NOTE TO USERS

This reproduction is the best copy available.

UMI®

Improving the Performance of Neural Networks through Parallel Processing in the Cell Broadband Engine

by

Yuri Boiko

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of
Master of Applied Science in Electrical and Computer Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering (OCIECE)

Department of Systems and Computer Engineering

Carleton University

Ottawa, Ontario, Canada, K1S 5B6

May 2010

Your file Votre référence
ISBN: 978-0-494-68647-8
Our file Notre référence
ISBN: 978-0-494-68647-8

The undersigned recommend to
the Faculty of Graduate Studies and Research
acceptance of the thesis

Improving the Performance of Neural Networks through Parallel Processing in the Cell Broadband Engine

submitted by

Yuri Boiko

in partial fulfillment of the requirements for

the degree of **Master of Applied Science in Electrical and Computer Engineering** of

Chair, Dr. H. M. Schwartz, Department of Systems and Computer Engineering

Thesis Supervisor, Dr. Gabriel Wainer

Carleton University
May 2010

Abstract

This Thesis focuses on the exploration of parallelization approaches for improving the performance of ANN. A main goal of this Thesis is to define the routes for the parallel computation of this problem using the multi-core Cell Broadband Engine. In particular, a new design for parallel tracing of the gradient descent algorithm showed the feasibility for efficient finding of viable solutions for the approximations of 2D non-linear functions and the predictions of 1D time series by neural networks. One objective was to identify the parameters of the gradient descent algorithm which can be used for parallelization of the tasks in 2D function approximation and 1D time series prediction in terms of speed and accuracy of the delivered solutions, and to obtain fast convergence to the optimal solutions. Specifically, for a 2D function approximation, the entrapment in the local minima has been addressed via parallel tracing of the converging trajectories, while verifying the optimality of the solutions. For a 1D function approximation, the original task involves multiple-input-multiple-output multi-dimensional neural networks and thus is challenging for the gradient descent algorithm, posing problems of speed and convergence. In this case, the goal was set to verify the efficiency of the splitting of the multiple-steps forecasting task into several sub-tasks with various forecasting horizons in order to achieve fast and accurate forecasting solutions. The sub-tasks with various forecasting horizons extracted from the complex task would require a simpler type of multiple-input-single-output neural networks. The objective was to demonstrate the improved efficiency of such approach in parallel computing environment to reach fast and accurate solutions with the gradient descent algorithm.

ACKNOWLEDGEMENT

Gratefully acknowledged are: the assistance of the Department's students, particularly that of Ph.D. Candidate Qi (Jacky) Liu, which enabled the smooth progress of this research; support and consistent care taken by Professor Halim Yanikomeroglu of Carleton University in his capacity of Chair for Graduate Studies, enabling this research effort; efforts of the Committee members, in particular that of Professor James Green of Carleton University as internal reviewer, Professor Tet H. Yeap of University of Ottawa as external examiner and Professor Babak Esfandiari of Carleton University as Committee Chair; and supervision of this research work by Professor Gabriel Wainer (Carleton University).

Table of contents

List of acronyms

Abbreviations from vocabulary of Cell/B.E. documentation:

Cell/B.E. – Cell Broadband Engine;

SDK – Software Development Kit;

SPU - Synergistic Processing Unit;

SPE - Synergistic Processing Element;

PPU – Power Processing Unit;

PPE – Power Processing Element;

DMA – direct memory access;

MLP – multi-layer perceptron;

RMS – root mean square;

RMSE – root mean square error;

MIMO – multiple-input-multiple-output;

MISO – multiple-inputs-single-output;

SS – single-step (example of use: "SS prediction" for "single-step prediction");

VOP – visual object plane;

NN – neural networks;

ANN – artificial neural networks;

OCR – optical character recognition;

ASR – automatic speech recognition;

Chapter 1. Introduction

An *Artificial Neural Network* (ANN) represents a mathematical model to simulate the operation of the neural systems, such as those of biological objects (including the brain), which have been applied to solve a variety of complex problems [1]. In particular, ANNs have multiple applications in adaptive control systems, where a controller has the ability to adapt to changes in the environment. Examples include systems with non-linearities of any kind, such as mechanical (friction, etc), electrical, thermal, temporal, etc. Other well known applications, include stock market prediction, temperature distribution controllers, robotics vision, signal processing, etc. [2 - 30]

Artificial neural networks (ANN) possess important abilities to infer functions from observations. In practice, they are especially useful for the cases of dealing with complex data, where the ANNs are capable of actually mimicking an underlying unknown function, in particular for non-linear function approximations [3 - 8], regression analysis [27] and time series prediction. [25 - 30]

Typically, a neural network (NN) consists of a set of highly interconnected neurons. Each neuron in the ANN represents an object that accepts incoming signals from neighboring neurons, multiplies it by a weight coefficient, adds all the signals obtained and produces output signal, which is a preset function of the obtained sum. An ANN possesses adaptivity in that it can change the input weights of the neurons in such a way that signals coming from the outside of the network can be memorized (in the form of the weights distribution inside the neural network). This means that ANN actually can *learn* about the environment by interacting with it, and changing its structure to reflect the record of such interaction. It can be seen as a *machine learning* technique, where the

ANN as a machine learns and adapts its properties in accordance with its interaction with the outside world. [9 - 11]

ANNs are usually organized in layered structures of neurons, in which the neurons of consecutive layers are interconnected (but with no interconnection between the neurons within the same layer). Such organization for an ANN is called *multi-layer perceptron* (MLP) [1]. MLPs are capable of mapping the sets of the incoming signals to the sets of the outgoing signals according to a specified *transformation function*, which MLP can approximate. According to existing terminology, the minimal number of layers for an MLP is 3: at the very least the MLP contains one layer of input nodes, one layer of neurons (called the *hidden layer* or *hidden neurons*), and the layer of the output neurons [1]. Any incoming signal is transferred from the input layer to the hidden layer through the connections of the NN. Then it proceeds to the output layer. Each neuron collects an input signal from each connection, multiplies them by the input weights, adds them, and performs a functional transformation using its internal function, passing the result to its output.

The input weights of the neurons are *free parameters* of the NN. These free parameters are subject to updates in the process of adapting the structure of the NN to the signal from the outside. The algorithm for adjusting the input weights of the neurons, called the *gradient descent algorithm* [1], is a mathematical procedure for calculating the required updates for the network's free parameters (such as the input weights of the neurons). This algorithm starts from obtaining an input from the environment (i.e. from the outside). For the MLP to perform the required mapping, it needs to undergo through a *training* process, during which the MLP is exposed to a set of standard incoming signals with known

outputs. During this training procedure the neurons' free parameters compares the actual output of the MLP with the expected value, and it calculates the error obtained. Then, the algorithm updates each free parameter based on the requirement to minimize the overall error. In this way, the error is propagates back along the network connections, and the result of this process is the incremental update of the free parameters (i.e. the input weights of the neurons), which reduces of the overall output error. For that reason this procedure is also called *error backpropagation*, and it does constitute the basis of the gradient descent algorithm [1]. The gradient descent is a first order optimization algorithm that allows finding a local minimum of the error function (thus optimizing the approximation of the function represented by ANN) by taking steps proportional to the negative of the gradient of this error function [1].

The gradient descent algorithm became very popular in the early 1990s [3 - 5] and it still remains one of the most widely used algorithms for machine learning [15, 17]. Nevertheless, the algorithm poses various implementation difficulties for NN training in single processor computing environments. The following list includes some of the major problems, which have inspired various research efforts for more sophisticated versions of it and/or alternative algorithmic approaches [1]:

(1) the gradient descent algorithm does not guarantee reproducible results;

(2) the systems to which gradient descent is applied to can be too complex, so that the convergence of the algorithm can become poor;

(3) the gradient descent can be too slow;

(4) the solutions offered by the gradient descent can be non-optimal.

Addressing the above listed problems in a single processor environment is sometimes unfeasible. For instance, the reproducibility of the results may be addressed by multiple repetitions of random initial starting conditions and memorizing those of them that led to the acceptable solutions. The complexity of the system involved can be addressed by splitting the task into smaller sub-tasks (each with better convergence). The sequential screening of the various step sizes would lead to finding its optimized value. The use of these methods on a single processor results in extra computational time (which is not feasible for real time applications). For instance, in time series prediction, the time itself is an essential parameter of the task, which makes it essential for the gradient descent to be sufficiently fast on the time scale of the series. A parallel computing environment can help to solve these problems, and increasing the available computational power may reduce the risks of failure due to the problems with implementation of the gradient descent algorithm may offer in facing the challenge [31 - 35].

There had been systematic efforts in exploring advantages of parallelization for the neural networks applications, in particular large size networks [34 - 37]. For example, Hasselström (2008) in his Thesis [37] investigated and analyzed the efficiency of parallelization for accelerating the gradient descent for large-size neural networks. The idea was to distribute the computational load between parallel processors of the Cell Broadband Engine [38 – 47], which has been used as a tool for supporting the parallel architecture. The effort had been successful in allowing parallel processors to jointly simulate large size neural net. The communication between processors allowed to retain the connections within portions of the networks, which were simulated by different processors. This however is redundant for small size neural networks which are

considered herewith. There also had been research implementations of the tasks parallelization for the neural networks [48 – 50]. In one case the same data sets were processed by multiple parallel processors to extract different features from them [49]. In another case a complex problem was split in a number of sub-tasks to be processed by different processors and the obtained solutions were combined together in order to obtain solution not available otherwise [50].

In this Thesis similar to the above mentioned [49, 50] parallelization approaches have been explored with regard to 2D function approximation and 1D time series prediction tasks. A main goal of this Thesis is to define the routes for the parallel computation of these problems. The implementation of the parallel tracing of the gradient descent algorithm showed the feasibility for efficient finding of viable solutions for the approximations of 2D non-linear functions and the predictions of 1D time series by neural networks. The main goal of this Thesis was to identify the parameters of the gradient descent algorithm which can be used for parallelization of the tasks in 2D function approximation and 1D time series prediction in terms of speed and accuracy of the delivered solutions, and to obtain fast convergence to the optimal solutions. The idea was to target the problems (1) - (4) described above. Specifically, for a 2D function approximation, the entrapment in the local minima in the parallel computing environment can be addressed via parallel tracing of the converging trajectories, while verifying the optimality of the solutions to which efficient convergence is reached. For a 1D function approximation, the original task involves multiple-input-multiple-output multi-dimensional neural networks and thus is challenging for the gradient descent algorithm, posing problems of speed and convergence. In this case, the goal was set to verify the

efficiency of the splitting of the multiple-steps forecasting task into several sub-tasks with various forecasting horizons in order to achieve fast and accurate forecasting solutions. The sub-tasks with various forecasting horizons extracted from the complex task would require a simpler type of multiple-input-single-output neural networks. The objective was to demonstrate the efficiency of such approach in parallel computing environment to reach fast and accurate solutions with the gradient descent algorithm.

The Cell Broadband Engine (Cell/B.E.) [36] was used as a tool for providing parallel computing in this study. The capacity of the Cell/B.E. to asynchronously trace several trajectories of implemented gradient descent algorithm from random sets of the starting points offered the advantage of revealing statistical trends and classifying viable optimal approximations delivered by simulated function generator.

The algorithms were applied and numerous simulations were conducted (which will be presented in this document). The first problem to be discussed is the approximation conditions of the surfaces of 2^{nd} and 3^{rd} order with saddle points (which are derived via implementation of the gradient descent algorithm for a MPL). The results of these experiments show the conditions for generating function approximations with an optimal error distribution. Then, the advantages of the use of the Cell/B.E. parallel environment were explored for a time series prediction, using two types of time series. One type of series has been designed to contain a long term trend. A second one is a deterministic chaos time series, such as the Mackey–Glass 30 time series [25] (regarded as one of the benchmark in the time series prediction field). For both the above time series, it has been shown that scaling up the base step for prediction (which acts as a factor of extending the base for prediction into the preceding history), does improve the accuracy of the long

term forecasts. Quantitatively, such an improvement is characterized by about proportional reduction of the root mean square error (RMSE) for 10-steps-ahead forecasts. Moreover, if the long term trend of the time series changes its sign, which herewith is regarded to be a negating trend as compared to the trend in training, then it is shown that scaling up of the base step B brings about a significant change in the dependence of the RMSE of the prediction from the duration of training session. Detection of such change in the above dependence is suggested to be the ground for detecting of the occurrences of the negating trend events in the time series.

For the deterministic chaos time series prediction two major results have been obtained. Firstly, the fast portion of the training process for Mackey-Glass series is identified, which provides about 10^2 reduction of the Root Mean Square Error in the predicted values and thus amounts to more than 95% of the training benefit (i.e. RMS Error reduction) within less than 2% of total amount of training time. The fast track of training takes less than 1000 epochs as compared to 50000 of a full training time length with detectable convergence. The parallel computing environment allows for an early engagement of the neural forecaster which is better than 95% ready while still continuing full training of the final forecaster with the additional benefit of supplying the newly arrived time points for ongoing training. The last procedure benefits in that the most recent time points would be accounted for when forecaster is finalized. It has also been shown that the long term forecasts can be improved by leveraging the forecasting time base by points from the prior history. Specifically, for 10 step forecasts based on immediately preceding points of the Mackey-Glass 30 time series the RMS Error can be improved up to the order of magnitude (ca. 10 times) when engaging forecaster with the

leverage depth of 10 times into the history. The leverage is provided by feeding in to the neural network the points separated by 10 consecutive time intervals instead of sequential time points. The Cell/B.E. provides an adequate parallel environment for taking advantage of availability of the long range forecaster with leverage in the prior history.

The rest of the thesis is structured as follows. Chapter 2 presents the state-of-the-art for neural networks in parallel computing environments, reviewing the subject of the research and its coverage in the related literature. Chapter 3 contains a concise formulation of the research question, its justification based on the analysis of the existing literature, and a discussion on its significance. Chapter 4 presents algorithm design, implementation and the results on 2Dfunction approximation tasks, while Chapter 5 – does the same for 1D time series prediction tasks. Concluding remarks and major discussions with the formulated contributions delivered by the work done are summarized in Chapter 6.

LIST OF CONTRIBUTIONS

The following contributions of *new* knowledge have been made by this thesis.

1. Using a neural approximation of the 2D functions of 2^{nd} and 3^{rd} order with saddle points, the parallel tracing of the gradient descent trajectories is demonstrated as a viable alternative in identifying the favorable starting conditions, avoiding local minima traps and obtaining the optimal solution in a quickest possible time frame. It is shown that optimality of the solution can only be determined by overall testing, until finalization of which all the viable candidates obtained in the parallel tracing need to be retained. For the

first time the close proximity extrapolation ability of the neural network is shown to reflect its interpolation quality in the 2D function non-linear approximation.

2. A new approach is introduced for performance improvement of the neural network MLP multi-step forecaster in the 1D time series. The approach includes simultaneous (i) redundancy removal in the 1D time series and (ii) base step increase for the prediction basis, implementation of which results in accuracy increase of the forecasts. For the Mackey-Glass 30 time series a 10 fold simultaneous redundancy removal and a base step increase is shown to result in almost order of magnitude improvement of the RMSE for the 10-steps and 20-steps forecasts, as well as more than double in accuracy for long-term forecasts of 50-, 100- and 150-steps ahead. In the parallel environment splitting the multi-task into the individual ones and distributing of the individual forecasting tasks between the processors is shown to provide fast-tracks for training of the neural networks, as well as to offer alternative forecasts for selection of the best available accuracy.

3. A new approach is demonstrated capable to identify occurrence of the negating trend in the compositional 1D time series. The approach is based on evaluating the performance of the neural network MLP forecaster and includes comparing RMSE dependences of the long range forecasts from the number of training epochs for the minimal base step and the scaled up base step of the prediction basis. Specifically, it is demonstrated, that for a steady continuous trend doubling the base step results in almost doubling the accuracy of the 10-steps forecasts, while for negating trend in the same time

series such procedure creates an expressed optimum in training length for the same forecasts. Parallel tracing of the above dependences enables an in-situ detection of the negating trend occurrences at the central control unit.

LIST OF PUBLICATIONS

1. Y. Boiko and G. Wainer, "Accelerating the computation of parallel trajectories of gradient descent with the Cell-BE multiprocessor environment". Submitted to SummerSim'10 conference, July 11-15, 2010, Ottawa, ON, Canada.

2. Y. Boiko and G. Wainer, "Parallel tracing of multiple trajectories in gradient descent algorithm with Cell Broadband Engine",- Proceedings of the 2009 Spring Simulation Multiconference, San Diego, California, 2009 poster track, Article No. 123 (2009)

3. Y. Boiko and G. Wainer, "Modeling of neural decoder based on binary spiking neurons in DEVS",- Proceedings of the 2009 Spring Simulation Multiconference, San Diego, California, 2009 poster track, Article No. 149 (2009)

4. Y. Boiko and G. Wainer, "Modeling spiking neural terminals in DEVS",- Proceedings of the 2008 Spring simulation multiconference, Ottawa, Canada, 2008 poster track, Article No. 19 (2008)

5. Y. Boiko and G. Wainer, "Modeling quantum dot devices in Cell-DEVS environment",- Proceedings of the 2008 Spring simulation multiconference, Ottawa, Canada, 2008 poster track, Article No. 18 (2008).

Chapter 2. Neural Networks in Parallel Computing Environments

2.1 Function approximation with neural network

The concept of a neural network (NN) stems from the attempts to simulate computationally and electronically operation of biological nervous systems, brain in particular [1]. Of about hundred billion of cells called *neurons* make up a human's brain. Neurons are highly interconnected with each other and are communicating via sending and receiving electrochemical signals, thus composing sophisticated network of interconnected and communicating neurons. The incoming signals are entering the neurons via junctions called *synapses*. Operation of each neuron is in collecting the incoming signals and producing the output, which is a function of the received integrated input. Multi-layer perceptron (MLP) represents a basic type of neural network, which is capable of implementing the multi-dimensional mapping of set of input variables into the designated set of output variables. The structure of the MLP includes at least three interconnected layers – (1) the layer of input nodes to take in the values of input variables, (2) the layer of hidden nodes which are simulating functional neurons and (3) the layers of output neurons to deliver the result of input processing. There is one additional variable, the bias, which is normally set to be equal to "-1" and is connected to all the neurons in the same way as the input variables are, i.e. via adjustable input weights. Adjusting the input weights of the hidden nodes and output nodes is called "training", which is the algorithmic process of finding combination of input weights for all neurons, hidden and output ones, so that to represent by MLP the desired mapping

function with the minimal error. The training of MLP may be looked at as a function approximation problem. In this view, the neural network plays the role of a mapping function, which provides a nonlinear mapping of the input values to the output. Wieland and Leighton (1987) [2] suggested that a neural network can efficiently generalize the mapping, thus exhibiting the effect of a good nonlinear interpolation for those input values, which never had been presented to the network before. Such interpolation is possible due to the fact that having continuous nonlinear function for its nodes the MLPs provide a continuous output function. Possible outcomes of the generalization are illustrated on Fig.1 for a hypothetical network. The input data supplied to the MLP consist of a set of points, called "fitting points", for which the output values are known. These data are provided to the network to adjust weights of neuron's inputs via algorithm, called gradient descent or error backpropagation algorithm [1]. This algorithm which involves calculation of the error for each of the supplied points and based on the value and the sign of that error the increment of each neuron's input weight is derived aiming to minimize the error at the output of MLP. When the number of "fitting points" is too small, the underfitting may occur. One way to counter it is by increasing the number of hidden nodes of the network and thus improving its learning capability. In contrast to that, increasing the number of the "fitting points" in training allows to pursue convergence under more stringent accuracy criteria. After the training is completed and all the input weights of the neurons are adjusted so that the resulting error for the set of "fitting points" is minimized, the MLP becomes ready to perform a role of function approximation by taking in the intermediate points which had never been offered to it before. The output values for those intermediate points are calculated by implementing

operations of applying the weights (and biases) derived via training based on "fitting points" only (it is called a *feed-forward process*). Such direct calculation provides the mapping values for any other points other than "fitting points", thus performing an interpolation as well as an extrapolation. The trained network needs to allow accurate mapping even for points other than "fitting points" not included in the training. However, training algorithm only ensures accuracy for the fitting points while values in the intermediate points (interpolation points) might differ significantly from the correct values.

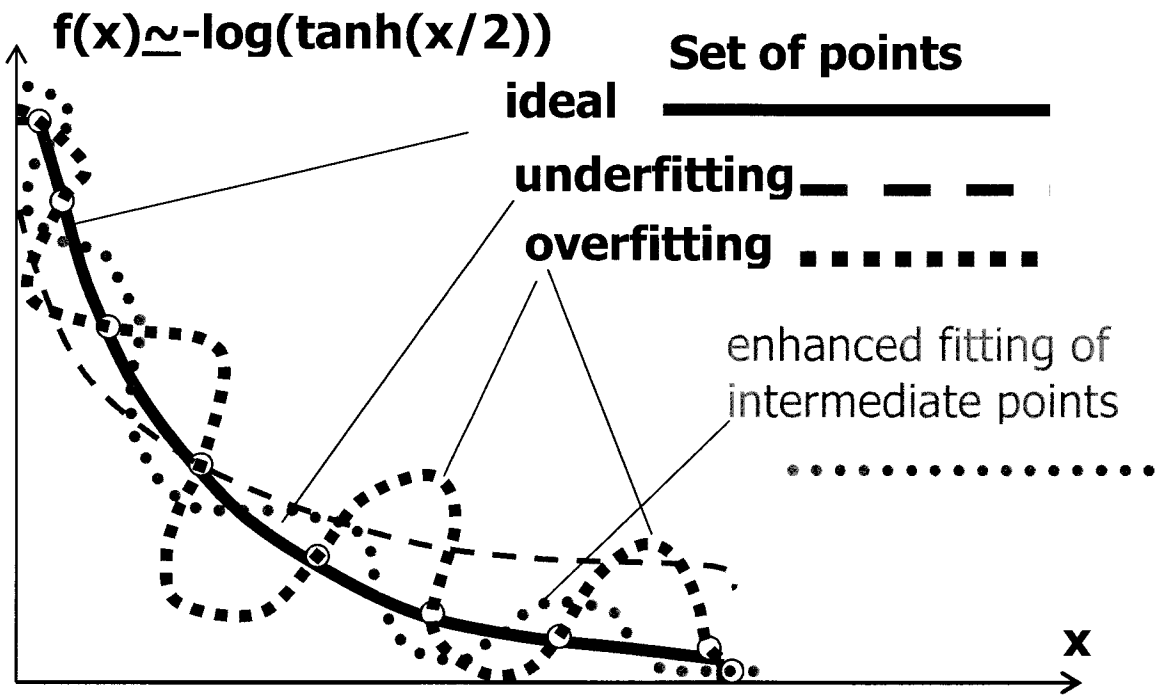


Fig.1. Generalization by hypothetical network: (a) ideal fit; (b) underfitting; (c) overfitting; (d) enhanced fitting of intermediate points.

Such situation when occurs is called *over-fitting* and needs to be avoided. An essential feature of the over-fitting (as compared to the ideal fit) is the lack of smoothness in the

fitting interpolation. Without involving extra "fitting points" the desired smoothness can be attained by selecting of the "simplest" function to meet the fitting criteria. There are two major factors that affect the accuracy in the generalization:

(1) the number of the "fitting points", which represent essential features of the model;

(2) the design/architecture of the neural network.

Additionally, the complexity of the model to fit the required mapping is supposed to be represented by the set of the selected fitting points. A good generalization is obtained by either: (i) fixing the architecture of the network and finding the representative set of fitting points; or by (ii) finding the best architecture for a given fixed set of the fitting points. A practical criterion to achieve a good generalization for fixed network architecture is to provide a training set of N fitting points,

$$N = O(W/\varepsilon), \qquad (2.1)$$

where W is the number of the network's adjustable parameters, such as weights of the neurons, ε is the fraction of classification errors allowed on the test, and O(.) is the order of inside quantity. As an example, for 1% error (i.e. ε=0.01) the number of training points need to be 100 times the number of W[1].

If m_0 is the number of inputs of the MLP and M=m_L - the number of its outputs, the MLP function provides mapping of the m_0-dimentional input space into M-dimensional output space. The minimum number of hidden neurons in the MLP with one hidden layer and input-output mapping that provides an approximate realization of any continuous mapping is determined by the Universal Approximation Theorem, which merely generalizes an approximation by finite Fourier series. It follows from this theorem that a

single hidden layer is sufficient for an MLP to compute a uniform ε approximation to a given training set represented by the set of inputs $x_1,...,x_{m0}$ and a desired output $f(x_1,...,x_{m0})$. The optimum solution may require the hidden neurons to be arranged in more than one hidden layer to minimize a learning time, a total number of hidden neurons, and to improve mapping accuracy.

There are bounds on the approximation errors, which were established by Barron in 1993 [3 - 5] under the following assumptions: (1) MLP contains only one hidden layer; (2) hidden neurons are using tanh(.) function; (3) the output neuron is linear. At the training stage the network learns the set of the training data and as a result reproduces the approximating function F. At the testing stage the network is faced with the new data and there the network function F assumes the role of the estimator of the output target function: $F = f^*$. The approximation error is measured by the *integrated squared error* with respect to an arbitrary probability measure μ on the ball $B_r = \{\mathbf{x}: \|\mathbf{x}\| \leq r\}$ of radius $r>0$. The final expression for the bound on the risk resulting from the MLP approximation mapping can be presented in the form [1]:

$$R \leq O(C_f{}^2/m_I) + O((m_I m_0 \log N)/N), \qquad (2.2)$$

This expression with two terms for the bound on the risk R contains two conflicting requirements on the size of the hidden layer and reflects the existing tradeoffs:

1. Accuracy of the best approximation: m_I must be large (as the universal approximation theorem suggests).

2. Accuracy of the empirical fit to the approximation: m_I/N must be small. Because N, the number of fitting points, is fixed, this contradicts to the first requirement.

The consequence of the above is that an exponentially large sample size N, large in the dimensionality m_0 of the input space, is not required to achieve an accurate estimate of the target function $f(\mathbf{x})$, provided that the first absolute moment C_f remains finite. This conclusion contributes greatly to the recognition of the MLP as universal approximators for the practical cases. If ε_0 denotes the mean-square value of the estimation error, then from the last bound it may be inferred that the size N of the training sample necessary for a good generalization is about $m_0 m_I/\varepsilon_0$. As $m_0 m_I \sim W$, the above relation for the number of fitting points N in a good generalization case becomes confirmed: $N >> W/\varepsilon_0$.

2.2. Structure of the neural network function approximator

Barron's consideration of the function approximation had been extended by G. Cheang on two-hidden-layers neural net [6, 7]. For one hidden layer neural network, the limitations were analyzed by Chui et al [8]. Application wise, the neural function approximators are of interest for adaptive control systems with non-linear actuators [9 - 12], for instance, for systems with one of the following non-linearities: friction, dead-zones, backlash, and time delays. Obtaining 2D non-linear function approximators in such systems is achieved by employing the concept of the function mapping with a neural network as shown in Fig.2.

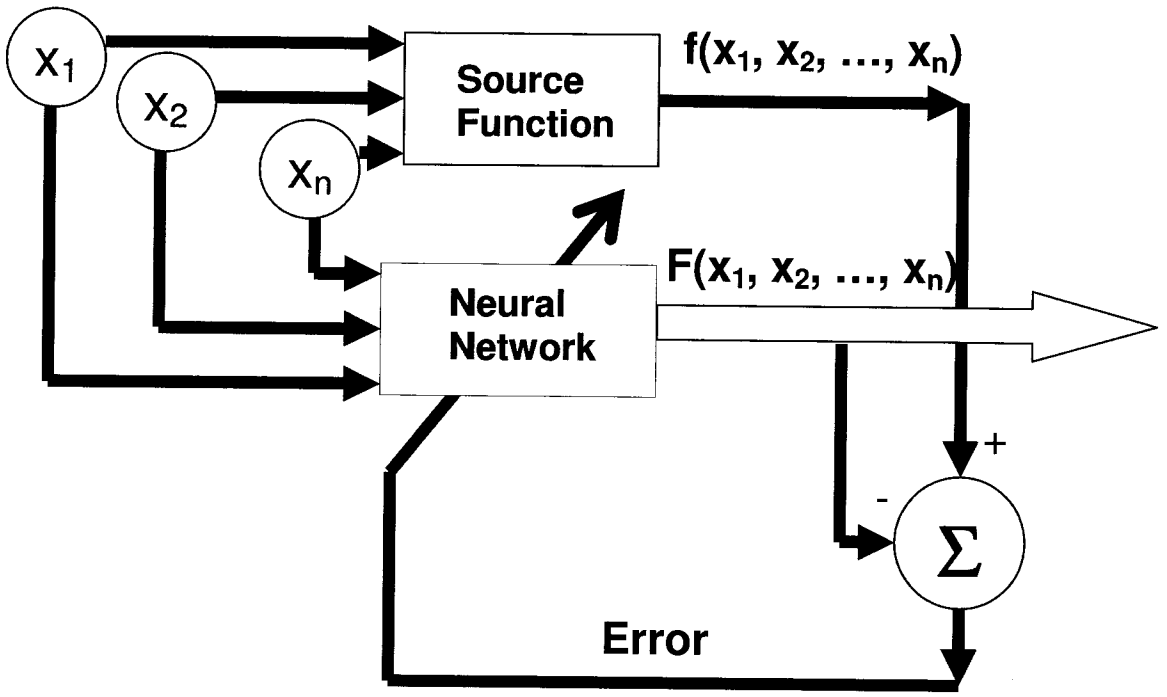


Fig. 2. The concept of the function-approximation training of a neural network.

In the training stage, the neural network is connected to a function source, and it is subjected to a set of the training data (which contain values of the respective output values of the source function to these data). Based on the error signal (generated by comparing the expected values from the function source and the current output of the network) the neural network undergoes adjustments to minimize the error signal.

General structure of the neural network for function approximation is shown in Fig.3. (Omitted there is additional bias node at the input, which has the input value of 1.0 and is connected to each hidden node as well as to the output node).

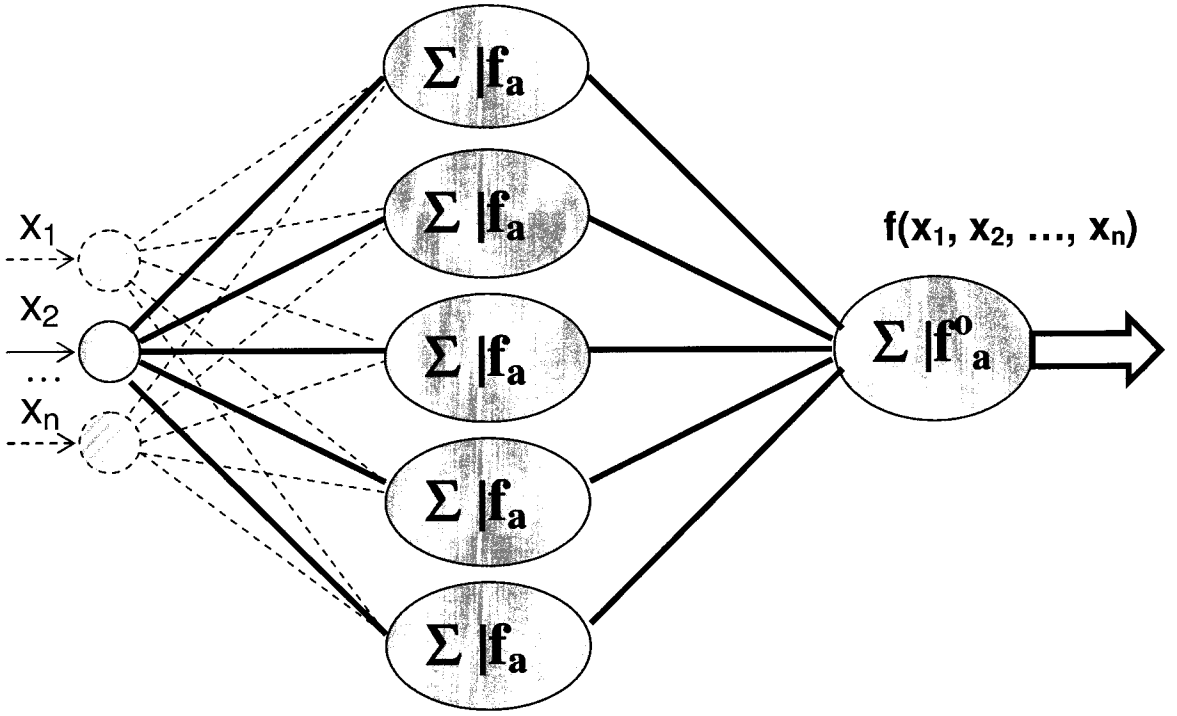


Fig.3. Feed-forward network for function generation (Bias is omitted).

Such neural network can be implemented, for instance, by employing Intel's chip 8017NW ETANN (Electrically Trainable Analogue Neural Network). An alternative is the design suggested by Malcolm Stagg [13]. Electronic versions of the neural network support the backpropagation algorithm, which is the feedback-based learning algorithm mentioned in Chapter 2 and also called *supervised learning*. For training, it requires the calculation of the correct output for an input pattern, but will learn to represent the correct output as a result. The neuron design uses an SRAM cell to enable/disable algorithm-dependent circuits, and also uses tanh(.) sigmoid circuit for forwards-propagation (range: [-1, 1]) and its derivative as $sech^2()$ for backwards propagation (range: [0, 1]).

2.3. Time series forecasting with neural network

Another class of functions in which neural network's mapping ability had been found to be useful is 1D times series [14 - 17]. A time series is a sequence of data points, representing values of the variables at successive times, usually spaced uniformly on the time scale. Examples of well known time series are the stock rates on financial markets, temperature changes in the weather forecasts, traffic volume in the communication channels. Very often the need is to actually predict the expected values of the time series in some remote future based on its past evolution. It was shown [14] that feed-forward type neural networks based on MLP are capable of efficient single-step predictions of the frames sizes in MPEG-coded streams, which reflect representation of those video streams by the time-series with very long range time dependencies. Multi-step prediction based on recurrent networks proved to be efficient for 2-steps-ahead and 4-steps-ahead prediction horizons [14].

The structure of the time series predictor based on a MLP is shown in Fig.4. It is seen that the time series stream is filling in the n inputs of the MLP based on which the prediction, $\underline{X}_{t+k}$, of the future value k-steps ahead is obtained at the output, the actual value being x_{t+k} (so that the Error = x_{t+k} - $\underline{X}_{t+k}$).

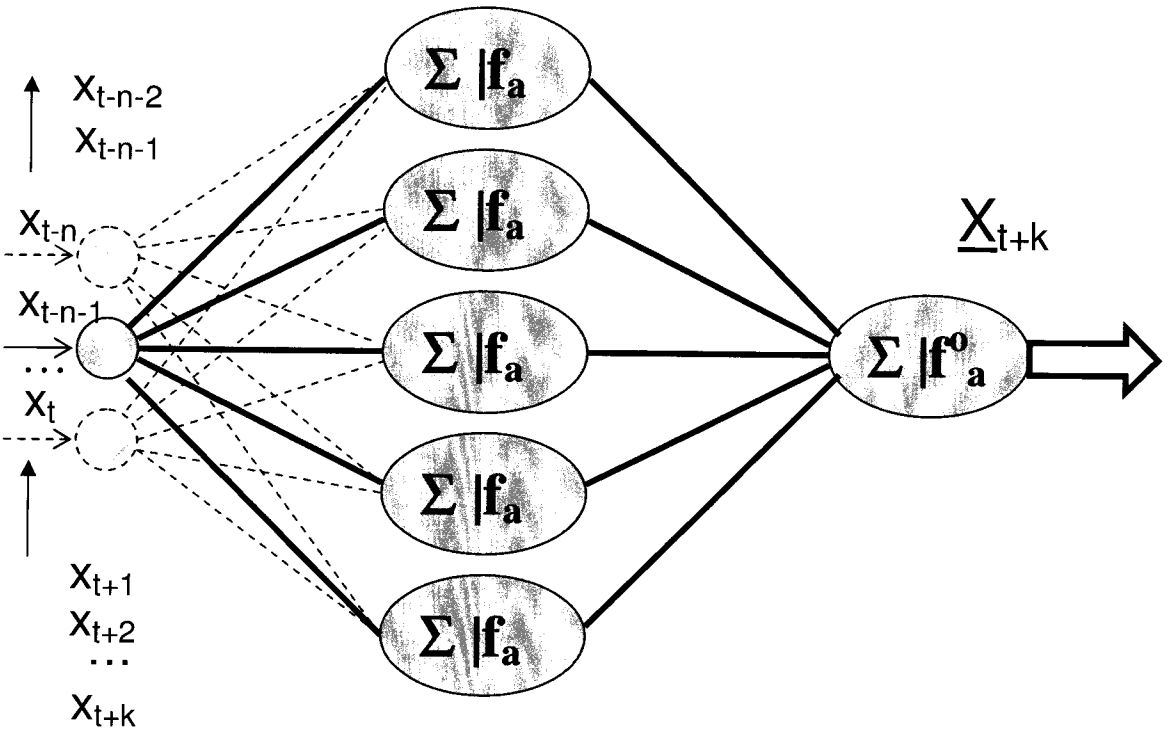


Fig.4. Time series prediction with the neural network (Bias is omitted).

