Accelerating the computation of parallel trajectories of gradient descent with the Cell-BE multiprocessor environment

Yuri Boiko and Gabriel A. Wainer
Carleton University
1125 Colonel By Drive,
Ottawa, ON, K1S 5B6 CANADA
yuri.boiko@rocketmail.com , gwainer@sce.carleton.ca

Keywords: neural networks, multilayer perceptron, non-linear function approximator, time series; predictor; parallel computing; Cell broadband engine (Cell-BE), Cell microprocessor, Cell/B.E.

Abstract
Neural networks offer various possibilities for function approximation. When provided a set of data points, the network learns to approximate the underlying function that generates those points. Although the network can be very efficient, the amount computation needed during the learning process can be very high. In order to improve this process, we explore the parallelization for the random scanning of starting points selected for the gradient descent algorithm using Cell-BE multiprocessor environment. We show the application of this method for approximating 3D nonlinear function, as well as for predicting 2D time series. We show that the parallel tracing of gradient descent trajectories of the 3D function approximation allows identifying a suitable starting condition for implementing an efficient gradient descent, while being able deliver the required accuracy of approximation in a shorter time. In 2D time series prediction the attained advantage is the possibility to achieve simultaneous prediction for various numbers of steps ahead. *It is shown how the Cell-BE multiprocessor offers a convenient parallel environment for the above solutions.*

1. INTRODUCTION
Artificial Neural Networks (ANN) allows achieving function approximation by learning the functionality underlying to available set of data points by either interpolating the non-linear function, or extrapolating its expected values [1-4]. The *gradient descent algorithm* is a popular method for training a popular kind of ANN: the *multilayer perceptron*. The idea is to back-propagate the errors found on output nodes in the ANN to hidden nodes; based on the adjustment of the hidden neuron's weights, the overall error is minimized [5,6]. One of the associated problems is the local minimum trapping which may occur and thus either prevent convergence or allow it to suboptimal value only. Another problem is the time required for training - which may be too long to reach a suitable solution. Both problems may be addressed using parallel computing. In this article we explore the advantages offered by the Cell-BE multiprocessor architecture. The objective here is to study the potential of Cell-BE multiprocessor parallel environment in optimizing the tasks of function approximation and/or interpolation, specifically for (1) 3D nonlinear function approximation and (2) 2D time series prediction. As a 3D nonlinear function the hyperbolic paraboloid has been taken, represented by the equation $z=x^2-y^2$, which is 2nd order polynomial function with saddle point. As a 2D time series the function $x=\text{int}[1000*\sin^2(t/2)] + \text{int}[1000*\sin^2(t/20)] + \text{int}[1000*\sin^2(t/30)] + \text{int}[1000*\sin^2(t/300)]$ has been taken, in which the last term represented long term trend for the scale of prediction considered. Multilayer percentron was the neural network of choice to resolve both task (1) and (2). It is demonstrated, that parallel tracing of gradient descent trajectories of 3D function approximator allows efficiently identifying the suitable starting condition for implementing gradient descent to realize diving trajectory and thus delivering the required accuracy of approximation in shortest time frame. The 2D time series reveal narrow distribution of the gradient descent trajectories, which in itself does not benefit from parallel tracing. The advantage from parallelization here is in splitting n-dimensional time series into n 2D ones because of naturally fast convergence track of its gradient descent training, which allows in shortest time frame to obtain simultaneous predictions for various numbers of steps ahead.

1.1. Related work
So far, at least two principally distinct parallelization approaches were realized [7] in processing data with ANN: (i) parallel connection of several networks so that each one extracts different features from the same set of data [8] (see Fig.1); (ii) splitting up a complex task into a number of subtasks to be solved by different networks [9] (see Fig.2). According to LiMin Fu [7], advantage of approach (i) is in making different analyses of the same data and as a result in extracting more information. First implementation of such scheme was by Gevins and Morgan (1988) when applying different networks to detect different types of contaminants in EEF signals [8] from the brain. Approach (i) carries potential for unexpected discoveries due to implementation

of multiple networks, because favorable combinations may occur however unexpected.

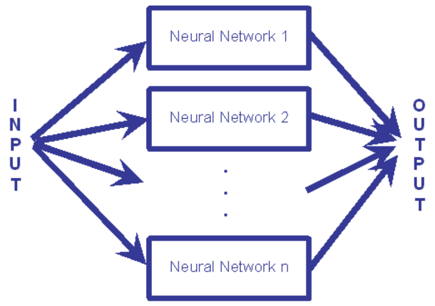

Figure 1. Parallel network **without** central control [7].

