Vol. 40 No. SIG 2(TOM 1) 情報処理学会論文誌 Feb. 1999
Regular Paper

Energy Function based on Restrictions for Supervised Learning on Feedforward Networks

ALEXANDRA I. CRISTEA† and TOSHIO OKAMOTO†

In this paper we present the construction and usage of an energy function for supervised learning on feedforward networks, based on restrictions. We focus on the mathematical deductions of the energy function, based on the Lyapunov (also called infinite) norm, from error minimization procedures. We will show how the movement equations derived from this energy function improve the learning and generalization capacity of the neural tool in the case of stock exchange (SE) prediction, in the sense of time-series (TS) prediction. We will also show some comparative results of our method and the classical backpropagation (BP) method, obtained by means of the T (Theill) test and the correlation computation. The verification of the proposed energy function is done through computer simulation.

1. Introduction and Delimitation

Optimizations of Neural Networks (NN) have been performed in multiple ways. Algorithmic improvements are the traditional concerns of the NN community (see e.g. 1), 7), 12), 20)). Among these, the most popular learning algorithms for feedforward networks try to minimize a certain cost function, which depends on an adjustable weight vector. In a statistical context, this is very often equivalent to the maximization of the likelihood of sample data of an input-output relation with respect to this weight vector[19]. A statistical and sound theoretical argumentation of learning from data can be found in 4). Another line of research is based on enhancing NNs' performance by using parallelization techniques and parallel machines[6),9),18)]. By now it is obvious that a combined approach is preferable[16)].

In the present paper we will confine ourself to the traditional optimization methods, keeping in mind that this represents only a part of the complete system (see 5), 6) for details). In fact, we will focus here on the mathematical deductions of an error metric for feedforward networks (also known as *multi-layer perceptrons*), based on the Lyapunov (also called infinite) norm and on error minimization procedures, in the sense of the "probably approximately correct" (PAC) model[3)]. This error function is an extension of the quadratic penalty function, and is applicable to SE TS forecasting. We will show how the effective use of this norm reduces the sensitivity of the network to minor errors.

Benchmarking of our system is done by comparison with the classical BP method, obtained by means of the T (Theill) test and the correlation computation.

The paper is structured as follows. First (section 2) we will describe the two step error function deductions for a general case, regardless of the actual dimensions of the feedforward net used for implementation. Then we will describe the frame of the stock exchange prediction problem, in section 3. In the following, we will build a specific network for the SE problem, following some general guidelines (section 4). Then, from the obtained error function, we will design the learning algorithm in section 5 and explain why we can use a BP based development. The implementation details, simulations and results will be presented in following sections (6,7) and conclusions will be drawn.

2. Deductions of the energy function

For time series prediction we can rewrite the prediction problem in a standard *Energy-minimization* form. We used a Lyapunov based development of an energy function, but while the Lyapunov☆ method is based on vector derivation, we used *weights' change*, therefore matrix derivation, as follows.

Let r be the error vector, $r_j = y_j - d_j; j = 1,..,n$, where $y_j = f(\sum_{i=1}^{m} w_{ij}x_i)$, is the actual, calculated output and d_j the desired output for neuron j in the output layer; x_i is one of the

† Graduate School of Information Systems, University of Electro-Communication

☆ Infinite Norm

inputs, with $i = 1, ..., m$, w_{ij} are the weights, m the number of inputs, n the number of outputs, f the external activation function. Then we can build an energy function☆ as follows.

$$E(w) = max_{j=1,..,n}\{|r_j(w)|\} \quad (1)$$

Finding the minimum of this error/energy function means finding

$$"min_{w\in\Re^{mxn}} max_{j=1,..,n}\{|r_j(w)|\}", \quad (2)$$

or, if we define w_0 as being the maximum error radius,

$$w_0 \geq |r_j(w)|; j = 1, .., n; w_0 \geq 0, \quad (3)$$

the minimization of the error becomes equivalent to the minimization of w_0.