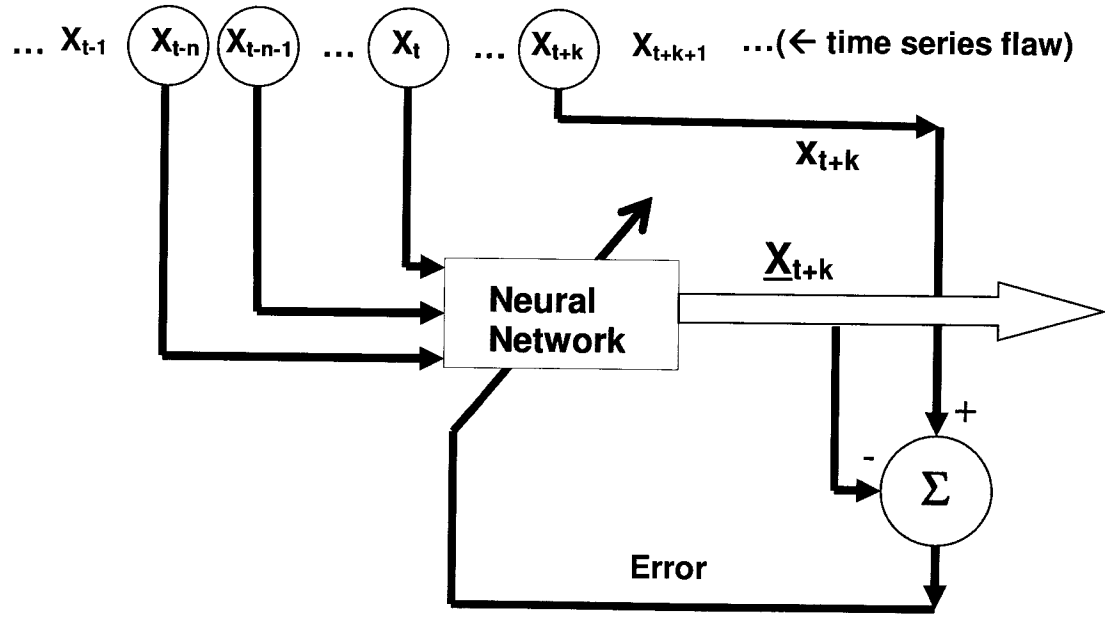


Fig.5. Training scheme of multilayer perceptron type neural network.

The error calculated at each iteration step of the gradient descent algorithm is used for training of the MLP, as it is depicted in Fig.5, where n is number of input points from the training set, k is the number of step for prediction target (also inside the same training set). Thus the error back-propagation allows adjusting the free parameters of the MLP, i.e. neuron's input weights and biases, to minimize the overall error. Converging to the optimal solution in training then allows employing the obtained MLP as a forecaster in the future values of the same time series. In the above scheme the forecasting was pursued for a fixed number of steps into the future. The forecast of only one single point was targeted there for each set of the input values.

In general, the forecasting may involve the requirement to deliver the forecasts for various numbers of steps ahead, i.e. include a short term forecast as well as a longer term forecasts based on the same data and/or preceding past. The neural network suitable for that would be inherently MIMO-type (multiple-inputs-multiple-outputs), as the one depicted in Fig.6. Advantage of the multi-step forecasts is that various horizons of the future values are obtained, which allows for more thoroughly weighted and flexible decision making approach due to more comprehensive view into the upcoming events and changes. However, the increase of the dimensionality of the task makes it more difficult to obtain the optimal solutions due to higher complexity of the problem.

There are number of time series [15], which are widely accepted by the research community, among which deterministic chaos time series is regarded as a benchmark for prediction. Example of such series is the Mackey-Glass time series [16], which originates from a numerical solution of the Mackey-Glass linear differential equation. This time series had been extensively studied with the neural network's approach.

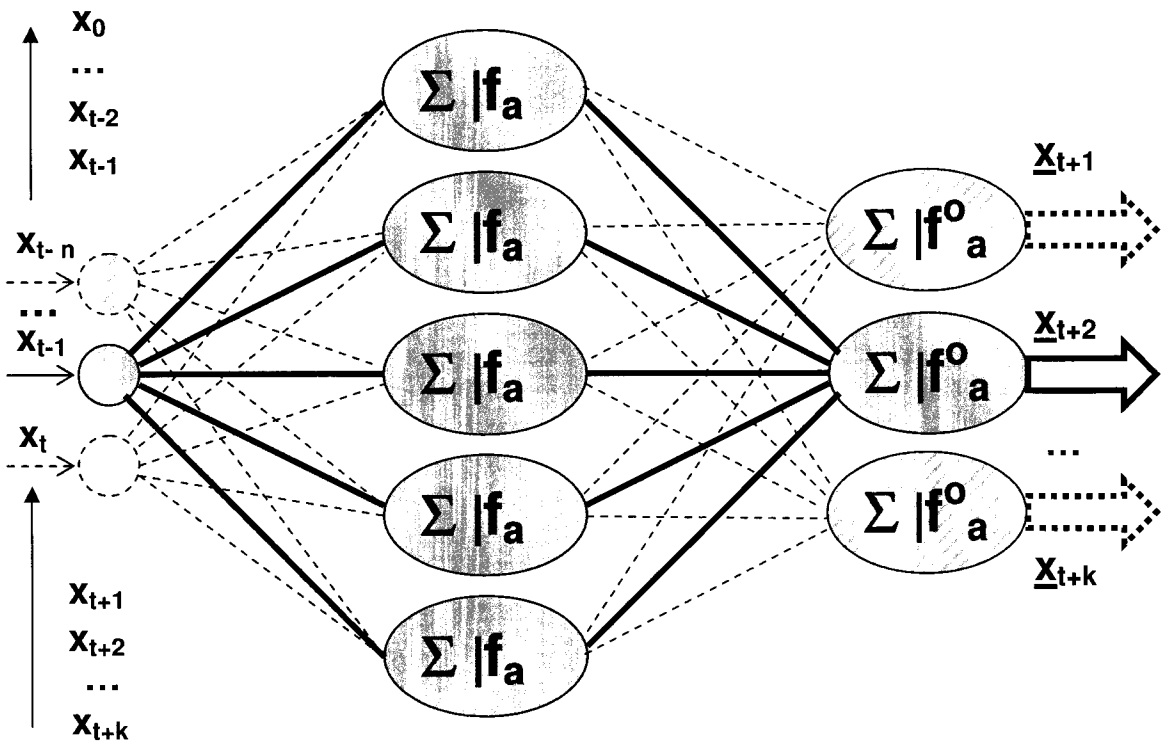


Fig.6. Multiple forecast neural network time series predictor (Bias is omitted).

A time-series forecasting model based on the flexible neural tree were introduced by Y. Chen et al [17, 18], allowing optimization of the neural net structure via evolution by mutations. The neuro-fuzzy systems were explored to integrate neural networks, fuzzy inference systems and evolutionary search procedures [19]. An incremental cascade network architecture based on error minimization were explored [20]. Demonstrated were advantages of the local approximation technique for non-linear function approximation [21]. The Support Vector Machines were reported as a viable alternative to a neural approach in the time series forecasting [22]. A direct adaptive output feedback control design procedure was developed for highly uncertain nonlinear systems that do not rely on state estimation [23]. Number of research efforts was dedicated to multi-step prediction algorithms [24 – 30], among which the neural approaches included B-spline

interpolation and adaptive time-delay neural network [24], superimposed noise modeling in the neural systems [25] and co-evolutionary recurrent neural network [28].

At the same time, the parallel programming approaches were in the focus of research efforts due to an expected increase in efficiency of iterative processing in the parallel computational environment. On this end the parallel evolutionary asymmetric subset-hood product fuzzy-neural inference system has been developed to take advantage of parallelization in message passing [31]. Another efficient effort to materialize advantages of the parallel geometry was in implementing a holographic neural predictor which was demonstrated for fractional Brownian motion predictions [32]. A multi-core environment as one of the recent trends in the computer architectures also was part of the parallelization research in the neural network algorithm area [33]. Such an effort is a continuation of the earlier push for efficiency increase when adopting the parallel computational environment [34] from Sharp Corp. The implementation of neural networks on FPGAs has several benefits, with emphasis on the parallelism and the real time capabilities, which had been addressed at Sapientia Hungarian University of Transilvania [35]. An attractive alternative of the parallel environment offers the Cell broadband engine (Cell/B.E.) multiprocessor recently produced by IBM [36]. The potential of the Cell broadband engine for modeling the large size neural networks has recently been addressed by Per Hasselström of Stockholm University [37], in which optimization of the model was sought via distributing the large computational task over the multiple processing units. Such potential of the Cell/B.E. is intrinsically present in its architecture and the simplest example of that is seen in the fast matrix multiplication [38].

The neural predictors have potential also in video compression via predicting quantification parameter of discrete cosine transform coefficients as shown in [39]. The neural classifiers were demonstrated to be efficient in improving characteristics of the video encoder [40].

2.4. Potential deficiencies of the gradient descent algorithm

Despite its continuous popularity [2 - 30], the gradient descent algorithm poses various implementation difficulties for the neural networks training in a single processor computing environments. The following list includes some major problems with the gradient descent [1], which have inspired various research efforts for more sophisticated versions of it and/or alternative algorithmic approaches: (1) The gradient descent algorithm does not guarantee reproducible results. (2) The systems to which gradient descent is applied to can be too complex, so that the convergence of the algorithm becomes poor. (3) The gradient descent can be too slow. (4) The solutions offered by the gradient descent can be non-optimal.

The first problem stems from the fact that typically the algorithm starts from setting up initial parameters, such as input weights of the neurons, to small random values. Moreover, the preferred also is the random order of the training patterns supplied to the neural network within the algorithm. These two sources of randomness are essential to ensure versatility of the algorithm in general, but it does poses an obstacle in reproducing once obtained results.

The second problem occurs when the task requires a complex system, for which the convergence of the gradient descent becomes negatively affected by sophistication of the

system. As an example of that can be multi-dimensional neural networks, such as those of multiple-input-multiple-output type, for which the gradient descents in various dimensions may contradict to each other.

The third problem may occur when the complexity of the task allows only slow convergence and/or for the large networks, where the number of the connections requires prohibitively high number of iterations to maintain the convergence. Alternatively, the problem of the speed of convergence may be the result of the lack of clarity in the task on what the step size in the gradient descent should be. Adopting infinitesimal step size would make the time frame to be indefinitely long to complete the algorithm, which is not acceptable. However what exactly the size of the step should be in each particular case is unclear and needs to be found first, which in itself requires extra-resources. The number of algorithms had been developed for adjusting the step size during the run time. However many of those sophisticated algorithms bring in the sensitivity to noise or negatively affect scalability, or both [1]. Consequently, the gradient descent approach still is used and it remains to be an important robust tool, where only minor improvements with a simple momentum heuristic address the adjustment of the step size and thus the speed of convergence.

The fourth problem, the optimality of the obtained solutions, is due to entrapment of the gradient descent trajectory in the local minimum. There are many optimization algorithms designed to address the local minima entrapment problem, such as variational learning, annealing [1]. However none of it offers guarantees against the local minima entrapment as well as versatility of the gradient descent algorithm keeps it as preferred method in many practical tasks, particularly those with on-chip learning [13, 35].

2.5. Cell Broadband Engine as a tool for parallel processing of algorithms

The Cell Broadband Engine (Cell/B.E.) has been recently introduced [41 - 47] by IBM as a new class of multi-core processors (Fig.7). The Cell/B.E. processor extends the trends in processors development by offering network of Synergistic Processor Elements (SPE) on the board, operating under command of Power Processor Element (PPE), as shown in Fig.8.

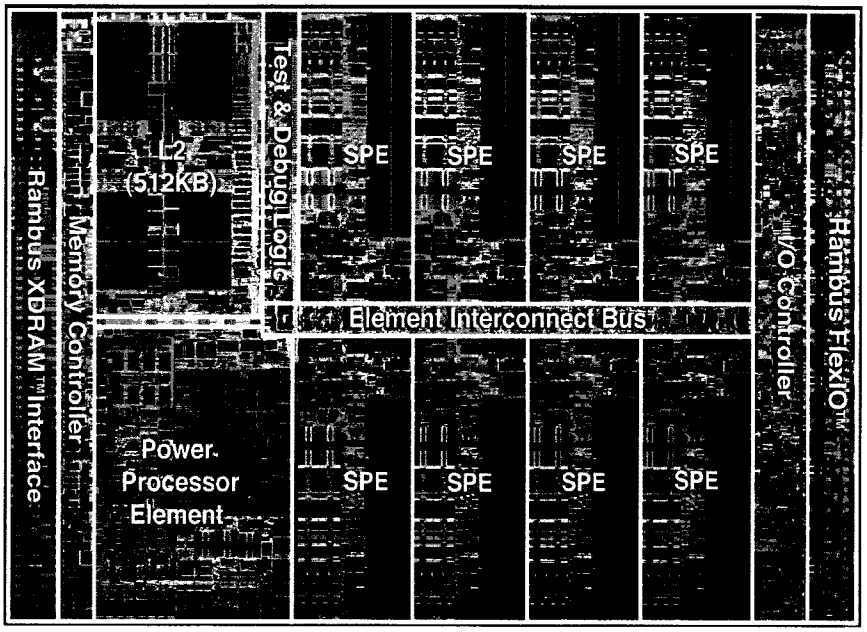


Fig.7. Hardware of Cell Broadband Engine (Cell/B.E.).
(http://domino.research.ibm.com/comm/research_projects.nsf/pages/multicore.CellBE.html/%24FILE/cellbe.jpg)

The SPEs are adapted to efficiently run computational tasks, while the PPE is more suitable to perform the controlling tasks. The Cell/B.E. can perform multithreading tasks of concurrent tracing of various trajectories of the numerical algorithms, in particular the gradient descent algorithm, which is in the basis of backpropagation training of the feed-forward neural networks.

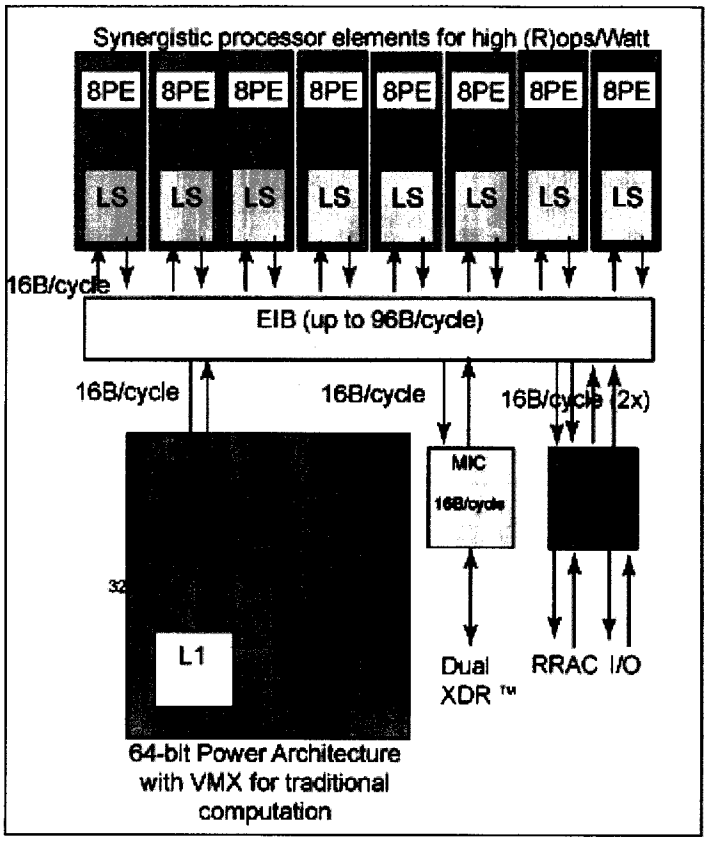


Fig.8. The architecture of the Cell/B.E. processor from IBM.

(http://www.ibm.com/developerworks/power/library/pa-celltips1/)

The exploration of tracing the parallel trajectories with neural network's algorithms has its history, summarized by LiMin Fu [48]. It stems from the pioneering work of Gevins and Morgan (1988) [49] and that of Casselman and Acres (1990) [50] for processing data with neural networks, which, as had been emphasized by LiMin Fu [48], represented two principally distinct parallelization strategies: (i) connecting different networks in parallel so that each network extracts different features from the same data [49] (see Fig.9) and (ii) breaking up a complex problem into a number of subtasks to be solved by different networks [50] (see Fig.10).

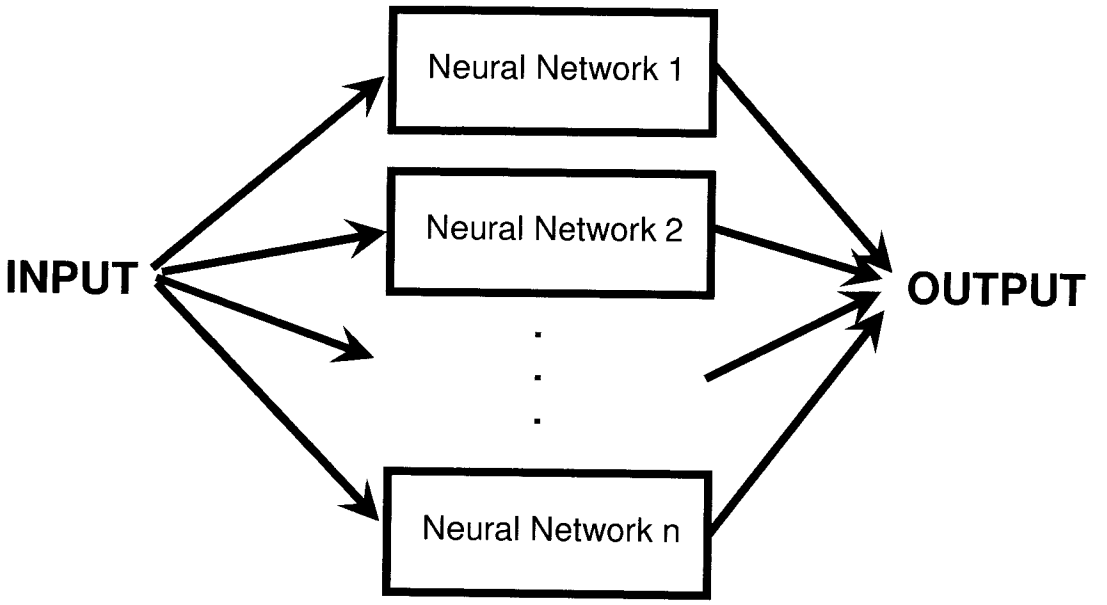


Fig.9. Parallel network model without central control (reproduced from [48]).

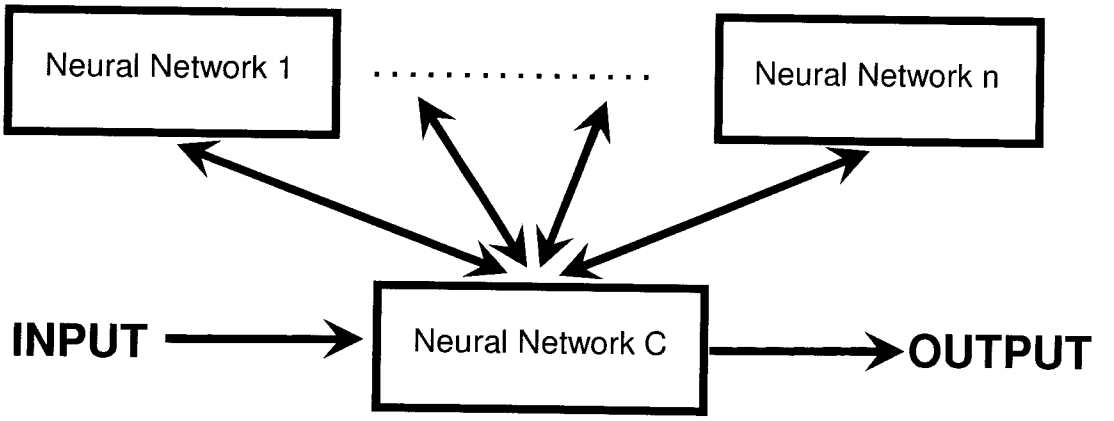


Fig.10. Parallel network model with central control (reproduced from [48]).

According to LiMin Fu [48], the usefulness of approach (i) is in the possibility of making different analyses of the same data (extracting more information as a result). The first implementation of such approach by Gevins and Morgan (1988) consisted of applying different networks to detect different types of the contaminants in the brain's EEF signals [49]. The approach (i) may potentially result in unexpected discoveries due to

implementation of the multiple networks, because unexpected favorable combinations may occur.

Alternatively, in the approach (ii), the results from different networks may be combined in order to reach the solution of the task. For instance, Casselman and Acres (1990) [50] had employed a parallelization scheme (ii) to conduct different diagnostic tasks of satellite spectral data. Essential here is the fact that the parallelization scheme (ii) allows obtaining a solution for the task, whose complexity is too high for a single network of equivalent size. In other words, the parallelization (ii) enables to reach a solution not available otherwise.

The application of approaches (i) and/or (ii) has so far not been considered with regard to 2D non-linear functions approximation as well as to 1D time series prediction problems, which is in the core of consideration herewith. For nonlinear function approximation, the obvious benefit of parallelization (i) is the extensive coverage of the starting conditions for gradient descent algorithm, which allows address the problem of potential trapping in local minima (and identify the most efficient trajectory to the global minimum). This will be discussed in Section 4.

Both schemes of parallelization may benefit the task of the time series prediction, because such task contain several dimensions suitable for parallelization, among which the obvious ones are: 1) varying the initial set of the time values for prediction; 2) varying the length of the forecast; 3) varying the depth in the history to rely on in making the forecast; 4) varying the structure and/or the parameters of the predicting neural network. The first three advantages will be explored in details in Section 5. As a hardware for parallel computing environment, the IBM's BladeCenter® QS22 based on

the innovative multi-core IBM PowerXCell 8i processor, a new generation processor based on the Cell/B.E. architecture, has been used in this work.

Chapter 3. Research Question Formulation

This thesis identifies and demonstrates the advantages offered by the parallel programming environment of the Cell/B.E. for the implementation of the gradient descent algorithm (in terms of speed and accuracy of the attained solutions of 2D non-linear functions approximations and 1D time series predictions by multilayer perceptron neural networks).

A parallel computational environment offers potential of extra power, which traditionally is used to improve the speed and the accuracy of the calculations. Such potential is readily available from the Cell/B.E. multiprocessor [36, 41 - 47]. In the field of neural networks, this is useful for the case of large-scale networks [37], where a large task can be split into several smaller sub-tasks, each of which is assigned to a separate SPE. Another well known example of large-scale tasks implemented on the Cell/B.E. is large matrix multiplication [38]. In both last cases the gain was in engaging combined computing power of the SPE processors to solve sub-tasks, which initially large whole task was divided into. On the other hand, for small scale tasks (sufficiently small for a single processor to efficiently handle), there are two commonly used parallelization schemes originated in [49] and [50], which had been reviewed in Section 2. Possible applications of these schemes to neural networks were shown in Fig.9 and Fig.10 and, up-to-date, neither of them had been explored. Single processor environment has limitations in specific implementations of the neural approximations of 1D and/or 2D nonlinear functions. Specifically, the gradient descent algorithm is known to be vulnerable to the local minima traps [1 - 12]. For a single processor the consequences of such entrapment

are double fold: (1) unproductive loss of the computational time; (2) uncertainty as to the optimality of the solutions found. Both of them can be addressed by a parallel computing environment via employing extra processors. Still, the efficiency of this solution need to be observed and therefore specific considerations are needed to determine under which circumstances and to which extent such brute-force approach is an adequate way to address the posed risks of losing the optimal solutions.

A parallel computing environment offers extra dimensions that can be used to boost the efficiency of the neural network application for the nonlinear function predictions. Two main aspects of the predictions are in the center of attention – speed and accuracy. Timely and accurate predictions are key in the successful targeting of applications involving neural forecasting in time series (such as stock markets). The same is true for the 2D non-linear function approximators, where controllers require accurate and speedy conclusions about properties of the 2D function underlying the input data in order to adequately manage the system. The limitations of single processor environment for employing gradient descent algorithm in neural predictions and approximations tasks become disadvantageous in quickly changing environments, because the changes may render the previous training of the neural network forecasters to be no longer useful and require either fast and/or efficient re-training or switch to the pre-trained forecaster from the stored ones, or combinations thereof. The sequential handling of those tasks with a single processor carries a risk of delivering the solutions too late and may become unacceptable. A parallel computing environment may face the required time cost with much better efficiency and therefore the aspects of the advantages available from its employment deserve a detailed study to account for in the applications. Availability of

the parallel computing environment from the portable sources, such as the multi-core processors, the Cell/B.E. multi-processor, the FPGAs, render such consideration to be beneficial for various kinds of the parallel environments due to potential applicability of the principles developed to all of them to at least some, if not equal, extent.

Moreover, even though the Cell/B.E. was used in this thesis as platform to support parallel computing environment, changing the platform itself would still keep the principles of parallelization intact (i.e., the same principles may be employed later in other systems supporting a parallel environment, such as with multi-core processors [33], computer network architectures [34], or FPGA hardware systems [35]). There are multiple applications in which the increased efficiency of the neural network algorithms can be beneficial. This includes adaptive control systems with non-linear actuators [9 - 12], such as systems with one of the following non-linearities: a friction, dead-zones, a backlash, time delays, and for which 2D nonlinear functions approximators are needed. The applications for 1D time series prediction includes those with the chaotic time series prediction, such as the forecasters of the frame or visual object plane sizes of MPEG-coded streams [14], the Mackey-Glass time series [15 – 30], the Brownian motion prediction [32], the stock and financial market forecasters [56 – 64].

The concept of tracing the parallel trajectories was recently formulated [65] for bio-chemical reaction simulation approach with Stochastic Simulation Algorithm. A trajectory in Stochastic Simulation Algorithm is a time-ordered sequence of states of the chemical system. For example, in a model with 4 species such as the metabolite-enzyme model, each state is a tuple (ordered set) of 4 integers. Each element's value in the tuple is the population of the corresponding species. At any time, the state of the system

possesses Markov property, i.e. depends only on the current state and not on any previous state. This is equivalent to that the system has no memory of any earlier state. Trajectories of the system in this case are Markov chains over the state-space of the chemical system. The state-space for such system consists of all possible combinations of populations of the species. Each trajectory in this case is a random walk through the state-space of the system. The idea in Stochastic Simulation Algorithm is to start with some amount of each species at initial time moment (set to be zero on the time scale) and then advance in small random steps through the time. The random numbers are used to decide which reaction should occur at each time-step (as in Markov chains) until we reach the stopping conditions (i.e. the time frame decided on). Such an individual simulation, which tracks the amounts of each species at each time-step, will represent a trajectory of the system. However, each individual trajectory represents only one possible way of the system advancement. In order to obtain a fully representative picture of system's behavior, many thousands of trajectories are needed. To accomplish the task, a power of parallel computing environment were individual processors may accommodate simulation of one possible trajectory is a suitable alternative to implement the simulation of multiple trajectories. A Cell multiprocessor had been suggested for this purpose by Emmet Caulfield and Andreas Hellander and implemented in multiplatform compiler for the Cell/B.E. and x86 processors [65]. However, this approach had not been explored in the area of neural networks, even though it does have potential to address the bottleneck of the grtadient descent algorithm approach, which is vulnerable to occurance of unfavorable staring conditions. This further contributes to justification of the research question formulation in this Thesis.

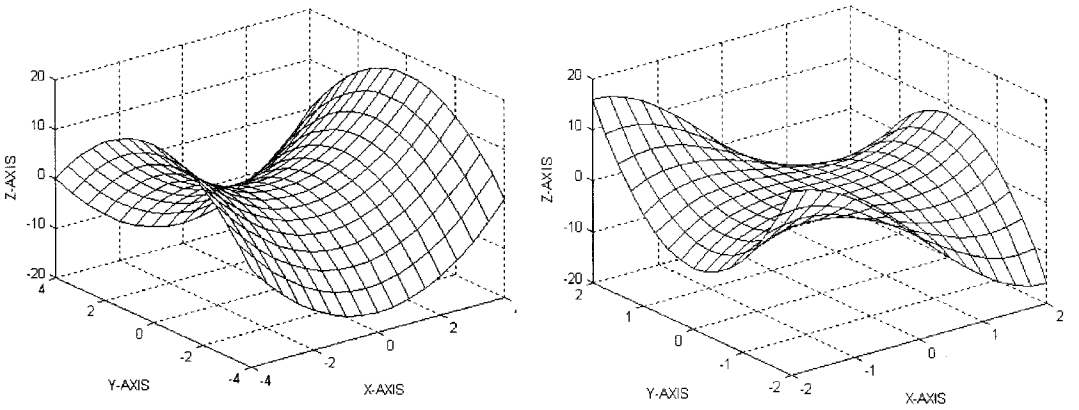
Another example of multiple trajectories generated for various starting points and simulated in massively parallel computing environment is represented by implementation of Magnetic Pendulum Fractal Simulation by René Müller [66]. The task of a magnetic pendulum movement over n magnets is described by a system of differential equations. The solution of the coupled differential equations is implemented numerically using Verlet leap-frog integration. The potential and kinetic energies of pendulum are computed for every iteration step. Stopping conditions for simulation requires the system energy to drop below a threshold E_T. As simulations had shown, a slightly different origin produces different trajectories, so that to see the full range of capabilities the dense coverage of the starting conditions is needed, which brings multiple trajectories as viable solutions. Software packages had been developed to implement this simulation in various parallel environments, such as Unix cluster consisted of 12 Alpha Workstations that ran on DEC Tru64 UNIX, Sony PlayStation3 with Cell/B.E.., demonstrating advantages of tracing various trajectories in the parallel computing environment. Similar approach is feasible for the gradient descent algorithm, which however was not explored so far and therefore may constitute part of the research question of this Thesis.

Chapter 4. Parallel tracing of multiple trajectories in gradient descent algorithm with Cell/B.E. multiprocessor

Efficient function generators are needed in the areas of signal processing systems, such as speech recognition [51], text-to-speech synthesizers [52], Optical Character Recognition (OCR), data mining, image compression, medical diagnosis, Automatic Speech Recognition (ASR) etc. Artificial Neural Networks (ANN) are recognized to be able of delivering efficient technical solutions to this problems [83 - 85]. The Cell/B.E. environment offers an efficient tool for algorithm implementation in parallel allowing for simultaneous tracing of multiple trajectories of such algorithms from randomly selected starting points, thus efficiently observing the diversity of possible solutions and allowing for the efficient selection of the better ones.

The first 2D non-linear function selected for this study was the standard saddle surface of hyperbolic paraboloid $z=x^2-y^2$, which is polynomial of the 2^{nd} order with saddle point as shown in Fig.11(a). The second selected function was defined by the equation $z = x^2 - 3*x*y^2$ and is polynomial of 3^{rd} order with saddle point as it is shown in Fig.11,(b). The choice of the 2D non-linear functions here is motivated to verify the concept with the polynomials of the lowest order of non-linearity, which also have saddle point as the center of the area of quasi-stability. While it is feasible that some practical system may require functions of that kind, in this Thesis the selected functions are not linked to specific application, but rather to their ability to provide features of non-linearity and area of quasi-stability.

The MLP's structure consisted of the three layers as that shown in Fig.3 with 3 inputs (x, y, 1), of which two were for the variables and one for the bias. There was one hidden layer with tanh(.) activation function for the hidden neurons and linear output neuron, thus allowing 2D function generation. Particular parameters of the neural network used for simulation of 2D function approximator were as follows: (1) two inputs (n=2) for the variables X and Y (2) one input for the bias set to be "1.0" (so that a set of the input data is represented by variables X=x and Y=y and the bias, so that each node has weighted adjustable input and adjustable bias); (3) the network has one output, so that a mapping function is represented by variable Z=f(x, y).



(a) (b)

Fig.11. Graphical representation of 2D functions selected for neural approximation: (a) hyperbolic paraboloid $z=x^2-y^2$; (b) $z = x^2 - 3*x*y^2$.

4.1. Software operation in Cell/B.E. simulation environment

A simulation environment was implemented using a multithreaded synergistic mode on the Cell/B.E. The idea is that the PPU initiates asynchronous threads of the gradient descent algorithm on the available SPU's concurrently. The diagram in Fig. 12 shows

operation of the software designed for tracing parallel trajectories of the gradient descent algorithm. The program written in C/C++ has been developed for Linux (Fedora 9) using IBM's Cell SDK 3.0. The program included four files: *main.c, m_spu.c, dimensions.h* and *control_block.h*. The data were organized in the structure, DataStruct{.} specified in the *control_block.h* file. *DataStruct* includes variables used for the transfers between PPU and SPUs, among which there were data for input and output training points, i.e. the sets of values for X, Y and Z, as well as the sets of values for neuron's input weights. The dimensions.h file contained values for the parameters used in the MLP training algorithm, among which there were *numInputs* for the number of MLP's inputs, including bias, *numOutputs* for the number of outputs, which here was set to be equal to 1, *numPatterns* for the number of patterns used in the training, *numHidden* for the number of hidden nodes of the MLP, *numEpochs* for the number of the training epochs. The algorithm also included the learning rates as parameters (specified in *dimentions.h* file): LR_IH for the input-hidden nodes connections, *LR_HO* for the hidden nodes-output connections, *LR_IO* for the input-output direct connections, as well as the momentum coefficients – *alphaHO*, *alphaIO* and *alphaIH* with the same notations for *HO*, *IO* and *IH* indices. The MLP's training algorithm was placed in the *m_spu.c* file to enable concurrent training of various MLPs by SPE processors. As it is indicated in Fig.12, the program starts by loading in the data. There are three sets of values for 2D function variables in the program: (1) the set for training, including X, Y and Z; (2) the set for the first level testing, including X_t, Y_t and Z_t; and (3) the set for comprehensive testing, which included values for X_d, Y_d and Z_d. Unlike X, Y and Z, the variables X_t, Y_t, Z_t, X_d, Y_d and Z_d were not included into the Datastruct{.}.

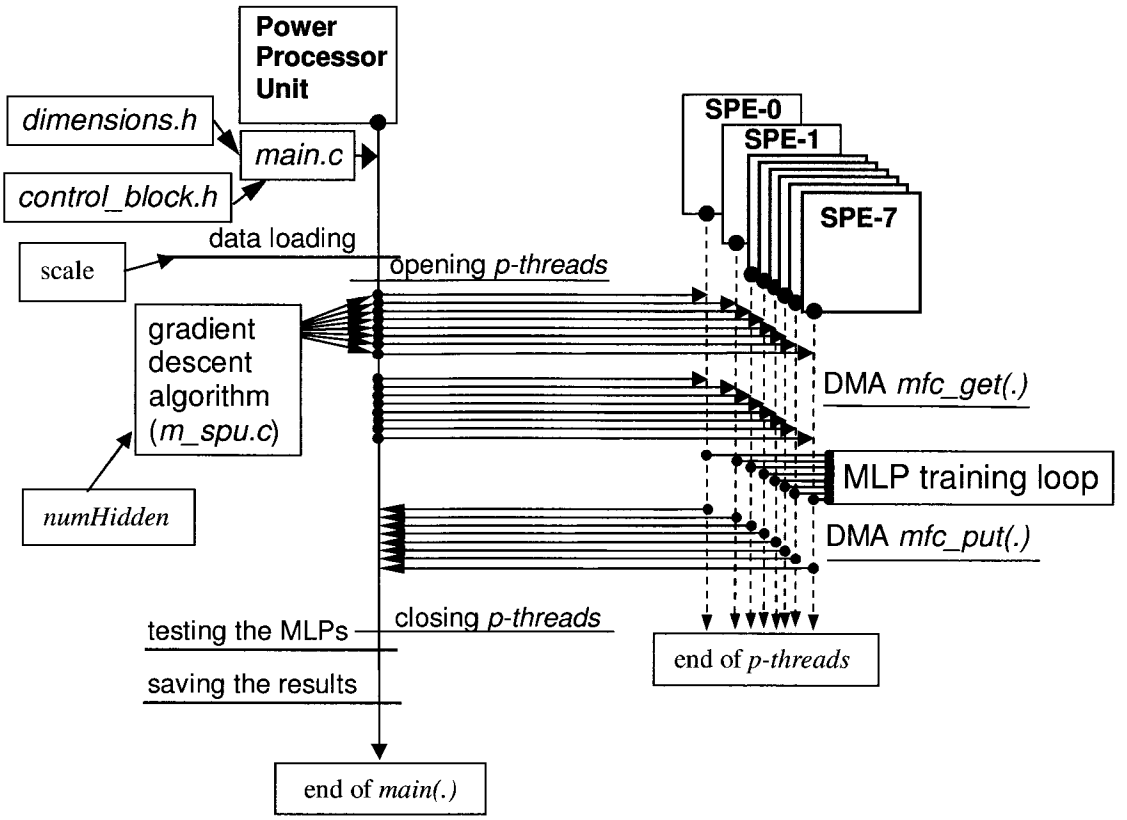


Fig.12. Software operation diagram for tracing parallel trajectories of the gradient descent algorithm for 2D function approximation. .

It is because the tests involving X_t, Y_t, Z_t, X_d, Y_d and Z_d are part of the procedures listed in the main.c file, all of which are ran by the PPU. Contrary to that, the variables X, Y and Z are part of the algorithm listed in the m_spu.c file for the gradient descent trajectory tracing conducted by the SPEs, which therefore required X, Y and Z to be placed in *DataStruct{.}* to enable transfers between the PPU and the SPEs via Direct Memory Access (DMA). The first level of testing was conducted for the points (X_t; Y_t), which were equally distanced from the closest neighbor points (X; Y). The comprehensive testing of resulting MLPs was implemented on points (X_d; Y_d), which apart from (X; Y)

and (X_t; Y_t) also included all intermediate cross-reference points, namely (X; Y_t) and (X_t; Y), as well as close-proximity interpolation point extending outside training region for half of the distance between two closest (X; Y) points. Specific coordinates for the training as well as both testing sets of points are listed below for each of the experiments. The data loading stage shown in Fig.12 also included assignment of the starting values for the input weights of the MLP's neurons. Such assignment was implemented by using either a random choice of relatively small values (<<1) or a specially chosen set of pre-assigned values. The last case was needed for verification of reproducibility of the trajectories. Parallelization can be started at this point via varying the scale of the quantization step for the input values, so that training of the MLPs can be conducted at different scale factors.

After completion of the data loading, the program proceeds to initiating p-threads by loading SPEs with the instructions generated by *m_spu.c*'s code (Fig.12). Threads are asynchronous, which is provided by *pthread(.)* function of the Cell SDK package and which interrupts the main program execution until completion of all the initiated threads. The part of the program in the *m_spu.c* file contains the declared structure of the MLP, so that as parallelization parameter the number of hidden neurons, *numHidden*, can be varied for the SPEs to allow algorithm convergence verification for various values of the parameter *numHidden*. This would start the number of specified in the program asynchronous p-threads at specified number of SPEs, each implementing the same algorithm but for a different number of the hidden neurons in the MLPs. Otherwise for the case of 2D functions the instructions are the same for each SPE and as a first step they start from the data loading into the local store via DMA transfer from the main memory

employing the function *mfc_get(.)*. Similar instructions are for all SPEs with the only difference being in the starting point for the gradient descent, which is stored in the variable of input weights for the neurons. Thus the parallel trajectories would reflect diversity of the gradient descent routes originating from the various starting points.