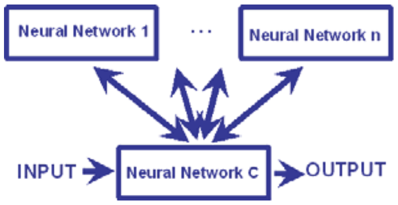

Figure 2. Parallel network model **with** central control [8].

In the scheme (ii) the results supplied by different networks may be combined in order to reach the solution of the complex task. For instance, Casselman and Acres (1990) [9] had employed parallelization scheme (ii) to conduct different diagnostic tasks of satellite spectral data. Effectively, parallelization scheme (ii) allows obtaining solution for the task, complexity of which may be too high for a single network with equivalent size, i.e. parallelization (ii) enables to obtain the solution not reachable otherwise.

For nonlinear function approximation the benefit of parallelization scheme (i) is in extensive coverage of starting condition of the gradient descent. This has potential in addressing problems of either trapping in local minima and/or identifying the efficient trajectory of descent to the global minimum. Both the above possibilities become part of present article.

Either of the parallelization schemes, (i) and/or (ii), may benefit the task of prediction in time series, which task contain several dimensions suitable for parallelization, in particular varying one of the following: 1)the initial set of time values for prediction; 2)length of the forecast; 3) depth in the history to rely in making forecast; 4)structure and/or parameters of the predicting neural network. In present article the first one of the above listed advantages will be explored in details by employing scheme (i) of parallelization. Scheme (ii) is regarded as going beyond the scope of present article and is left for future research, as well as listed advantages (2)-(4).

For parallelization purposes the environment offered by newly developed Cell multiprocessor [10-12] is suitable and will be exploited in this study.

2. PROBLEM FORMULATION

2.1 Selection of 3D function for parallel tracing

As nonlinear functions selected were those of 2nd and 3rd order with saddle points. The first function was the standard saddle surface or hyperbolic paraboloid $z=x^2-y^2$, which is shown in Fig.3. Second selected function is defined by the equation $z = x^2 - 3{*}x{*}y^2$, which is depicted in Fig.4. As a function approximator the multilayer perceptron (MLP) has been designed, which consisted of three layers with 3 inputs (x, y, 1), one hidden layer with tanh(.) activation function for the neurons and linear output neuron, thus allowing 3-D function generation.

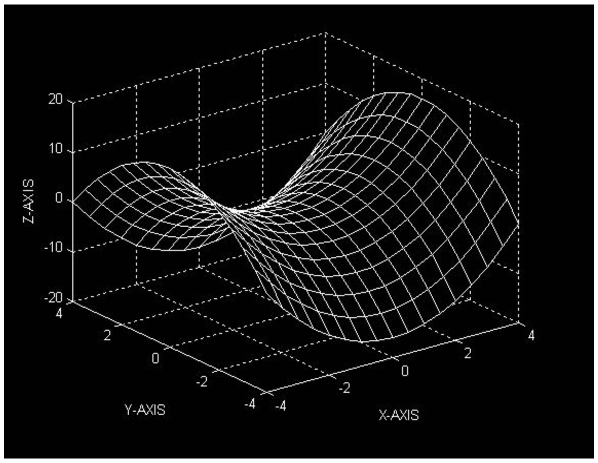

Figure 3. Analytical representation of hyperbolic paraboloid $z=x^2-y^2$ for the area under simulation.

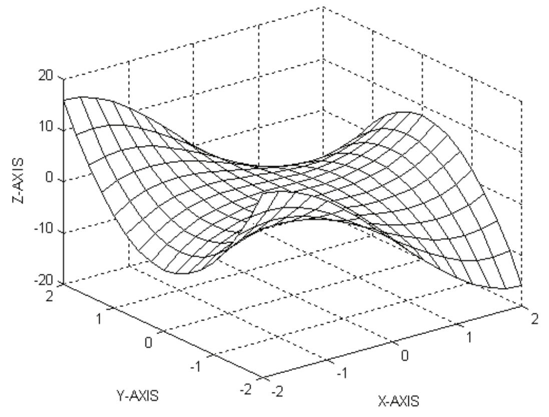

Figure 4. Analytical representation of $z=x^2 - 3{*}x{*}y^2$ for the simulated area.