Condition 3 can be divided into 2 inequalities (restrictions):

$$g_{j1}(w) = w_0 + r_j(w) \geq 0;$$
$$g_{j2}(w) = w_0 - r_j(w) \geq 0;$$
$$w_0 \geq 0; j = 1, .., n, \quad (4)$$

where g_{j1}, g_{j2} are notations.

In order to achieve a more robust representation, we develop a new energy function by applying the standard quadratic penalties on the newly constructed restriction inequalities, and obtain:

$$E(w) = \nu w_0 + \frac{k}{2}\sum_{j=1}^{n}\{([g_{j_1}(w)]_-)^2 + +([g_{j_2}(w)]_-)^2\}, \quad (5)$$

with $[y]_- = min(0, y); \nu > 0, k > 0$, constants. Equation 5 states that the energy of the system depends only on the maximum error radius w_0, as long as the restrictions in 4 are satisfied. If not, the terms g_{j1}, g_{j2} also contribute to the error function.

Let $S(y) = (1/y) * [y]_-$, therefore:

$$S(y) = \begin{cases} 0, & \text{for } y \leq 0 \\ 1, & \text{rest} \end{cases}$$

$S_{j1} = S(w_0 + r_j)$ and $S_{j2} = S(w_0 - r_j)$, then apply Gradient Descent reasoning. The weight changing equations will look as follows:

$$\frac{dw_{i,j}}{dt} = -\frac{\delta}{\delta w_{ij}}(E(w)) = \quad (6)$$
$$-kx_i\sum_{j=1}^{n}\{(w_0 + r_j)\,S_{j1} - (w_0 - r_j)\,S_{j2}\}$$

Similarly, we obtain a movement equation for the maximum error radius, as follows:

☆ error, or cost function

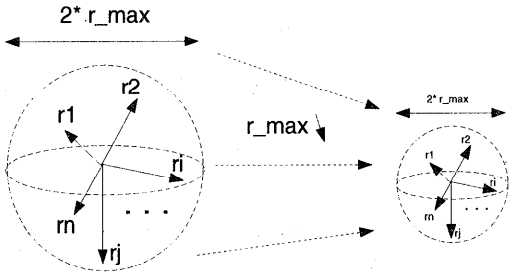

Fig. 1 Delimitation through the external (maximum) error sphere

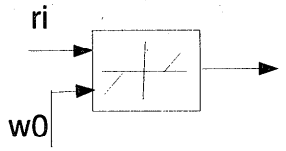

Fig. 2 Dead zone delimiter for dwij/dt

$$\frac{dw_0}{dt} = -\frac{\delta}{\delta w_0}(E(w)) = \quad (7)$$
$$-\nu - k\sum_{j=1}^{n}\{(w_0 + r_j)\,S_{j1} + (w_0 - r_j)\,S_{j2}\}$$

The basic idea of this approach is that w_0 not only controls all the other errors, but also ensures convergence by decreasing monotonously. If the maximum error(ME) radius will decrease, so will all other errors contained in the error sphere. Which is more, the decreasing step is set by the decreasing step of the ME, that can be adjusted in order to assure convergence (**Fig. 1**). In the figure, r_max is ME, $r_j, with j = 1, .., n$ has the same meaning as in the above equations, and the convergence takes place when shifting from the sphere on the left side, to the one on the right side of the picture.

We can rewrite the components that are added in the sums of equations 6,7 as:

for $\frac{dw_{ij}}{dt}$ computation :

$$(w_0 + r_j(w))S_{j1} - (w_0 - r_j(w))S_{j2} =$$
$$= \begin{cases} w_0 + r_j(w), & \text{for } r_j < -w_0 \\ 0, & r_j \in [-w_0, w_0) \\ -w_0 + r_j(w), & r_j \geq w_0 \end{cases} \quad (8)$$

for $\frac{dw_0}{dt}$ computation :

$$(w_0 + r_j(w))S_{j1} + (w_0 - r_j(w))S_{j2} =$$
$$= \begin{cases} w_0 + r_j(w), & \text{for } r_j < -w_0 \\ 0, & r_j \in [-w_0, w_0) \\ w_0 - r_j(w), & r_j \geq w_0 \end{cases} \quad (9)$$

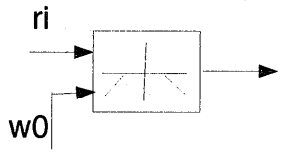

Fig. 3 Dead zone delimiter for dw0/dt

Both movement equations are functions of r_j and linear ramps of absolute slope 1 and dead-zone $[-w_0, w_0]$. We interpret the obtained dead-zone delimiters as follows.

