Energy Function based on Restrictions for Supervised Learning on Feedforward Networks

Alexandra I. Cristea and Toshio Okamoto
The Grad. School of Info Systems, Univ. of Electro-Comm.
IS, AI lab., Chofu, Chofugaoka 1-5-1, Tokyo 182, Japan

Abstract

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.

教師ありフィードフォワード・ニューラルネットワークにおける
制約に基づくエネルギー関数

アレクサンドラ・イオアナ・クリステア、岡本 敏雄
電気通信大学大学院 情報システム学研究科
情報システム設計学専攻 知識処理システム学講座
〒182 東京都調布市調布ヶ丘 1-5-1

本稿では、教師あり学習をおこなうフィードフォワード型のニューラルネットワークにおけるエネルギー関数の設計と使用に関して詳述する。このエネルギー関数は Lyapunov (infinite) ノルムをベースにし、そこでの計算上の制約に基づいて設計されている。特に、誤差最少化手続きから得られるエネルギー関数の数学的演繹に焦点をあてる。提案するエネルギー関数から導かれる等式の変化が、時系列データの予測ツール、特に株価を予測するツールのニューラルネットワーク部の学習と汎化能力を如何に高めるかを論じる。提案手法の能力を確認するために、典型的なバックプロパゲーション法の動作との比較を行うためにシミュレーション実験を行った。そして t 検定等を行った結果から、提案手法の妥当性が確認された。

1 Introduction

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], [5]). 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 the present paper we will confine ourself to the traditional optimization methods (see [4] for details), and focus 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. 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.

Forecasting of TS 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*, therefore predictable, *fully random*, therefore unpredictable, or, as in most cases, somewhere in-between. 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.

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.

2 Deductions of the energy function

For TS 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[1] 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 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[2] as follows.

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

If we define w_0 as being the maximum error radius, $w_0 \geq |r_j(w)|; j = 1,..,n; w_0 \geq 0$, the minimization of the error becomes equivalent to the minimization of w_0. This condition can be divided into 2 inequalities (restrictions):

$$g_{j1}(w) = w_0 + r_j(w) \geq 0;\ g_{j2}(w) = w_0 - r_j(w) \geq 0;\ w_0 \geq 0; j = 1,..,n, \tag{2}$$

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:

$$E(w) = \nu w_0 + \frac{k}{2}\sum_{j=1}^{n}\{([g_{j_1}(w)]_-)^2 + ([g_{j_2}(w)]_-)^2\}, \tag{3}$$

with $[y]_- = min(0, y); \nu > 0, k > 0$, constants. Equation 3 states that the energy of the system depends only on the maximum error radius w_0, as long as the restrictions in 2 are satisfied. If not, the terms g_{j1}, g_{j2} also contribute to the error function.

Let $[y]_- = y * S(y)$, where

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

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

$$\frac{dw_{i,j}}{dt} = -\frac{d}{dw_{ij}}(E(w)) = -kx_i\sum_{j=1}^{n}\{(w_0 + r_j)\,S_{j1} - (w_0 - r_j)\,S_{j2}\} \tag{4}$$

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

$$\frac{dw_0}{dt} = -\frac{d}{dw_0}(E(w)) = -\nu - k\sum_{j=1}^{n}\{(w_0 + r_j)\,S_{j1} + (w_0 - r_j)\,S_{j2}\} \tag{5}$$

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.

We can rewrite the components that are added in the sums of equations 4,5 as:

[1] Infinite Norm

[2] error, or cost function

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

$$(w_0 + r_j(w))S_{j1} - (w_0 - r_j(w))S_{j2} = \begin{cases} w_0 + r_j(w), & \text{for } r_j < -w_0 \\ 0, & r_j \in [-w_0, w_0) \\ -w_0 + r_j(w), & r_j \geq w_0 \end{cases} \quad (6)$$

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

$$(w_0 + r_j(w))S_{j1} + (w_0 - r_j(w))S_{j2} = \begin{cases} w_0 + r_j(w), & \text{for } r_j < -w_0 \\ 0, & r_j \in [-w_0, w_0) \\ w_0 - r_j(w), & r_j \geq w_0 \end{cases} \quad (7)$$

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.6 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.7 for w_0 compares the maximum error to the other errors, and tries to shrink the sphere, 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, but r_j must correct both deviations in the negative and positive domain, so it can change both ways.

3 Backpropagation of Error

The designed net is a feedforward net 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. [6]) showed that a 2 layer net is enough (see [9] for a discussion upon network parameter selection).

We have deducted the weights' changing equation in section 2, and we have established the net design, 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[3], 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. 4, 5, 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:

$$\frac{dv_{k,i}}{dt} = -kx_k f'\left(\sum_{k=1}^{m+1} v_{k,i}x[k] + v_{0,i}\right) * \sum_{j=1}^{n} w_{i,j}\left\{(w_0 + r_j)\,S_{j1} - (w_0 - r_j)\,S_{j2}\right\},$$
$$k = 1, .., m; i = 1, .., p; \quad (8)$$

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

$$\frac{dv_{0,i}}{dt} = -kf'\left(\sum_{k=1}^{m+1} v_{k,i}x[k] + v_{0,i}\right) * \sum_{j=1}^{n} w_{i,j}\left\{(w_0 + r_j)\,S_{j1} - (w_0 - r_j)\,S_{j2}\right\},$$
$$i = 1, .., p; \quad (9)$$

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.