Once started, each SPE's thread would run in asynchronous mode to enter the training cycle for its MLP. The algorithm includes screening the available input/output mapping relations in order to calculate the respective error which current MLP produces at its output. From the obtained errors the updating increments are derived for the input weights of the neurons so that updating the weights reduces the overall root mean square error of the MLP as 2D function approximator. At the end of each updating cycle the program calculates and reports the current RMSE attained for the given epoch, ε. The sequence of these reported ε values forms the trajectory of the MLP training, which is observed in the conducted experiments as the dependence $\varepsilon = \varepsilon(numEpochs)$. Termination of the training cycle and thus the end of the trajectory occurs when either the specified accuracy of the approximation or the allowed number of training epochs is reached. In Fig.12 this corresponds to ending the MLP training loop and proceeding to initiating DMA transfer of the resulting parameters of the MLP, which are kept in the structure *DataStruct{.}* back to the main memory of the PPU. The function *mfc_put(.)* is employed to carry out this operation.

In asynchronous mode, the PPU waits for all the p-threads to be completed in order to proceed with the programmed testing of the obtained MLPs. The two above mentioned tests are based on the sets $\{X_t; Y_t; Z_t\}$ and $\{X_d; Y_d; Z_d\}$, which brings about the respective

RMSE values E_t and E_d to be saved along with the obtained parameters of the data structure *DataStruct{.}* thus completing the program.

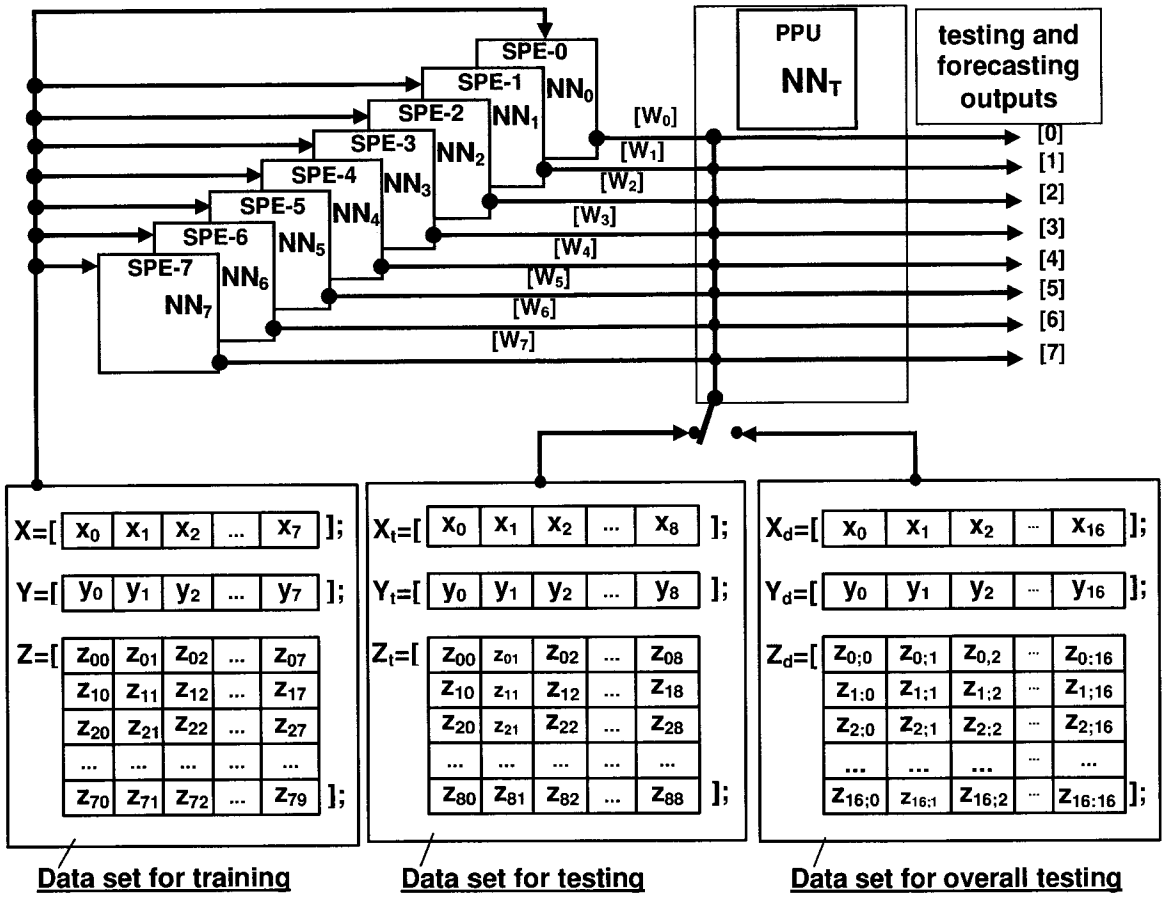


Fig.13. Parallelization scheme and software operation for tracing the trajectories of the gradient descent algorithm to obtain 2D function approximator with the neural networks.

A parallelization scheme for software operation used in tracing parallel trajectories of the gradient descent algorithm of neural 2D function approximator is shown in Fig.13. Three

sets of data were loaded into the main memory: (i) data set {X, Y, Z} for training of the MLPs included vectors X =[x_0, x_1, x_2, ..., x_7] and Y=[y_1, y_2, y_3, ..., y_7] of (x; y) coordinates and matrix Z=[z_{00}, z_{01}, z_{02}, ..., z_{07}; z_{10}, z_{11}, z_{12}, ..., z_{17}; ...; z_{70}, z_{71}, z_{72}, ..., z_{77}] of the 2D function values supplied to SPEs to implement gradient descent algorithm; (ii) data set {X_t, Y_t, Z_t}of testing points with vectors X_t =[x_0, x_1, x_2, ..., x_8] and Y_t=[y_1, y_2, y_3, ..., y_7] of (x_t; y_t) coordinates chosen as most remote ones from the training points, and with the matrix Z_t=[z_{00}, z_{01}, z_{02}, ..., z_{08}; z_{10}, z_{11}, z_{12}, ..., z_{18}; ...; z_{80}, z_{81}, z_{82}, ..., z_{88}] of the 2D function testing values; and (iii) data set {X_d, Y_d, Z_d} for comprehensive testing which included coordinates (x_d; y_d) of points presented by vectors X_d =[x_0, x_1, x_2, ..., x_{17}] and Y_d=[y_1, y_2, y_3, ..., y_{17}] and function values Z_d=[$z_{0;0}$, $z_{0;1}$, $z_{0;2}$, ..., $z_{0;17}$; $z_{1;0}$, $z_{1;1}$, $z_{1;2}$, ..., $z_{1;17}$; ...; $z_{17;0}$, $z_{17;1}$, $z_{17;2}$, ..., $z_{17;17}$] for comprehensive testing with intermediate points inclusive. The training sets {X; Y; Z} were equally distributed to all SPEs to run the gradient descent algorithm from random starting points. The random starting points were generated by supplying to each SPE the matrix [W_i], where SPE's index i=0, 1, 2, ...M; (M+1) being the maximum number of SPEs (in Fig.13 it is shown to be 8) of input weights for the hidden and the output neurons, which thus represented the essential information about the structure of the MLP under training. The elements of the matrices [W_i] were initiated by the PPU at small random values thus selecting the starting points for each SPE and stored in the main memory. Initiation of the gradient descent algorithm was accompanied by the DMA transfer of the data sets {X, Y, Z) and the matrices [W_i] to the local memory store of each SPE-i. Running the gradient at SPE-i, i=0, 1, 2, ..., M, resulted in updating the values of the matrices [W_i]. After completion of the gradient descent algorithm by either satisfying the conditions for the required RMSE value ε, or

by exhausting the allocated resource of the training epochs, controlled by the parameter numEpoch (both parameters were programmed in the file control_block.h of the program), the final versions of the matrices $[W_i]$ were DMA transferred from the SPE's local stores back to the main memory for the testing procedures (as shown in Fig.13 by the arrows from the SPEs to the PPU). These $[W_i]$ matrices contained the resulting weights information for neuron's inputs, which the MLP marked in Fig.13 as NN_T was using for the final testing procedures. Two testing procedures were used in the program – one for the points represented by the set $\{X_t, Y_t, Z_t\}$ and which were most remote from the training ones, and another one, the overall comprehensive test, for the points represented by the set $\{X_d, Y_d, Z_d\}$, which additionally included intermediate points. These two testing procedures were conducted in the program sequentially, which was provided by the switch shown in Fig.13. The results of the testing procedures were then directed to the output [0] – [M] (M=7 in Fig.13) of the program together with the data of matrices $[W_i]$, thus enabling further analysis of the obtained function approximators, such as graphic analysis with Matlab program to visualize the approximated 2D surfaces as well s error distributions.

The parallelization scheme shown in Fig.13 is close to the one represented by Fig.10 as the control module of the program does contain the testing neural network NN_T to conduct the comparison of the resulting solutions and selecting the most optimal one based on supplied information from the SPEs about training trajectories as well as results of the tests by the network NN_T in the central control module of the program.

4.2. Simulation results for hyperbolic paraboloid $z=x^2-y^2$ case

For the hyperbolic paraboloid, the MLP's training algorithm converges with a minimum of 7 hidden neurons and the training points sets of X=Y=[3.5 2.5 1.5 0.5 -0.5 -1.5 -2.5 -3.5]. The tests below have been conducted for the number of hidden neurons *numHidden*=10 for improved convergence.

Table 1 shows the data for the parallel tracing of the training trajectories on 8 SPEs (namely SPE-0 through SPE-7). The targeted precision value was □=10^{-5}, after achieving which the training was terminated (or continued until reaching the limit of max training cycles otherwise). The first column of the Table 1 shows the level of the training reached in terms of conducted training sessions for MLPs , i.e. the number of epochs spent in the training. The rest of the columns in the Table 1 show the reached by the MLPs of each SPE level of accuracy, represented by the RMSE value, ε. For each training level listed there is a winner by the reached accuracy level, i.e. the MLP demonstrated the lowest RMSE thus far. Graphically the same data are presented in Fig.14 (a-d) as tracings of trajectories followed by SPEs in training.

Some of the trajectories (such as those the SPE-0, the SPE-2, the SPE-6 and the SPE-7), exhibit the desired "diving" behavior, i.e. finding and following a quick descent track to converge and deliver smallest RMS Error values at smallest computational cost (in other words, requiring minimal number of epochs for training of the neural net). Others (such as the SPE-4 and the SPE-5), after a quick start, saturate the progress at certain level of accuracy in function approximation. In this example, the SPE-5 did not reach targeted precision level of ε=10^{-5} for the allocated training interval, while the SPE-4 did achieve

$\varepsilon=10^{-5}$ but at much higher calculating cost than the pack of leaders (the SPE-0, the SPE-2, the SPE-6 and the SPE-7).

Table 1. Training progress of the MLP on various SPEs of the Cell/B.E. (RMSE vs. *numEpochs*) for the hyperbolic paraboloid function. (Rate 2500 epochs/sec.)

epoch	SPE-0	SPE-1	SPE-2	SPE-3	SPE-4	SPE-5	SPE-6	SPE-7
0	6.48	6.48	6.48	6.48	6.48	6.48	6.48	6.48
70	2.35	1.45	1.11	2.022	1.3825	1.9625	1.65	1.7925
125	1.38	1.5	0.4275	1.633333	1.78333333	0.8325	1.625	1.4625
191	1.2225	1.45	0.25	1.625	0.2425	0.1925	0.18625	1.345
280	0.7175	1.4225	0.1925	1.4225	**0.06975**	0.1824	0.04475	0.7875
400	0.25	1.3025	0.09625	1.5275	0.04225	**0.0445**	0.05525	0.885
540	**0.08075**	1.19	0.0295	1.475	0.02175	0.03225	0.025	0.7375
1164	0.0165	0.98	0.001965	1.282	**0.006275**	**0.003175**	0.0099	**0.011**
1675	**0.007525**	0.9955	0.00090675	1.46	0.001122	0.002175	0.004825	**0.003375**
1755	0.00435	0.875	0.00040125	1.285	0.0011165	0.0032245	0.003275	0.003
2775	0.001266	1.05	0.00008625	1.385	**0.00017875**	0.0018	0.00097	**0.000485**
3171	**0.0006435**	1.25	0.00007825	1.54	0.00025075	**0.00095**	0.0004915	0.0002585
4635	**0.00007575**	1.135	0.00004925	1.35	0.00011725	0.00096175	0.0001485	**0.00002225**
4951	0.00004775	1	0.00004075	1.335	0.00010025	0.0008945	0.00011525	0.00001475
5087	0.000038	0.93	0.00003625	1.355	**0.00009125**	0.0010355	0.000105	0.00001
6785	0.00001	1.11	0.00002275	1.37	0.000124	0.001183	0.00003025	
7410		1.36	0.00001	1.485	0.0000785	0.0006735	0.0000175	
7745		1.14		1.14	0.00007625	0.0007015	0.00001	
15240		0.93		1.545	0.0000435	0.00031675		
18775		0.94		1.645	0.00003775	0.00027825		
25345		1.015		1.8	0.00002925	0.00022475		
43540		1.025		1.29	0.00001725	**0.00003725**		
48298		1.1		1.32	0.00001	0.00004475		

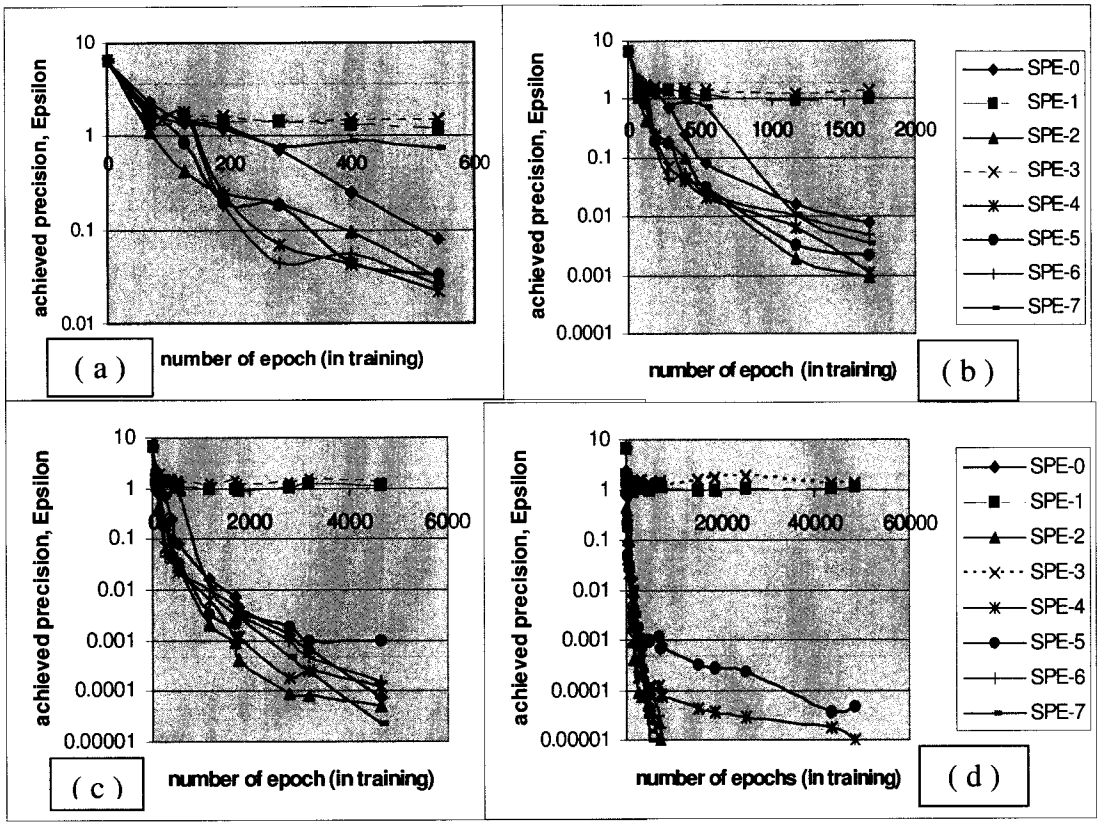


Fig.14. Trajectories of the SPE's training on the Cell/B.E. in achieving the levels of precision ε equal to: (a) 0.1; (b) 0.01; (c) 0.001 and 0.0001; (d) 0.00001. Windows (a), (b), (c) and (d) show various parts of the same trajectories at different magnification. Training rate is 2500 epochs/sec.

The resulting function approximation differs not only in the value of ε reached, but also in the error distribution, which can be characterized by evaluating separately RMS Errors for the training points (ε), the testing points (E_t) and the full set of all points (E_d). Herewith for hyperbolic paraboloid the training points sets of X=Y=[3.5 2.5 1.5 0.5 -0.5 -1.5 -2.5 -3.5] have been used, while the testing points have been X_t=Y_t=[4.0 3.0 2.0 1.0 0.0 -1.0 -2.0 -3.0 -4.0] and the sets of overall evaluation points have been X_d=Y_d=[4.0 3.5 3.0 2.5 2.0 1.5 1.0 0.5 0.0 -0.5 -1.0 -1.5 -2.0 -2.5 -3.0 -3.5 -4.0]. Table 2 shows the case, in which three winning the SPEs (the SPE-1, the SPE-2 and the SPE-6) achieved the

designated level of the precision prior to expiration of allocated iterations limit of 5400 epochs. Among them there is a clear winner, which exhibited best value at each parameter, including ε, E_t and E_d while spending smallest amount of calculations, i.e. 1419 epochs as compares to 1874 for the SPE-1 and 2151 for the SPE-5. The training tracks which lead to this outcome are shown in Fig.15. It is seen that three SPEs (the SPE-0, the SPE-4 and the SPE-7) got trapped in the local minima and did not progress. Their traces in Fig.15 remained at shallow level of RMS Error of ca. 1.0. Two SPEs , namely the SPE-2 and the SPE-3 there, managed to escape the entrapment and eventually reached appreciable RMS Error level of less than 10^{-2}.

Table 2. Training results for the MLP in approximating hyperbolic paraboloid function $z=x^2-y^2$ to achieve either precision ε=Eps=0.001 or maximal number of training epochs of 5400. (The input parameters see in the Appendix B).

	SPE-0	SPE-1	SPE-2	SPE-3	SPE-4	SPE-5	SPE-6	SPE-7
RMSE, ε training	0.897521	0.000974	0.006523	0.004064	1.576885	0.000954	0.000999	0.760430
RMSE, Et, testing	2.779444	2.079857	1.989340	1.979378	2.677969	1.978961	1.879715	2.690461
RMSE, E_d overall	2.250431	1.637820	1.570713	1.524210	2.276174	1.505976	1.435486	2.150663
number of epoch	5400	1874	5400	5400	5400	2151	1419	5400

The next experiment has been designed to verify the stability of the observed trajectories. The source of instability in the algorithm is the randomness of the pattern sequences in the training epochs. There are total of 8*8=64 different patterns in this training experiment (as variables X and Y have 8 values each in the set), each pattern

corresponding to one combination of values (x, y, z) representing one point on the surface of 2D function. In the training algorithm one epoch consists of 64 patterns, which are selected randomly from the initial set of total 64 various patterns. Within each epoch the selected patterns are ordered randomly too. Such randomness is meant to avoid entrapment in local minima and to increase chances of convergence. However the trajectory of the convergence may deviate from trial to trial even when originating from the same starting conditions due to remaining randomness of the order of the patterns offered to MLP in the training. Deviation of trajectories from the original track may be stochastic and small enough to keep within its close proximity and eventually converging to the same global minimum, in which case the trajectory is seen as a stable one. In the alternative case, the deviation from the original track may be significant and lead to essentially different route of the convergence, rendering the trajectory to be unstable. For the experiment depicted in Fig.14, the stability of the trajectories can be verified by repeating this experiment from the same starting conditions (i.e. the same values of the initially assigned weights for the neuron's inputs) but keeping random order of the pattern's flow in the training. The result of this experiment is shown in Table 3 and Fig.16. While the overall trend remains similar to that in Table 2 and Fig.15, the trajectories differ, showing some instability. For example, the SPE-6 on a second run abandoned the leader's group, while still delivering good performance at the end. The rest of the trajectories where not far from that of the first run as well as sufficiently close by the end results on error values, as can be seen by comparing the RMSE values for respective trajectories in Table 2 and Table 3. .

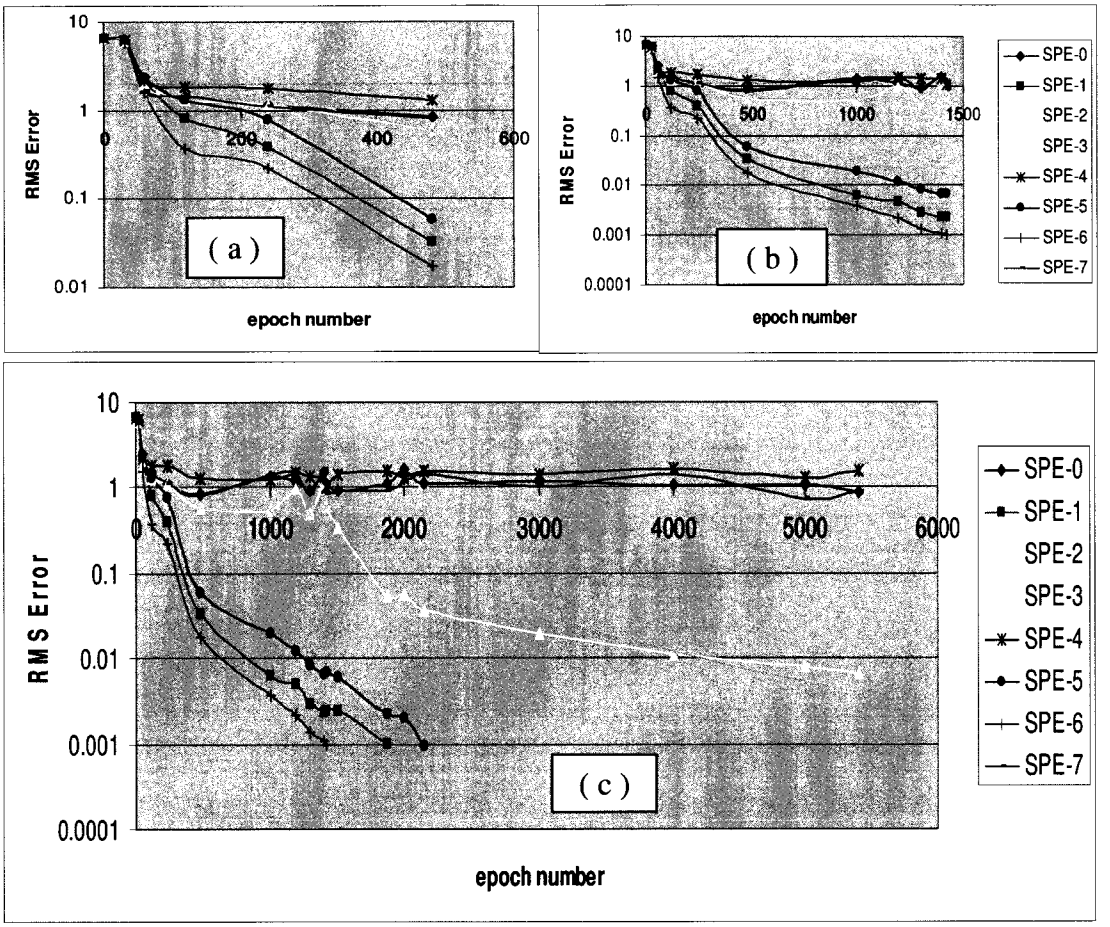


Fig.15. Gradient descent tracks of the SPE's neural networks training, which lead to RMS Error values shown in Table 2. Windows (a), (b) and (c) show different parts of the same trajectories at various scales. Training rate is 2500 epochs/sec.

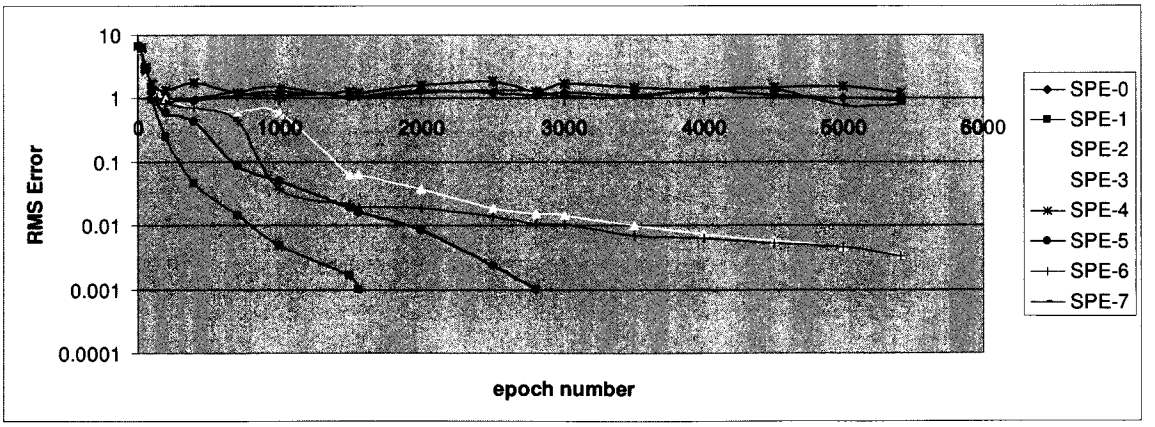


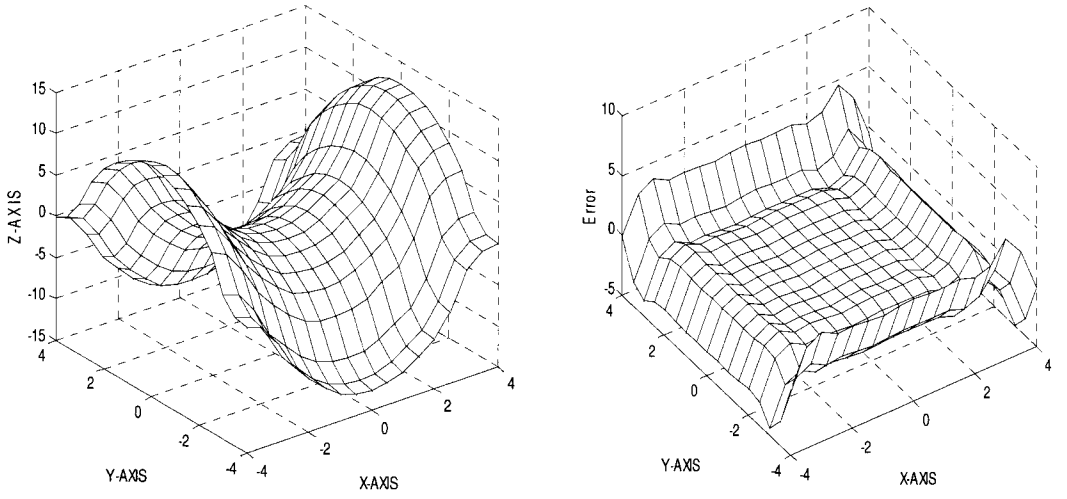
Fig.16. Gradient descent traces for reproducing the same starting conditions as those in Table 2 and Fig.15, but with random selection of the pattern sequence in the epochs. Training rate is 2500 epochs/sec.

The overall winner of the last run (Fig.16) is the SPE-1, which delivered the function approximation presented in Fig.17,(a). Respective error distribution for that case is presented in Fig.17,(b). It is worth to note, that the minimal deviation from the original function is located inside the training area, while at the edges error tends to significantly increase, particularly in the extrapolation part of it, where MLP's approximation exhibits stronger vulnerability to errors. (The errors are evaluated as deviations from the ideal surface shown in Fig.11(a)).

Table 3. Results of reproduction of the same starting conditions as those in Table 2 and Fig.15, but with random selection for pattern sequence in the epochs left intact.

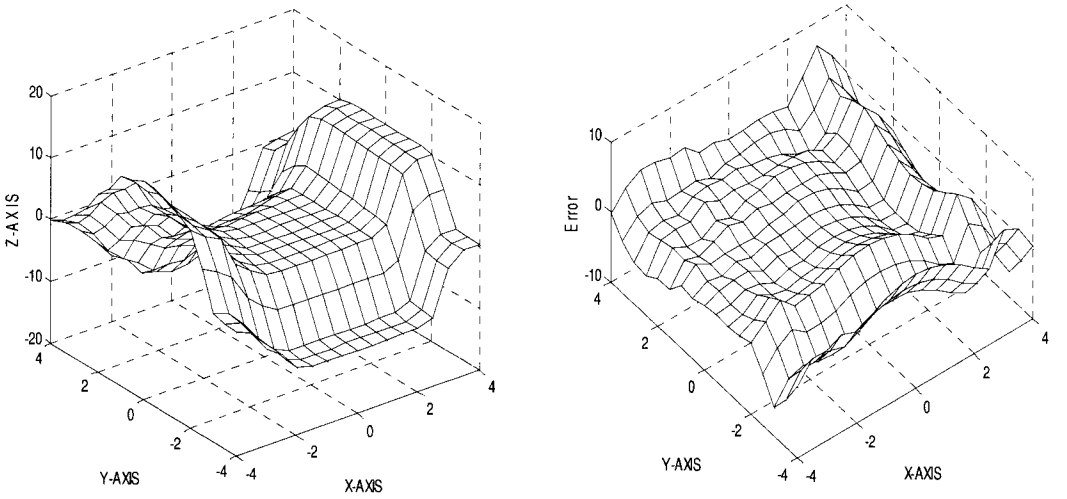
	SPE-0	SPE-1	SPE-2	SPE-3	SPE-4	SPE-5	SPE-6	SPE-7
RMSE, ε training	0.909161	0.000977	0.003556	0.011714	1.199642	0.000993	0.003308	0.808976
RMSE, Et, testing	2.776967	2.078992	1.994234	2.053515	2.707388	1.953186	1.970799	2.678789
RMSE, E_d overall	2.24974	1.633622	1.54502	1.605672	2.184742	1.479834	1.495387	2.165177
number of epoch	5400	1563	5400	5400	5400	2803	5400	5400

The worse performance in the run reflected in Table 3 and in Fig.16 is shown by the SPE-4, which was trapped at the shallow level of precision. The surface of hyperbolic hyperboloid ($z=x^2-y^2$) approximated by the MLP trained on the SPE-4 is shown in Fig.18,(a), in which string distortions are seen.



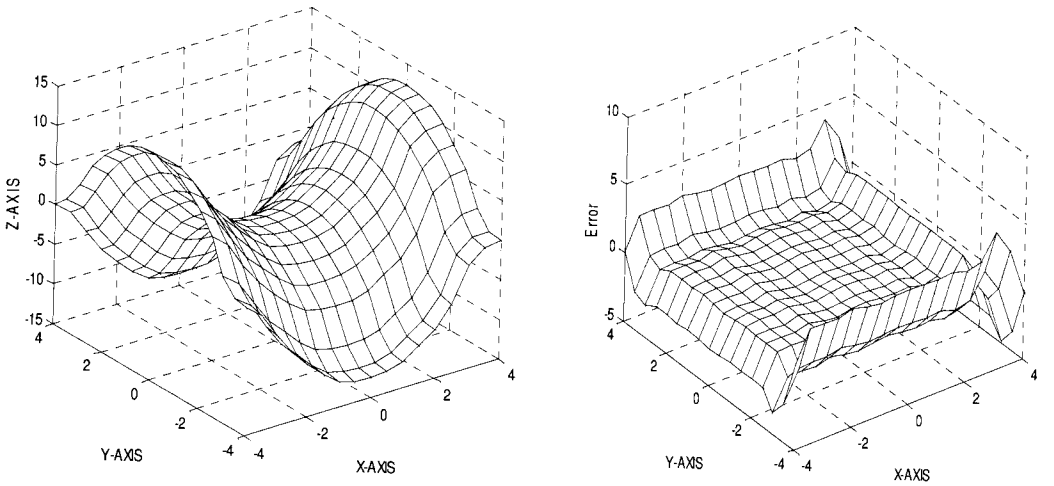
(a) (b)

Fig.17. Results of the MLP's training delivered by the SPE-1 from Fig.16 and Table 3: (**a**) hyperbolic paraboloid's ($z=x^2-y^2$) approximation and (**b**) the error distribution for it.



(a) (b)

Fig.18. (**a**) Highly distorted approximation of $z=x^2-y^2$, delivered by the SPE-4 from Table 3 and Fig.16 and (**b**) the error distribution for it.



(a) (b)

Fig.19. (**a**) The approximation of hyperbolic paraboloid $z=x^2-y^2$ by MLP on the SPE-5 (from Table 3) and (**b**) the error distribution for it (i.e. deviation from the original $z=x^2-y^2$ surface shown in Fig.11,(a)).

As we can see, the strong errors have affected the internal area of the area of training. It is also instructive to compare the final result delivered by second runner-up of Table 3 and Fig.16, the SPE-5, which closely matched the quality of the function approximation to that of the winner, the SPE-1, but at almost twice higher computational cost (2803 epochs by the SPE-5 compare to 1563 epochs taken by the winner, the SPE-1). The surface of hyperbolic paraboloid approximation rendered by the MLP on SPE-5 (from Table 3) is shown in Fig.19,(a) with respective error distribution shown in Fig._23.

The tests reveal that significant enhancement of the quality of the approximation can be achieved by scaling down by factor of 2 the distance between training points, i.e. for X=Y=[1.75 1.25 0.75 0.25 -0.25 -0.75 -1.25 -1.75]. This case is illustrated in Fig.20,

where the enhancement of the edges is seen as compared to that in Fig.17,(a), which was the best approximation in its series.

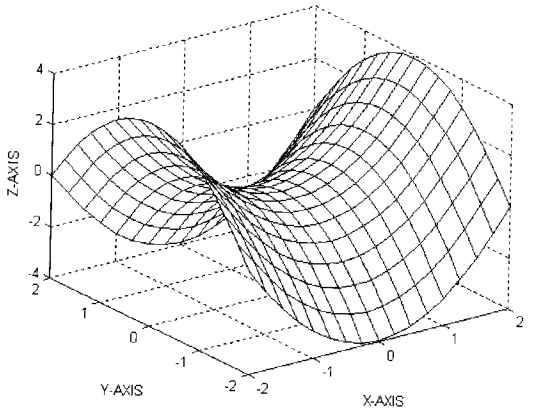


Fig.20. Neural approximation of the hyperbolic paraboloid by MLP with 10 hidden neurons and the training points sets of X=Y=[1.75 1.25 0.75 0.25 -0.25 -0.75 -1.25 -1.75]; ε.= 0.000999.

4.3. Simulation for $z = x^2 - 3*x*y^2$ case

For the case of $z = x^2 - 3*x*y^2$, the convergence of the gradient descent algorithm converges with the least number of hidden nodes being 10 and under training sets of points X=Y= [1.75 1.25 0.75 0.25 -0.25 -0.75 -1.25 -1.75], which has been followed by the test on the set of points $X_t = Y_t$ = [2 1.5 1.0 0.5 0.0 -0.5 -1.0 -1.5 -2.0] and finally evaluated overall at the set of points $X_d = Y_d$ = [2.0 1.75 1.5 1.25 1.0 0.75 0.5 0.25 0.0 -0.25 -0.5 -0.75 -1.0 -1.25 -1.5 -1.75 -2.0].

For a stable performance the number of hidden nodes needs to be slightly increased from its minimum value of 10 up to 12, the result of which is shown in Fig.21, where approximated is the function $z = x^2 - 3*x*y^2$ from the Fig.11,(b) at the accuracy level characterized by the RMSE for training ε= 0.050009.

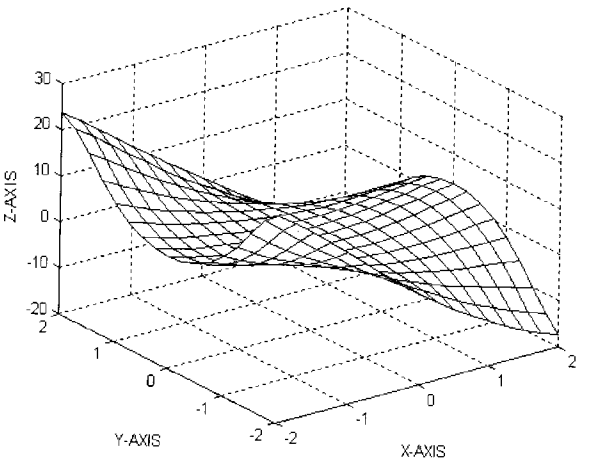


Fig.21. The function generated to approximate $z = x^2 - 3*x*y^2$ with 12 hidden neurons and the training points sets of X=Y=[1.75 1.25 0.75 0.25 -0.25 -0.75 -1.25 -1.75]; ε= 0.050009.

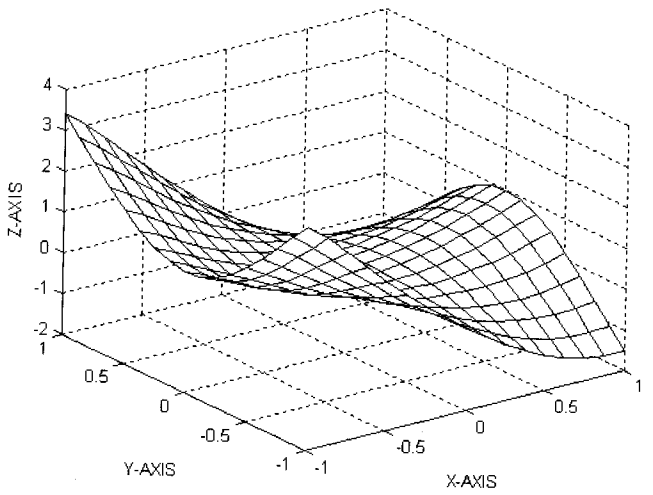


Fig.22. The function generated to approximate $z = x^2 - 3*x*y^2$ with 12 hidden neurons and the training points sets of X = Y = [0.875 0.625 0.375 0.125 -0.125 -0.375 -0.625 -0.875]; ε.= 0.050001.