They both compare the maximal error with the other errors. The dead-zone linear ramp obtained for w_{ij} in eq.8 compares each error to the maximum error radius w_0 and establishes how the weights should change in order to bring the errors inside the range of the maximal error, by prohibiting that other errors become greater than the maximal one. Eq.9 for w_0 compares the maximum error to the other errors, and tries to shrink the sphere (see Fig. 1), i.e., to adjust the maximum error w_0 in order to become closer to the other errors. That is why w_0 can only become smaller (only negative domain in **Fig. 3**), but r_j must correct both deviations in the negative and positive domain, so it can change both ways(**Fig. 2**).

3. Stock Exchange Forecasting

Development of prediction tools is, according to 14), one of the "basic subjects in science". Moreover, forecasting of time-series is a challenging task that attempts to find the rule/mechanism behind data generation. However, in such cases, there is always the question of the predictability of the data, in other words, if the data is *fully deterministic(FD)*, therefore predictable, *fully random*, therefore unpredictable, or, as in most cases, somewhere in-between.

Most of the economical functions can be represented as TS. In this work we study the TS of SE events, that can be forecasted with certain accuracy with NN ([2),8),16)]). Still, according to a well-known financial law called the *efficient market hypothesis*, as soon as a predictable regularity appears on the market, it is immediately exploited by all the market actors and therefore tends to disappear. In reality, the law would be true only in a transparent market where the information would be perfectly transmitted and equally available for everybody. But as this is not the case on the real market, where information is often private or becomes public with a delay, and also as the participants at a stock trading possess different forecasting tools and act driven by their own private theories, it can be presumed that the process is not influenced only by one theory☆. The present research is done in view of this point.

There are three ways in which a NN can make predictions upon a TS:

1) the NN can *find rules/functions*,

2) it can make *predictions in a fixed window of time-values*, of which some are recent past states and some are future values, and

3)it can *learn from any number of past values and predict any number of future values.*

From these we selected the second type, for the following reasons: First, because if there might be some deterministic features in SE data, there will surely never be enough to construct a function of it, or, in other words, because SE data are not FD. On the other hand, if we select the third option, although it might increase the generality degree of the learning process, the input data may loose some of the connection information☆☆.

The inputs can be:

1) the *predictable data* or

2) the *predictable data and other economical influence factors* or

3) *only economical influence factors.*

Further on we will call the predictable data the *"Mathematical Data(MD)"*, because it reflects the mathematical relation of the past terms of a series with its future terms, and we will call the economical influence factors the *Economical Data*, as being other data from the economical scene, with only indirect connection to the prediction data. Here we selected again the second type, for the obvious reason that it's the richest in information, although a lot of the training takes place on simple MD (case 1).

4. The Network

The designed net is based on the *Lyapunov Gradient Descent NN*. The simplest net construction would be an 1 layer NN (1 input layer, 1 output layer) which isn't enough for the complexity of the analysed data. The next step is a 2 layer NN. Previous results (see for e.g. 10)

☆ this is a simplified hypothesis, because, for instance, the market is influenced by the long term windows used by most of the predictors, that usually take fix values, as 25/50 days, etc.

☆☆ as being adjacent data

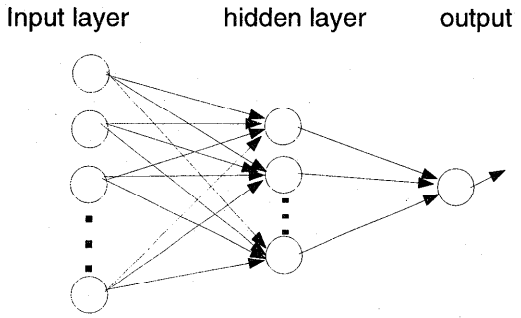

Fig. 4 The net layout

showed that a 2 layer net is enough (for a discussion upon network parameter selection, see 17). Therefore, if the Lyapunov Gradient Descent NN idea shows itself to be successful, it should at least be able to work on a net of a similar dimensional complexity.