2.2 Selection of Time Series

Time series predictor has been implemented on Cell/B.E. architecture, in which individual SPE processors compete to predict 3 steps ahead of the 5 current values of times series. As test functions two series were chosen: (1) series x(t) representing underlying long term trend as x=int [1000 * $\sin^2(t/2)$] + int [1000 * $\sin^2(t/20)$] + int [1000 * $\sin^2(t/30)$] +

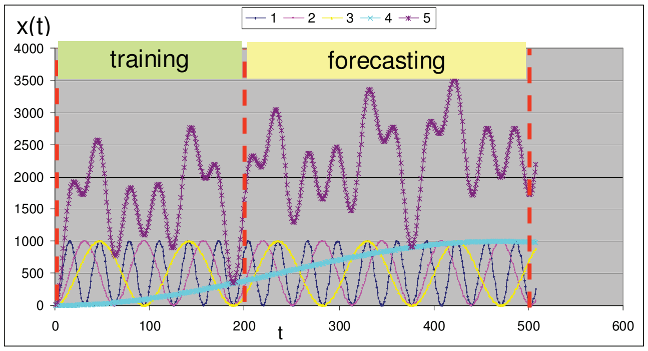

Figure 5. Time series of x=int [1000 * $\sin^2(t/2)$] + int [1000 * $\sin^2(t/20)$] + int [1000 * $\sin^2(t/30)$] + int [1000 * $\sin^2(t/300)$], where t=1, 2, 3…, T_b+t_{sp}+n. The terms of the expression are shown as Series 1 to Series 4, while the full function is represented by Series 5. (Here T_b=500; t_{sp}=10; n=5).

int [1000 * $\sin^2(t/300)$], where t=1, 2, 3…, T_b+t_{sp}+n; and which is shown graphically in Fig.5 for T_b=500; t_{sp}=10; n=5). For time series prediction the structure of three layer MLP included n=5 inputs, which tested the n consecutive time series values 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} - $\underline{x}_{t+k}$. 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, which is calculated via T_{max} = T_b + t_{sp} + n, where t_{sp} is the number of time-steps for prediction.

The above mentioned parallelization scheme (i) was employed to trace the gradient descent of training from various starting points.

For comparison, the time series with the trend changing from positive to the negative one has also been constructed, as it is shown in Fig. 6. For this series the training is performed within the same function as in Fig.5, while in the forecasting area the sign of the last term is changed to the opposite (from plus to minus) as shown in Fig.6, thus representing negating trend. Here 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 int [1000 * $\sin^2(t/300)$], but taking it in the equation with the opposite sign, for which coefficient h(t) is hcanging the value from “+1” to “-1”. It is essential here that the training set did not include the negating trend at all, so that in the forecasting area the predictor would face it for the first time.

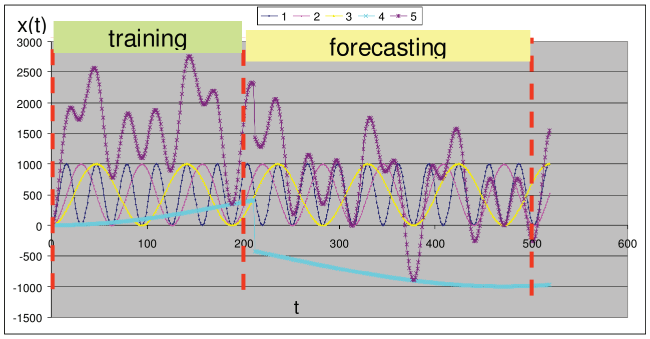

Figure 6. Time series of x=int [1000 * $\sin^2(t/2)$] + int [1000 * $\sin^2(t/20)$] + int [1000 * $\sin^2(t/30)$] + h(t) * int [1000 * $\sin^2(t/300)$], where t=1, 2, 3,…, T_b+t_{sp}+n, and h(t)= {+1 for $1 \leq t \leq 210$; -1 for $t \geq 211$ }; (here T_b=500; t_{sp}=3; n=5). The terms of the expression are shown as Series 1 to Series 4, while the full function is represented by Series 5.