Scaling down the distance between training points by factor of 2 also gives good results for the hyperbolic paraboloid as it is illustrated in Fig.22. Coordinates in question were X

= Y = [0.875 0.625 0.375 0.125 -0.125 -0.375 -0.625 -0.875]; $X_t = Y_t$ = [1.0 0.75 0.5 0.25 0.0 -0.25 -0.5 -0.75 -1.0] and $X_d = Y_d$ = [1.0 0.875 0.75 0.625 0.5 0.375 0.25 0.125 0.0 -0.125 -0.25 -0.375 -0.5 -0.625 -0.75 -0.875 -1.0].

Table 4 shows the data on MLP's training to approximate $z = x^2 - 3*x*y^2$ function in the Cell/B.E. environment by various SPEs, using training set of points X=Y= [1.75 1.25 0.75 0.25 -0.25 -0.75 -1.25 -1.75]. When comparing to the data for the hyperbolic paraboloid (Table 1), it is seen that the "diving depth" becomes at least 2 orders of magnitude shallower (1.E-3 instead of 1.E-5), which is attributed to the higher complexity of the function (a 3rd order non-linearity instead of a 2nd one). The earliest possibility to take advantage of the function approximator becomes available after 400 epochs from the SPE-4, which demonstrated the convergence of RMSE more than order of magnitude from its initial value (RMSE=**0.302902** in Table 4). The next order of magnitude improvement of RMSE is delivered at the cost of 25000 epochs, i.e. more than 60 times later, by the SPE-3 (RMSE=**0.021230** in Table 2). And the last order of magnitude improvement comes at a cost of 200 times more training, i.e. 500000 epochs, by the SPE-3 again (RMSE=**0.002463**), coming ahead of possibly trapped in local minima the SPE-1 (RMSE=**0.011991**), the SPE-4 (RMSE=**0.010353**), the SPE-5 (RMSE=**0.011089**), the SPE-6 (RMSE=**0.016755**) and the SPE-7 (RMSE=**0.010210**). Closest rivals were the SPE-0 (RMSE=0.007165) and the SPE-2 (RMSE=**0.003166**). Here the parallelization of the gradient descent algorithm also brought about two advantages – (1) an early possibility of having intermediate level of the function approximator, while still continuing to pursue (2) the improved version of it at a higher training cost.

Table 4. Training progress of the MLP to approximate $z = x^2 - 3*x*y^2$ function of Fig.11,(b) on various SPEs of the Cell/B.E.: the achieved root mean square error (RMSE) as a function of the number of training epochs. (Training rate is 2500 epochs/sec. The input parameters see in the Appendix B).

epoch	SPE-0	SPE-1	SPE-2	SPE-3	SPE-4	SPE-5	SPE-6	SPE-7
0	6.009953	5.945440	5.950127	5.990887	5.956882	6.008109	5.955856	5.994685
1	4.836313	4.681596	4.706274	4.778773	4.715749	4.821934	4.708642	4.815393
10	4.103190	4.109827	4.105010	4.107424	4.104908	4.113448	4.106591	4.099485
100	4.000068	3.910591	4.001476	4.001890	4.004594	4.009252	3.986856	3.997863
200	1.297651	**1.102347**	2.832628	3.984855	**1.131285**	3.975515	**1.128888**	2.387645
400	0.876783	**0.359232**	1.785159	1.538579	**0.302902**	4.057113	**0.333089**	0.836074
1000	0.715561	0.647003	0.378379	0.961722	0.413758	1.764666	0.365806	0.376895
2500	0.557923	0.191916	0.137183	0.155060	0.136230	0.289552	0.130241	**0.097422**
5000	0.193410	0.159165	0.123817	0.149576	0.196570	0.200460	0.117920	**0.073319**
10000	0.059317	0.093106	**0.037009**	0.065814	0.074203	0.126986	0.051245	**0.050357**
25000	0.029546	0.050399	0.028597	**0.021230**	0.039612	0.080636	0.036883	**0.022541**
50000	0.018220	0.024786	0.019204	**0.010966**	0.027617	0.085836	0.029983	0.019581
100000	0.018314	0.024303	0.012499	**0.009595**	0.042930	0.035079	0.048818	0.013050
250000	0.015588	0.032691	**0.003565**	**0.005046**	0.015257	0.026570	0.035521	0.012381
400000	0.014561	0.015149	**0.002409**	**0.002874**	0.007456	0.014684	0.011710	0.008363
500000	0.007165	**0.011991**	**0.003166**	**0.002463**	**0.010353**	**0.011089**	**0.016755**	**0.010210**
555000	**0.011785**	**0.00938**	**0.002317**	**0.001963**	**0.006767**	**0.006677**	**0.017233**	**0.008357**

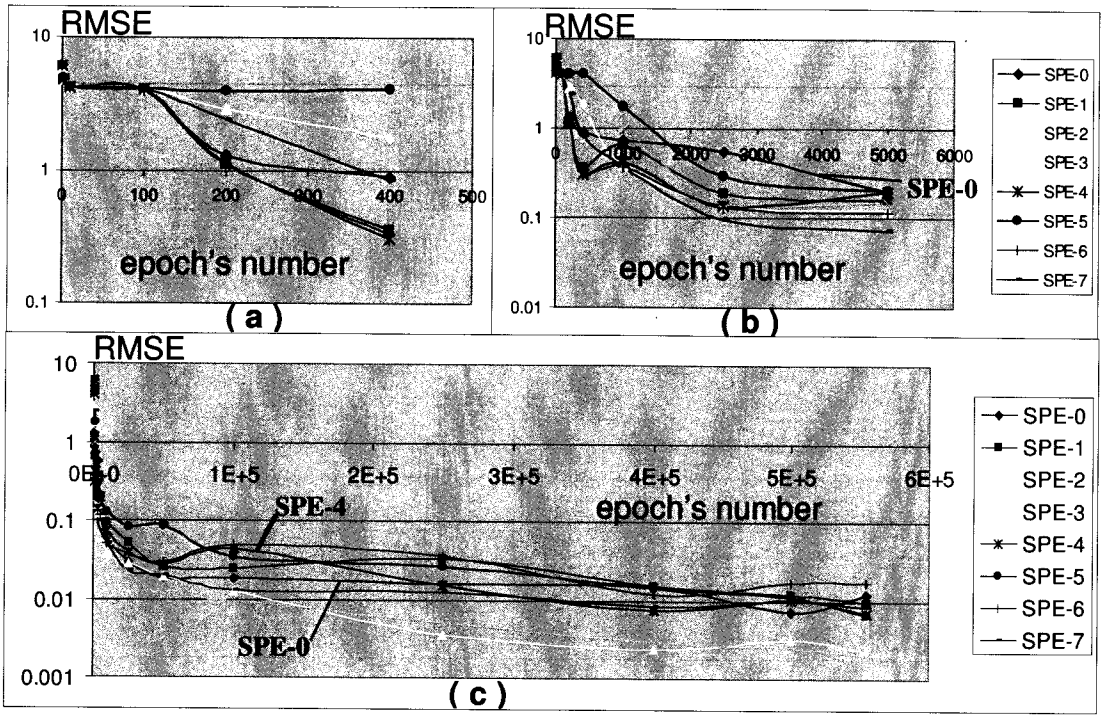


Fig.23. Trajectories of the MLP's training to approximate $z = x^2 - 3*x*y^2$ function of Fig.11,(b), as presented in Table 4. Windows (a), (b) and (c) present various views of the same trajectories at different magnification. Training rate is 2500 epochs/sec.

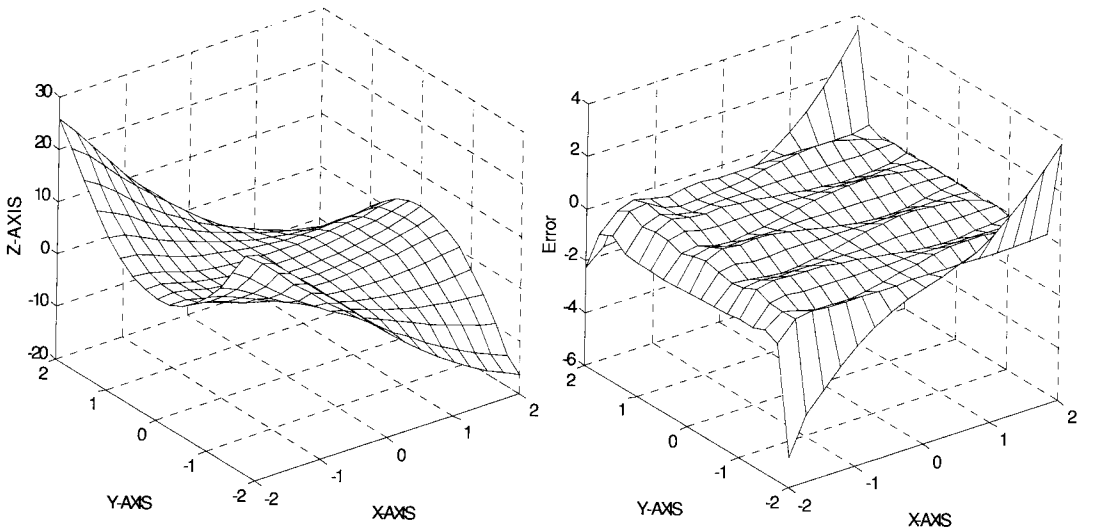
Table 5. Resulting RMSE values for training (ε), testing (E_t) and overall (E_d) evaluation of the delivered approximation of $z = x^2 - 3*x*y^2$ function by the session conducted in Table 4 and Fig.23 after reaching number of epochs *numEpochs* = 555000.

RMSE: ε, E_t, E_d	SPE-0	SPE-1	SPE-2	SPE-3	SPE-4	SPE-5	SPE-6	SPE-7
ε, training	0.011785	0.00938	0.002317	0.001963	0.006767	0.006677	0.017233	0.008357
E_t, testing	0.450612	0.649413	0.486791	0.901734	0.430226	0.619031	0.632174	0.510427
E_d, overall	0.383039	0.529887	0.490481	0.712765	0.586881	0.47007	0.614342	0.481186

The Fig.23 shows the trajectories of the MLP's training presented in Table 4 to approximate $z = x^2 - 3*x*y^2$ function (shown in Fig.11,(b)), the training being conducted on the sets of X=Y= [1.75 1.25 0.75 0.25 -0.25 -0.75 -1.25 -1.75]. It is seen that none of the trajectories are trapped in the local minima but rather all converge efficiently, splitting in the two major groups: 1) the SPE-2 and the SPE-3, which converge to lowest values of RMSE on training, $\varepsilon << 0.01$; 2) the rest of the pack (the SPEs ## 0, 1, 4, 5, 6 and 7), closely positioned at demonstrated RMSE values on the training session, i.e. $\varepsilon \simeq 0.01$. Based on the training results, the first group looks as that of the winners of the training race in Fig.23, which however appears to be misleading based on the following testing and verification of the overall performance, summarized in Table 5. The tests of the trained MLPs delivered by the SPEs and performed on the sets $X_t = Y_t$ = [2 1.5 1.0 0.5 0.0 -0.5 -1.0 -1.5 -2.0] have revealed that the leaders appeared to be the two MLPs from the second group, namely those delivered by the SPE-0 and the SPE-4 with RMSE E_t = **0.450612** and **0.430226** respectively. The overall evaluation on the sets $X_d = Y_d$ = [2.0 1.75 1.5 1.25 1.0 0.75 0.5 0.25 0.0 -0.25 -0.5 -0.75 -1.0 -1.25 -1.5 -1.75 -2.0] left only one leader namely the SPE-0, which showed the lowest RMSE overall value E_d = **0.383039**. The below Fig.24,(a) through Fig.26,(b) illustrate quality of the function approximation, delivered by the potential winners, namely the SPE-0, the SPE-3 and the SPE-4 selected based on the data of Table 5, which in turn already accounts for the argument derived from that of Table 4 and Fig.23. Here the SPE-0 holds a promise to be a winner based on the lowest E_d value of **0.383039** and close to the lowest E_t value of **0.450612**, while the SPE-3 have been the winner of the training session with the lowest ε value of **0.001963** and the SPE-4 showed lowest testing RMSE, E_t= **0.430226**. While appearance of the surfaces in

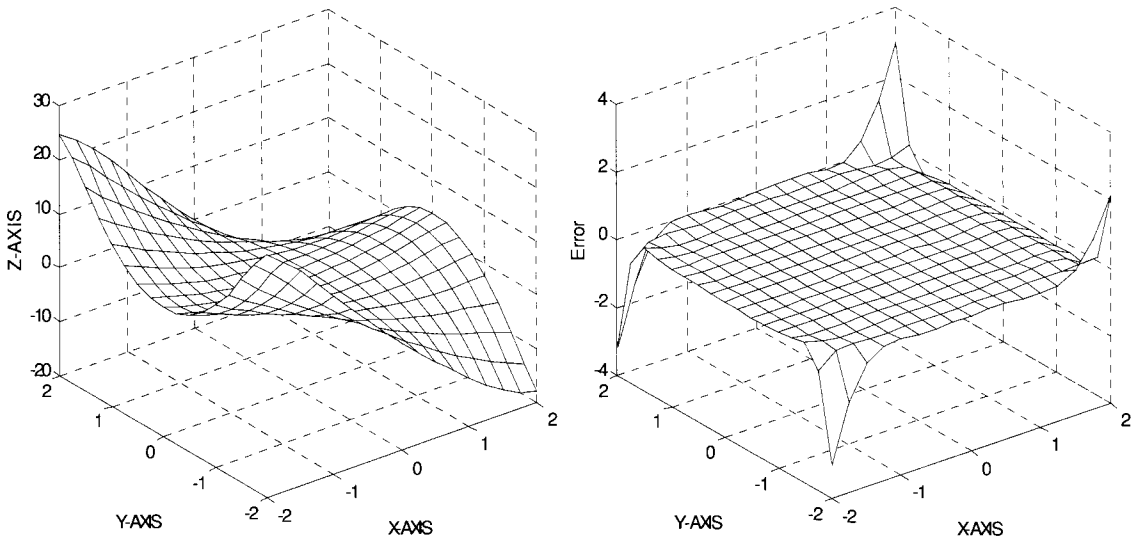
Fig.24,(a), Fig.25,(a) and Fig.26,(a) does confirms that approximation is tends to approach that of the sought function $z = x^2 - 3*x*y^2$, it still does not reveal the actual quality of the achieved approximation. The last issue is best addressed by the plots of distribution of errors on the approximated surfaces as compared to ideal one from the Fig.11,(b), which are depicted in Fig.24,(b) for the SPE-3, Fig.25,(b) for the SPE-0 and Fig.26,(b) for the SPE-4. A clear winner here is the SPE-0, which is indicated by several features: 1) the lowest E_d value, i.e. lowest overall RMSE in surface approximation; 2) error distribution in Fig.25,(b) appears to be obviously more uniform than that of the SPE-3 and the SPE-4; 3) the extrapolation close proximity area is represented with the smallest distortion. Noticeable here is the uniformity of the internal area of the error surface in Fig.25,(b), which represents interpolation area (i.e. where the tested and the overall evaluation points represented interpolations of the training data). Most of the error is accumulated there in the corners, which represent extrapolations from the border training points and which are indeed expected to contain worse errors than the interpolation points. Moreover, even at the interpolation edges the function representation in Fig.25,(b) appears to be very accurate and explicitly better than that of the SPE-3 and of the SPE-4 in Fig.25,(b) and Fig.26,(b) respectively. It is therefore seen that among 8 SPEs only one, the SPE-0, actually delivers the approximator with a uniform error distribution in the interpolation area and the lowest overall RMSE value (E_d = **0.383039**). Trajectory for the SPE-0 in Fig.23 lies always in the middle of the pack, so that in the training session nothing indicates it to be a potential winner even on the completion of the training. Moreover, testing itself also appears to be insufficient to identify the SPE-0 as a clear winner. Only overall comprehensive testing clearly shows strong advantages of

the MLP delivered by the SPE-0 over its competitors. Under conditions when testing points are not available and training is conducted on the limited amount of the data from the plant, which normally is the case (otherwise the training set should be expanded accordingly), the parallel environment allows to attain set of the solutions with potential hidden winner, which can be identified at a later stage after overall in-line testing of all the obtained approximators (such as the one of the SPE-0 in the above example). Moreover, as it is seen in Fig.23, significant part of the training is completed at the stage of 25000 epochs with RMSE value of ε=0.029546 with the further progress to be very slow. Therefore, a parallel environment offers the advantage of having an early approximator with slightly suboptimal properties for immediate usage and concurrent continuing training to further optimization, if such still needed.



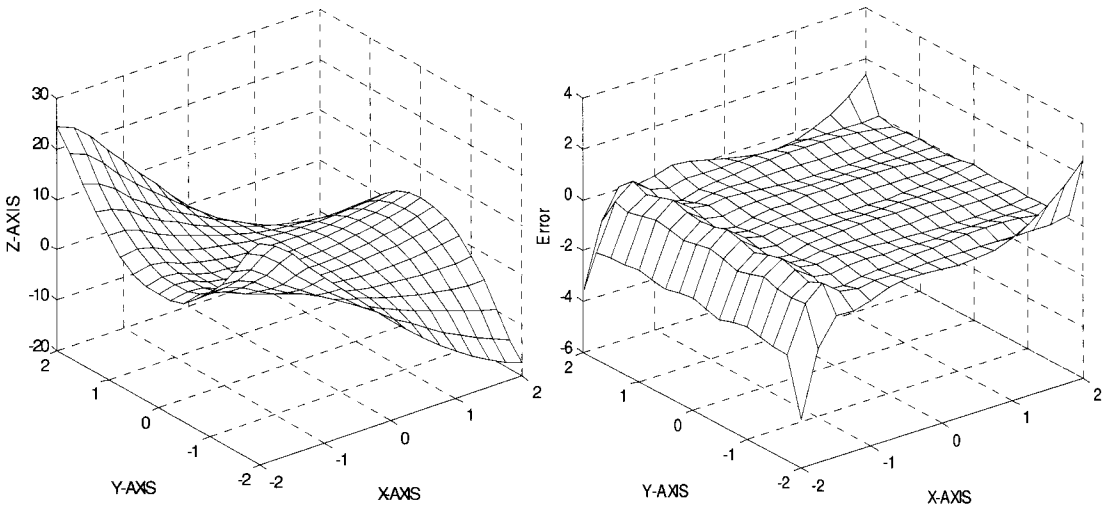
(a) (b)

Fig.24. (a) The surface generated by the SPE-3 (Table 4, Fig.23) to approximate $z = x^2 - 3*x*y^2$ function with 24 hidden neurons (*numEpochs* = 555000) and (b) the error surface associated with it.



(a) (b)

Fig.25. (**a**) The surface generated by the SPE-0 (Table 4, Fig.23) to approximate $z = x^2 - 3*x*y^2$ with 24 hidden neurons (*numEpochs* = 555000) and (**b**) the error surface associated with it (supporting the best performance).



(a) (b)

Fig.26. (**a**) The surface generated by the SPE-4 (Table 4, Fig.23) to approximate $z = x^2 - 3*x*y^2$ with 24 hidden neurons (*numEpochs* = 555000) and (**b**) the error surface associated with it.

4.4. Comparison with single CPU

A simultaneous tracing of the multiple trajectories in parallel via involving the power of multiprocessor's environment offers advantages in comparison with sequential tracing of the same or similar trajectories by a single processor (CPU) when there are obstacles in finding the optimal or close to the optimal trajectory, or even risk of getting trapped in the local minima, which is the case in the above examples. The benefit is therefore not in the speeding up the algorithm itself but rather engaging parallelization power in getting through the bottleneck set by starting conditions. Additionally, the statistical distribution of the trajectories becomes available in the parallel environment, such as that of the Cell/B.E., which brings about the confidence that the identified solution is close to the optimal one under given conditions. Therefore, while the same data can be obtained in the single processor environment by engaging multi-threading approach, the benefit of the parallel computations here in possibility of engaging higher computational power when the need is in obtaining optimal solution within limited time frame as Cell/B.E. provides possibility of multi-threading with multiple processors dedicating single processor for each thread. The number of processors of 8 used in the described experiments can be significantly increased depending on the hardware used. Additionally, the observed results are not Cell/B.E. specific in terms of that they should be valid also for other parallel computing hardware, such as multi-core, computer clusters, FPGAs.

4.5. Summary on approximations of 2D non-linear functions.

Summarizing the above consideration, we can see that the Cell/B.E. is allowing for efficient tracing of the multiple trajectories of the gradient descent algorithm, gathering

the best approximation conditions for 2D non-linear functions. The specific advantages achieved by employing a parallel computational environment of the Cell/B.E. for the gradient descent algorithm implementation of non-linear 2D function approximation of 2^{nd} and 3^{rd} order with saddle points are as follows.

1. Parallelization of the various starting conditions between the SPE processors allows coping with the encounters of trapping of the algorithm trajectory in the local minima, as well as identifying favorable starting conditions, which deliver a fast convergence track and thus minimize training calculation cost.
2. A close range extrapolation ability of the MLP's neural network is shown to reflect its interpolation capacity in that low root mean square error (RMSE) for close proximity extrapolation range correlates with low interpolation RMSE for selected 2D non-linear function approximator. In a parallel environment the criteria of obtaining reduced RMSE for close proximity extrapolation region may serve as an identification parameter to distinguish between the high quality approximator for interpolation region prior to or under limited possibility for comprehensive testing of the obtained in parallel tracing candidates for 2D non-linear approximator.
3. The optimal solution among delivered in parallel tracing candidates for 2D non-linear function approximation can only be identified in the aftermath via overall testing of the solutions, in which the winner does not necessarily coincide with that in training. Therefore it is advantageous to retain all the potential candidates from the SPE processors intact available for final determination of the winner, which contributes further to advantages of the parallel environment.

Chapter 5.

Time series prediction in the parallel Cell/B.E. environment

5.1. Task formulation for the time series prediction

The task of the present research is to explore potential of the Cell/B.E.'s parallel environment in order to improve a time series prediction in terms of it's speed and accuracy. Two types of the time series have been considered herwith: (a) a fully deterministic one with the long term trend; (b) a deterministic chaos time series. As a type (a), the series of linear combination of trigonometric functions has been designed. As a type (b) - a benchmark, the Mackey-Glass 30 deterministic chaos time series has been selected. Despite the extended literature covering the prediction problem for both types, with the long term trend one as well as for the Mackey Glass time series, advantages of the parallel computing environment for these tasks have only limited coverage, and specifically the Cell/B.E.'s contribution to the field is missing. A justification for the task of this research is due to the importance of having in-situ quick and accurate predictions, so that training of the forecaster would not render it to be outdated when it becomes ready to actually perform. A specific focus has been on (1) the short and (2) the long term forecasts based on immediate past as well as on (3) the long term forecasts based on the leverage of extended past.

5.2. Time series function with the long term trend

A time series predictor has been implemented on the Cell/B.E. architecture, in which individual SPE's processors compete to predict n (with n=1; 3 ; 5 or 10) steps ahead of

the 5 current values of the times series. The first test function of the time series x(t) has been constructed to represent an underlying long term trend as $x(t) = \text{int}\,[10^3 * \sin^2(t/2)] + \text{int}\,[10^3 * \sin^2(t/20)] + \text{int}\,[10^3 * \sin^2(t/30)] + \text{int}\,[10^3 * \sin^2(t/300)]$, where t=1, 2, 3..., T_b+t_p+n; and which is shown graphically in Fig.27 for $T_b=500$; $t_p=10$; n=5). For the time series prediction the structure of three layers MLP has included n=5 inputs, which tested the n consecutive values of the time series based on which the prediction of the future value k-steps ahead, $\underline{X}_{t+k}$, is obtained at the output, the actual value being x_{t+k}, so that Error= $x_{t+k} - \underline{X}_{t+k}$. A total number of basic t points was $T_b=500$, out of which the first 200 points formed the training set, while the rest 300 points were used for testing, i.e. verification of the predicting ability of the trained MLP. The maximal number of t points involved T_{max} also includes extra points above t=500 and is calculated via $T_{max} = T_b + t_p + n$, where t_p is the number of time-steps for the prediction.

The above mentioned in Fig.9 parallelization scheme (i) has been employed to trace the gradient descent of the training from various starting points.

For comparison with the above time series containing a continuous long term trend, another time series with the trend changing from positive to the negative one has also been constructed, as it is shown in Fig.28. For this series the training is performed within the same function as in Fig.27, while in the forecasting area the sign of the last term is changing to the opposite one (from plus to minus) as shown in Fig.28, thus representing the negating trend. Herewith the meaning of the negating trend includes retaining the analytical form of the trend, which is represented by the last term of the equation, namely $\text{int}\,[10^3 * \sin^2(t/300)]$, but taking it in the equation with the opposite sign, for which coefficient h(t) is changing the value from "+1" to "-1".

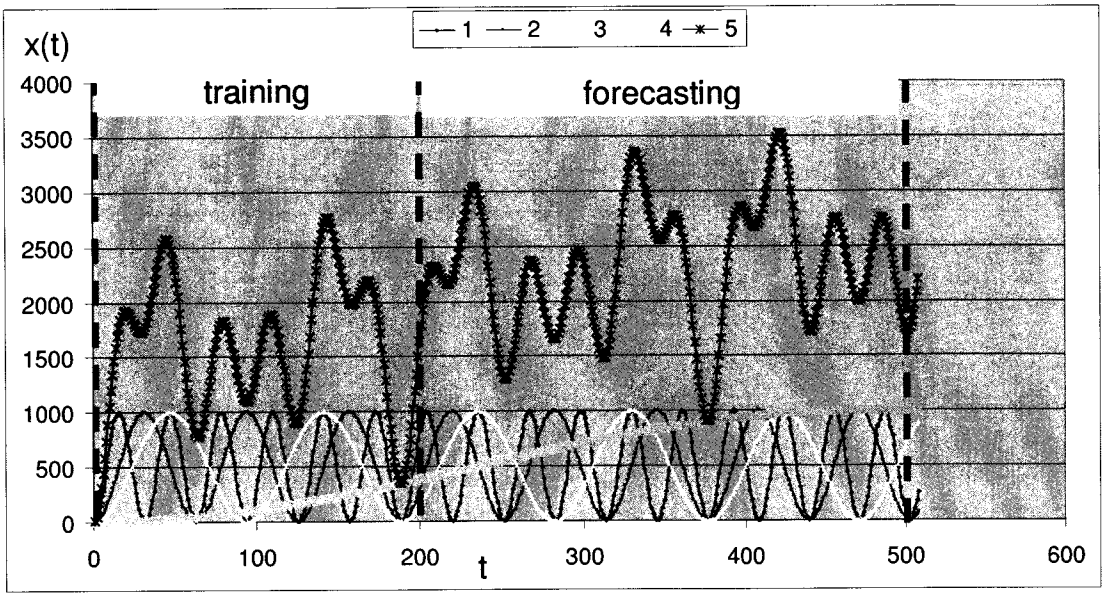


Fig.27. Time series of x=int [10^3 * $\sin^2(t/2)$] + int [10^3 * $\sin^2(t/20)$] + int [10^3 * $\sin^2(t/30)$] + int [10^3 * $\sin^2(t/300)$], where t=1, 2, 3..., T_b+t_p+n. The terms of the expression are shown as Series 1 through Series 4, while the full function is represented by Series 5. (Here T_b=500; t_p=10; n=5).

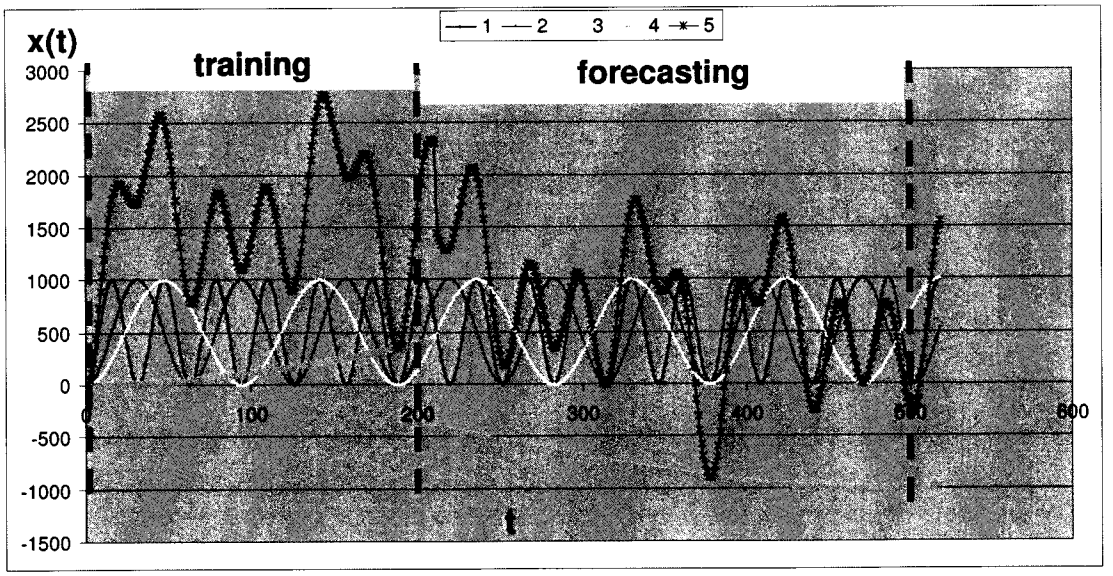


Fig.28. Time series of x=int [10^3 * $\sin^2(t/2)$] + int [10^3 * $\sin^2(t/20)$] + int [10^3 * $\sin^2(t/30)$] + h(t) * int [10^3 * $\sin^2(t/300)$], where t=1, 2, 3,..., T_b+t_p+n, and h(t)= {+1 for $1 \leq t \leq 210$; -1 for $t \geq 211$ }; (here T_b=500; t_p=3; n=5). The terms of the expression are shown as Series 1 through Series 4, while the full function is represented by Series 5.

It is essential here that the training set has not included the negating trend at all, so that in the forecasting area the predictor would face it for the first time. Mathematically this function could be described as $x(t) = \text{int}\,[10^3 * \sin^2(t/2)] + \text{int}\,[10^3 * \sin^2(t/20)] + \text{int}\,[10^3 * \sin^2(t/30)] + h(t) * \text{int}\,[10^3 * \sin^2(t/300)]$, where $t=1, 2, 3,\ldots, T_b+t_p+n$, and $h(t)= \{+1$ for $1 \leq t \leq 210$; -1 for $t \geq 211\ \}$.

5.3. Mackey-Glass as deterministic chaos time series

In the area of time series the Mackey–Glass series represent the sequence of deterministic chaos events, which mathematically are based on nonlinear time delay differential equation of the first order:

$$\frac{dx}{dt} = A \times \frac{x(t-h)}{1 + x(t-h)^{10}} - B \times x(t)\ ; \qquad A, B, h > 0 \qquad (5.1)$$

where A and B are real numbers, and h represents time delay, so that the value of the variable x at the time t is connected with that for the time (t - h) by means of differential equation (5.1). Depending on the values of the parameters, this equation displays a range of periodic and chaotic dynamics, which brings about the time series of values x(t) via numerical solution of the equation (5.1), usually obtained by the Runge–Kutta method of 4^{th} order. The Mackey-Glass time series have been generated by Matlab under parameters A=0.2, B=0.1 and h=30 and it is shown in Fig.29.

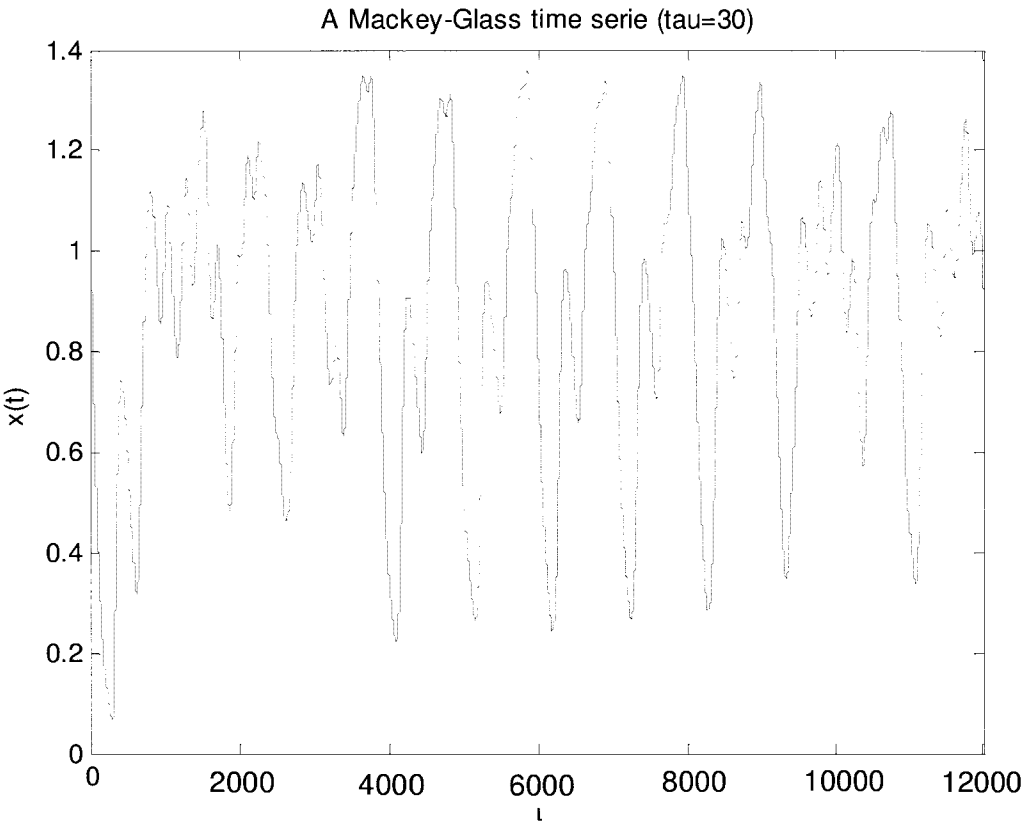


Fig.29. A generated Mackey-Glass time series with the parameters of equation (6.1) set as h=30; A=0.2; B=0.1 .

5.4. Parallelization strategy for the time series

A general task has been considered here of delivering the forecasts simultaneously for various number of steps ahead, which include short term as well as long term forecasts based on the same preceding history and therefore based on the same initial data set. The solution of such complex task with one processor would lead to a neural predictor of the multiple-input-multiple-output (MIMO) type shown in Fig.6.

Formulating such multiple forecasting tasks to a single neural network leads to an increased complexity of the network. This in turn is the major cause of the problems with the convergence, which is more difficult to achieve for the multiple variables task, as well as would take much longer to complete as the number of the required message passing events is increasing respectively. It also would be a disadvantage to require completion of such a task from one processor as compared to the multiple processors of inherently parallel environment. On the other hand, in itself, single forecasting task based on the neural network of MISO type (multiple-inputs-single-output), shown in Fig.4, is free from the above mentioned difficulties for a single processor. This would give a reason to justify the following parallelization strategy, namely to address the complex multiple-forecasts task by splitting it into more simple single-forecast sub-tasks between multiple processor units (which are the SPEs units for the Cell/B.E.), as it is shown in Fig.30.

Herewith the approach (i) of [48] (see Fig.9) is employed for implementing the neural predictor of the time series, as it is illustrated in Fig.30. The incoming values of x(t), where t is discrete time moments of preceding history t, t-1, t-2,…,t-n, based on which the prediction of the expected values has been made for the future time moments starting from t+1 and following the sequential time moments t+2, t+3,.., t+k, where k is the number of the time steps of the prediction. This approach comprises splitting of the task by parameter k, which is the number of prediction steps, while prediction is still based on the n preceding time points of immediate past. Additionally, the task to different processor units can be modified in terms of the data basis – for example the input points can be separated so that only each 10^{th} point gets connected to the network, thus increasing the leverage for the forecast, as this approach would allow 10 times deeper

reliance on the past history, which may be of particular importance for the long term forecasts (see the focus point #3 above).

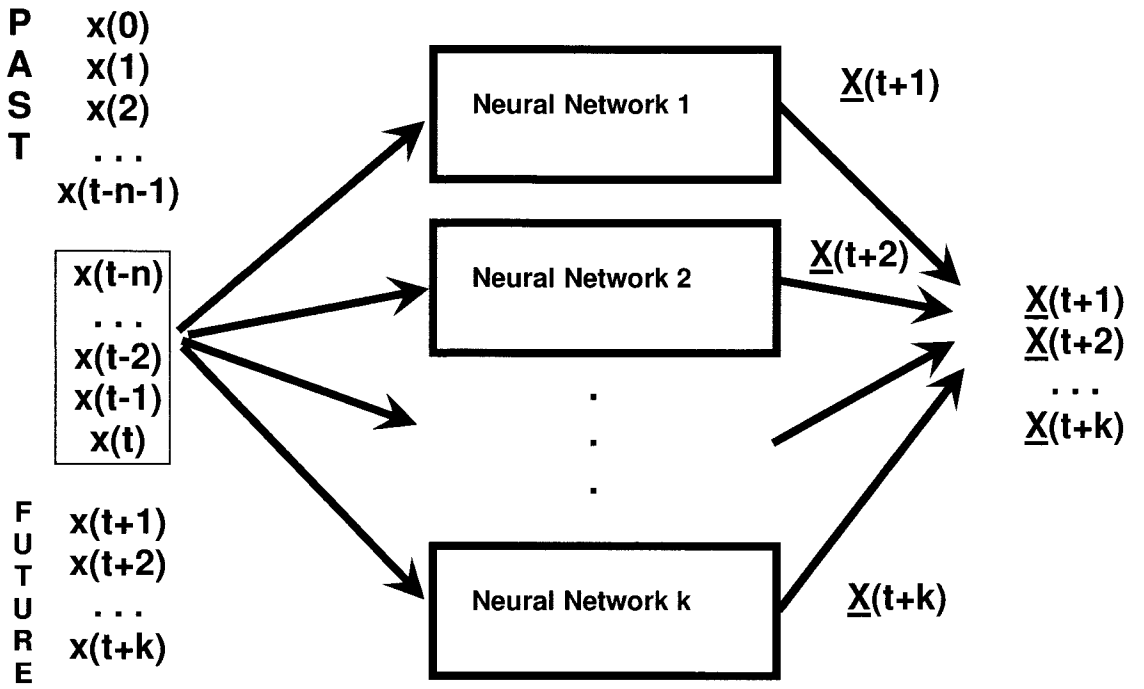


Fig.30. Parallelization of the neural time series predictor: $\underline{X}(t+k)$ is the predicted value for x(t+k) based on the input set [x(t); x(t-1);…;x(t-n)]; where n is the number of time points in the preceding history as a basis for forecast, k – the number of the time steps of prediction.