Figure 4 shows the used net, containing a m-dimensional input layer, a hidden layer of dimension p and a 1-dimensional output layer. Each layer is fully connected with the following layer (feedforward network). We consider 1 step ahead prediction, and for n step prediction (long-term prediction) we reinforce the system's output as a feedback to the input (from $x_1, x_2, ..., x_m$ we predict y^t, from $x_2, x_3, ..., x_m, y^t$, we predict y^{t+1}, a.s.o.)☆. We have also worked with predictions like $x_1, x_2, ..., x_m$ to $y_1, y_2, ..., y_n$ but the results are poorer.

Evidently, the previously deducted weight changes (eq. 6, 7) cannot work for a 2-layer NN directly, so we have to take into consideration some error backpropagation.

5. Backpropagation of Error

We have deducted the weights' changing equation in section 2, and we have established the net design in section 4, subsequently we need to see how the weight changes will propagate for the different layers of the network.

The previously calculated weight changes can be used for the external layer. For the hidden layer, we need a backpropagated error computation☆☆, because we cannot compute a direct error, as in the case of the external layer.

The weight changes for the output layer can be deducted directly as being the ones in eq. 6, 7, with the only difference of letting x_i be h_i, the hidden layer values.

By making an analogy to the backpropagation mechanisms, the weight $(v_{k,i})$ changes before the hidden layer must be:

$$\frac{dv_{k,i}}{dt} = -kx_k f'\left(\sum_{k=1}^{m+1} v_{k,i}x[k] + v_{0,i}\right) * \sum_{j=1}^{n} w_{i,j}\left\{(w_0 + r_j)\, S_{j1} - (w_0 - r_j)\, S_{j2}\right\},$$
$$k = 1, .., m; i = 1, .., p; \quad (10)$$

and for the biases $(v_{0,i})$:

$$\frac{dv_{0,i}}{dt} = -kf'\left(\sum_{k=1}^{m+1} v_{k,i}x[k] + v_{0,i}\right) * \sum_{j=1}^{n} w_{i,j}\left\{(w_0 + r_j)\, S_{j1} - (w_0 - r_j)\, S_{j2}\right\},$$
$$i = 1, .., p; \quad (11)$$

with f being, for instance, the sigmoidal function, $f(x) = \frac{1}{e^{\alpha x}+1}, \alpha \in \Re$, p the dimension of the hidden layer and the rest of the notations the same as in section 2.

6. Implementation notes and simulation details

The system can perform **training** (learning) through weight computing, *forecasting* with a 2-layer feedforward NN and can also serve as a *user-interface*. These 3 processes are implemented as independent programs. The communication between these processes is assured by the common resources.

The designed program provides a full help support at each step, both for current state explanations and for advice about future possible steps.

The *prediction error* is displayed in percentage to the maximum value (price) that occured in the given time-period in an error dispersion window (see **Fig. 6**). Information about the *Mean Square Error (MSE)* is also available to the user, although it is less informative in respect to local errors.

6.1 The Data

The data consists of some real-world data from 10), as well as some user-designed, synthetic series for testing (see indications about the usage of real, realistic and synthetic data for algorithm benchmarking in 15)). As only graphical charts were available, the reading procedure for the real data may include some reading errors of the above mentioned real-market data, but that shouldn't harm the generaliza-

☆ t is a time variable
☆☆ as in standard backpropagation

tion power of the resulting network. The data are divided into two subsets: a *training set*, that contains pairs of { *[old data], [new data],* } and that is fed into the system in order to learn the correspondence between these pairs, and a second set, the **test set**, of similar data, out of which only the *[old data]* are fed into the system, in order to check the system's capabilities. The feeding of the first set into the system is the *Supervised Learning* part, the second part is the *Testing*, and it contains no learning whatsoever (therefore, no weights changing).