2.3 Cell multiprocessor parallel environment

As a simulation environment a multithreaded synergistic mode on Cell/B.E. was implemented by allowing PPU to initiate asynchronous threads of gradient descent algorithm on available SPU's concurrently. Solutions were attained via subjecting MLP to training for selected training points until the convergence lead to attaining the desired level of mean squared error ε. The output parameters of each SPU run are weights of the MLP's neurons, which compose the neural function generator/ approximator. The algorithm solutions, which are above mentioned sets of the input weights for MLP's neurons, are classified according to the statistical characteristics of the function approximation they provide, such as under-fitting, over-fitting, minimal mean squared error fitting. The starting points (i.e. weights of the neurons) for the gradient descent algorithm deterministically define the convergence route to the solution, which satisfies the conditions of optimality in that the average mean square error for a predetermined set of training points falls below chosen limit ε..Evaluation of the quality of the achieved solution was conducted by employing two more error parameters: (i) mean squared error for the most remote from training coordinate points E_t, and (ii) overall mean squared error E_d for combined coordinates including joint set of X and Y, both including all of training and testing (used in (i)) coordinates.

According to the adopted parallelization strategy, the training of MLPs were programmed for SPEs, transferring each SPE randomly chosen starting point. The following variables are used in the programs: numInputs for number of MLP inputs, including bias, numOutputs for number of outputs, which here was = 1, numPatterns for number of patterns used in training, numHidden for number of hidden nodes of MLP, numEpochs for number of training epochs. The algorithm also included learning rates: LR_IH for input-hidden nodes connections, LR_HO for hidden nodes-output connections, LR_IO input-output direct connections as well as momentum coefficients: alphaHO, alphaIO and alphaIH with the same notations for HO, IO and IH indices. Testing of the trained MLPs was conducted by PPU processor.

The main programming feature enabling the task solution was direct memory access (DMA), which allowed initiation of parallel trajectories by SPEs by multi-threading, where each trajectory represented one thread. Threads are asynchronous, for which is provided by pthread(.) function of Cell SDK package, which interrupts main program execution until completion of all threads. The data were organized in the struct{.} in the control block file and DMA transferred to SPEs by the PPU. Termination of the pthread is achieved either by reaching the required precision Eps, or by exhausting the resource of allowed training epochs (parameter numEpochs). After termination of all threads, the PPU conducts tests for the networks obtained from each SPE.

3. RESULTS AND DISCUSSION

3.1 Simulation for hyperbolic paraboloid $z=x^2-y^2$ case

For hyperbolic paraboloid, the algorithm convergence is achieved at minimum 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]. Tests below were conducted for number of hidden neurons numHidden=10 for improved convergence.

Table 1 gathers the data of parallel tracing of training trajectories presented by 8 SPEs of Cell/B.E. engine, namely SPE-0 through SPE-7). Targeted precision value was $\varepsilon = 10^{-5}$, after achieving which the training was terminated (or continued until reaching limit of max training cycles otherwise).

Table 1. Training progress of MLP to approximate hyperbolic paraboloid function of Fig.3 on various SPEs of Cell/B.E. engine: achieved root mean square error as function of number of training epoch. (Input parameters are: 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)

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		

It is seen from Table 1 that some of the trajectories 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), such as those SPE-0, SPE-2, SPE-6 and SPE-7 in Table 1. At the same time others, after a quick start, such as SPE-4 and SPE-5 in Table 1, saturate the progress at certain level of accuracy in function approximation. In this example, SPE-5 did not reach targeted precision level of $\varepsilon=10^{-5}$ for the allocated training interval, while SPE-4 did achieve $\varepsilon=10^{-5}$ but at much higher calculating cost than the pack of leaders (SPE-0, SPE-2, SPE-6 and SPE-7).

The result of the above experiment therefore demonstrates that parallel environment allows detecting the overall nature of the distribution of gradient descent trajectories for a given task. In the above example the trajectories are divided on those (1) non-converging (such as SPE-1 and SPE-3), (2) saturating convergence (such as SPE-4 and SPE-5) and (3) quickly converging ("diving") to the global minimum (such as SPE-0, SPE-2, SPE-6 and SPE-7). Advantage offered by the parallel environment of Cell multiprocessor allows therefore avoiding time losses associated with possible trapping of the trajectories in local minima if sequentially attempting to find the solution with single processor.