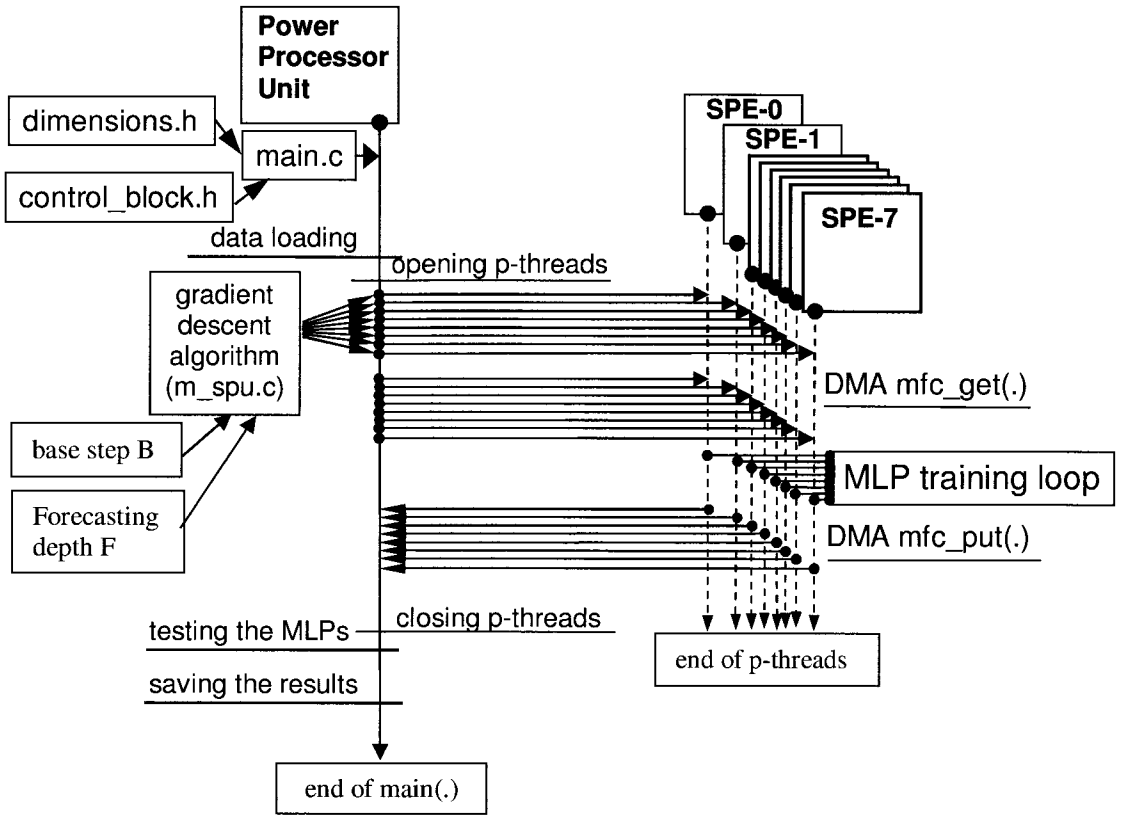


Fig.31. Software operation diagram for tracing parallel trajectories of the gradient descent algorithm for time series prediction.

Algorithm implementation for the time series prediction, shown in Fig.31, differs from that for 2D function approximator (Fig.12), in that there is only one dimension for the input variable, t, and parallelization parameters become associated with the base step parameter, B, and forecasting depth, F. A base step parameter, B, is defined as a number of time steps between consecutive intake values for the forecast. A forecasting depth, F, is defined as a number of time steps along original time series from the last intake value and the forecasted point in the future. The use of these parameters for parallelization is via the embedded portion of the program run by SPEs, i.e. via m_spu.c file, as shown in the Fig.31.

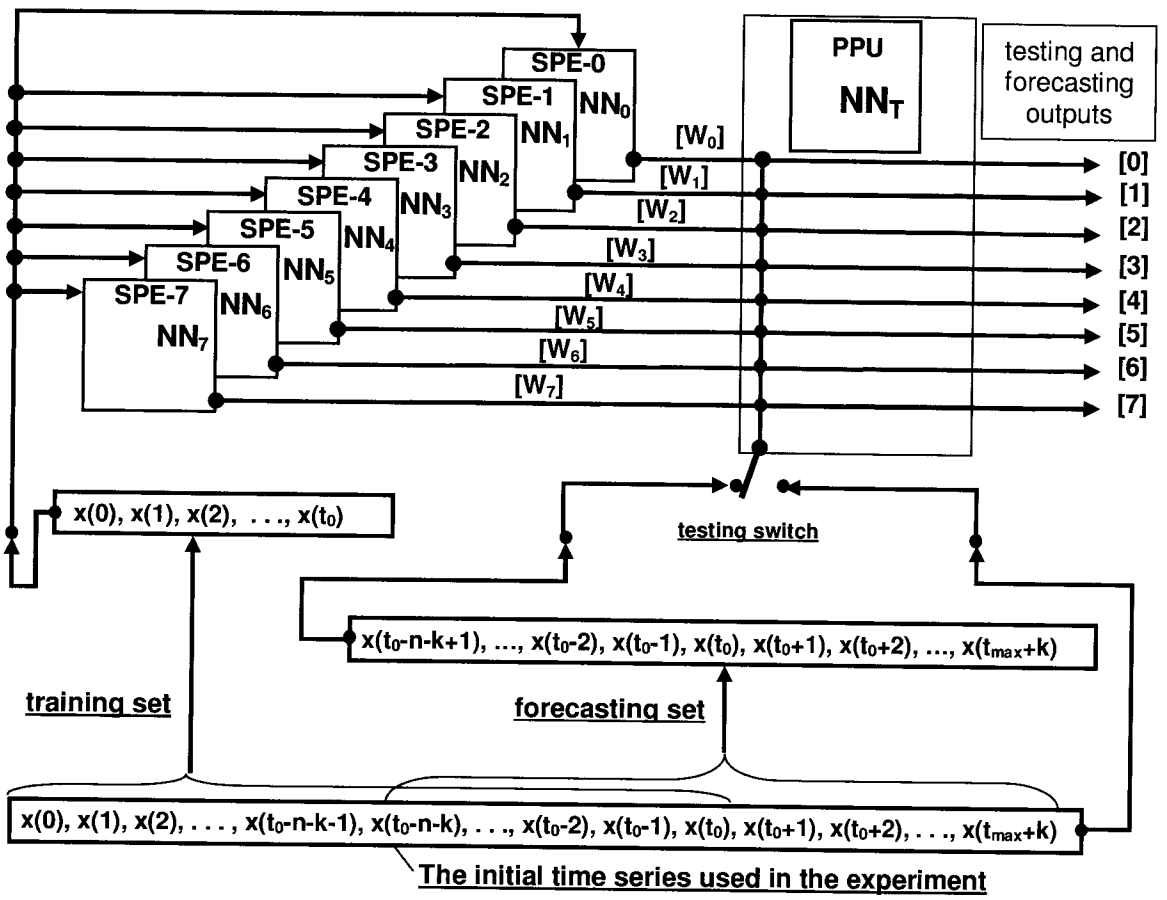


Fig.32. Parallelization scheme and software operation for tracing the trajectories of the gradient descent algorithm for time series prediction experiments.

For tracing the parallel trajectories of the gradient descent algorithm to train neural time series predictors the parallelization scheme shown in Fig.32 was used, which is similar to that in Fig.13 except that the data were organized as three series, programmed as multi-vectors in the program: (i) initial time series, from which the training set (ii) and testing set (iii) were extracted. The initial time series is represented by the sequence of x(t) values: **x(0), x(1), x(2), . . . , x(t_0-n-1), x(t_0-n), . . ., x(t_0-2), x(t_0-1), x(t_0), x(t_0+1), x(t_0+2), ..., x(t_{max}+k).** Here t=t_0 is the moment in time at which the data necessary for prediction are collected for the training, so that all available information from t=0 to t=t_0 inclusive is

supplied for training purposes. Additionally, n is the number of points taken as a base for prediction for the input of the MLP, and k is the required maximum forecasting depth, k=max(F); $\mathbf{t_{max}}$ is the last point taken as the basis for prediction input. Initialization of the weight matrices $[W_i]$ here is similar to that in Fig.13 – the matrices are initialized at random starting points with small random values put in it, stored in the main memory and DMA transferred to the local SPE stores after p-threads initiation to run the parallel trajectories of the gradient descent algorithm. Again, as in Fig.13, after completion of the gradient descent, the resulting matrices $[W_i]$ are transferred back from the SPE's local stores to the main memory for testing (as shown in Fig.32). The two testing procedures, controlled in the program by the switch shown in Fig.32, employ forecasting set (iii) of the time series, which includes values from $t=t_0-n-k$ till $t=t_{max}+k$ inclusive. It is seen that there is overlap with the training set, and such overlap assumes availability of the points $\mathbf{x(t_0-n-k+1)}$, ..., $\mathbf{x(t_0-2)}$, $\mathbf{x(t_0-1)}$, $\mathbf{x(t_0)}$ for both sets – the training and the forecasting one. However in the training set points $\mathbf{x(t_0-n-k)}$, $\mathbf{x(t_0-n-k+1)}$, ..., $\mathbf{x(t_0-k)}$ are only used as the basis to predict the last value $\mathbf{x(t_0)}$ of the training set, while in the forecasting set the points $\mathbf{x(t_0-n-k+1)}$, ..., $\mathbf{x(t_0-k)}$, $\mathbf{x(t_0-k+1)}$ are used to make forecast for the first future point $\mathbf{x(t_0+1)}$ in the series. Therefore the points $\mathbf{x(t_0-k)}$, ..., $\mathbf{x(t_0)}$ are playing role of "dummies" to only account for the forecasting depth at which the point $\mathbf{x(t_0)}$ stands.

5.5. Forecasting time series with the long term trend

The training of the time series predictor of the sequence with long term trend, described by the equation $x(t) = \text{int}\,[10^3 * \sin^2(t/2)] + \text{int}\,[10^3 * \sin^2(t/20)] + \text{int}\,[10^3 * \sin^2(t/30)] +$

h(t) * int [10^3 * $\sin^2(t/300)$], where t=1, 2, 3..., T_{max} and int(.) is the function taking integer part of the argument, has been conducted in Cell/B.E. environment in order to demonstrate best predictive capability between the 8 SPEs, each modeling MLP neural network with 12 hidden nodes and trained for 5000 epoch on 200 training patterns. The results for $T_{max} = T_b + t_p + n$, with T_b=500; t_p=3; n=5 are summarized in Table 6, where it is seen that the best performance has been attained by the SPE-3 and the SPE-7 with final Root Mean Square Error (RMSE) values of **0.056455** and **0.056087** respectively for the training points. It is seen however that the best predictive capability is demonstrated by the SPE-3, which provides lowest RMSE value of **0.121441** for novel points versus that of **0.121573** for the SPE-7. The worst performance here was by the SPE-2 with RMSE values of **0.058436** for training points and **0.125095** for novel ones, which are respectively ca. 4% and 3% worse than that of the winner (the SPE-3). It is seen that final values for all trajectories are falling within close proximity of each other and there were no instances of entrapment in local minima. Therefore, the parallelization here is not beneficial unless another dimension in forecasting is present. Such a dimension can be the parameter t_p – the number of steps for prediction. The parallelization on parameter t_p for the time series is therefore seen as the viable target.

A forecasting property of the best MLPs (those of the SPE-3 and the SPE-7 from Table 6) is illustrated in Fig.33, where the plot 1 represents the actual series, of which first 200 points were used for training while the rest 300 points for actual test in prediction capability. The plots 2 and 4 show predictions made by the SPE-3 and the SPE-7, while plots 3 and 5 – the errors made by the same. It is worth to note that for the training region the error in prediction oscillates around the axis x, while for the testing points it stays

entirely in the negative area, which reflects the influence of the persistent long term trend. It is seen that errors are located in the vicinity of the local trend change areas. Otherwise, the forecasting is accurate and it reflects the underlying long term trend. Changing the trend from positive (as in Fig.27) to negative one (as in Fig.28) practically retains the predictive capability of the MLP, as it is illustrated in Fig.34 and in Table 7. Despite the fact that training was entirely performed on the part of the function which only contained a positive trend (points ##1 to 200 in Fig.28 and Fig.34), the network appeared to fully retain the forecasting capability for the part where the trend was the opposite one, i.e. for the negated trend in Fig.28 in the forecasting area. The partial cost however for that is the higher RMS Error as it is seen by comparing related RMSE values of Table 6 and Table 7. Specifically, for the best performing SPEs, namely the SPE-3 and the SPE-7, the change in RMSE for the forecasting area become respectively ΔRMSE = **0.171191 - 0.121441 = 0.049750** and **0.170680 - 0.121573 = 0.049107** due to negating the trend. These changes amount to 40.97% and 40.39% respectively of its initial values (or 29.06% and 28.77% respectively of its final values). Thus the costs of negating the trend become less then twice increase in RMS Error for the forecasting area. Apart from being higher in amplitude, the error also changes the sign, as it is seen by comparing plots #3 and #5 of the Fig.33 with the same of the Fig.34.

Table 6. Results of the competition for prediction of time series seen in Fig.27 and described by x(t) = int $[10^3 * \sin^2(t/2)]$ + int $[10^3 * \sin^2(t/20)]$ + int $[10^3 * \sin^2(t/30)]$ + int $[10^3 * \sin^2(t/300)]$, where t=1, 2, 3..., T_b+t_p+n, where $T_b=500$; $t_p=3$; n=5. (The input parameters see in the Appendix B).

	SPE-0	SPE-1	SPE-2	SPE-3	SPE-4	SPE-5	SPE-6	SPE-7
RMSE training	0.057047	0.056689	0.058436	0.056455	0.056678	0.057080	0.056961	0.056087
RMSE new points	0.122910	0.122039	0.125095	0.121441	0.121953	0.123024	0.122692	0.121573
RMSE overall	0.101813	0.101102	0.103707	0.100616	0.101037	0.101903	0.101636	0.100629

Table 7. Results of the competition for 3 steps ahead in time series prediction with the negating trend seen in Fig.28 and described by x(t) = int$[10^3 * \sin^2(t/2)]$ + int$[10^3 * \sin^2(t/20)]$ + int$[10^3 *\sin^2(t/30)]$ + h(t) * int $[10^3 * \sin^2(t/300)]$, where t=1, 2, 3,..., T_b+t_p+n, and h(t)= {+1 for $1 \leq t \leq 210$; -1 for $t \geq 211$ }; (here $T_b=500$; $t_p=3$; n=5). (The input parameters see in the Appendix B).

	SPE-0	SPE-1	SPE-2	SPE-3	SPE-4	SPE-5	SPE-6	SPE-7
RMSE training	0.057047	0.056689	0.058436	0.056455	0.056678	0.057080	0.056961	0.056087
RMSE forecast	0.171012	0.171538	0.170898	0.171191	0.170940	0.170927	0.170956	0.170680
RMSE overall	0.137291	0.137625	0.137439	0.137327	0.137176	0.137233	0.137234	0.136885

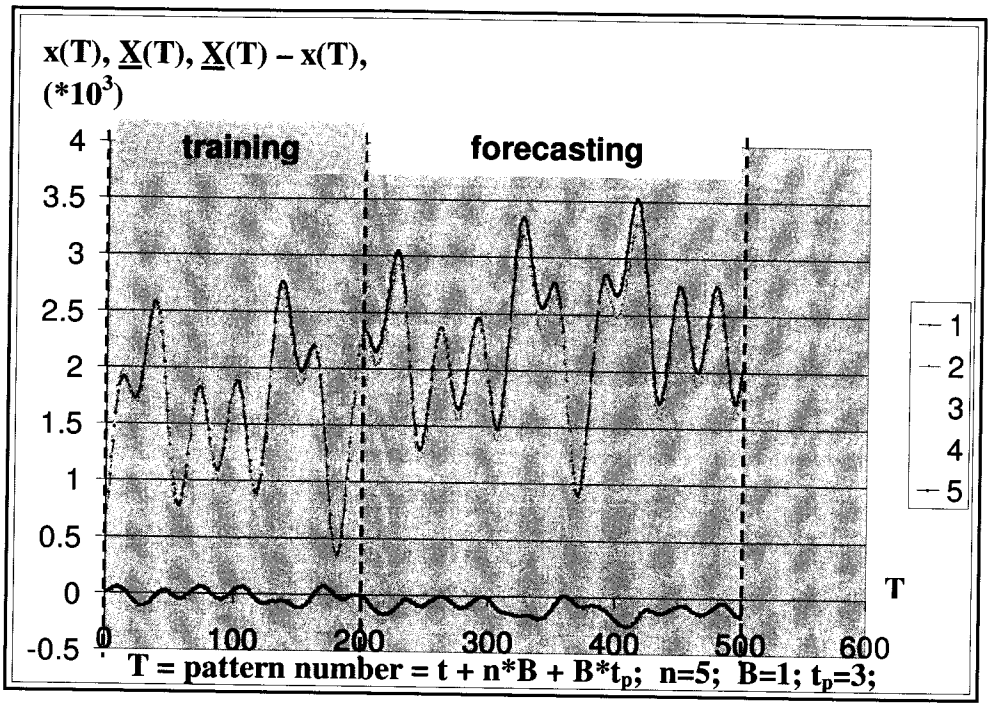


Fig.33. Prediction of the time series of Fig.27 for 3 steps ahead (t_p =3) by trained MLPs of the SPE-3 and the SPE-7 from Table 6. Here: **1** is for the values x(T) of initial function x(t)=int [10^3 * $\sin^2$(t/2)] + int [10^3 * $\sin^2$(t/20)] + int [10^3 * $\sin^2$(t/30)] + int [10^3 * $\sin^2$(t/300)], where t=1, 2, 3..., T_b+t_p+n, T_b=500; t_p=3; n=5; **2** and **4** – the MLP's outputs $\underline{X}$(T) in the tests of forecasting from the SPE-3 and the SPE-7 respectively; **3** and **5** – forecasting errors =[$\underline{X}$(T) – x(T)] in tests of the resulting MLP from the SPE-3 and the SPE-7. T is the forecasted pattern number, T = t+n+t_p = t+5+3. All units (for T, x(T) and $\underline{X}$(T)) are arbitrary. The values x(T) and $\underline{X}$(T) are scaled down 10^3 times.

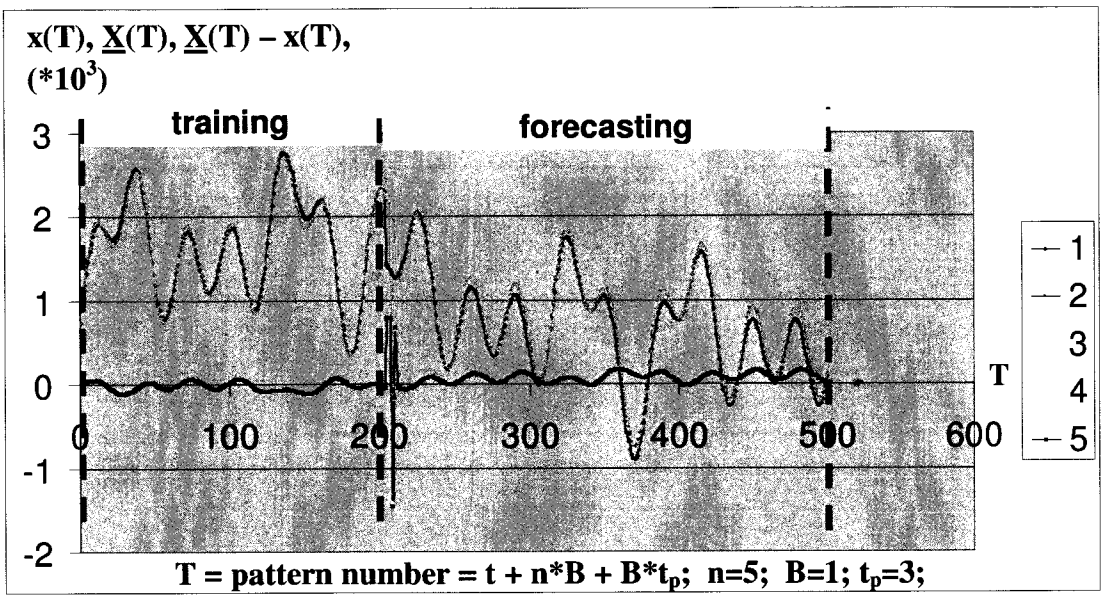


Fig.34. Prediction of the time series with the negating trend (Fig.28) 3 steps ahead (t_p=3) by trained MLPs of the SPE-3 and the SPE-7 in Table 7. Here: **1** is for the values x(T) of initial function x(t)=int [10^3 * $\sin^2$(t/2)] + int [10^3 * $\sin^2$(t/20)] + int [10^3 * $\sin^2$(t/30)] + h(t) * int [10^3 * $\sin^2$(t/300)], where t=1, 2, 3,..., T_b+t_p+n, and h(t)= {+1 for 1≤t≤210; -1 for t≥211 }; T_b=500; t_p=3; n=5; **2** and **4** – the MLP's outputs $\underline{X}$(T) in the tests of forecasting from the SPE-3 and the SPE-7 respectively; **3** and **5** – forecasting errors =[$\underline{X}$(T) – x(T)] in tests of the resulting MLP from the SPE-3 and the SPE-7. T is the forecasted pattern number, T = t+n+t_p = t+5+3. All units (for T, x(T) and $\underline{X}$(T)) are arbitrary. The values x(T) and $\underline{X}$(T) are scaled down 10^3 times.

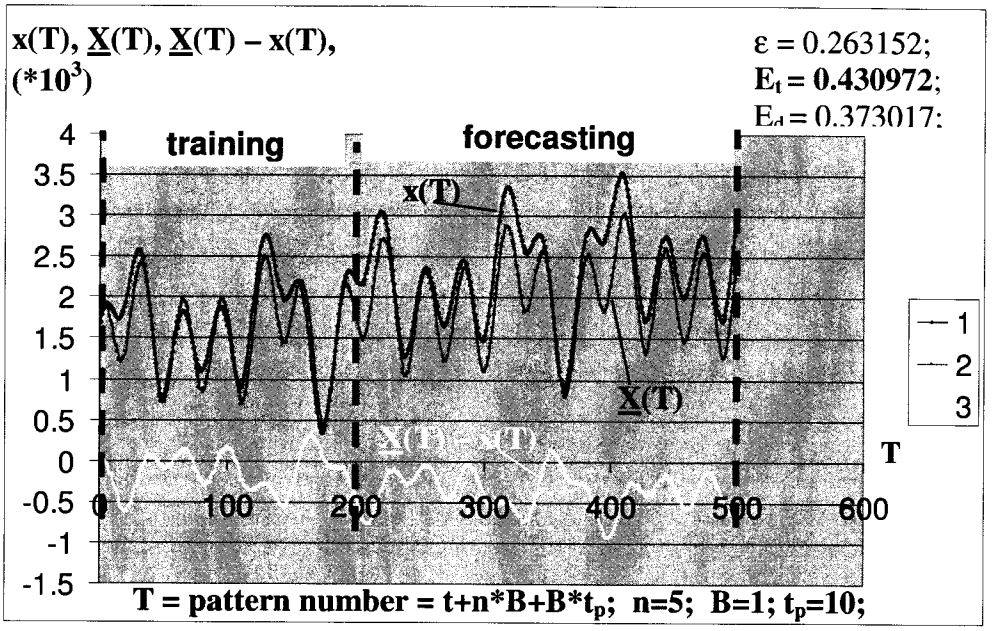


Fig.35. Prediction of the time series of Fig.27 for 10 steps ahead ($F=B*t_p=10$) by trained MLP (*numEpochs*=5000). Here: **1** is for the values x(T) of initial function $x(t)=int\ [10^3 * \sin^2(t/2)] + int\ [10^3 * \sin^2(t/20)] + int\ [10^3 * \sin^2(t/30)] + int\ [10^3 * \sin^2(t/300)]$, where t=1, 2, 3..., T_b+t_p+n; $T_b=500$; $t_p=10$; n=5; T is the forecasted pattern number, $T = t+n+t_p = t+15$. **2** is the MLP's output $\underline{X}(T)$ in the tests of forecasting; **3** is the forecasting error $=[\underline{X}(T) - x(T)]$ in the tests of the resulting MLP. All units (for T, x(T) and $\underline{X}(T)$) are arbitrary. The values x(T) and $\underline{X}(T)$ are scaled down 10^3 times.

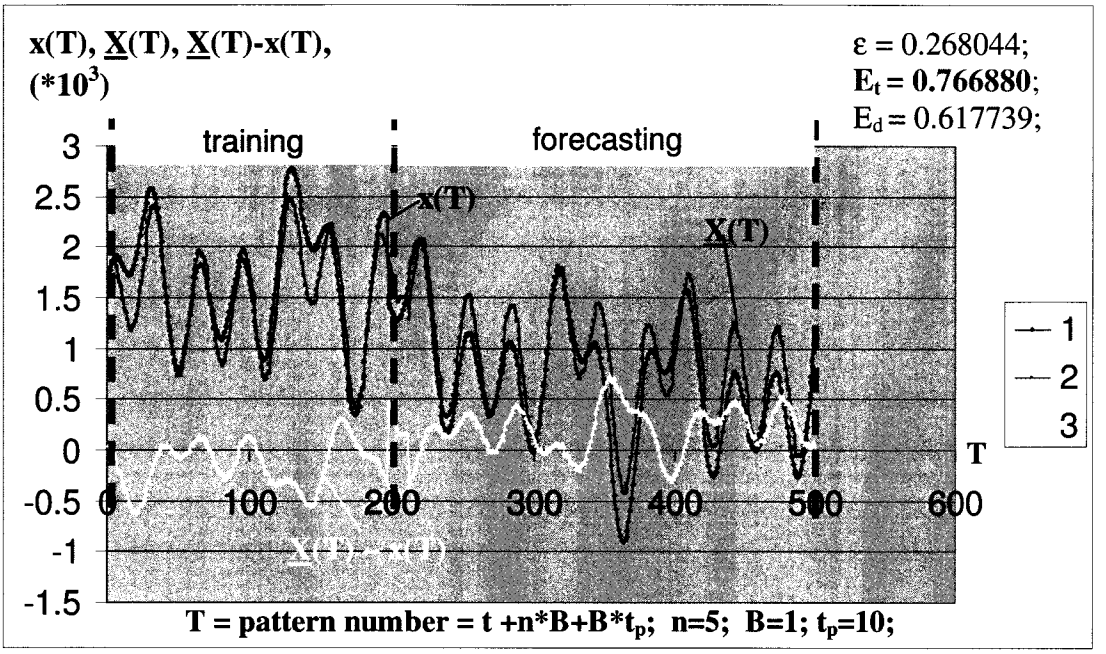


Fig.36. Prediction of the time series of Fig.28 for 10 steps ahead ($F=B*t_p=10$) by trained MLP (*numEpochs*=5000). Here: **1** is for the values x(T) of initial function $x(t)=\text{int}\,[10^3 * \sin^2(t/2)] + \text{int}\,[10^3 * \sin^2(t/20)] + \text{int}\,[10^3 * \sin^2(t/30)] + h(t) * \text{int}\,[10^3 * \sin^2(t/300)]$, where $t=1, 2, 3,\ldots, T_b+t_p+n$, and $h(t)= \{+1 \text{ for } 1\leq t\leq 210; -1 \text{ for } t\geq 211\}$; $T_b=500$; $t_p=10$; $n=5$; T is the forecasted pattern number, $T = t+n+t_p = t+15$. **2** is MLP outputs $\underline{X}(T)$ in tests of forecasting; **3** is forecasting error $=[\underline{X}(T) - x(T)]$ in tests of the resulting MLP. All units (for T, x(T) and $\underline{X}(T)$) are arbitrary. The values x(T) and $\underline{X}(T)$ are scaled down 10^3 times.

The increase of the prediction range to $t_p=10$ leads to the increase in RMSE both for training and forecasting areas, as it is shown in Fig.35 and Fig.36 for the continuing and the negating trend respectively. The RMS Error for the training sessions of 5000 epochs long for both cases (Fig.35 and Fig.36) are close (0.263152 and 0.268044) as the training sets are the same. It is ca. 4 times higher than that for $t_p=3$ in Table 6 and Table 7, which suggest linear trend for RMSE increase with the t_p parameter. For the forecasting area the values of RMSE are $E_t=0.430972$ and $E_t=0.766880$ respectively, which indicates that

forecasting of the negating trend (Fig.36) is almost twice less accurate than that of the continuing one (Fig.35).

The behavior of the neural net in predicting the time series seen in Fig.35 and Fig.36 suggests that increasing the forecasting range t_p values makes it increasingly difficult for the MLP to anticipate the expected value based on provided current information and previous training. As it is seen in Fig.35 and Fig.36 even training areas become prone to significant errors. The suggested here solution for this problem is in trying to increase the length of the base the prediction is based upon while retaining the same number of input points for the MLP. It is important however first to complete optimization of the MLP training in terms of the applied number of training epochs (parameter *numEpochs*). The result of such optimization is shown in Fig.38 (a), where it is seen that to minimize the forecasting RMSE there is an optimum number of the training epochs, which is located in the range of $5*10^4$ for t_p=10 and which provides the best forecasting results possible under the settings. The settings in particular include n=5, i.e. the number of input data points for the MLP is 5 consecutive values of the time series of that in Fig.27, forecasting length – 10 steps ahead ($F=B*t_p=10$). The performance of the optimized forecaster of Fig.35 is presented in Fig.39. It is seen that there is certain improvement as compared to the performance in Fig.35: while the case of Fig.35 demonstrates the RSME values of ε= 0.23384; E_t= 0.35773; E_d= 0.314094, the optimized case in Fig.39 demonstrates appreciably lower values of ε= 0.199224; E_t= 0.242113; E_d= 0.22595. Thus, the best attainable value of forecasting RMSE for 10 step prediction in B=1, n=5 mode becomes E_t= 0.242113. As it is seen from Fig.39, this optimized performance is still prone to significant errors even at the training stage, which leads to even higher errors at the

forecasting stage. As it is mentioned earlier, now it is no longer possible to achieve further improvement via the training, therefore the suggested alternative needs be explored via increasing the length of the base the prediction is based upon (while retaining the same number of input points for the MLP). The approach here is that instead of taking consecutive time points for the MLP inputs, which in this terminology would constitute the base step value B=1, the intake would be conducted of every second time point, i.e. making the base step value B=2 and with n=5 inputs this would cover the total length for the forecast to be $L_B = n * B = 5 * 2 = 10$, which is twice longer (in proportion to the increase of the base step coefficient B from 1 to 2). At the same time, the parameter t_p has to be set at 5 in order to maintain the forecasting length at equivalent level of $F = B * t_p = 2 * 5 = 10$.

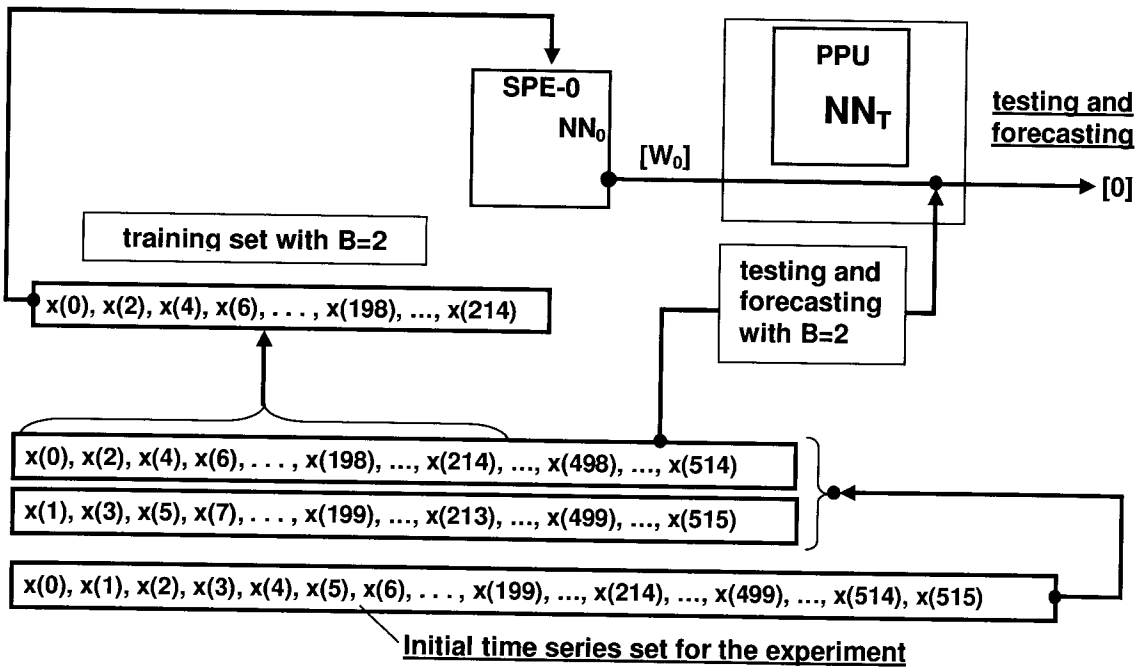


Fig.37. Redundancy removal by the factor of 2 in the time series prediction experiment via setting base step parameter B=2.

The data manipulation aiming to have simultaneous effect of the increased base step B for prediction equal 2 and redundancy removal in the data set by the same factor of 2, is shown in Fig.37 for the case of the continuous function from Fig.28 and negating function from Fig.29. It is seen in Fig.37 that the initial time series is split in two: one with values for the even time points t= 0, 2, 4, 6, ..514; and another one for the odd time points t=1, 3, 5, 7, ..., 515. The value of 514 is obtained by adding 10 for F=10 steps for prediction used in the experiment and 5 for n=5 input values to the 499 as the index of the selected 500 points. One more value is added on the top of that to equalize the number of points in both sub-series, which becomes 208. After that one of the sub-sets is used to extract the training and forecasting sub-set from and follow the procedure described for the Fig.32 to obtain the testing and forecasting, but this time based on the increased value of B=2 and both made sequentially in automatic manner by using the whole sub-set as the input to testing neural network NN_T.

The results of testing such a procedure are summarized in Fig.38,(b) while covering comparable to the previous case range of the training epochs up to 500000 in order to insure that the optimized solution is not missed. As it is seen in Fig.38,(b), such optimized solution is in fact present at the same level of training expense of 50000 epochs as before, but it is demonstrating significant improvement in RMSE values, delivering $\varepsilon = 0.060972$; $E_t = 0.142678$; $E_d = 0.117052$; at the optimal of 50000 training epochs, where the forecasting RMSE parameter E_t is almost twice lower than initially. The actual performance of the suggested procedure at its optimal point is presented in Fig.40, which shows a sensible improvement as compared to that in Fig.39. It is worth noting that a deterioration of the predictive capability of the MLP took place when attempting to make

predictions to the length twice that of the predictive base itself. Such deterioration occurred notwithstanding of the fact that the functionality of the time series was still maintained the same, i.e. as that in Fig.27. Intuitively, this may be interpreted as redundancy of short term information supplied to the MLP and insufficiency of the long range information. Because in the suggested procedure the number of the input values retained to be the same, n=5, expanding the forecasting base via doubling the base step B was equivalent to sacrificing the short term redundancy in favor of the longer term information. This appears to be beneficial when attempting the long range forecast, because allowed to equalize forecasting range with the base for prediction. To put it in another way, attempting to forecast to the length of $F = B * t_p = 1 * 10 = 10$ based on the base of $L_B = n * B = 5 * 1 = 5$ is more prone to errors than that to the length of $F = B * t_p = 2 * 5 = 10$ based on the base of $L_B = n * B = 5 * 2 = 10$. It can be therefore concluded that expanding the base for prediction allowed recovering the predictive capability of the MLP, which deteriorated when attempting to exceed the predictive base L_B by the forecasting length F.

In terms of parallelization, the predictive base step B becomes one more dimension, in which parallelization according to either scheme – that of **Fig.9** and/or **Fig.10** – may benefit the forecaster. The simplest case of implementing the module according to **Fig.10** scheme would include distributing between SPUs computations for various values of B, followed by the selection switch on the PPU module to utilize a better predictor when such is identified based on the on-line tests for the current incoming values of the controlled time series.

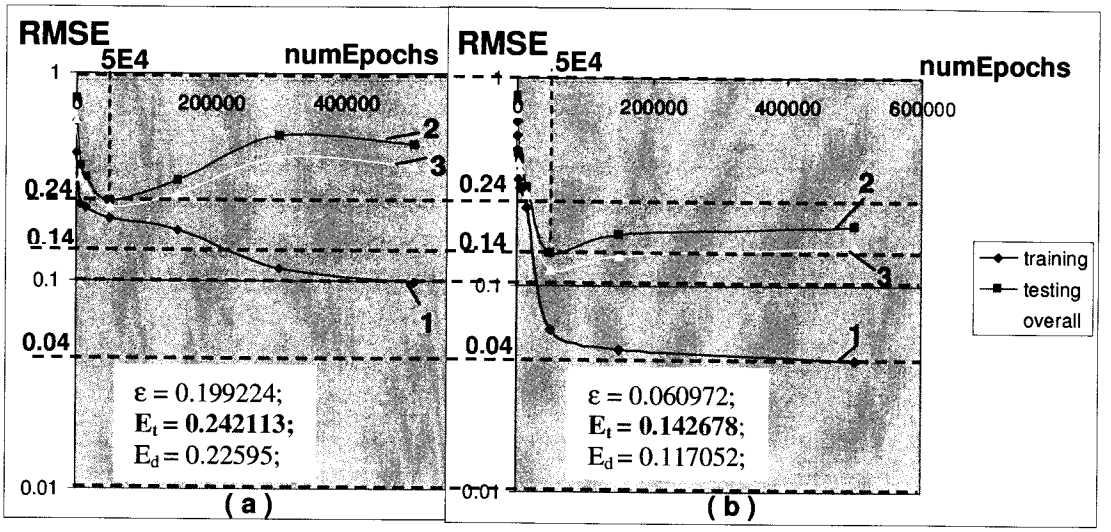


Fig.38. The results of the RMSE optimization of the neural time series predictor for the series of Fig.27 via scanning the number of training epochs (*numEpochs*) for two different base-lengths B for prediction: (a) B=1 with t_p=10; (b) B=2 with t_p=5.