The *data* are values (prices on SE market) over any desired period of days and are represented by the user-interface program on a screen on the 0y axis (see **Fig. 5**). The 0x axis represents the time. The data are scaled for a better representation and for a uniformization of input data for the learning algorithm, so they range from [0-min; 1-max]. The known, predicted and true future values (if available) are represented on the same screen.

Fictional (synthetic) values with a high degree of irregularity were also used. The considered time-period is usually a month, of which about 20 values are known, and 10 to be predicted. This is not a fixed rule, though, as both learning and display functions can handle any amount of input and/or output data, by using dynamic memory allocation functions.

6.2 Convergence Considerations

In 11) a method for convergence improvement is suggested: *Weight Decay*, also related to the one known in the literature under the name *Momentum*. A method which is similar, but much easier to implement is the *step-changing method*. The idea is that, if the convergence is following a certain path for a given time-period, the step should be increased:

$$step = 2 * step,$$

and if the computation tends to oscillate around a value, the step should be decreased, in order to allow the algorithm to reach the so called "valleys of lower potential":

$$step = .5 * step.$$

This procedure requires less memory than the weight decay method (where all the last changes of the weights have to be stored) and only requires an integer vector of directions, with values of $\{ -1, 0, +1 \}$. If the last weight-change had a negative sign, the new value of the direction vector will be [-1], if it was positive, [+1], and if there was no change, [0].

Then, "going in the same direction" means having the direction vector [-1] and a new negative weights change value, or having the direction vector [+1] and a new positive weights change value. Obviously, "going in opposite directions" are the pairs ([-1],positive change) and ([+1],negative change). There appears to be a little problem for [0] values, but the simplest way to avoid them is not to do any change in that case.

This simple procedure has two consequences we observed: if the initial (random) weights are far from the desired ones, then this procedure tends to determine a shorter convergence time; on the contrary, if the starting weights are close to the final ones, the net has a slight tendency of oscillation, till it finds the correct combination, because of the rougher approach of bigger steps for a short time period. A comparative training has been done in order to observe this phenomena, and the results are presented in the following.

7. Results

First we will introduce the elements of result display and analysis:

System Learning Display: We present here a learning example of an equal share of 10 input data and 10 outputs (Fig. 5). That means that out of 10 input daily SE values, 10 output daily values were to be learned. The weight matrix dimension therefore had a 10x10=100 members dimension. The continuous, zig-zag line on the left side of the graphical chart represents the past (or input) data, while on the right side, the desired outputs☆ and the prediction can be seen. The prediction is normally displayed by a dotted line, while the desired outputs are displayed by a continuous line☆☆ but here, with "0" error, the two outputs overlap, and therefore, a single line is visible. The program here displays the learning of the correspondence of 10 past prices with 10 future prices. The outputs are scaled from 0 to 1 (Oy axis), and the time (days) is represented on 0x.

Error display: The error display is shown, for a better understanding, in a separate, two-dimensional error-display window (Fig. 5 upper right corner, Fig. 6).☆☆☆ The display of error was designed so that a qualitative reading can

☆ as in *Supervised Learning*

☆☆ just like the inputs

☆☆☆ The **prediction error** is displayed in percentage to the maximum value(price) that occured in the given time-period.