3.2 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 become attainable 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]. hyperbolic paraboloid, the algorithm convergence. Scaling down the distance between training points by factor of 2, which was very much successful for the hyperbolic paraboloid, brings a success, which is worth mentioning. Coordinates in question become $X_t = Y_t$ = [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 2 shows data on MLP training to approximate $z = x^2 - 3 * x * y^2$ function in Cell/B.E. environment by various SPEs. Comparing to the data for hyperbolic paraboloid (Table), it is seen that the "diving depth" becomes at least 2 orders of magnitude shallower (E-3 vs E-5), which is attributed to the higher complexity of the function (3rd order nonlinearity instead of 2nd one). The earliest possibility to take advantage of the function approximator become available after 400 epochs from SPE-4, which demonstrated the converge of RMSE more than order of magnitude from its initial value (RMSE=**0.302902** in Table 2). Next order of magnitude improvement of RMSE is delivered at the cost of 25000 epochs, i.e. more than 60 times later, by 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 SPE-3 again (RMSE=**0.002463**), coming ahead of possibly trapped in local minima SPE-1 (RMSE=**0.011991**), SPE-4 (RMSE=**0.010353**), SPE-5 (RMSE=**0.011089**), SPE-6 (RMSE=**0.016755**) and SPE-7 (RMSE=**0.010210**). Closest rivals were SPE-0 (RMSE=0.007165) and SPE-2 (RMSE=**0.003166**). Here the parallelization of the gradient descent algorithm also brought about two advantages – (1)early possibility of having intermediate level of function approximator, while still continuing to pursue (2) the improved version of it at higher training cost.

Table 2. Training progress of MLP to approximate $z = x^2 - 3 * x * y^2$ function of Fig.4 on various SPEs of Cell/B.E. engine: achieved root mean square error as function of number of training epoch. (Input parameters are: numInputs=3, numOutputs=1, numPatterns=64, numHidden=24, max numEpochs=500000; 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)

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***

Table 3. Results of the competition for Time Series Prediction described by x(t) = int[1000 * $\sin^2$(t/2)] + int[1000 * $\sin^2$(t/20)] + int[1000 *$\sin^2$(t/30)] + int[1000 * $\sin^2$(t/300)], where t=1, 2, 3,…, $T_b+t_{sp}+n$; (here T_b=500; t_{sp}=3; n=5). 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

	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	0.122910	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 4. Results of the competition for 3 steps ahead in time series prediction with negating trend described by x(t) = int[1000 * $\sin^2$(t/2)] + int[1000 * $\sin^2$(t/20)] + int[1000 *$\sin^2$(t/30)] + h(t) * int [1000 * $\sin^2$(t/300)], where t=1, 2, 3,…, $T_b+t_{sp}+n$, and h(t)= {+1 for $1 \leq t \leq 210$; -1 for $t \geq 211$ }; (here T_b=500; t_{sp}=3; n=5). 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

	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**

3.3 Simulation for time series prediction

The training of time series predictor of the sequence with long term trend, described by equation x(t) = int [1000 * $\sin^2$(t/2)] + int [1000 * $\sin^2$(t/20)] + int [1000 * $\sin^2$(t/30)] + h(t) * int [1000 * $\sin^2$(t/300)], where t=1, 2, 3…, T_{max} and int(.) takes the 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 $T_{max} = T_b+t_{sp}+n$, with T_b=500; t_{sp}=3; n=5 are summarized in Table 2, where it is seen that best performance has been attained by SPE-3 and 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 best predictive capability is demonstrated by by SPE-3, which provides lowest RMSE value of **0.121441** for novel points versus that of **0.121573** for SPE-7. The worst performance here was by 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 (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, parallelization here is not beneficial unless another dimension in forecasting is present. Such dimension can be the parameter t_{sp} – the number of steps for prediction.

Parallelization on parameter t_{sp} for time series is therefore seen as the viable target.

Forecasting property of the best MLPs (those of SPE-3 and SPE-7 from Table 3) is illustrated in Fig.7. It is seen that errors are located in the vicinity of the local trend change