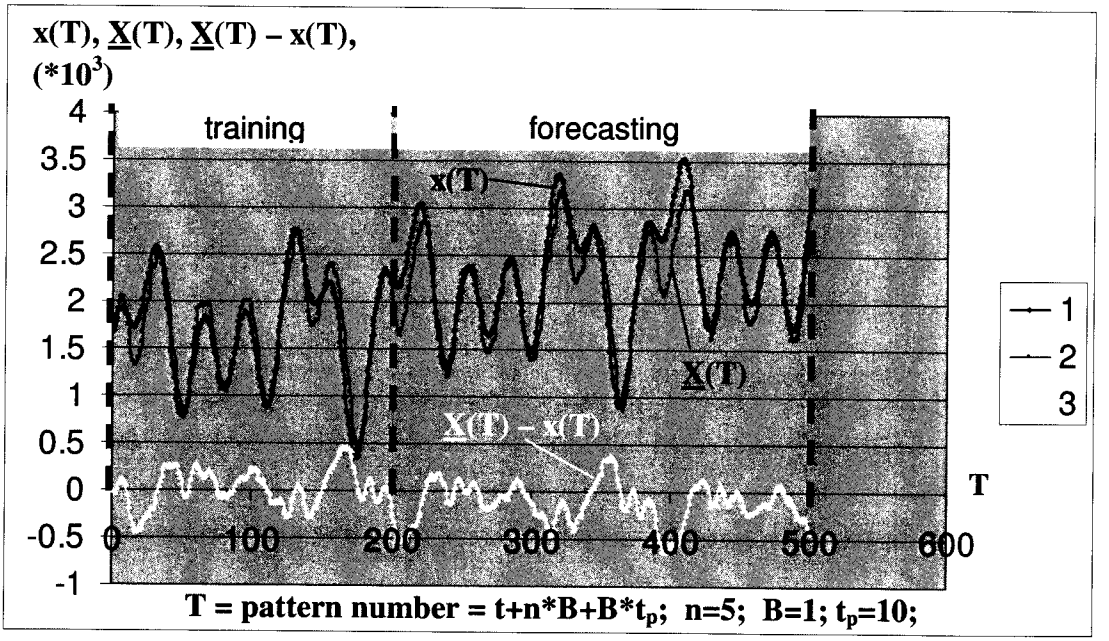


Fig.39. Optimized performance of the forecaster of Fig.35.

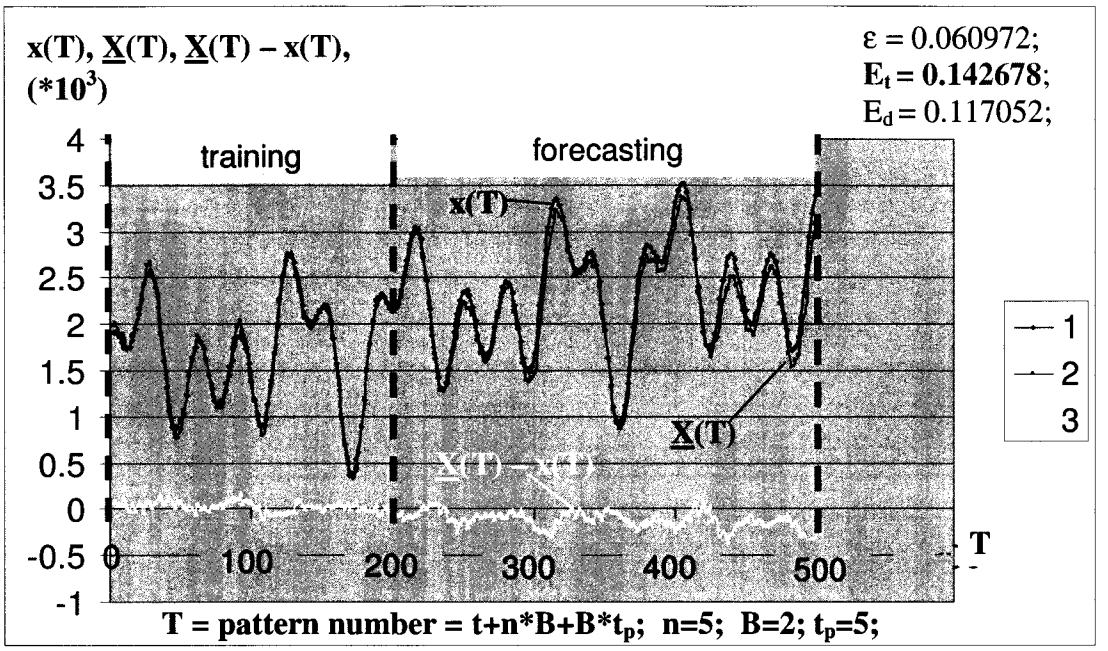


Fig.40. Performance of the forecaster with increased base step (B=2) and t_p=5 steps for the function of **Fig.27** with the depth F=B*t_p=10. Every second point has been excluded from the initial time series to form a subset to operate on.

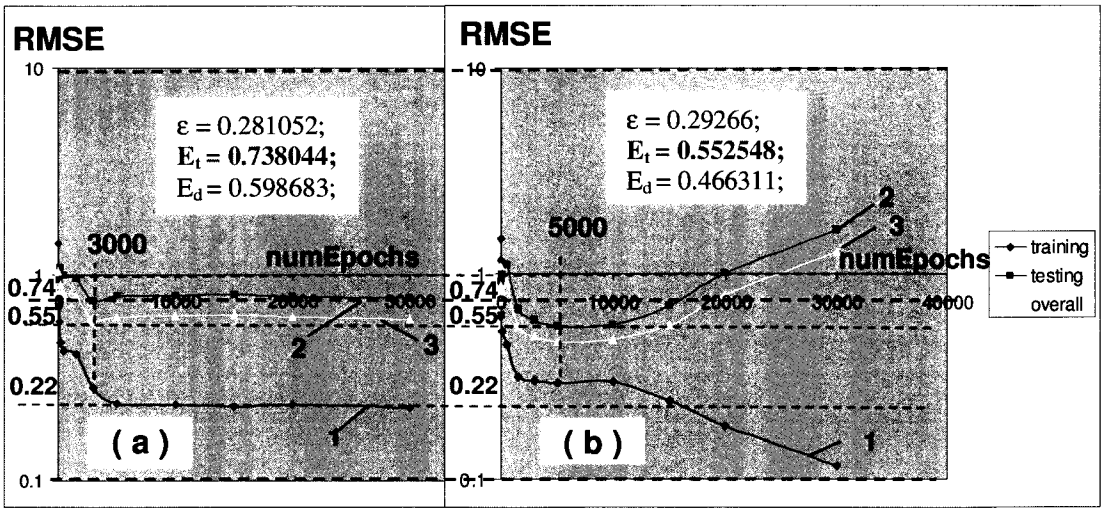


Fig.41. Comparative performance of the forecasters of series with negating trend (see Fig.28) for 10 steps ahead (F=B*t_p =10) with (a) B=1 (as in Fig.36) and (b) B=2 for various training length *numEpochs* (number of epochs). The following RMSE parameters are used as performance measures: **1** - ε in training; **2** - E_t in testing/ forecasting; **3** - E_d overall over both procedures. Optimal RMSE values attained are shown in insertions. .

The parameter *numEpochs*, representing number of epochs in training, does not represent further parallelization dimension, as it may be perceived from Fig.38, because various stages of the training can be retrieved sequentially from the same SPU within the same training process.

The efficiency of the same procedure of extending the predictive base as above needs to be also verified in the case of the function with the negating trend (Fig.28), as the problem of deterioration of the forecasting accuracy when exceeding the predictive base $L_B = n * B$ by the length of forecast $F = B * t_p$ here appears to be even worse (see Fig.36). As before, the scanning of the *numEpochs* parameter reveals somewhat optimized training of about 3000 epochs, which delivers the following RMSE parameters $\varepsilon = 0.281052$; $E_t = 0.738044$; $E_d = 0.598683$, which however is quite close to those values for which Fig.36 has been generated, namely $\varepsilon = 0.268044$; $E_t = 0.766880$; $E_d = 0.617739$. In fact, as it is seen in Fig.41,(a), the training does not have much of the effect on the MLP predictor for this range of the forecast as the forecasting capability keeps at the same level for the training range from 5000 to 30000 epochs (see Fig.41,(a)). Intuitively it may be understood, because the training is done for the positive trend, while actual testing is taking place for the negating trend not seen by the MLP on the training. Effect of extending the predictive base by doubling the base step parameter B, i.e. setting B = 2, is summarized in Fig.41,(b). It is seen that the plot of E_t vs *numEpochs* acquires convexity with the minimal value $E_t = 0.552548$ at *numEpochs*=5000. The full set of RMSE values at this point become $\varepsilon = 0.29266$; $E_t = 0.552548$; $E_d = 0.466311$; which constitute detectable improvement as compared to the best case for B=1 situation in Fig.41,(a).

Important here is not the improvement of the performance itself, as it remains poor, but a significant and quite characteristic change of E_t vs *numEpochs* dependence. It is because such pattern of behavior may be used to identify occurrences of negating trends in the time series. Application example may potentially include algorithmic modules in stocks prediction software, which may be able to identify occurrences of negating behavioral trends of large scale stock holders.

In terms of parallelization in the Cell/B.E. environment, the benefit here is in possibility for simultaneous training of the two neural forecasters for various B values (B=1 and B=2 in our particular case) in different SPUs and finally comparing their performance in E_t vs *numEpochs* dependencies with the real time series at the PPU module (i.e. according to scheme of **Fig.42**), at which detecting the relations depicted in Fig.41 would give the ground for conclusions about occurrences of the negating trends. Again, as parallelization parameter here would be the value of the base step B. The *numEpochs* parameter would be scanned within the same SPU module thus not requiring the parallelization.

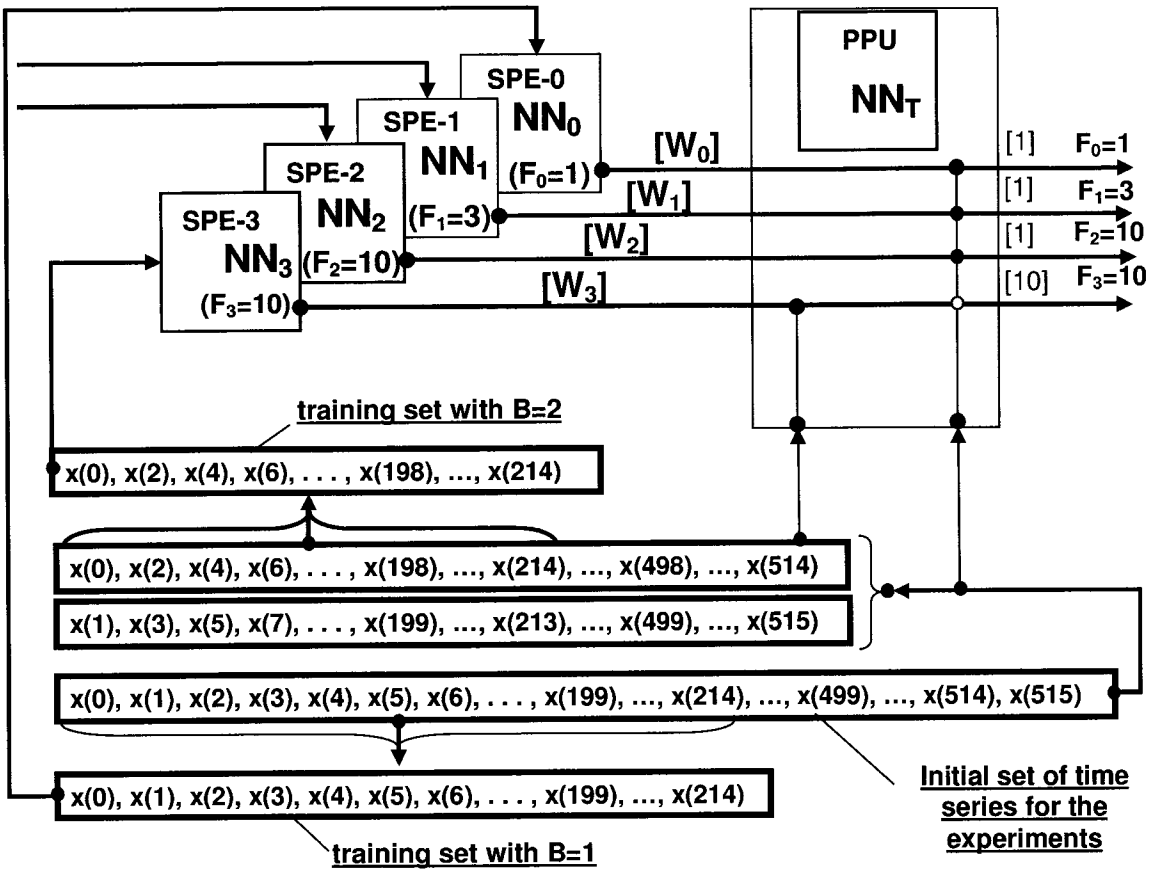


Fig.42. Resulting parallelization scheme for the prediction of time series with the long term trend, which is designed to accommodate routes explored in Section 5.5 (i.e. those shown in Fig.33 through Fig.41)

The routes explored in Section 5.5 for the time series with long trend, such as those in Fig.27 and Fig.28 and the results of which are presented in Fig.33 through Fig.41, are summarized in the software design shown in Fig.42. First three SPEs (SPE-0 through SPE-2) are taking in the training set with B=1, while the SPE-3 is accepting the training set with B=2. The resulting forecasts for the first three output lines are for the values of F=1; 3 and 10 with B=1 all, while the last line outputs the forecasting results for F=10 and B=2.

5.6. Experimental results of prediction for Mackey-Glass series

The Mackey-Glass time series generated (see Fig.29) has been used to train and to test the neural forecasters. For training purposes the 200 points have been taken, after which the following 300 points have been used for the forecasting purposes, i.e. to test the quality of the forecasts made. The root mean square error has been used as the criteria parameter for training as well as for testing.

The initial tests revealed that the training of the individual single-forecasters is not prone to the trapping into local minima, but rather is swiftly reaching the diving trajectory, followed by the saturation and slow lengthy finalizing training track, as it is shown in Fig.43. It is therefore possible to take advantage of the diving part of the training trajectory by deploying the forecaster after completion of the training in the diving part of the trajectory, which results in almost two orders of magnitude improvement of the RMS error value and comprises more than 90% of the forecasting capacity at a cost of the short but efficient initial "diving" part of training trajectory (about 300 epochs in Fig.43). In the parallel environment the remaining part of the training can be continued while the partially trained forecaster can be already employed for the forecasting purposes. Another advantage here is that newly incoming time points may also be included in training, thus ensuring that the forecaster is up-to-date at all the time.

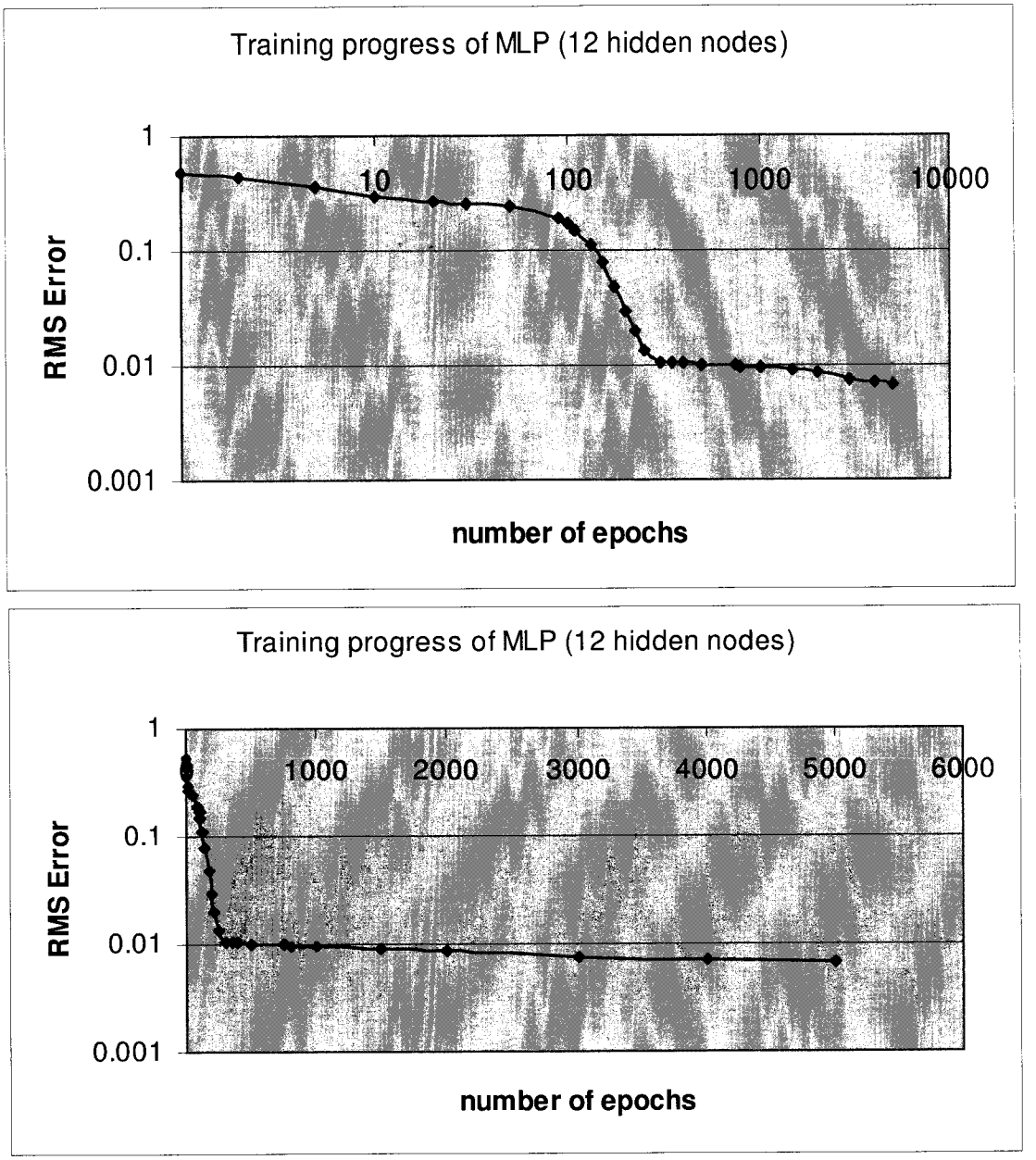


Fig.43. Diving and saturation parts of the training trajectory of the neural forecaster (12 hidden nodes, 5-inputs, 1-output; k=1).

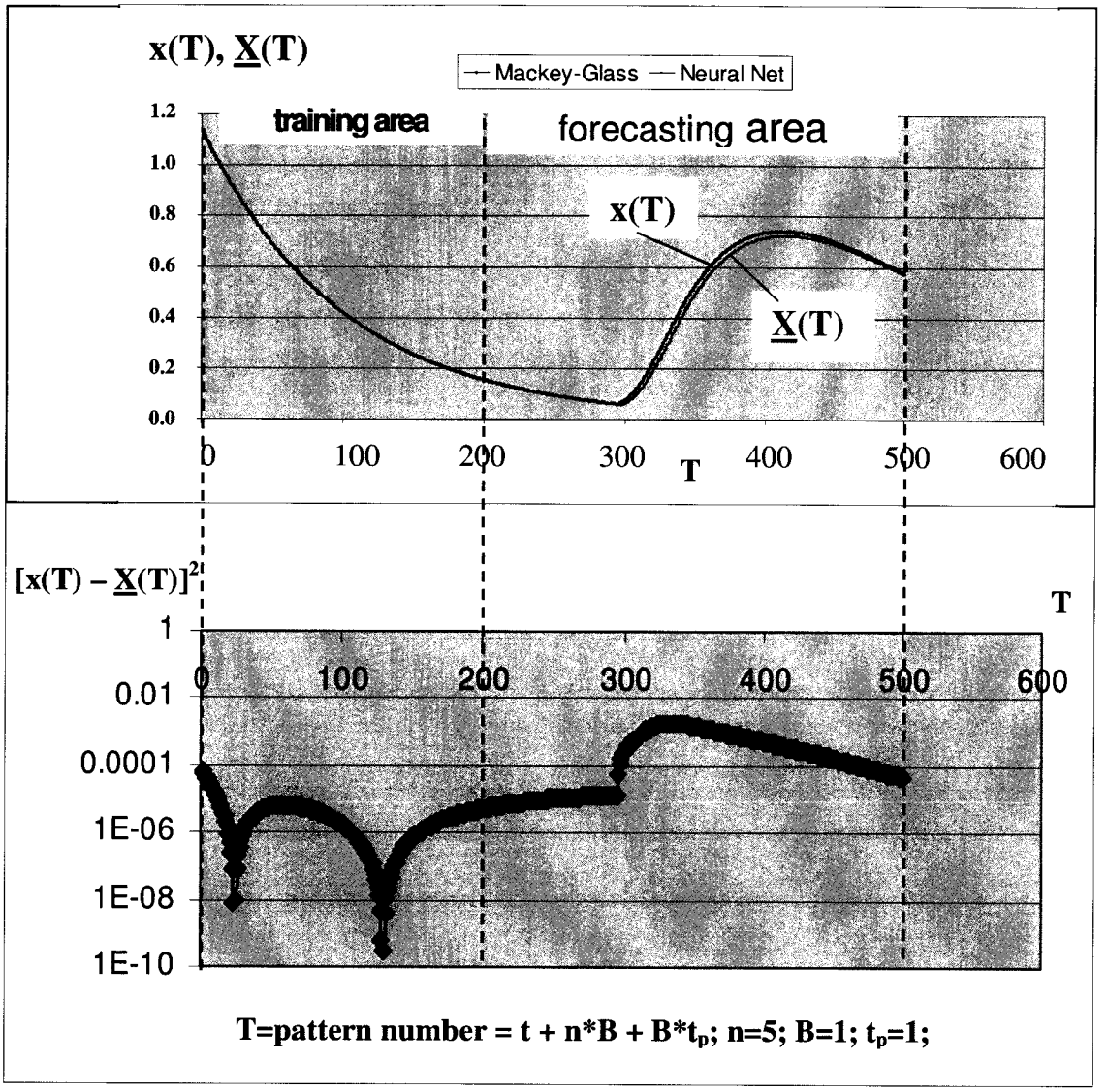


Fig.44. A neural forecaster's performance of 1-step ahead based on 5 prior input values (36 hidden nodes MLP) over initial 500 patterns of the Mackey Glass time series.

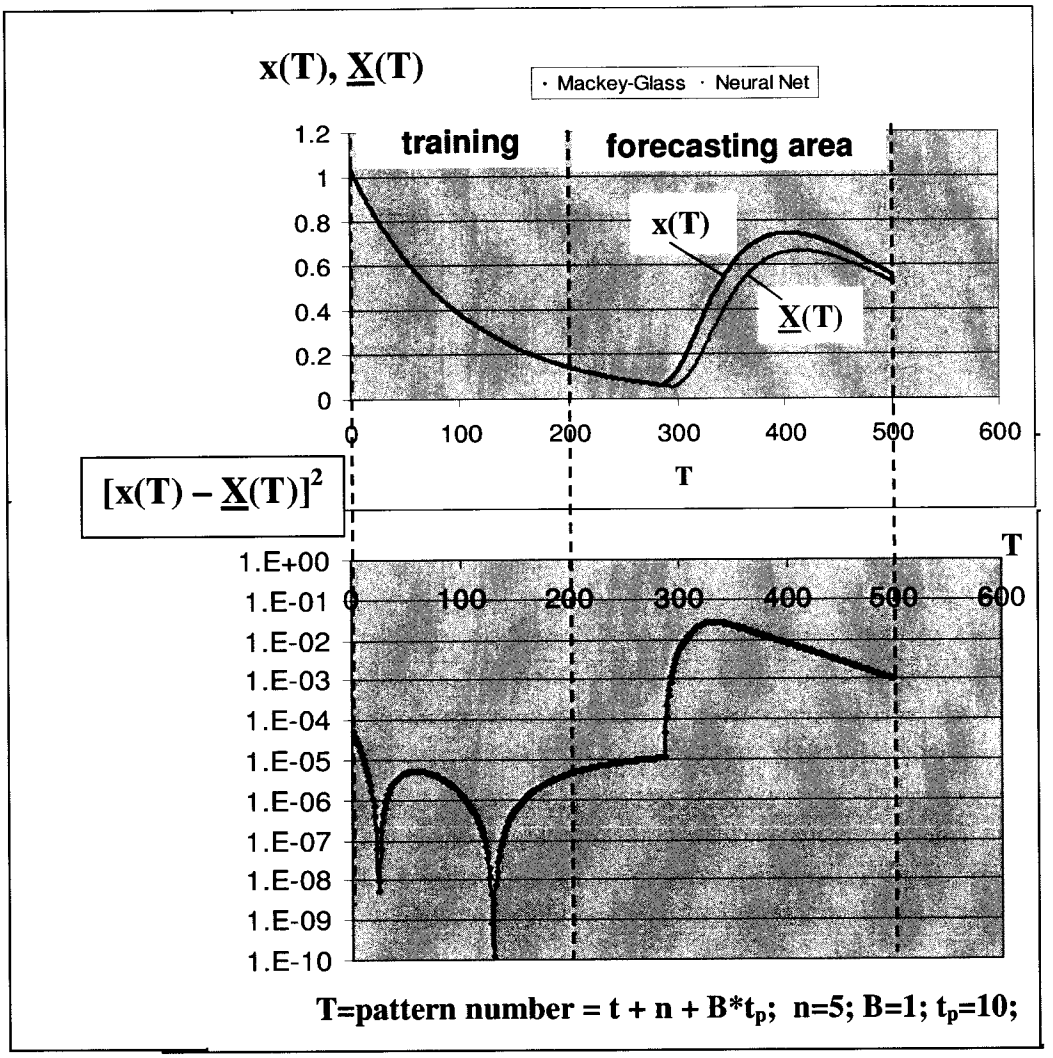


Fig.45. Performance of the MLP's forecaster of 10-step ahead based on 5 consecutive input points, trained and tested within initial 500 patterns of the Mackey-Glass time series

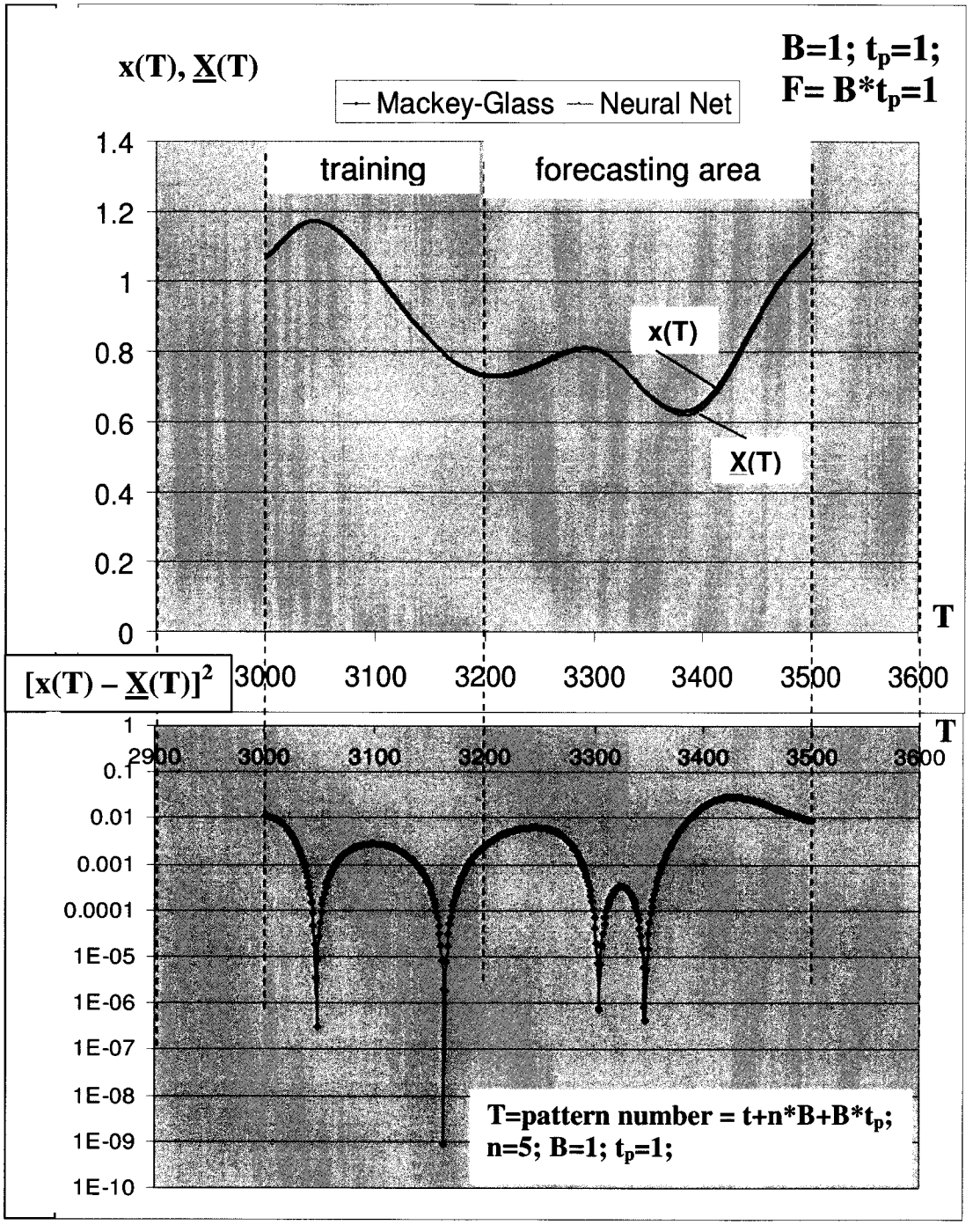


Fig.46. Performance of the 1-step-ahead MLP forecaster (n=5; B=1; t_p=1) for the time patterns T=[3000; 3500].

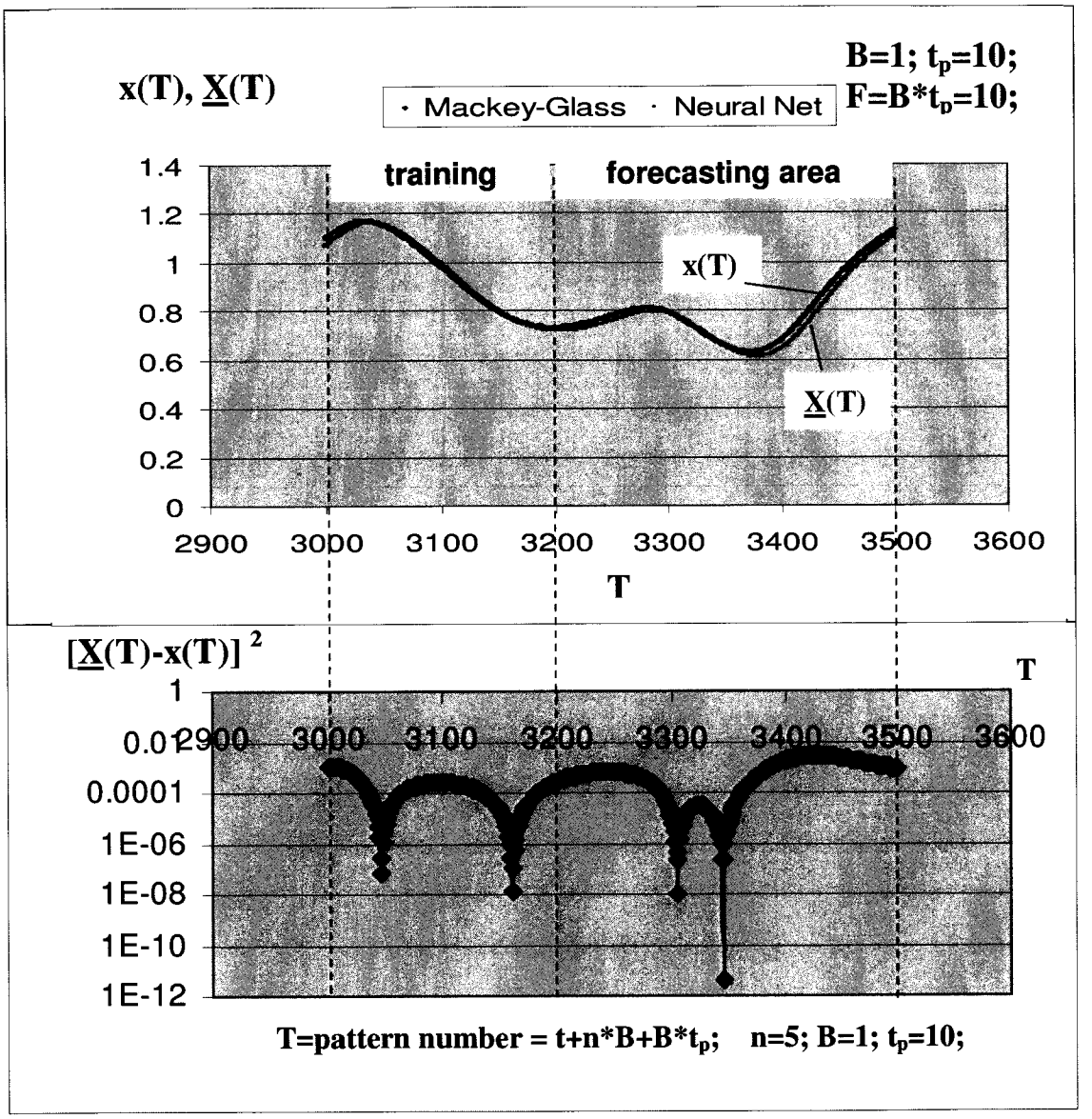


Fig.47. Performance of the 10-step-ahead MLP forecaster (n=5; B=1; t_p=10) for the time patterns T=[3000; 3500].

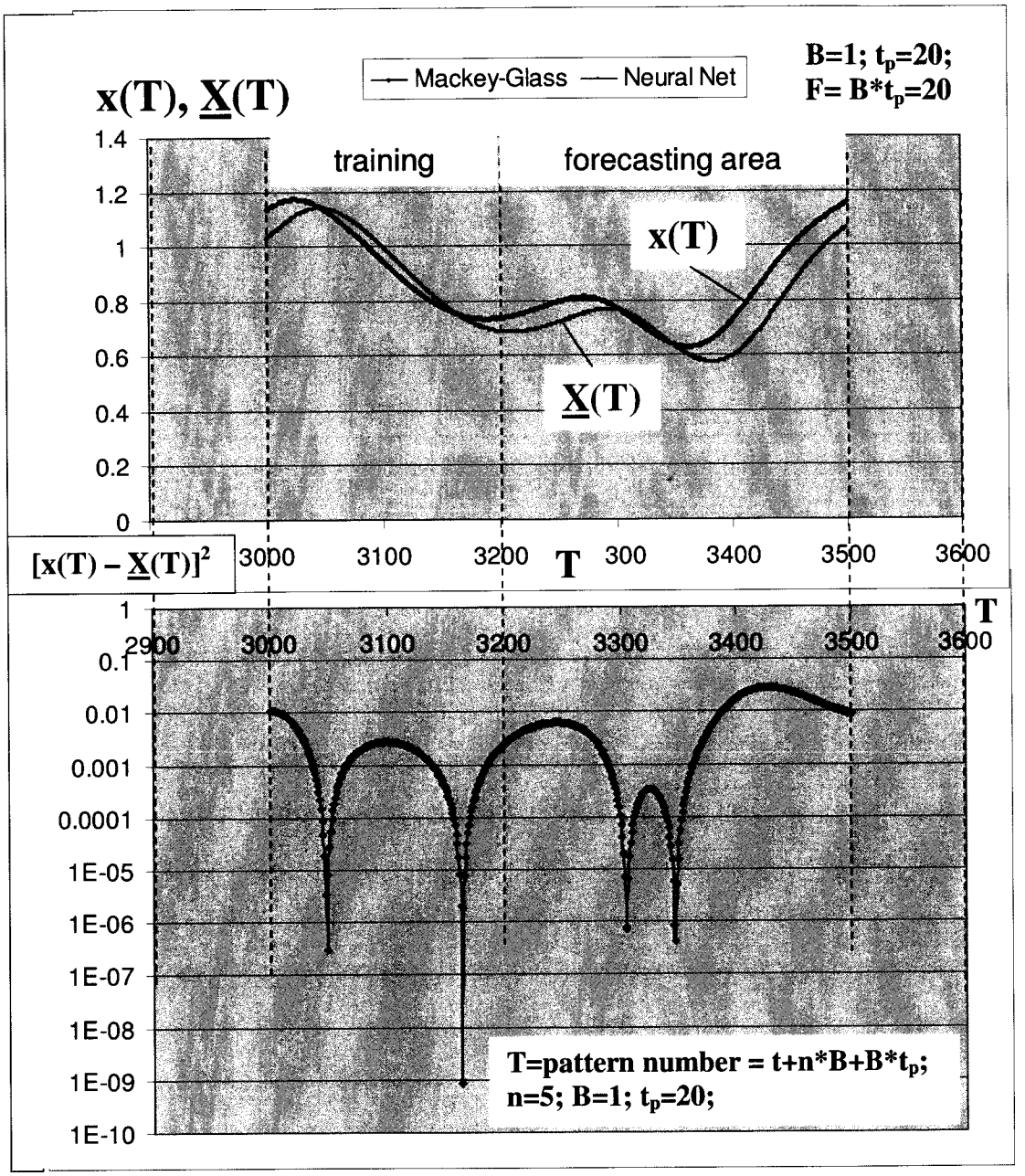


Fig.48. Performance of the 20-step-ahead MLP forecaster (n=5; B=1; t_p=20) for the time patterns T=[3000; 3500].