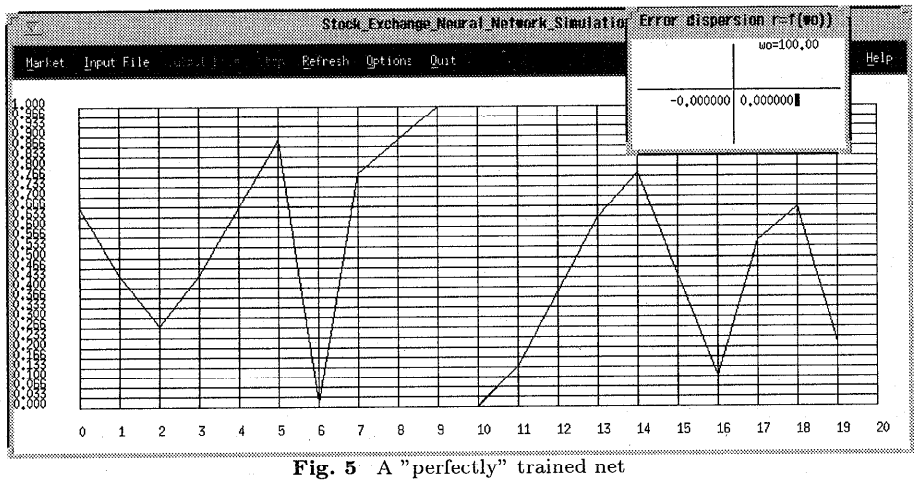

Fig. 5 A "perfectly" trained net

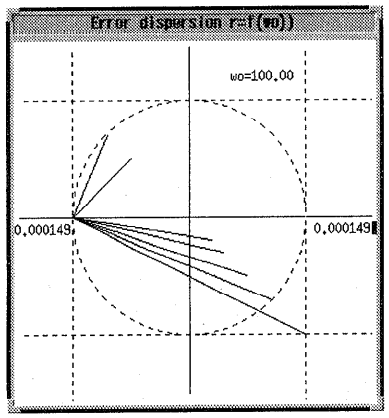

Fig. 6 Error display window

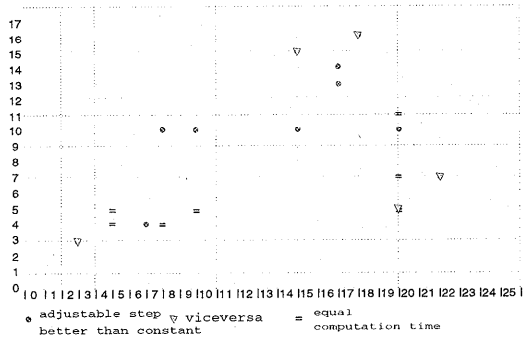

Fig. 7 Variable and constant step; the influence on the convergence.

be done easily. For this reason, on both vertical and horizontal axis of the error-window, individual errors from different times are represented. Errors can occur if the desired values and the predicted output values don't overlap. The dimension of the error vector is the same as the dimension of the output vector (here, 10). Of interest are: first, the display of the maximum error of the whole interval - represented in the window by the exterior square - and then, the other errors, with lower values - represented by the lines starting in the left corner of the picture and ending at the intersection with the second diagonal, on which all the errors (including the maximum) are represented. In this way, the error structure can be understood at a single glance.

These were the elements of the result display. We used them to view first the behaviour of a regular BP net, then the Lyapunov net training. One of the similarities that appears is that both methods tend to increase the weights during learning (if no momentum term is added). As a convergence difference we mention that the growth-rate is larger in the BP case, while our method seems to show a smoother pattern.

7.1 Adjustable step vs. constant step

We present here some results concerning the adjustable convergence step versus the constant computational step. These computations were performed on a small quantity of irregular, user-designed data (**Fig. 7**). The 0x-axis shows the number of input values, the 0y-axis shows the number of output-values. The convergence step number would be on a 3rd axis, which is not represented. The step on both axis is 1 day's time.

As can be seen in Fig. 7, out of 300 random starting points, that we used for training with constant step and then with a variable step, there were 165 input-output combinations for which the adjustable step was better, and only 81 where the constant one showed better con-

	correlation of set	perf. compared to random walk
BKP		
trained	0.951364	0.447806
test1	0.226734	0.853706
test2	0.308115	0.810235
modified Lyapunov		
trained	0.960783	0.356597
test1	0.480251	0.871593
test2	0.511318	0.795510

Fig. 8 Backpropagation and Lyapunov method's behaviour regarding the algebraic indicators.

vergence. There were also 54 cases in which the two types had similar convergence times. The most frequently used pattern was the [20,10] one, so, 20 input values, 10 outputs (together adding up to a month=30 days). It is interesting that, for this particular pattern, all test examples showed the adjustable step to be the quicker converging one.

