumption for the test interval of retraining 35. The real data are represented with thin lines and the neural network output values with thick lines. There is a "tube" (dotted lines) around the real data, given by the function $f(n) = 200+0.0005\cdot n$ (where $n = 1 \ldots 168$). The abscissa shows the index number of the corresponding lines in the database when predictions are performed.

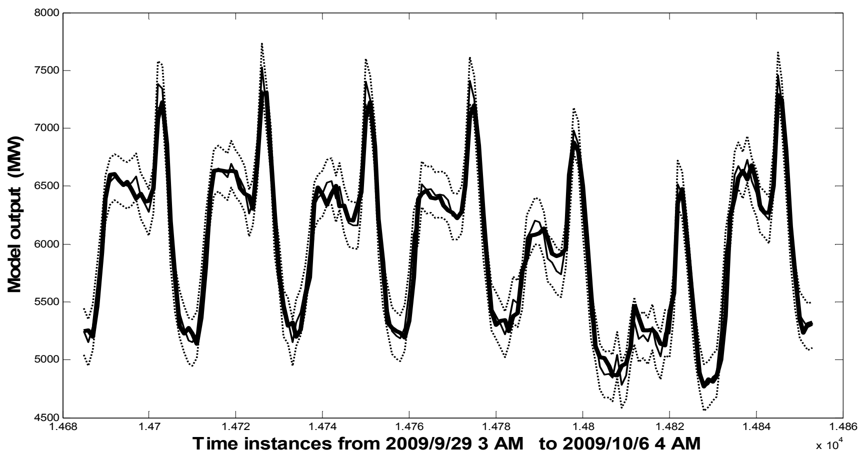

Fig. 3 – Data forecasting for the test interval of retraining 35 (Case I). ERR = 1.1396.

We noticed that, except for a few situations (retrainings:1, 5, 8 and 9), in almost all graph representations of the predictions, the trends were well captured, even better than in the previous example from Fig. 3 (which was preserved just to show the slight differences between the thin and thick lines).

For the cases II and III it was difficult to find a proper value of the parameter Q_{shake}. As an example, when Q_{shake} was set tentatively to 0.001, it resulted to be a too high value, since the ERR increased progresively after successive retrainings (Fig. 4). In this situation, we were forced to stop the retraining process after only 11 succesive steps. By decreasing this parameter at progressively smaller values, we finaly obtained a good result for $Q_{shake} = 10^{-8}$.

Having established a proper value for the shaking parameter, we obtained a visible improvement in the Case II for the retrainings 8 and 9, as we can easily observe in Fig. 5, where a comparison of ERR graphs was performed between the cases I and II.

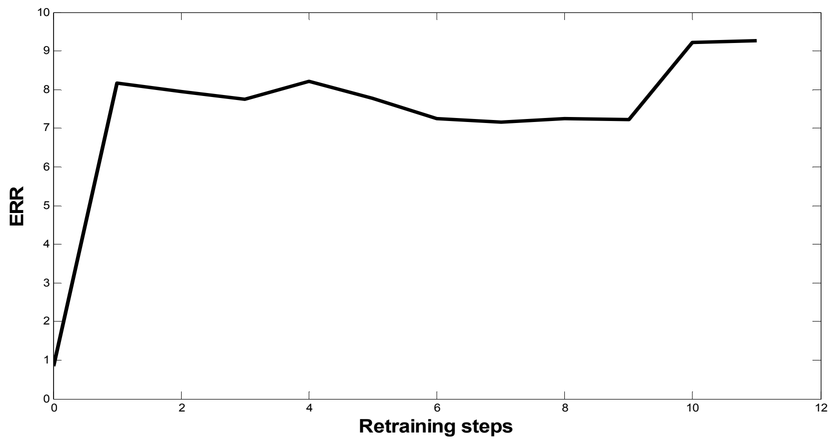

Fig. 4 – ERR trends for Case II when Qshake was inadequately set to 0.001.

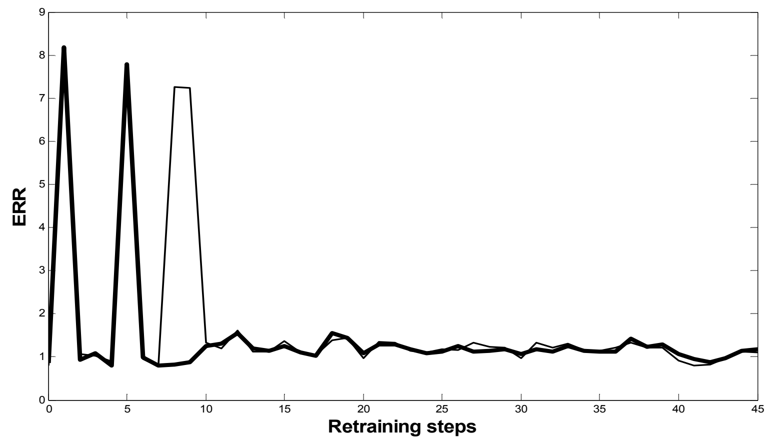

Fig. 5 – ERR trends for Case I (thin line) and Case II (thick line).