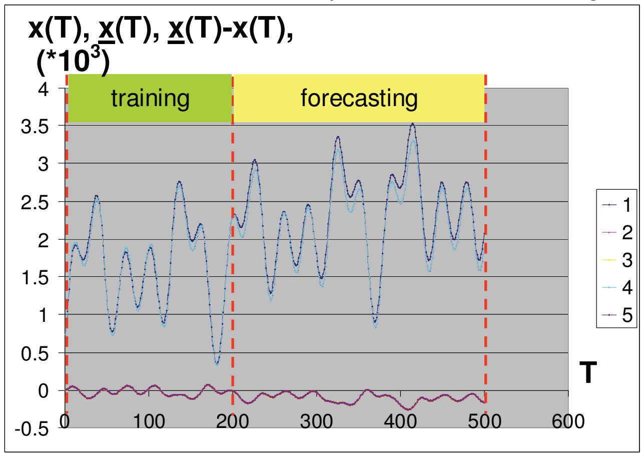

Figure 7. Prediction of time series of Fig.5 for 3 steps ahead (t_{sp} =3) by trained MLPs of SPE-3 and SPE-7. Here: 1 is function x=int [1000 * $\sin^2(t/2)$] + int [1000 * $\sin^2(t/20)$] + int [1000 * $\sin^2(t/30)$] + int [1000 * $\sin^2(t/300)$], (where t=1, 2, 3…, 507, 508); 2 and 4 – MLP output (forecasting) from SPE-3 and SPE-7 respectively; 3 and 5 – RMSE for MLPs from SPE-3 and SPE-7. All units (for t, x(t) and RMSE) are arbitrary. Values x(t) are scaled down 1000 times, thus compensatory coefficient *1000 is to be used as shown.

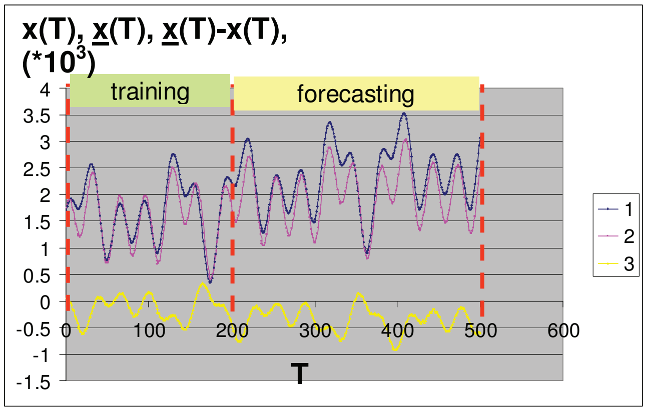

Figure 8. Prediction of time series of Fig.5 for 10 steps ahead (t_{sp} =10) by trained MLP.. Here: 1 is function x=int [1000 * $\sin^2(t/2)$] + int [1000 * $\sin^2(t/20)$] + int [1000 * $\sin^2(t/30)$] + int [1000 * $\sin^2(t/300)$], (where t=1, 2, 3…, 507, 508); 2 is MLP output (tests of forecasting); 3 is RMSE for testing the resulting MLP. All units (for t, x(t) and RMSE) are arbitrary. Values x(t) are scaled down 1000 times, thus compensatory coefficient *1000 is to be used as shown.

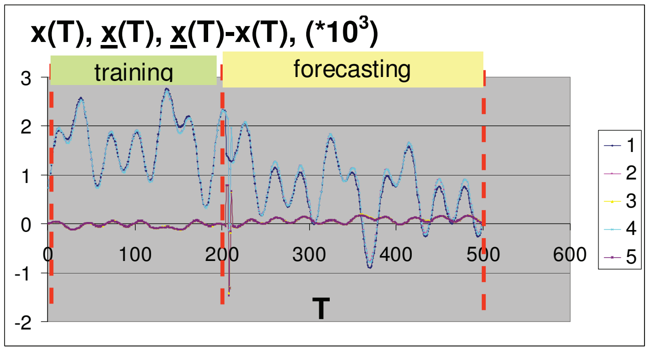

Figure 9. Prediction of time series with negating trend (Fig.6) 3 steps ahead (t_{sp}=3) by trained MLPs of SPE-3 and SPE-7. Here: 1 is function x=int [1000 * $\sin^2(t/2)$] + int [1000 * $\sin^2(t/20)$] + int [1000 * $\sin^2(t/30)$] + h(t) * int [1000 * $\sin^2(t/300)$], where t=1, 2, 3,…, $T_b+t_{sp}+n$, and h(t)= {+1 for $1 \le t \le 210$; -1 for $t \ge 211$ }; (here T_b=500; t_{sp}=3; n=5).); 2 and 4 – MLP output (forecasting) from SPE-3 and SPE-7 respectively; 3 and 5 – RMSE for MLPs from SPE-3 and SPE-7. All units (for t, x(t) and RMSE) are arbitrary. Values x(t) are scaled down 1000 times, thus compensatory coefficient *1000 is to be used as shown.