7.2 Algebraic indicator comparison

We used two coefficients for comparison:

- the **correlation** coefficient, that measures the linear correlation between the forecasted value y_j and the real value b_j:

$$R : \frac{\sum_{j=1}^{n} (b_j - b')(y_j - y')}{\sqrt{\sum_{j=1}^{n} (b_j - b')^2} \sqrt{\sum_{j=1}^{n} (y_j - y')^2}} \quad (12)$$

with $x' = \frac{1}{n} \sum_{i=1}^{n} x_i$ and n the number of observations (on the validation set).

- the **t test** or **Theill** coefficient, that measures the out-performance of the NN over the random walk (the predictor that estimates the future value as $y_{j+1} = b_j + Eps_j$, i.e. the actual value plus a white noise):

$$T_r : \frac{\sqrt{\sum_{j=1}^{n} (y_j - b_j)^2}}{\sqrt{\sum_{j=1}^{n} (b_j - b_{j-1})^2}} \quad (13)$$

If $T_r \leq 1$, the NN predictor is better than the random walk predictor.

Some average results over a number of 100 learning experiments (with error margin of 0.1, and convergence time interval of 5-10 min.) are presented in **Fig. 8**.

We observed two methods: our method, versus the classical BP method. As can be seen in fig.8, we have tested these coefficients on the trained set of data, and also on two test sets. The Theill coefficient is best (lowest), as expected, in both cases, on the already trained set. The correlation of prediction and real value also are the best for this case (closer to 1). Except for the performance measure for test1 set (when compared to random walk) that has a higher value than the BP case (0.87.. when compared to BP's 0.85..), the rest of the values show that our method is superior: for the correlation coefficient, the values are closer to 1, so there is a higher correlation between prediction and real values, and for the comparison with the random walk, the Theill coefficient has lower values for the method we proposed, when compared to the BP method.

Furthermore, one of the similarities that appear is that both methods tend to increase the weights during learning (if no momentum term is added). However, the growth-rate is larger in the BP case, while our method seems to show a smoother pattern.

These results experimentally validate our theoretical energy function deduction and show that it outperforms the classical BP algorithm (see 13) for experimental validation suggestions for NN algorithms).

8. Conclusions

In this paper we described the development and usage of an energy function that reduces the sensitivity of the network in respect to minor errors, based on the Lyapunov infinite norm concept. From this model, a NN was designed and a SE forecasting tool was constructed, using the TS behaviour of SE.

We compared the network constructed based on this energy function with a network using the classical Backpropagation method and showed the differences between the two.

The proposed energy function is less sensitive to minor errors when compared with the standard LMS-function of the backpropagation method. We proved this by using indicator comparison, and showed how our method is superior to the traditional BP for SE forecasting.

For further work we intend to extend the system and find out what other problems it's applicable to. In constructing the new function we exploited the TS structure of SE events, therefore we expect that it can be useful for other problems which deal with TS forecasting.

References

1) Amari, S., et al.: *Asymptotic Statistical Theory of Overtraining and Cross-Validation*, RIKEN, Japan, METR 95-06 (1995).