A further, less spectacular, improvement could be noticed in Fig. 6, where another comparison of ERR graphs is performed for the cases II and III. Supplementary, a more intuitive comparison of these three cases is shown in Table I, where the mean values for whole interval, second half, and last quarter of the interval (with 46 values) are presented.

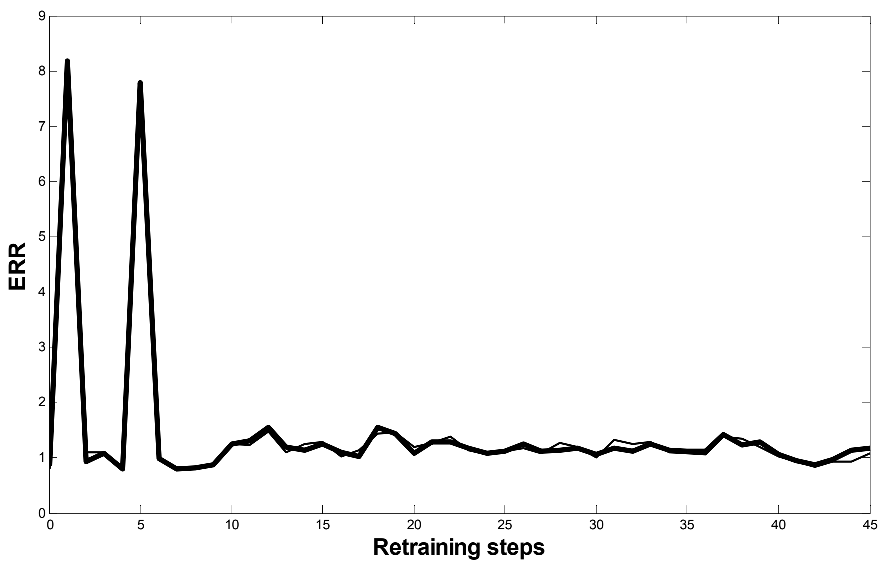

Fig. 6 – ERR trends for Case II (thick line) and Case III (thin line).

Table 1

The mean values of ERR for whole interval, second half, and the last quarter of the interval

	Case I	Case II	Case III
$m_{\text{ERR whole interval}}$	1.706176	1.429903	1.426344
$m_{\text{ERR for second half}}$	1.145609	1.144767	1.134491
$m_{\text{ERR for last quarter}}$	1.083165	1.123910	1.075560

We can easily observe in Table I that even if it is a general improvement of the output accuracy for Case II versus Case I, there is still a slighter higher value of the mean of the last quarter in Case II. The Case III, which used progressive shaking, provided the best results. We have to mention that the parameter N_{rep} was set to 10 in all cases under discussion.

A comparison with classical approaches, such as the naive or ARMA models [15], proved the superiority of the proposed model, since it was the only one able to follow so closely the real shape of the power consumption as we can easily remark for instance in Fig. 3. Another model proposed in [9] was also far from our results. In Fig. 7 we can see a comparison of the adaptive retraining technique with shaken initialization (called ARTIS) versus two standard models, which are an ADALINE model [16] and a Focused Time-Delay Neural Network (FTDNN) [16].

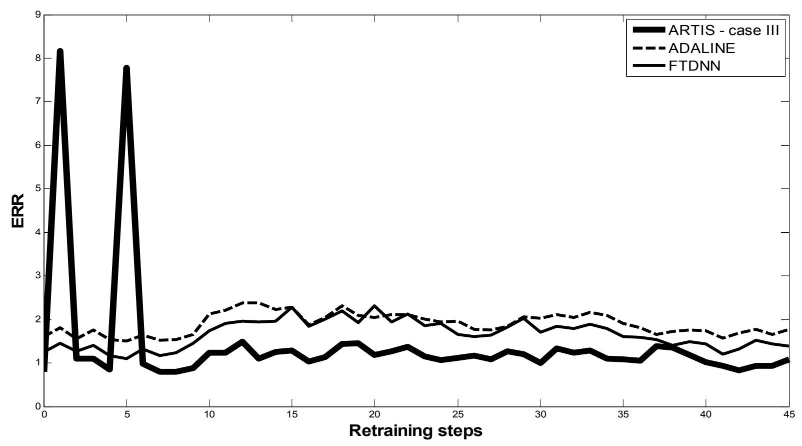

Fig. 7 – Comparison of ERR trends for ARTIS (Case III depicted with thick line) versus ADALINE (dashed line) and FTDNN (thin line).