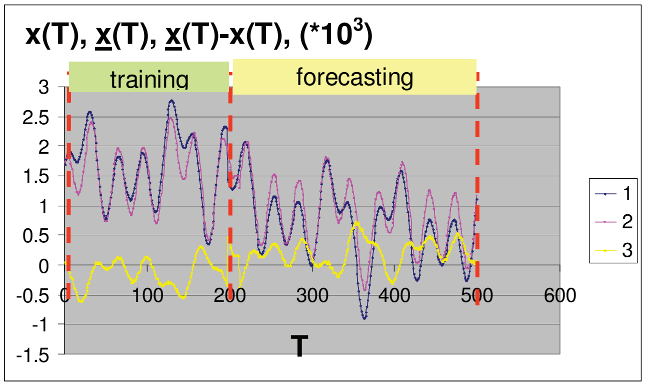

Figure 10. Prediction of time series of Fig.6 for 10 steps ahead (t_{sp} =10) by trained MLP. Here: 1 is function x=int [1000 * $\sin^2(t/2)$] + int [1000 * $\sin^2(t/20)$] + int [1000 * $\sin^2(t/30)$] + h(t) * int [1000 * $\sin^2(t/300)$], where t=1, 2, 3,…, $T_b+t_{sp}+n$, and h(t)= {+1 for $1 \le t \le 210$; -1 for $t \ge 211$ }; (here T_b=500; t_{sp}=3; n=5).); 2 is MLP output (tests of forecasting); 3 is RMSE for testing the resulting MLP. All units (for t, x(t) and RMSE) are arbitrary. Values x(t) are scaled down 1000 times, thus compensatory coefficient *1000 is to be used as shown.

areas. Otherwise, forecasting is accurate and it reflects the underlying long term trend. Changing the trend from positive (as in Fig 5) to negative one (as in Fig.6) practically retains the predictive capability of the MLP, a it is illustrated in Fig. 9 and in Table 4. 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.6 and Fig.9), the network appeared to fully retain forecasting capability for the part where the trend was opposite one, i.e. for the negated trend in Fig.6 in the forecasting area. The partial cost however for that is involved of higher RMS Error as it is seen by comparing related RMSE values of Table 3 and Table 4. Specifically, for the best performing SPEs, namely SPE-3 and 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 with that of the Fig.7 and Fig.9.

Increase of the prediction range to t_{sp}=10 leads to increase in RMSE both for training and forecasting areas, as it is shown in Fig.8 and Fig.10 for the continuing and negating trend respectfully. The RMS Error for the training sessions of 5000 epochs long for both cases (Fig.8 and Fig.10) are close (0.263152 and 0.268044) as the training sets are the same. It is ca. 4 times higher than that for t_{sp}=3 in Table 3 and Table 4, which suggest linear trend for RMSE increase with the t_{sp} parameter. For the forecasting area the values of RMSE are 0.430972 and 0.766880 respectfully, which indicates that forecasting of the negating trend (Fig.10) is almost twice less accurate as that of the continuing one (Fig.8).

Thus, it is seen that parallel tracing allows for simultaneous generation of the multiple range forecasts.

3. CONCLUSIONS

The following conclusions can be drawn form the above considerations.

1. Cell multiprocessor parallel environment allows for efficient scanning of starting points of gradient descent algorithm via parallel tracing of training trajectories from random locations.
2. For 3D nonlinear function approximator Cell multiprocessor efficiently address the problem of local minima entrapment by identifying trajectories free of that.
3. For the time series prediction parallelization within Cell multiprocessor environment brings the advantage of simultaneous forecasting for various steps ahead into the future.
4. For both, nonlinear function approximator as well as time series predictor, the parallel tracing of the trajectories allows to access an early approximations, achievable at significantly lower calculating cost (number of training epochs) , while still continuing further training and achieving higher accuracy at later stage.

REFERENCES

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

[2]. 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).

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

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

[5]. Lewis, Frank L.; Campos, J.; Selmic, R.; "Neuro-fuzzy control of industrial systems with actuator nonlinearities",- Frontiers in applied mathematics ; Society for Industrial and Applied Mathematics, Philadelphia, 2002, 244 p.

[6]. Stavroulakis, Peter., "Neuro-fuzzy and fuzzy-neural applications in telecommunications",- Signals and communication technology; Engineering online library, Springer-Verlag, Berlin; New York, 2004, 339 p.

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

[8]. 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.

[9]. 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.

[10]. 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

[11]. 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

[12]. 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]