4 Implementation

The system can perform **training** (learning) through weight computing, *forecasting* with a 2-layer feedforward NN and can also serve as a *user-interface*. The designed program provides a full help support at each step.

[3] as in standard backpropagation

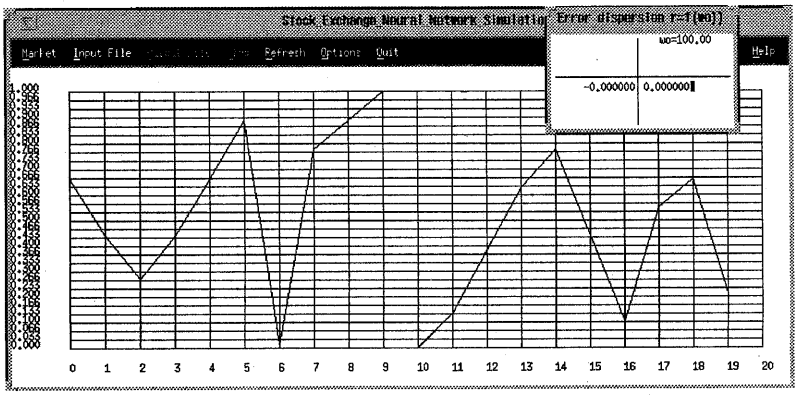

Figure 1: A "perfectly" trained net

The *prediction error* is displayed in percentage to the maximum value (price) of the time-period (see fig. 1). The data consists of some real-world data from [6], 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 [8]). The data are values (prices on SE market) over any desired period of days and are represented in fig. 1 on the 0y axis, scaled for a better representation [0-min; 1-max]. The 0x axis represents the time. The considered time-period is usually a month, but the system can handle any amount of input and/or output data, by using dynamic memory allocation functions.

For speed-up we used 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 (step = .5 * step), to converge to the valleys of lower potential. This procedure requires less memory than the weight decay method. 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; but 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.

5 Results

First we introduce the learning display. We present here a learning example of an equal share of 10 input data and 10 outputs (fig. 1). 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[4] 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[5] 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.

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.

[4] as in *Supervised Learning*
[5] just like the inputs

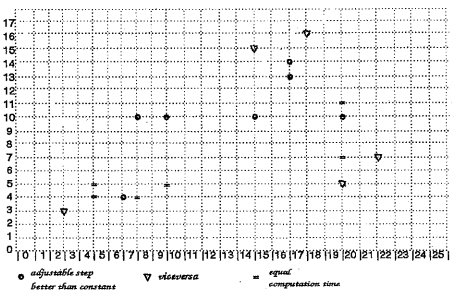

Figure 2: Variable and constant step; the influence on the convergence.

5.1 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. 2). 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. 2, 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 convergence. 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.

5.2 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:

$$R : \frac{\sum_{j=1}^{n} (b_j - b')(y_j - y')}{\sqrt{\sum_{j=1}^{n} (b_j - b')^2} \sqrt{\sum_{j=1}^{n} (y_j - y')^2}} \tag{10}$$

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):

$$T_r : \frac{\sqrt{\sum_{j=1}^{n} (y_j - b_j)^2}}{\sqrt{\sum_{j=1}^{n} (b_j - b_{j-1})^2}} \tag{11}$$

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. 3.

We observed two methods: our method, versus the classical BP method. As can be seen in fig.3, 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

	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

Figure 3: Backpropagation and Lyapunov method's behaviour regarding the algebraic indicators.

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 [7] for experimental validation suggestions for NN algorithms).

6 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, by using indicator comparison, the network constructed based on this energy function, with a network using the classical Backpropagation method and showed the differences between the two. For further work we intend to extend the system and find out what other problems it's applicable to.

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] Cristea, A. and Okamoto, T.: *NN for Stock Exchange prediction; a Lyapunov based training. ICCIMA'98 Proceedings*, Australia, (Selvaraj, H and Verma, B. (ed.)), World Scientific, pp. 416-421, (1998).

[5] Dasgupta, D. and McGregor, D.R.: *Designing Application-Specific Neural Networks using the Structured Genetic Algorithm. COGANN-92 Proceedings*, (Whitley & Schaffer (ed.)), IEEE Comp. Soc. Press, (1992).

[6] Komo, D., et al.: *Neural Network Technology for Stock Market Index Prediction, ISSSIPNN'94 Proceedings*, Hong-Kong, IEEE, pp. 543-546, (1994).

[7] Lukowicz, P., et al.: *Experimental Evaluation in Computer Science: A Quantitative Study*, Journal of Systems and Software, (1995).

[8] Prechelt, L.: *Some Notes on Neural Learning Algorithm Benchmarking* , Neurocomputing, (1995).

[9] Ripley, B.D.: *Statistical Ideas for Selecting Network Architectures NIPS'95 Proceedings*, (1995).