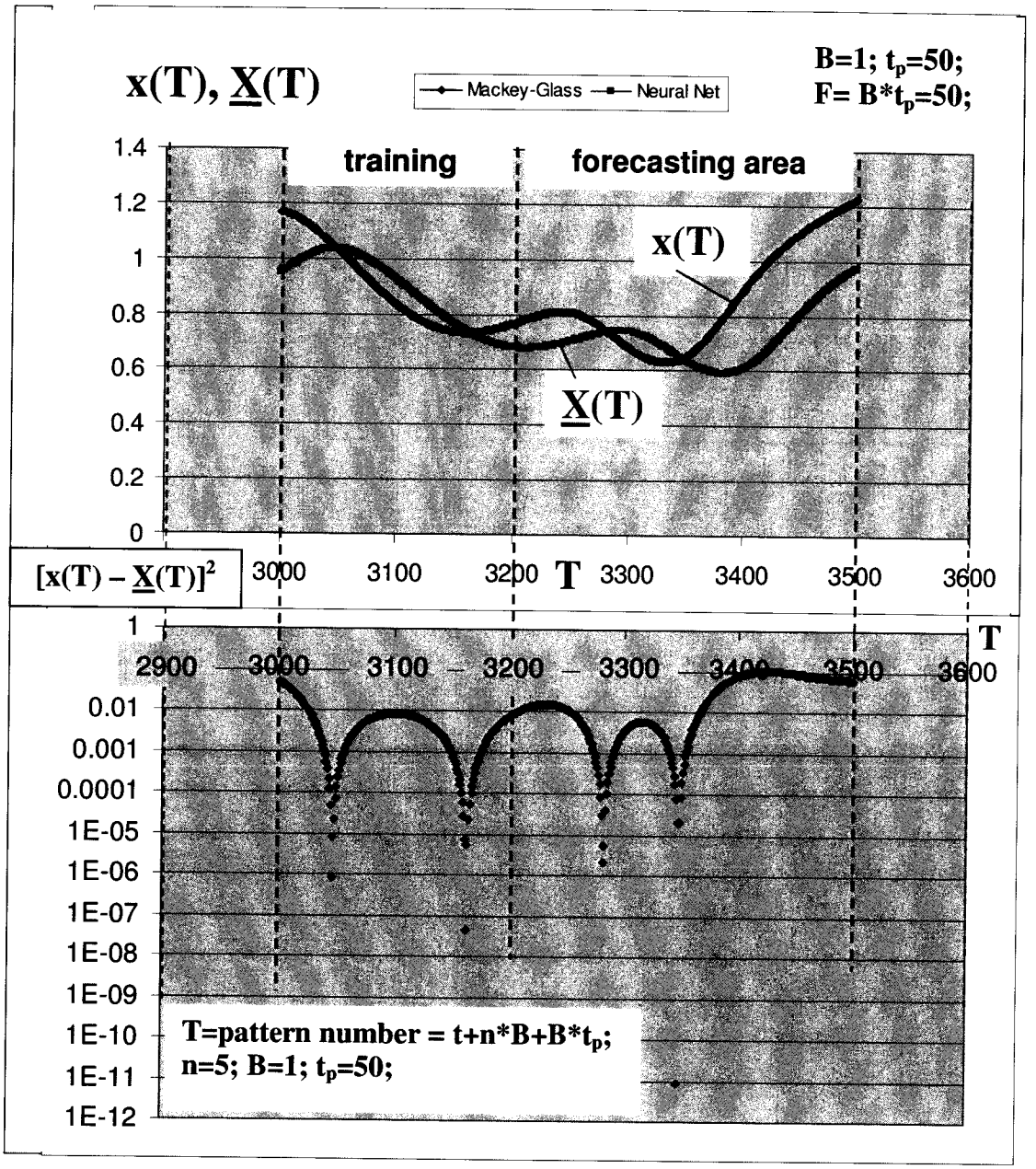


Fig.49. Performance of the 50-step-ahead MLP forecaster (n=5; B=1; t_p=50) for the time patterns T=[3000; 3500].

In Fig.44 the operation of the forecaster of 1 step ahead (t_p=1; B=1; n=5)) is shown based on MLP with 36 hidden nodes and 5 input points. It is seen that the low forecasting error is attributed to the area which has similarity to the training area, while significant increase of the quadratic error occurs when the trend is changing to a different one. Similar situation is for the 10-steps-ahead forecaster (t_p=10; B=1; n=5), shown in Fig.45, where quadratic error for the forecasting area increases more than 3 orders of magnitude for the area of changing trend. For the points presented in Fig.44 and Fig.45 there is not much of the prior history, as it is the very beginning of the time series. This problem has similarity to the considered earlier deterioration of the forecasting capability for the time series with the long term trend (as shown in Fig.35). Here in the same way the forecasting errors increase when the forecasting depth $F = B * t_p = 1 * 10$ exceeded predicting base's length $L_B = n * B = 5 * 1 = 5$. The difference between the two cases (that in Fig.45 with Fig.44 and the one previously considered in Fig.35), however, is in the nature of the function involved. More specifically, as it is seen in Fig.44, the scale chosen for the training purposes allows covering only portion of the function with the expressed trend – the trend down, with no oscillation at all. Shifting to another area, when training portion has more similarity of the features involved in the forecasting is shown in Fig.46 for 1 step prediction ($B = 1$; $t_p = 1$) and in Fig.47 for 10 step prediction ($B = 1$;. $t_p = 10$). Comparison of the later with that in Fig.45 reveals, that introducing into training area features present later at the testing/forecasting stage does lead to reduction of forecasting errors (as the quadratic errors in Fig.45 quite strongly exceed the level of 10^{-2}, while in Fig.47 it keeps below 10^{-2} level. However, as it is seen there, it comes at the expense of having somewhat higher level of the errors at the training stage. More results of the

multi-step training and forecasting tests are reproduced in the next figures: the 20 steps MLP predictor (B=1; t_p=20) in Fig.48 and the 50 steps MLP predictor (B=1; t_p=50) in Fig.49 (for the same portion of the Mackey-Glass series as in Fig.44 and Fig.45). It is seen that increasing the forecasting depth F = B * t_p leads to increase in the forecasting errors. In terms of RMSE as a measure parameter, the trend of the error increase with the forecasting depth is shown for the above cases of multi-step neural predictor in Fig.51,(a). It is seen that in the range of small values for F = B * t_p, from 1 to about 20 steps ahead, both RMSEs, for training, ε, as well as for prediction, E_t, are increasing jointly, with some trend of the spread between them increasing too, which however is not very significant until F reaches the value of 50. Starting from about this point, F = 50, the spread between ε and E_t accelerates, to which a big deal of contribution is added by the change of the trend for ε. After that the training and forecasts are less and less connected, so that the MLP predictor is becoming "blind" to the forecasting depth F.

As earlier in the case of the function in Fig.35, it would be natural to address the problem of increasing disparity between the training and the forecasting via increase of the predicting step base B in order to at least equalize the predicting base length L_B = n * B and the forecasting depth F = B * t_p. This concern seems to be addressed by setting B=10, which provides 10 fold increase in predictive base to L_B = n * B = 5 * 10 = 50. Because this value becomes comparable to the size of the whole training area itself, which is 200 points, the training area needs to be proportionally expanded. Also the apparent redundancy in the Mackey-Glass time series could be removed by considering only every 10^{th} point from it for the training purposes, while eventually recovering the whole series at the forecasting stage. To accomplish that, the initial set of the Mackey-Glass series is

split into 10 subsets, each assigned an index k, where k=1, 2, 3, ... , 10. Each set k is composed by taking in the points numbered (10 * m + k) into it from the initial time series, where m=0, 1, 2, 3, ... , *maxMG*/10, with *maxMG* being the maximum number of the points in the initial time series, which in our case is *maxMG*=12000 (see Fig.29). Reducing redundancy in such a way, we then form the training set by selecting the first 200 points from the set k=1, the point's numbers being (10 * m + 1), where m=0, 1, 2, 3, ... , 199. To keep the same ratio of the size of the training set to the testing set, the next 300 points will form the forecasting area. To the forecasting area also the $F = B * t_p$ points need to be added to account for the border effect, when the predicting base at the end of the selected area still needs a full coverage of the predictive depth to be present to complete verification of the forecast made.

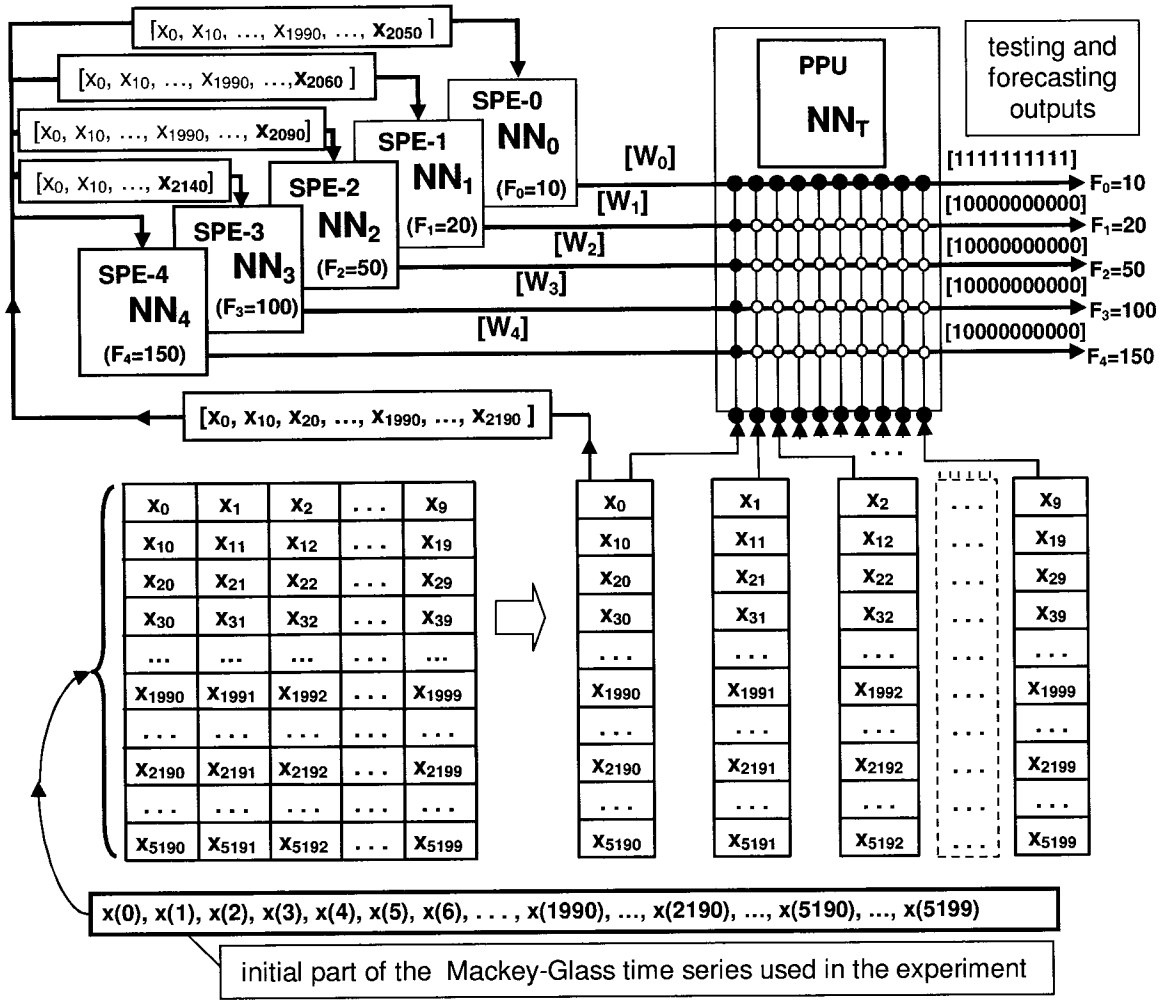


Fig.50. Parallelization scheme and software operation for the Mackey Glass series prediction with the neural networks, which takes the advantage of the redundancy removal by setting B=10 and allows multiple-step forecasts of F= 10; 20; 50; 100; 150. The switches at the lines intersection mean the respective connections to be: "ON" ● and "OFF" ○ .

Operation of the software to take the advantage of 10-fold redundancy removal and the base step increase to B=10 for the Mackey-Glass time series is shown in Fig.50. Here the initial set from the Mackey-Glass time series, chosen to accommodate the maximum forecasting depth F=150 and the prediction base L_B=n*B=5*10=50 is split into 10 equal sub-sets, first of which is used to extract the training sets from. The testing and

forecasting operations are sequential and automatically performed on the whole subset. All the subsets are available to the neural network NNT at the PPU and whether or not it is used in the forecast is indicated by the color of the fill for the switches (circles) at the line intersections inside the PPU area: white color means "connection is ON", black color means "connection is OFF". In the Fig.50 all the sub-sets are delivered to the testing and forecasting line [0] with F_0=10, which is indicated by the switches descriptor [1111111111] at the output and where "1" in the escriptor stands for "ON" (alternatively "0" would be for "OFF") for the respective switch. The examples of the output generated from the line [0] of Fig.50 with switches descriptor being [1111111111] are presented in Fig. 52, Fig.57 and Fig.58. The remaining output lines [1] through [4] in Fig.50 are shown in configuration of accepting only the first sub-set for the testing and forecasting with B=10 and F=20, 50, 100 and 150, which is indicated by the colors of the switches and the configuration of the switches descriptor being [1000000000]. The examples of the outputs for that are shown in Fig.53 for the line [1], in Fig.54 for the line [2]. Results of the outputs for the lines [3] and [4] are summarized in Fig.51,(b). The configuration of the switches for the line [0] characterized by the descriptor [1000000000] brings about the output illustrated in Fig.52 and Fig.55.

Application of the above procedure of the 10-fold increase of the base step B as well as related to that 10-fold reduction of the redundancy of the initial time series, is equivalent to reliance in the forecast on 10 times deeper prior history, while retaining the same structure of the MLP forecaster. Such forecaster has a 10-fold leverage for both – the prior history in training as well as for the forecasting perspective. It means that 1-step-

ahead prediction (t_p=1) for such forecaster is in fact providing forecasting depth of F = B* t_p = 10 * 1 = 10, i.e. 10 steps on the scale of initial time series.

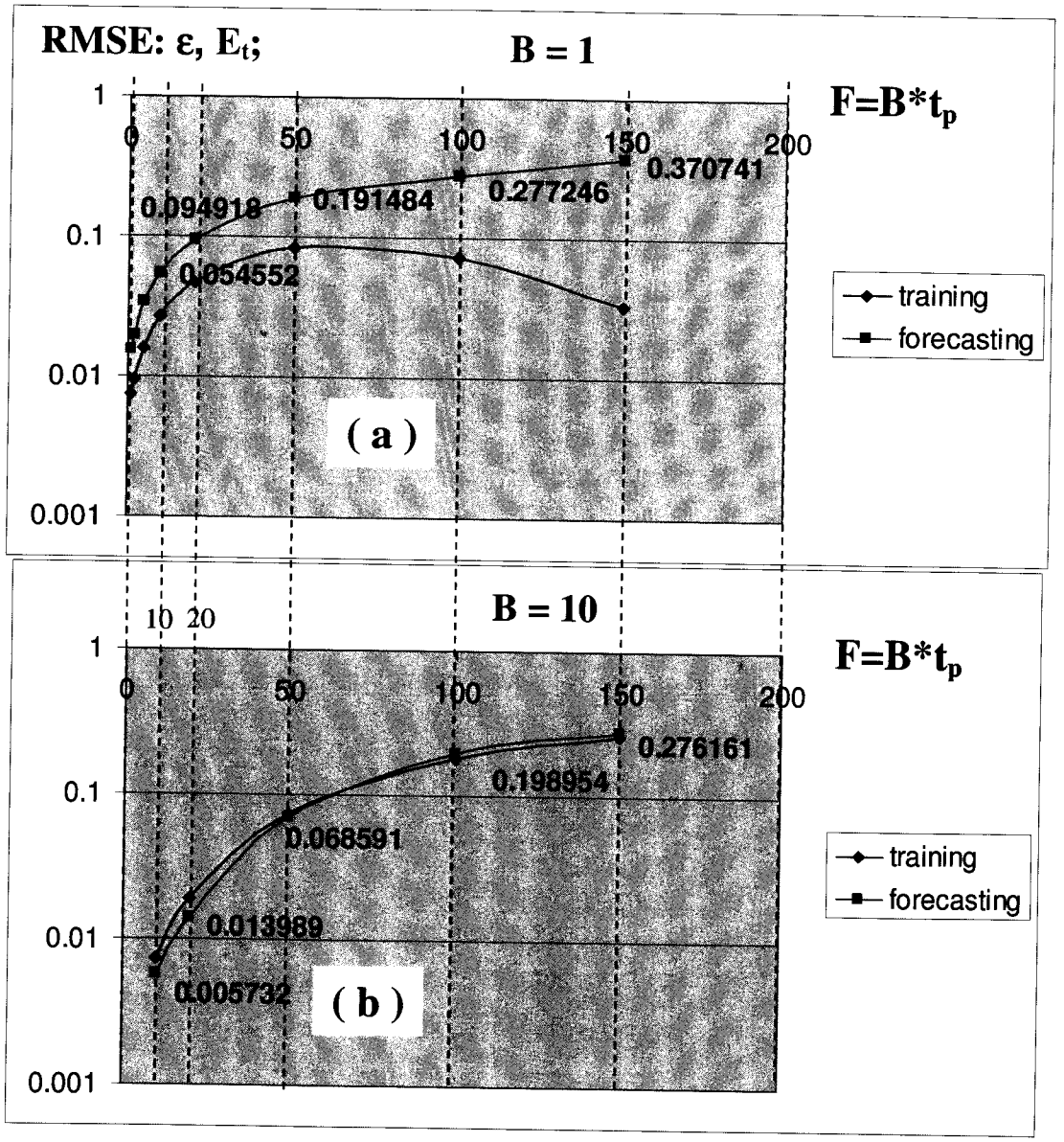


Fig.51. Comparative summarized performances of the MLP's forecasters for Mackey-Glass 30 series with the RMSE as a performance measure: (a) B=1; (b) B=10; ε and E_t are RMSE for training and forecasting.

The result of the implementation of the above approach for B=10, t_p=1 and n=5 is shown in Fig.52, where the forecaster with 5 inputs is trained within points from the designed set (indexed k=1), covering time intervals [1; 2000] and then tested as a forecaster on the interval [2000, 5000]. The quality of the forecaster in the area [2000; 5000] can be compared to that of the regularly trained forecaster, performance of which is shown in Fig.47 for the overlapping area of pattern numbers T = [3000; 3510].

Training and verification of the forecasters for the higher depths, namely for F = 20 and F = 50, are shown in Fig.53 and in Fig.54 respectively. Comparisons of those should be performed with that of Fig.48 (a 20-steps-ahead forecaster) and in Fig.49 (a 50-steps-ahead forecaster), with the time intervals adjusted accordingly to address the pattern numbers T = [3000; 3500 + B * t_p]. Here the pattern numbers T are connected to the time point numbers t via relation:

$$T = \text{pattern number} = t + n * B + B * t_p ; (5.2)$$

which reflects the fact that the earliest forecast involving the time point t needs to be made for the point T, which is separated from t by the predicting base length $L_B = n * B$ and the forecasting depth $F = B * t_p$.

The performance of the forecasters with the base step B=10 for various depths F from 10 to 150 is summarized in Fig.51,(b), in which the RMSE parameters ε and E_t of training and verification sessions respectively are used for the quality assessment. Two improvements are seen as compared to the same shown in Fig.51,(a): (1) the spread between RMSE trends for training and testing is eliminated; and (2) demonstrated values of RMSE become smaller. So, firstly, the increase of the base step B allows to keep training in line with the forecasting, when improvement in the training leads to

proportional improvement in the forecasting. Moreover, the forecasting accuracy actually becomes better than that in the training, as the E_t stays below ε for F<50. And secondly, the RMSE errors become smaller for increased B, particularly for intermediate values of forecasting depth F < 50. Indeed, E_t for F=10 in case of B=10 goes down to 0.005732 from 0.054552 in B=1 case, which is 9.5 times improvement, i.e. almost order of magnitude in RMSE's scale. This is seen as a reduction of the quadratic errors $[\underline{X}(T) - x(T)]^2$ in Fig.52 in comparison with the same in Fig.47 – in the first case (B=10) the $[\underline{X}(T) - x(T)]^2$ keeps under 10^{-4}, while in the second case (B=1) it mostly stays above 10^{-4} level and even reaching almost 10^{-2} level in the vicinity of T=3400. Only slightly smaller improvement takes place for the next point of Fig.51,(b), i.e. F = 20. The E_t for F = 20 in the case of B = 10 is 0.013989 while it was 0.094918 in B = 1 case, which constitutes 6.78 times improvement. Again, in terms of quadratic error reduction it can be seen when comparing the F = 20 forecaster of Fig.53 (B=10) with that of Fig.48 (B=1): the $[\underline{X}(T) - x(T)]^2$ keeps under 10^{-3} level in B=10 case, while in B=1 case it mostly stays above 10^{-3} level and breaking 10^{-2} level for 3400 < T < 3500.

Less dramatic, but still appreciable improvement of the RMSE is for F=50 of Fig.51,(a,b): 0.068591 versus 0.191484 of E_t for B=10 and B=1, i.e. 2.79 fold improvement, which also reflects the $[\underline{X}(T) - x(T)]^2$ change from mostly under 10^{-2} level for F=50 forecaster in Fig.54 to reaching almost 0.1 level at 3400 < T < 3500 for the same F=50 forecaster in Fig.49. Some improvement remains for further points, F=100 and F=150, but it is vanishing.

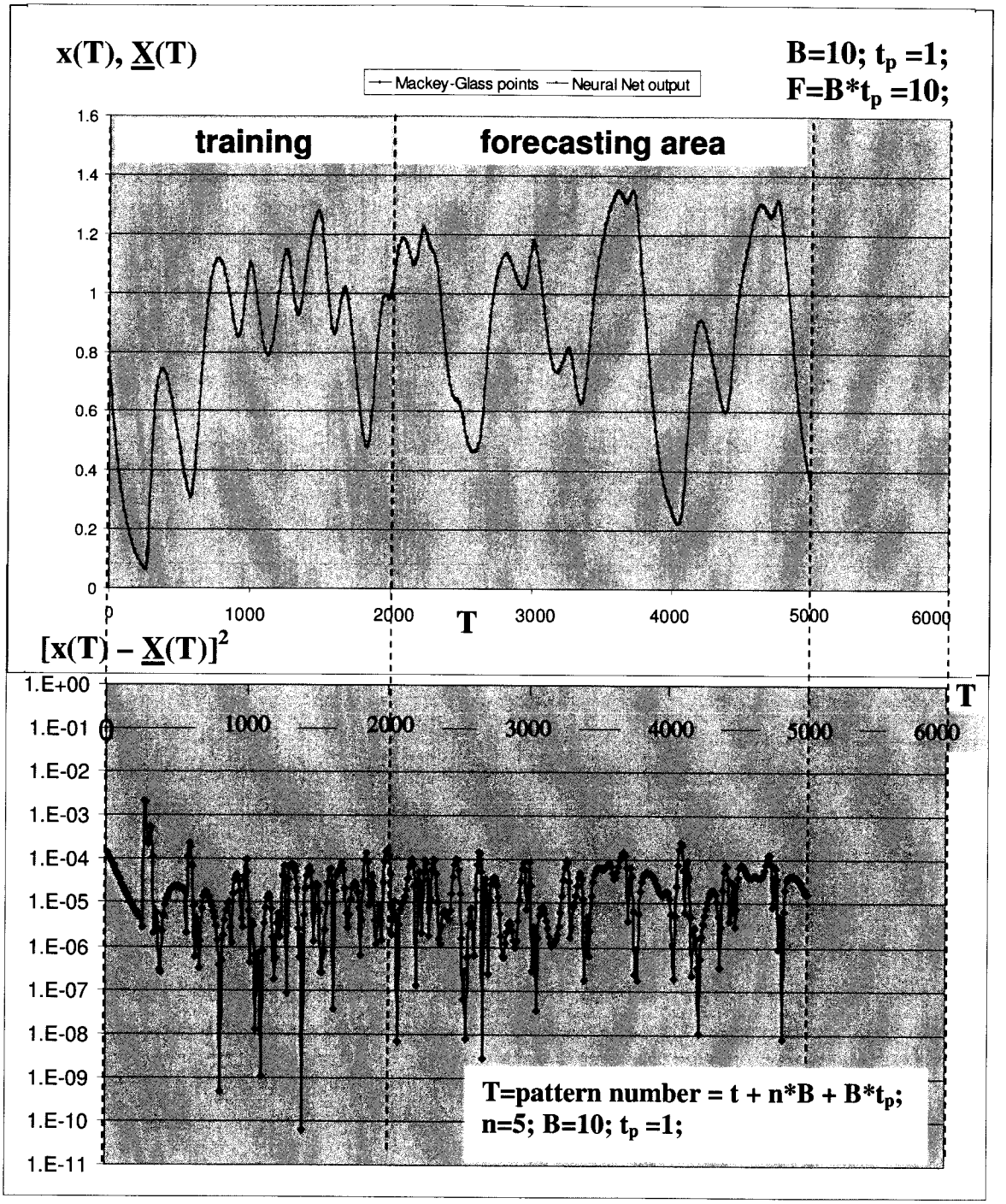


Fig.52. Performance of the F=10 MLP's forecaster trained on the reduced subset of the Mackey-Glass time series, consisting of each 10^{th} point of the initial set starting from the first one (indexed k=1).

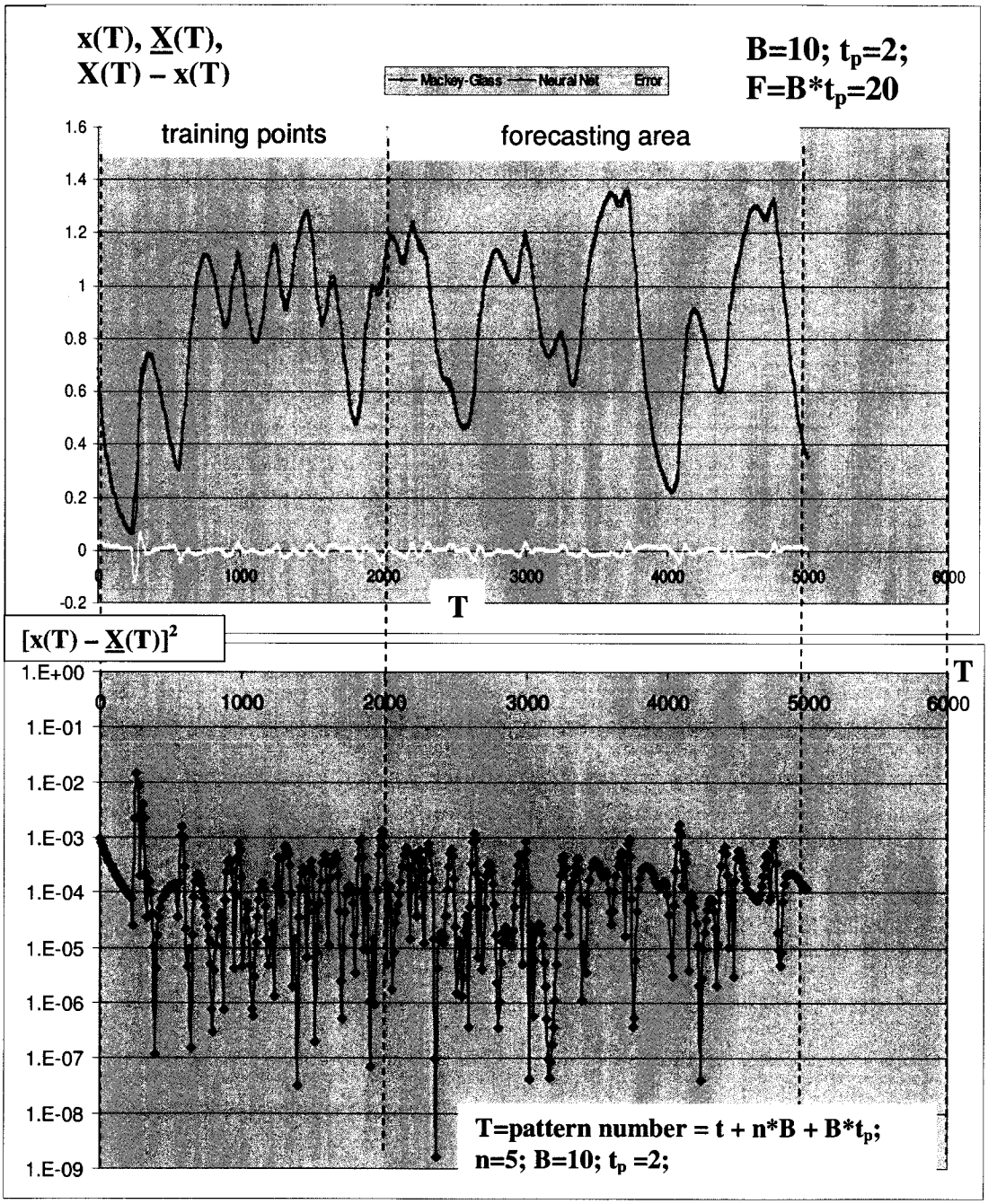


Fig.53. Performance of the F=20 MLP's forecaster trained on the reduced subset of the Mackey-Glass time series, consisting of each 10^{th} point of the initial set starting from the first one (indexed k=1).

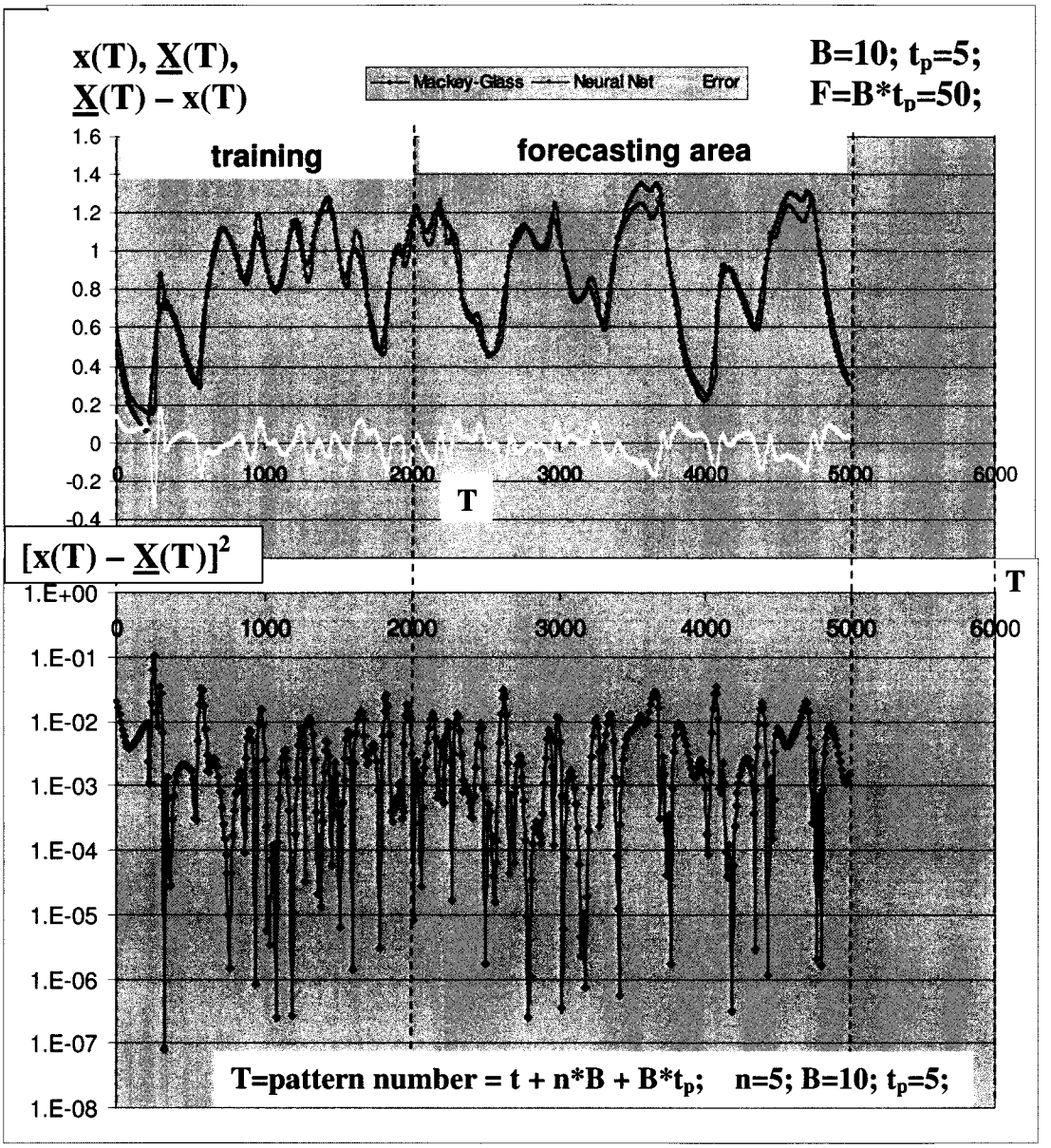


Fig.54. Performance of the F=50 MLP's forecaster trained on the reduced subset of the Mackey-Glass time series, consisting of each 10^{th} point of the initial set starting from the first one (indexed k=1).

The low RMSE level of 0.005732 and the quadratic error under 10^{-4} level demonstrated by the F=10 forecaster in Fig.52 allows to accurately recompose the initial time series forecast in full point-by-point by feeding into the MLP's F=10 forecaster trained on the k=1 subset the data points for the rest of subsets of k=2, 3, … , 10. Implementation of this procedure is shown in Fig.55 through Fig.58. First, in Fig.55 the results of the forecasting are shown of the forecaster with F=10 (n=5; B=10; t_p =1;) for the pattern numbers 3000 < T < 3500 only with the set of points indexed k=1 and used for both the training and the forecasting purposes. It is seen that the accuracy of the forecasting is very high and the sole problem remains that only every 10^{th} point is forecasted, missing the intermediate ones with k=2, 3, …, 10. This can be addressed by re-using the forecaster on the data points from the rest of the sets and combining the forecasts together. Implementation of this approach is shown in Fig.56 for the whole range of the patterns T = [1; 5000]. In this Fig.56 the training points included are only those from the subset k=1, while testing of all of the rest of the subsets has been conducted without any further training, but rather by directly feeding the data points from the sets k=2, 3, …, 10 into the forecaster trained on the set k=1. It is seen that lack of the training on the set points k=2, 3 … 10 does not have detectable negative effect and the accuracy remains the same for the extra sets as it was for the initial one with k=1. This supports the presence of redundancy in the time series, which had been eliminated by splitting it into subsets in the first place. Composing all of forecasts together restores the whole time series required. Enlarged portion of the restored series composed of the 10 sets together in one plot is shown in Fig.57. It is seen that the prediction points strongly correlate producing consistent forecasts. To show the separate points of this picture, the stronger enlargement is used in Fig.58, where

forecasting points for subsets j = k = 1, 2, 3, ..., 10 go one after another mostly overlapping with the initial time series, indexed there as j=0. The error points [$\underline{X}$(T) – x(T)] are shown there too in the same scale indexed j=11, 12, ..., 20 with j = k + 10 being the error of the forecasting pattern points of the subsets with k=1, 2, ..., 10. This demonstrates the operational ability of the suggested method.

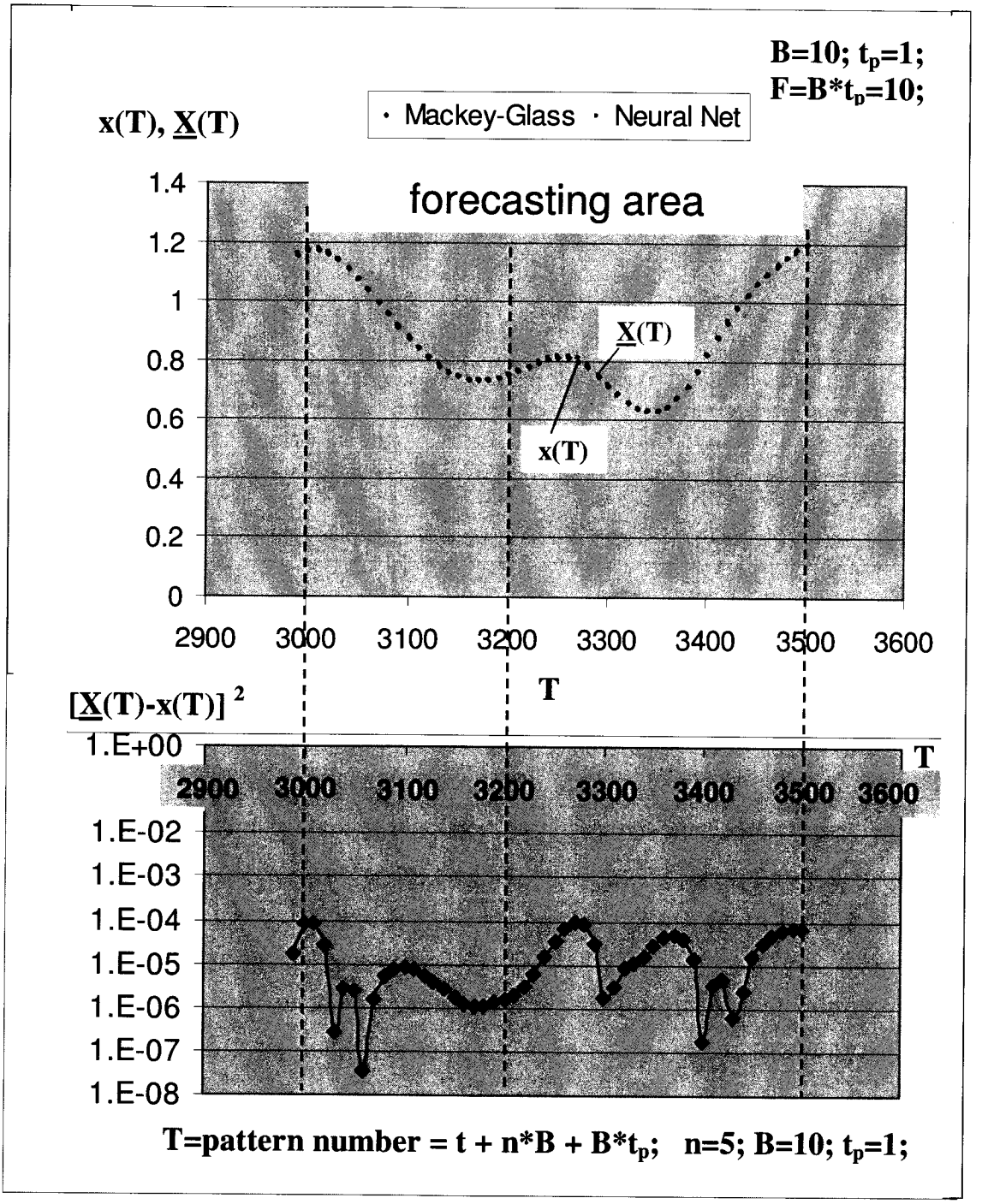


Fig.55. Forecasting results by the forecaster trained on the designed training set (k=1) consisting of each 10^{th} of the Mackey-Glass initial series. Forecasting by 1 step here is equivalent to the10 steps in the initial series.