It is obvious that the retraining technique outperforms the other two models.

4. **CONCLUSIONS**

The outcome of this work confirmed an improvement of the adaptive retraining model when a so called "shaken parameter" was taken into consideration. We are still looking for a better decreasing trend of the ERR during more successive retraining steps. In our previous approaches, by using other data sets, we obtained a visible decreasing trend, even if there we didn't employed the shaken initialization of the ANN's weights. For the data used in this article, all three cases show similar graphs, with the clear difference that the last two cases have stabilized much better the evolution of ERR, even after the first six retrainings.

Some similarities with the Simulated Annealing algorithm can be noticed, but here the method works in somewhat opposite way, since the shaking parameter induces a growing of the disorder among the weights. This parameter must be kept at small values in order to avoid the disturbance of the forecasting system. Further research is necessary to establish its proper range of values, in accordance with each specific set of data.

ACKNOWLEDGEMENTS

We are grateful to Prof. Paul Ulmeanu for the data and helpful discussions. The work was supported by the Sectorial Operational Programme Human

Resources Development (SOP HRD), financed from the European Social Fund and by the Romanian Government under the contract number SOP HRD/89/1.5/S/59758.

Received on 29 August, 2012

REFERENCES

1. D.I. Nastac, *An Adaptive Retraining Technique to Predict the Critical Process Variables*, TUCS Technical Report, No. 616, June 2004, Turku, Finland. Available: http://tucs.fi/publications/view/?pub_id=tNastac04a.
2 J. Strackeljan, K. Lankers, *Eunite Competition 2003 – Prediction of product quality in glass manufacturing process. Summary of the Results and Collection of Reports*, Available on: http://www.eunite.org/eunite/index.htm.
3. D.I. Nastac, P.D. Cristea, *An ANN-PCA Adaptive Forecasting Model*, Proceedings of IEEE-IWSSIP 2012, Vienna, Austria, April 2012, pp. 532-535.
4. D.I. Nastac, P.D. Cristea, *ANN Flexible Forecasting for the Adaptive Monitoring of a Multi-Tube Reactor*, Proceedings of the IEEE-IWSSIP 2007, Maribor, Slovenia, June 2007, pp. 205-208.
5. D.I. Nastac, E. Dobrescu, E. Pelinescu, *Neuro-Adaptive Model for Financial Forecasting*, Romanian Journal of Economic Forecasting, **4**, *3*, pp. 19-41, 2007.
6. D.I. Nastac, N. Tanase, P.D. Cristea, *Smart predictive model for air pollutants*, in Proceedings of GSP 2011 – 2nd International Workshop on Genomic Signal Processing, Bucharest, Romania, June 27-28, 2011, pp. 131-134.
7. P. D. Cristea, Rodica Tuduce, I. Năstac, J. Cornelis, R. Deklerck, M. Andrei, *Signal Representation and Processing of Nucleotide Sequences*, International Journal Functional Informatics and Personalized Medicine, **1**, *3*, pp. 253-268, 2008. Available: http://www.inderscience.com/browse/index.php?journalID=295&year=2008&vol=1&issue=3.
8. P. D. Cristea, *Phase and fractal analysis DNA and reoriented reading frame genomic signals*, Rev. Roum. Sci. Techn. – Électrotechn. et Énerg., **48**, *2-3*, pp. 429-440, 2003.
9. M. O. Popescu, Claudia Laurenţa Popescu, Petruţa Mihai, *Mid Term Load Forecasting Using Analog Neural Networks*, Rev. Roum. Sci. Techn. – Électrotechn. et Énerg., **54**, *2*, pp. 147-156, 2009.
10. R. J. P. de Figueiredo, *A neural-network-based approach to speech signal prediction*, Rev. Roum. Sci. Techn. – Électrotechn. et Énerg., 55, 1, p. 42-48, 2010.
11. C. Oros, C. Rădoi, Adriana Florescu, *Comparison among computational intelligence methods for engine knock detection*, Part 1, Rev. Roum. Sci. Techn. – Électrotechn. et Énerg., **56**, *4*, pp. 418-427, 2011.
12. I. Jolliffe, *Principal component analysis*, 2nd Edition, Springer, NY, 2002.
13. J.E. Jackson, *A user guide to principal components*, John Wiley, New York, 1991.
14. M.F. Moller, *A scaled conjugate gradient algorithm for fast supervised learning*, Neural Networks, **6**, pp. 525-533, 1993.
15. M.T. Hagan, H.B. Demuth, M.H. Beale, *Neural Networks Design*, MA: PWS Publishing, Boston, 1996.
16. M.H. Beale, M.T. Hagan, H.B. Demuth, *Neural Networks Toolbox – User's Guide*, The MathWorks, Inc., 2012. Available: www.mathworks.com/help/pdf_doc/nnet/nnet_ug.pdf.