2) Ankenbrand, T. and Tomassini, M.: *Multivariate time series modeling of financial markets with artificial neural networks*, ANN and GA, Springer Verlag, Wien, pp. 257–260 (1995).
3) Anthony, M.: Probabilistic Analysis of Learning in Artificial Neural Networks: The PAC Model and its Variants, *Neural Computing Surveys*, Vol. 1, pp. 1–47 (1997).
4) Cherkassky, V. et al.: *Learning from Data: Concepts, Theory and Methods*, John Wiley & Sons (1998).
5) Cristea, A. and Okamoto, T.: NN for Stock Exchange prediction; a Lyapunov based training, *Proc. Int'l Conf. on Computational Intelligence and Multimedia Applications '98* , World Scientific, pp. 416–421 (1998).
6) Cristea, A. and Okamoto, T.: A Parallelization Method for NNs with Weak Connection Design, *Proc. Int'l Symp. on High Performance Computing '97*, Springer, pp. 397–404 (1997).
7) Dasgupta, D. and McGregor, D. R.: Designing Application-Specific Neural Networks using the Structured Genetic Algorithm, *Proc. Combination of Genetic Algorithms and Neural Networks* , IEEE CS Press, pp. 87–96 (1992).
8) Gas, B. and Natowicz, R.: Unsupervised learning of temporal sequences by neural networks, *ANN and GA*, Springer Verlag, Wien, pp. 253–256 (1993).
9) Jabri, M., Tinker, E. and Leerink, L.: MUME - a multi-modules multi-algorithms NN, http://www.sedal.usyd.edu.au/mume/mume.html
10) Komo, D., et al.: Neural Network Technology for Stock Market Index Prediction, *Proc. IEEE Int'l Symp. on Neural Networks, Image and Speech Processing '94* , pp. 543–546 (1994).
11) Krogh, A. and Hertz, J.A.: A Simple Weight Decay Can Improve Generalization, *Proc. Neural Information Processing Systems*, pp. 950-957 (1992).
12) Lin, F., et al.: *Time Series Forecasting with Neural Networks*, Complexity International, Vol. 2, ISSN 1320-0682, (1995).
13) Lukowicz, P., et al.: *Experimental Evaluation in Computer Science: A Quantitative Study*, Tech. Rep. 17/94, Univ. Karlsruhe, ftp:/pub/papers/techreports/1994/ (1994).
14) Miyano, T. and Girosi, F.: *Forecasting Global Temperature Variations by Neural Networks*, AI memo 1447, MIT AI Lab. (1994).
15) Prechelt, L.: *Some Notes on Neural Learning Algorithm Benchmarking* , Neurocomputing, Vol. 9, No. 3, pp. 343-347 (1995).
16) Ossen, A. and Schnauss, M.: Practical Tools for Derivative Instruments based on Nonlinear Series Prediction, *Proc. Int'l. Workshop on Parallel Applicat. in Statistics and Econ.* , Neural Network World, Vol. 5, pp. 525–536 (1995).
17) Ripley, B. D.: Statistical Ideas for Selecting Network Architectures, *Neural Information Processing Systems '95* , http://www.stats.ox.ac.uk/pub/bdr/NIPS/ (1995).
18) Rogers, R.O. and Skillicorn, D.B.: *Strategies for Parallelizing Supervised and Unsupervised Learning in ANN using the BSP Cost Model.*, TR97-406, Dept. of Comp. & Inf. Sci., Queen's Univ. (1997).
19) Rueger, S.M. and Ossen, A.: Performance Evaluation of Feedforward Networks Using Computational Methods, *Proc. Int'l Conf. on Neural Networks and their Applications 95/96*, pp. 35–39, (http://ini.cs.tu-berlin.de/~ao/pubs/pitfalls_submitted.ps.gz) (1995).
20) Tang, Z. and Fishwick, P. A.: *Feedforward Neural Nets as Models for Time Series Forecasting*, TR91-008, Comp. & Inf. Sci., Univ. of Florida (1991).

(Received April 15, 1998)
(Revised June 10, 1998)
(Accepted June 29, 1998)

Alexandra Ioana Cristea graduated 1994 as a Computer Science Eng. from 'Politehnica' Univ.,Bucharest (PUB) and did her Master thesis study at Lyngby Univ., Denmark; she received 1996 a Master in Economical Eng., from the Dept. of Eng.Sciences, PUB and Techn. Hochsch. Darmstadt. She worked as a teaching assistant at PUB and Univ. of Electro-Communic.,Tokyo (UEC). At present she does a PhD. research at UEC, Grad. School of Info Syst.(IS). She is a student member of IEEE, IEEE Computer Soc. and IEICE.

Toshio Okamoto graduated 1971 Kyoto Univ. of Education, received his Master 1975 from Tokyo Gakugei University, and his Dr. of Eng. from Tokyo Instit. of Technology. He is presently a professor at UEC, engaged in research on AI models for intelligent CAI system. He is a member of the Board of CAI Soc., and Jap. Soc. Educ. Tech., Educ. Eng., AI & Knowledge Eng. in IEICE, and has a lot of important official functions for promoting the field of computer and education in Japan.