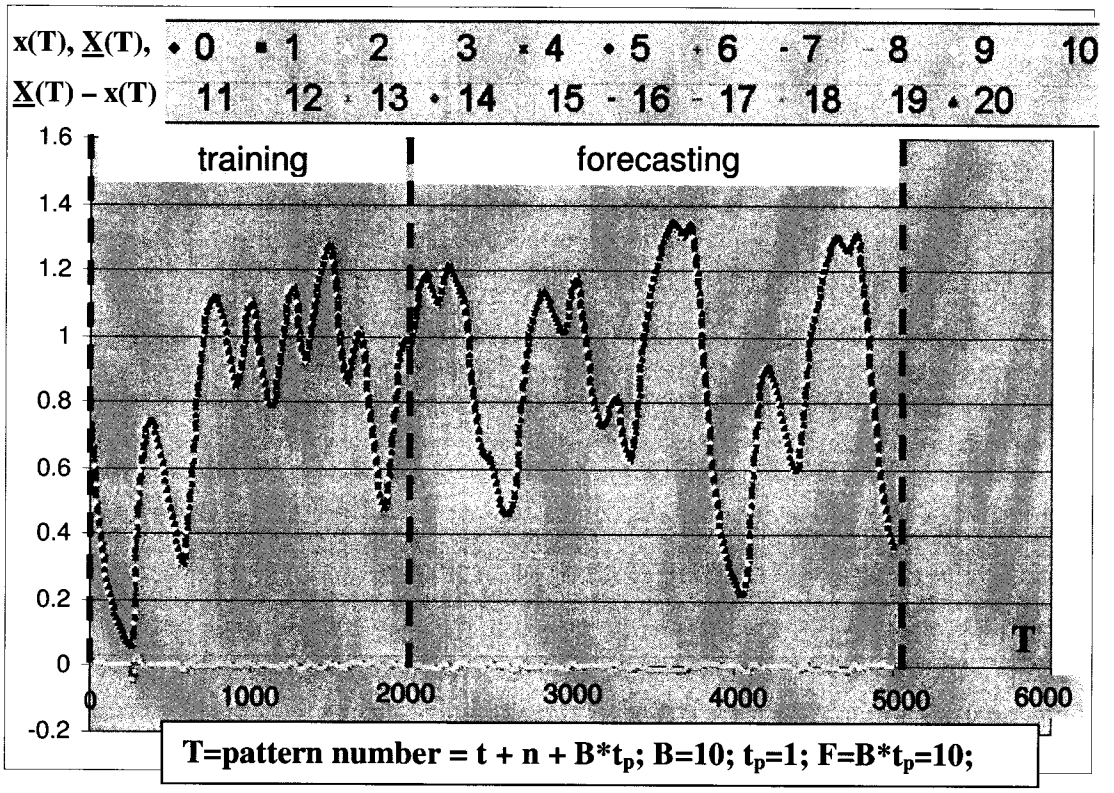


Fig.56. Joint forecast built by the MLP with F=10 trained on the subset indexed k=1 and tested by sequentially incorporating subsets with k=2, 3, ..., 10 with no further training on extra-subsets (so that for k=2, 3, ..., 10 all outputs are forecasts only within the whole range of the points). Indexed j=0, 1, 2, ..., 20 are the following: j=0 are the pattern points from the original Mackey-Glass series; j=1, 2, .., 10 are for the subsets k=1, 2, ..., 10; j=11, 12, ..., 20 are the errors for the subsets k=j-10.

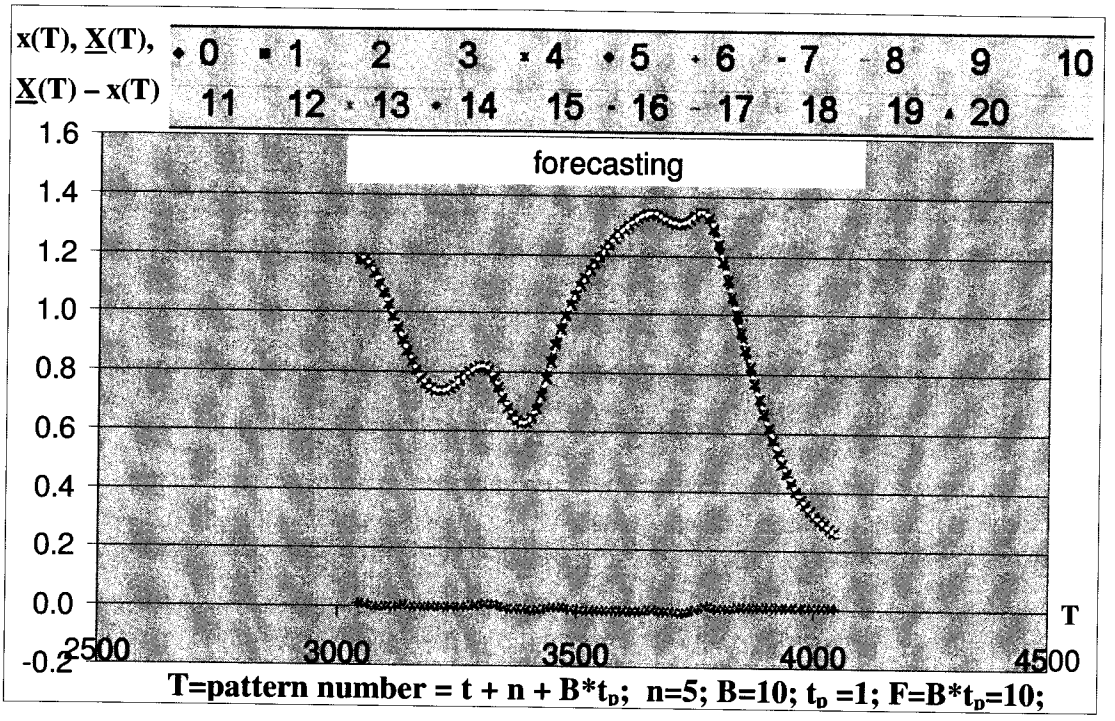


Fig.57. Enlarged portion from Fig.56 of times series built point-by-point by the F=10 forecaster for the 3015<T<4016 patterns.

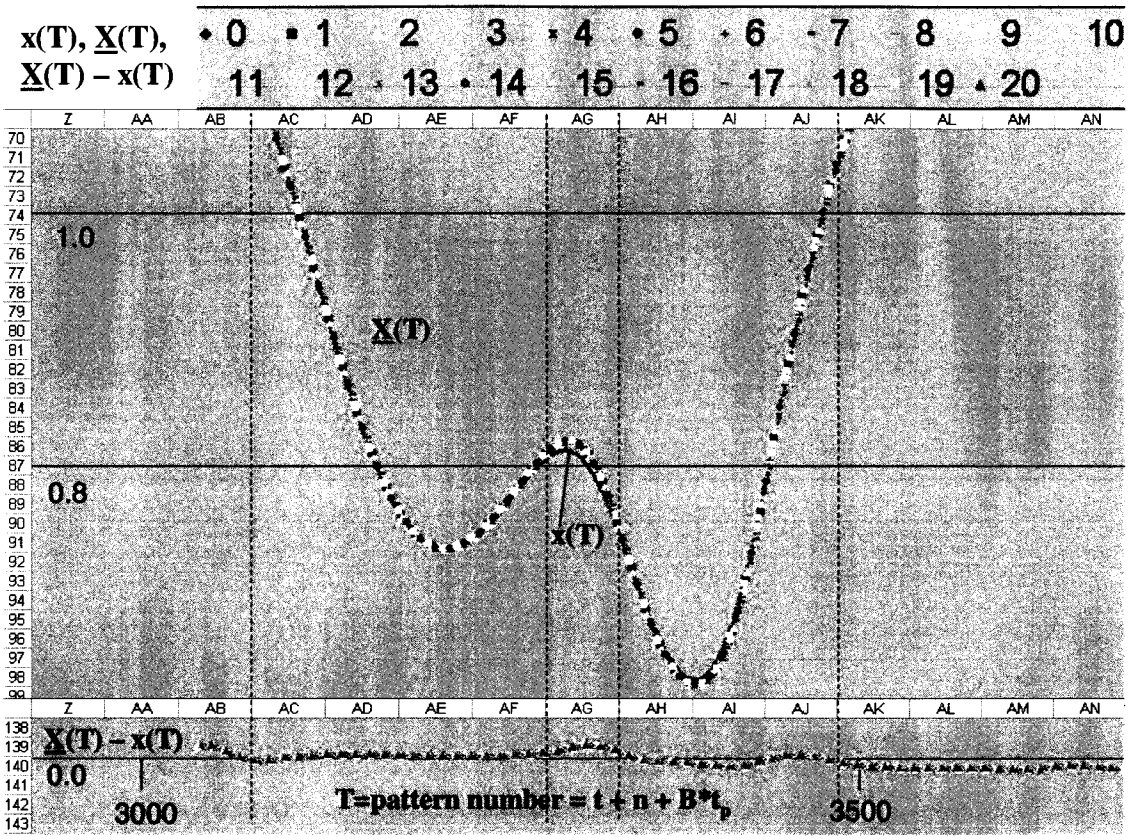


Fig.58. Enlarged portion from Fig.56 and Fig.57 of the Mackey-Glass times series built point-by-point by the F=10 forecaster for the 3015 < T < 3515 patterns.

To conclude, in parallel Cell/B.E.'s environment, the multiple steps forecasts can be distributed between the available SPUs and gathered together on the PPU module. Separation of the forecasts allows accelerating the process of obtaining the joint picture of the multiple step forecasts, which is essential for the time series case. Fast track for the individual forecasts is available and allows obtaining most of the forecasting power in the short training session. Varying the base step parameter B offers one more useful dimension for parallelization, which in some cases, such as above mentioned forecasts by F=10 and F=20 allows essential accuracy increase due to prediction base expansion effect.

5.7. Comparison with a single CPU

As it is seen above, parallelization procedure for the time series prediction allows to simplify the initial task – instead of solving single complex problem of multiple forecasts from the given data set, the parallelized approach allows pursuing the same forecasts separately by individual processors assigning one processor for one forecast. Simplification of the task for each processor from multi-dimensional to two-dimensional brings about the fast track for convergence for each processing unit and thus opens up a possibility not available otherwise – to complete the training process within the fast track and to obtain predictor within shortest possible time frame. Additional possibility – extending the leverage of the training points into the preceding history – is available equally for a single processor as well as in the parallel environment. However, only in the parallel environment it is available simultaneously with the forecasts from the short term perspectives.

5.8. Summary on predictions of the 1D time series

The above consideration supports that the Cell/B.E. is allowing an efficient multi-step prediction in the 1D time series, offering the following advantages.

1. Splitting the complex multi-step forecasting task into the separate individual fixed-step forecasting sub-tasks and distributing them among the parallel processors allows attaining the fast track in the training for each individual sub-task. The above goes in addition to the natural acceleration of the calculations due to simplification of the individual sub-tasks. Attaining the same fast track in the training with a single processor carries the cost

of sequential implementation of each, which therefore favors the parallel computing environment.

2. Scaling up the base step used for the 1D time series prediction by the neural network MLP trained via the gradient descent algorithm allows: (i) removing redundancy in the data and (ii) at the same time improving the accuracy of the multi-step long range forecasts; (iii) identifying the occurrence of the negating trend. In the parallel computing environment these advantages (i) - (iii) are attainable when using the base step as one of the parallelization parameter and engaging for each base step value a separate processing unit, thus covering otherwise unavoidable time cost in a single processor approach.

Chapter 6. Overall conclusions

As summarized below, to answer the research question of this thesis identified and demonstrated are the following advantages, offered by the parallel programming environment of the Cell/B.E. for the gradient descent algorithm implementations in terms of speed and accuracy of the attained solutions of the 2D functions approximations and of the 1D time series predictions by multilayer perceptron neural network.

6.1. Summary of conclusions

1. For nonlinear 2D function approximation of second and third order with saddle point the parallel tracing of the gradient descent trajectories from various starting conditions allows for separation of those which lead to local minima traps and those with the fast track convergence to the optimal solution with minimized training computational cost.

2. The parallel tracing of the gradient descent allows obtaining a set of viable candidates for the 2D non-linear function approximation, suitable for final selection of the optimal solution in overall testing procedure when available. The winner does not necessarily coincide with that in the training, which makes it advantageous to retain all the potential candidates from the SPE processors intact available for the final determination of the winner. This way the parallel tracing enhances the probability in attaining the best optimal solution in a fastest way, which contributes further to the advantages of the parallel computing environment.

3. The criteria of obtaining the reduced RMSE for the close proximity extrapolation region is demonstrated to serve as the indication parameter to distinguish the high quality

approximator for the interpolation region among those candidates obtained in the parallel tracing mode. Thus the close range extrapolation ability of the MLP neural network is shown to reflect its interpolation capacity in that low root mean square error (RMSE) for the close proximity extrapolation range correlates with the low interpolation RMSE for selected 2D non-linear function approximators. This enables an early identification of possible optimal solutions prior to or in the absence of a possibility for the comprehensive testing of the obtained solutions.

4. For the 1D time series prediction the splitting of the complex multi-step forecasting task into the individual fixed-step forecasting sub-tasks and distributing them among the parallel processors allows attaining a fast track in training for each individual subtask. The parallel computing environment is shown to allow efficient tracing and gathering of the solutions, offering speed advantage with respect to a single processor situation.

Scaling up the base step used for the 1D time series prediction by the neural network MLP trained via the gradient descent algorithm allows: (i) removing redundancy in the data and (ii) at the same time improving the accuracy of the multi-step long range forecasts; (iii) identifying the occurrence of the negating trend. In the parallel computing environment these advantages (i) - (iii) are attainable when using the base step as one of the parallelization parameter and engaging for each base step value a separate processing unit, thus covering otherwise unavoidable time cost in a single processor approach.

6.2. Summary of contributions

The following contributions of *new* knowledge have been made by this thesis.

1. Using a neural approximation of the 2D functions of 2^{nd} and 3^{rd} order with saddle points, the parallel tracing of the gradient descent trajectories is demonstrated as a viable alternative in identifying the favorable starting conditions, avoiding local minima traps and obtaining the optimal solution in a quickest possible time frame. It is shown that optimality of the solution can only be determined by overall testing, until finalization of which all the viable candidates obtained in the parallel tracing need to be retained. For the first time the close proximity extrapolation ability of the neural network is shown to reflect its interpolation quality in the 2D function non-linear approximation.

2. A new approach is introduced for performance improvement of the neural network MLP multi-step forecaster in the 1D time series. The approach includes simultaneous (i) redundancy removal in the 1D time series and (ii) base step increase for the prediction basis, implementation of which results in accuracy increase of the forecasts. For the Mackey-Glass 30 time series a 10 fold simultaneous redundancy removal and a base step increase is shown to result in almost order of magnitude improvement of the RMSE for the 10-steps and 20-steps forecasts, as well as more than double in accuracy for long-term forecasts of 50-, 100- and 150-steps ahead. In the parallel environment splitting the multi-task into the individual ones and distributing of the individual forecasting tasks between the processors is shown to provide fast-tracks for

training of the neural networks, as well as to offer alternative forecasts for selection of the best available accuracy.

3. A new approach is demonstrated capable to identify occurrence of the negating trend in the compositional 1D time series. The approach is based on evaluating the performance of the neural network MLP forecaster and includes comparing RMSE dependences of the long range forecasts from the number of training epochs for the minimal base step and the scaled up base step of the prediction basis. Specifically, it is demonstrated, that for a steady continuous trend doubling the base step results in almost doubling the accuracy of the 10-steps forecasts, while for negating trend in the same time series such procedure creates an expressed optimum in training length for the same forecasts. Parallel tracing of the above dependences enables an in-situ detection of the negating trend occurrences at the central control unit.

6.3. Future research

The extension of the presented research is seen in applications to specific systems based either on Sony Playstation-3, which includes the Cell/B.E. processors or on IBM's Cell/B.E. Blade Center with multiple Cell/B.E. processors.

More specific examples are:

- applications to controllers of the 2D functions with Sony Playstation-3;
- applications of the neural predictors in software for stocks prediction and/or video compression with Sony Playstation-3 and/or IBM Blade Center.

References:

[1]. Simon Haykin. Neural Networks: A Comprehensive Foundation. - Pearson Education (Singapore), 1999.

[2]. Wieland, A. and Leighton, R. "Geometric Analysis of Neural Network Capabilities", IEEE First International Conference on Neural Networks, San Diego, California, June 1987, Vol. 3, pp. 385 – 392 (1987).

[3]. A. R. Barron, "Neural net approximation", - Proc. of the 7-th Yale workshop on adaptive and learning systems, (1992).

[4]. A. R. Barron, "Universal approximation bounds for superpositions of a sigmoidal function",- IEEE Trans. Information Theory, vol. 39, pp. 930-944 (1993).

[5]. A. R. Barron, "Approximation and estimation bounds for artificial neural networks",- Machine Learning, vol. 14, pp. 113-143 (1994).

[6]. G. Cheang, "Non-linear network approximation and estimation of functions", - Proc. of the 1994 IEEE-IMS Workshop on Information Theory and Statistics, p.59 (1994).

[7]. G. Cheang and A. R. Barron, "Estimation with two hidden layer net",- Neural Networks, IJCNN'99 International Joint Conference on, vol. 1, pp. 375-378 (1999).

[8]. C. K. Chui, Xin Li and H. N. Mhaskar, "Limitations of the approximation capabilities of neural networks with one hidden layer"- Advances in Computational Mathematics, Volume 5, Number 1 / December, pp.233-243 (1996).

[9]. F. L. Lewis, J. Campos and R. Selmic, "Neuro-fuzzy control of industrial systems with actuator nonlinearities",- Society for Industrial and Applied Mathematics (SIAM), Philadelphia, 2002, 244 p.

[10]. J. Park and I. W. Sandberg, "Criteria for the approximation of non-linear systems",- IEEE Trans. Circuits Systems, 39 , pp. 673-676 (1992).

[11]. G. A. Rovithakis and M.A. Christodoulou, "Adaptive control of unknown plants using dynamical neural networks",- IEEE TransSystems Man Cybernet., 24 pp. 400-412 (1994).

[12]. Cellular neural networks: theory and applications. Editors A. Slavova and V. Mladenov, New York, Nova Science Publishers, Inc., 2004, 183 p.

[13]. Malcolm Stagg, "A dynamic analog concurrently-processed adaptive chip" http://www.virtualsciencefair.org/2006/stag6m2/credits.html

[14]. Adel Abdennour, "Evaluation of Neural Network Architectures for MPEG-4 video traffic prediction", IEEE Trans. Broadcasting, vol.52, No.2, pp.184-193 (2006).

[15]. Shasha, Dennis (2004). High Performance Discovery in Time Series. Berlin: Springer. ISBN 0387008578.

[16]. M.C. Mackey and L. Glass: Oscillation and chaos in physiological control systems. Science, Vol 197: 287-289, (1977).

[17]. Chen, Yuehui, Yang, Bo, Dong, Jiwen , Abraham, Ajith , "Time-series forecasting using flexible neural tree model"- Information Sciences. Vol. 174, no. 3-4, pp. 219-235. Aug. 2005.

[18]. Y Chen, B Yang, J Dong, "Non-linear system modeling via optimal design of neural trees"- International Journal of Neural Systems, Vol. 14, No. 2, pp.1–13 (2004).

[19]. Abraham, Ajith, "EvoNF: A Framework for Optimization of Fuzzy Inference Systems Using Neural Network Learning and Evolutionary Computation"- The 17th IEEE

International Symposium on Intelligent Control, ISIC'02, IEEE Press, ISBN 0780376218, pp 327-332, 2002. http://arxiv.org/abs/cs/0405032

[20]. Littmann, E. and Ritter, H., "Cascade network architectures"- in Proc. Intern. Joint Conference On Neural Networks. (1992) http://eprints.kfupm.edu.sa/29300/

[21]. Steve Lawrence, Ah Chung Tsoi, Andrew D. Back, "Function Approximation with Neural Networks and Local Methods: Bias, Variance and Smoothness"- Australian Conference on Neural Networks, ACNN 96, Edited by Peter Bartlett, Anthony Burkitt, and Robert Williamson, Australian National University, pp. 16-21, 1996.

[22]. Sayan Mukherjee, Edgar Osuna and Federico Girosi, "Non-linear Prediction of Chaotic Time Series Using Support Vector Machines"- Proc. of IEEE NNSP'97, Amelia Island, FL, 24{26 Sep., 1997),

https://eprints.kfupm.edu.sa/53373/1/53373.pdf

[23]. Anthony J. Calise, Naira Hovakimyan, "Adaptive output feedback control of non-linear systems using neural networks"- Automatica, Volume 37 (8), pp.1201-1211 (2001).

[24]. Jing-Xin Xie, Chun-Tian Cheng, Bin Yu and Qing-Rui Zhang, "Multi-step-ahead Prediction Based on B-Spline Interpolation and Adaptive Time-Delay Neural Network",- Artificial Neural Networks: Formal Models and Their Applications - ICANN 2005, Springer, Volume 3697/2005, Pages 565-570 (2005).

[25]. Sanjay L. Badjate, Sanjay V. Dudul, "Prediction of Mackey-glass Chaotic time series with effect of superimposed noise using FTLRNN model", - Proceedings of the Fourth IASTED International Conference on Advances in Computer Science and Technology, Langkawi, Malaysia, Publisher ACTA Press Anaheim, CA, USA, Pages: 384-389 (2008).

[26]. A. Girard, C. E. Rasmussen, J. Q. Candela, R. Murray-Smith, "Gaussian Process Priors With Uncertain Inputs Application to Multiple-Step Ahead Time Series Forecasting",- http://books.nips.cc/papers/files/nips15/AA06.pdf (2002).

[27]. Liu Zunxionga, and Liu Jianhuia, "Chaotic time series multi-step direct prediction with partial least squares regression",- Journal of Systems Engineering and Electronics, Volume 18, Issue 3, Pages 611-615 (2007).

[28]. Ma Qian-Li, Zheng Qi-Lun, Peng Hong, Zhong Tan-Wei and Qin Jiang-Wei, "Multi-step-prediction of chaotic time series based on co-evolutionary recurrent neural network",- Chinese Physics B Volume 17, Number 2, p.536 (2008).

[29]. M. P. Pushpalatha, N. Nalini, "An integrated multistep prediction system based on wavelet filter analysis and improved instance based learning (IIBL)",- Proceedings of the International Symposium on Biocomputing, Calicut, Kerala, India, POSTER SESSION: Poster session, Article No.: 47 (2010)

[30]. R. Boné, M. Crucianu, "Multi-step ahead prediction with neural networks: a review",- 9èmes rencontres internationales « Approches Connexionnistes en Sciences Économiques et en Gestion » (ACSEG 2002), 21-22 novembre 2002, Boulogne sur Mer, France, pp. 97-106 (2002).

[31]. Singh, L. , Kumar, S. , "Parallel Evolutionary Asymmetric Subset-hood Product Fuzzy-Neural Inference System with Applications"- Fuzzy Systems, 2006 IEEE International Conference, Dayalbagh Educ. Inst., Agra, pp. 1858-1865 (2006).

[32]. A. V. Pavlov, R. Z. Zakirov, V. S. Bilyk and V. V. Vedeneev, "Holographic neuro-predictor for fractional Brownian motion"- Proc. SPIE, Vol. 5036, 454 (2003).

http://dx.doi.org/10.1117/12.498462

http://www.springerlink.com/content/r677030m27p15411/

[33]. Hubert Eichner, Tobias Klug and Alexander Borst, „Neural Simulations on Multi-Core Architectures",- Front Neuroinformatics. 2009; 3: 21; 10.3389/ neuro. 11.021.2009. PMCID: PMC2715271;

http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2715271/

[34]. Fujimoto Y, Fukuda N, Akabane T., "Massively parallel architectures for large scale neural network simulations"- IEEE Trans Neural Netw. 1992;3(6):876-88.

[35]. Brassai, S. Bako, L. Dan, S. , "FPGA Parallel Implementation of CMAC Type Neural Network with on Chip Learning",- in: Applied Computational Intelligence and Informatics, 2007. SACI '07. 4th International Symposium on, May 18 2007, pp.111-115

[36]. Kistler, M., Perrone, M., and Petrini, F. (2006). Cell multiprocessor communication network: Built for speed. IEEE Micro, 26(3):10–23.

[37]. P Hasselström, "Artificial Neural Networks on the Cell Broadband Engine Architecture", - Master´s Thesis in Computer Science at Stockholm University, Sweden 2008.

http://www.nada.kth.se/utbildning/grukth/exjobb/rapportlistor/2008/rapporter08/hasselstrom_per_08079.pdf

[38]. Hackenberg, D. (2007). "Fast matrix multiplication on cell (smp) systems",- http://www.tu-dresden.de/zih/cell/matmul

[39]. Yong Zhang, Ming Ming Zhang, "Application of Artificial Neural Network in Video Compression Coding",- Int. Conf. Inform. Manag., Innovation Manag. And Ind. Eng. 2008 (ICIII'08), vol.1, 19-21 Dec.2008, pp.207-210.

[40]. B.Markoski, S. Pelemis, J. Mihailovic, "Applying Neuron Network to Improve Characteristic of Video Encoder", -Proc. 26th Int. Conf. Microelectronics (MIEL 2008), NIS, Serbia, 11-14 May, 2008, pp. 131-134 (2008).

[41]. Sandeep Koranne, "Practical Computing on the Cell Broadband Engine", Springer, 2009, XXXV, 485 p. http://www.springer.com/computer/communications/book/978-1-4419-0307-5

[42]. Abraham Arevalo, Ricardo M. Matinata, Maharaja Pandian, Eitan Peri, Kurtis Ruby, Francois Thomas, Chris Almond, "Programming the Cell Broadband Engine™ Architecture: Examples and Best Practices" Publisher: IBM® Redbooks®, 2008, 642 p. http://www.redbooks.ibm.com/redbooks/pdfs/sg247575.pdf

[43]. Matthew Scarpino, "Programming the Cell Processor: For Games, Graphics, and Computation" Prentice Hall, 2008, 744 p. [Sample Chapter: Introducing the Cell Processor @ http://www.informit.com/articles/article.aspx?p=1250899]

[44]. Michael Gschwind, David Erb, Sid Manning, and Mark Nutter, "An Open Source Environment for Cell Broadband Engine System Software"- IEEE Computer, Vol. 40, No. 6, June 2007.

http://www.research.ibm.com/people/m/mikeg/papers/2007_ieeecomputer.pdf

[45]. Michael Gschwind, "The Cell Broadband Engine: Exploiting Multiple Levels of Parallelism in a Chip Multiprocessor",- International Journal of Parallel Programming, Vol. 35, No. 3, June 2007.

http://domino.research.ibm.com/library/cyberdig.nsf/papers/1B2480A9DBF5B9538525723D0051A8C1/$File/rc24128.pdf

[46]. Michael Gschwind, H. Peter Hofstee, Brian Flachs, Martin Hopkins, Yukio Watanabe, Takeshi Yamazaki, "Synergistic Processing in Cell's Multicore Architecture"- IEEE Micro, Vol. 26, No. 2, pp.10-24 (2006).

http://www.research.ibm.com/people/m/mikeg/papers/2006_ieeemicro.pdf

[47]. A. Eichenberger, K. O'Brien, K. O'Brien, P. Wu, T. Chen, P. Oden D. Prener, J, Shepherd, B. So, Z. Sura, A Wang, T. Zhang, P. Zhao, M. Gschwind, "Optimizing Compiler for the Cell Processor"- PACT 2005, September 2005.

http://domino.research.ibm.com/comm/research_people.nsf/pages/alexe.publications.html/$FILE/paper-eichen-pact05.pdf

[48]. LiMin Fu, Neural Networks in Computer Intelligence, 1994 by McGraw-Hill, p.211.

[49]. Gevins, A.S. and Morgan, N.H. (1988) Application of neural network (NN) signal processing in brain research. IEEE Transactions on Acoustics, Speech and Signal Processing, 36, pp.1152-1166.

[50]. Casselman, F. and Acres J.D. (1990) DASA/LARS, a large diagnostic system using neural networks. In Proceedings of IJCNN (Washington, D.C.), pp.II-539-542.

[51] Computational Auditory Scene Analysis: Principles, Algorithms, and Applications, Wang, D. and Brown, G.J., Eds. (2006).

[52] Shaw-Hwa Hwang Sin-Horng Chen, "Neural network synthesizer of pause duration for Mandarin text-to-speech,"- Electronics Letters, Vol. 28 (8), pp.720-721 (1992).

[53] D. Larkin, A. Kinane, V. Muresan and N. O'Connor, "An Efficient Hardware Architecture for a Neural Network Activation Function Generator,"-

in: Advances in Neural Networks - ISNN 2006, Ed.Springer Berlin / Heidelberg, Vol.3973, pp.1319-1327 (2006).

http://www.springerlink.com/content/h27j70h25680j765/fulltext.pdf

[54] J.J. Soraghan, A. Hussain, "Neural network for predicting values in non-linear functional mappings,"- US Patent 6,453,206 (2002).

http://www.freepatentsonline.com/6453206.html

[55]. H.K. Rising, "Method and apparatus of compressing images using localized radon transforms,"- US Patent 6,424,737, (2002).

http://www.freepatentsonline.com/6424737.html

[56]. Karl Nygren, Stock Prediction – A Neural Network Approach. Master Thesis. Royal Institute of Technology, KTH. (karlnyg@kth.se) . Supervisor: Prof. Kenneth Holmstr¨om, Examiner: Dr. Torkel Erhardsson. 28th March 2004.

http://www.f.kth.se/~f98-kny/thesis.pdf

[57]. Bin Fang and Shoufeng Ma, "Application of BP Neural Network in Stock Market Prediction",- Book Series: Advances in Neural Networks – ISNN 2009, Publisher: Springer Berlin / Heidelberg, Volume 5553/2009, pp.1082-1088 (2009)

[58]. Robert R. Trippi and Efraim Turban, Neural Networks in Finance and Investing: Using Artificial Intelligence to Improve Real-World Performance, Revised Second Edition (with software), 800 pp., 1996.

[59]. Artificial Intelligence in the Capital Markets: State-of-the-Art Applications for Institutional Investors, Bankers & Traders. Roy S. Freedman, Robert A. Klein, and Jess Lederman, Editors. ISBN 1-55738-811-3, 750 pp., 1995.

http://www.sigma-research.com/bookshelf/rtbooks3.htm

[60]. Robert R. Trippi, Chaos & Non-linear Dynamics in the Financial Markets: Theory, Evidence, and Applications. 500 pp., 1995.

[61]. Virtual Trading: How Any Trader with a PC Can Use the Power of Neural Nets and Expert Systems to Boost Trading Profits. Jess Lederman and Robert A. Klein, Editors. ISBN 1-55738-812-1, 300 pp., 1995.

[63]. Dimitris N. Chorafas. Chaos Theory in the Financial Markets: Applying Fractals, Fuzzy Logic, Genetic Algorithms, Swarm Simulation, and the Monte Carlo Method to Manage Market Chaos and Volatility. ISBN 1-55738-555-6, 400 pp, 1994.

[64]. The Handbook of Investment Technology: A State-of-the-Art Guide to Selection, Implementation & Utilization. Joseph Rosen and Kevin M. Merz, Editors. ISBN 0-7863-0996-2, 400 pp., 1996.

[65]. Caulfield E, Hellander A. "CellMC--a multiplatform model compiler for the Cell Broadband Engine and x86",- Bioinformatics. 2010 Feb 1;26(3):426-8. Epub 2009 Dec 8. http://bioinformatics.oxfordjournals.org/cgi/content/abstract/26/3/426

http://polacksbacken.net/wiki/Trajectory

[66]. René Müller, Magnetic Pendulum Fractal Simulation, Project description

http://www.inf.ethz.ch/personal/muellren/pendulum/index.html

Appendix A.

List of variables

To simplify reading, herewith all variables used in this thesis are gathered and essential relations between them are summarized.

Variables used as input to neural network:

numInputs – for number of inputs, including bias; *numOutputs* – for number of outputs; *numPatterns* – for number of patterns used in training, *numHidden* – for number of hidden nodes; *numEpochs* – for number of training epochs; *LR_IH* - learning rate for input-hidden nodes connections, *LR_HO* learning rate for hidden nodes-output connections, *LR_IO* learning rate input-output direct connections; *alphaHO*, *alphaIO* and *alphaIH* - momentum coefficients with the same notations for *HO, IO and IH* indices as for learning rates (hidden nodes-output, input-output and input-hidden nodes connections); ε=Eps – root mean square error for set of training points of neural network.

ε, E_t and E_d. – RMS Errors for training points (ε), testing points (E_t) and full set of all points (E_d).

X and Y – sets of input points x and y for 2D functions z=z(x, y); X_t and Y_t are sets of testing points as well as X_d and Y_d are full sets of points for 2D functions z=z(x, y).

Variables in the sections on time series:

x(t) – time series of x values; t – time points;

$\underline{X}(t)$ – predicted values of the time series x(t);

T_b – total number of basic time points t in the time series;

t_p – the number of time-steps for prediction;

n – the number of consecutive time series values based on which the prediction of the future value k-steps ahead $\underline{X}_{t+k}$ is obtained, the actual value being x_{t+k};

T_{max} – maximal number of t points used in simulations from the time series;

$T_{max} = T_b + t_p + n$;

B – base step for prediction, i.e. the time increment between sequential time points selected from the time series to make prediction; B=1 means that time points are taken from the time series consecutively; B=2 means that every second time point is taken to make prediction, etc.

$L_B = n*B$ is predictive base, i.e. total time length from the series used to make a forecast;

$F = B*t_p$ is the length of forecast, i.e. the length in time point units from the last obtained value of the time series and the forecasted time point in the future;

t_p – a number of predictive base steps in the forecast; if B=1, then t_p coincides with F;

T – the pattern number, i.e. time point in the series, number of which is calculated as $T= t + n*B + B*t_p$; the introduction of a variable T for pattern number reflects the fact, that first $(n*B + B*t_p)$ points of the time series are in the blind spot of the forecaster as n*B of them form the basis for the forecast (starting from the very first one), while next $B*t_p$ points are those above which the forecaster is trying to see into the future, which shifts enumeration between points in the time series and pattern numbers is both are to start enumeration from #1 and up by $(n*B + B*t_p)$;

maxMG – the maximum number of the time points in the initial time series;

Appendix B.

Input parameters in the simulations

Listed below are the sets of the input parameters, used in specific simulations, results of which are presented in Tables and Figures of this thesis.

For Table 2. Training results for the MLP in approximating hyperbolic paraboloid function $z=x^2-y^2$ to achieve either precision ε=Eps=0.001 or maximal number of training epochs of 5400. Training rate was 2500 epochs/sec.

The input parameters were: *numInputs*=3, *numOutputs*=1, *numPatterns*=64, *numHidden*=10, max *numEpochs*=5400, Learning rates are: *LR_IH*=0.007, *LR_HO*=0.007, *LR_IO*=0.007; momentum coefficients: *alphaHO*=0.007, *alphaIO*=0.007, *alphaIH*=0.007; while precision ε=*Eps*=0.001 .

For Table 4. Training progress of the MLP to approximate $z = x^2 - 3*x*y^2$ function of Fig.11,(b) on various SPEs of the Cell/B.E.: the achieved root mean square error (RMSE) as a function of the number of training epochs. Training rate was 2500 epochs/sec.

Input parameters were: *numInputs*=3, *numOutputs*=1, *numPatterns*=64, *numHidden*=24, max *numEpochs*=555000; Learning rates are: *LR_IH*=0.007, *LR_HO*=0.007, *LR_IO*=0.007; momentum coefficients: *alphaHO*=0.007, *alphaIO*=0.007, *alphaIH*=0.007.

For Table 6. Results of the competition for prediction of time series seen in Fig.27 and described by $x(t) = \text{int}\,[10^3 * \sin^2(t/2)] + \text{int}\,[10^3 * \sin^2(t/20)] + \text{int}\,[10^3 * \sin^2(t/30)] + \text{int}\,[10^3 * \sin^2(t/300)]$, where t=1, 2, 3..., T_b+t_p+n, where T_b=500; t_p=3; n=5. . Training rate was 200 epochs/sec.

Final number of epoch = 5000. Input parameters are: *numInputs*=6, *numOutputs*=1, *numTrainPat*=200, *numHidden*=12, max *numEpochs*=5000, Learning rates are: *LR_IH*=0.000700, *LR_HO*=0.000700, *LR_IO*=0.000700,

momemtum coefficients: *alphaHO*=0.000700, *alphaIO*=0.000700, *alphaIH*=0.000700; while precision *Eps*=0.001000

For Table 7. Results of the competition for 3 steps ahead in time series prediction with the negating trend seen in Fig.28 and described by $x(t) = int[10^3 * \sin^2(t/2)] + int[10^3 * \sin^2(t/20)] + int[10^3 * \sin^2(t/30)] + h(t) * int [10^3 * \sin^2(t/300)]$, where t=1, 2, 3,..., T_b+t_p+n, and h(t)= {+1 for $1 \leq t \leq 210$; -1 for $t \geq 211$ }; (here T_b=500; t_p=3; n=5). Training rate was 200 epochs/sec.

Final number of epoch = 5000. The input parameters are: *numInputs* = 6, *numOutputs* = 1, *numTrainPat* = 200, *numHidden* = 12, max *numEpochs* = 5000, Learning rates are: *LR_IH* = 0.000700, *LR_HO* = 0.000700, *LR_IO* = 0.000700, momemtum coefficients: *alphaHO* = 0.000700, *alphaIO* = 0.000700, *alphaIH* = 0.000700; the precision *Eps* = 0.001000.