Journal of Circuits, Systems, and Computers

FPGA Implementation of Wavelet Neural Network Training with PSO/iPSO

Suhap Sahin*
Computer Engineering, Kocaeli University,
Izmit, Kocaeli, Turkey†
† *suhapsahin@kocaeli.edu.tr*

Mehmet Ali Cavuslu§
Computer Engineering, Kocaeli University,
Izmit, Kocaeli,Turkey
§ *alicavuslu@gmail.com*

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

In this study, FPGA-based hardware implementation of the Wavelet Neural training using Particle Swarm Optimization and Improved Swarm Optimization algorithms is presented. The WNN architecture and wavelet activation function approach that is proper for the hardware implementation is suggested in the study. Using the suggested architecture and training algorithms, test operations are implemented on 2 different dynamic system recognition problems. The test results obtained it is observed that WNN architecture generalizes well and the activation function suggested has approximately the same success rate with the wavelet function defined in the literature. In the FPGA-based implementation, IEEE 754 floating-point numbers format is used. Experimental tests are done on Xilinx Artix 7 xc7a100t-1csg324 using ISE Webpack 14.7 program.

Keywords: WNN; FPGA; PSO; iPSO; Dynamic System.

1. Introduction

Using Field Programmable Gate Arrays (FPGA), which allows design changes to be made easily by means of the programmable intermediate blocks, it is possible to design hardware that is special to the application[1]. At the same time in the designs implemented using FPGAs, by making changes easily, significant gains from time and cost are provided[2]. These properties, in addition to being used in many fields

*Typeset names in 8 pt Roman. Use the footnote to indicate the present or permanent address of the author.
†State completely without abbreviations, the affiliation and mailing address, including country. Typeset in 8 pt Times Italic.

such as digital signal processing[3], communication[4], robotics[5], is also preferred in the Artificial Neural Network applications[6,7,8].

In hardware implementation of FPGA-based Artificial Neural Network, the activation function, the solution approach belonging to this function and the number format that will be used are very important[9]. The most widespread methods used in implementation of activation function in the literature are look-up table and piecewise linear approaches. The number format that will be used affects the precision and the hardware resource consumption. Mostly, floating-point numbers and fixed-point numbers are used in hardware imlementation[1]. FPGA based ANN implementations presented in the literature are summarized according to their activations and number format types below.

Ferreira et. al.[10] implemented the hardware belonging to the feedforward ANN architecture in the 32-bit floating point number format and for the activation function, used hyperbolic tangent function approach that consists of 256 linear parts. Won[11], implemented his 5-6-1 structured forward feeding ANN using 8-bit integer format. For the logarithmic sigmoidal, which is used as the activation function, they constituted a look-up table using single port block RAM of 16 bits wide and 1024 length. Ferrer et. al.[12] implemented the multilayer perceptron hardware using the fixed-point number on FPGA. For the activation function, they used the 2-7 scaled hyperbolic tangent function approach. Chalhoub et. al.[13] implemented the multilayer perceptron hardware using the 18-bit fixed-point number format and the activation function was run using the ROM-based look-up table. Mohammed et. al.[14] implemented the FPGA-based ANN in the integer format using the piecewise linear activation approach. Nedjah et. al.[15], used 32-bit floating point number format and the parabolic approach of logarithmic sigmoidal function. Cavuslu et. al.[16], used the 16-bit floating point number format and the mathematical approach that is similar to the logarithmic sigmoidal function[17]. Stepanova et. al.[18] 2007 implemented Hopfield neural network on FPGA to determine the structure of DNA patterns. In their study, they used 32-bit fixed point number format and tangent hyperbolic function's piecewise linear approach. Lazaro et. al.[19] implemented the general regression neural network structure on FPGA. In the implementation, using 32 and 64-bit floating point number, 12 and 64-bit fixed point number formats comparisons are made. And for cell activations look-up tables are used. Boubaker et. al.[20] presented the implemented of LVQ (Learning Vector Quantization) neural network on FPGA. Savich et. al.[21] implemented ANN's training using back propagation algorithm in terms of hardware using FPGA. In the study, fixed and floating point number formats and logarithmic sigmoidal function's piecewise linear approach is used as the activation function. Cavuslu et. al.[2] implemented ANN training on FPGA in floating point number format with the back-propagation algorithm. The mathematical approach of the logarithmic sigmoidal function[17] is used as the activation function. Farmahini-Farahani et. al.[22] implemented ANN training based on FPGA using the PSO algorithm. In the application fixed point number

format is used as the number format and the look-up table is used for the tangent hyperbolic activation function. Lin et. al.[23] implemented wavelet neural network training with PSO algorithm on the FPGA. The fixed point number format is chosen as the number format. For the activation function, Taylor series and look-up table are used. Harikumar et. al.[24] implemented wavelet neural network on FPGA in terms of the hardware for determination of epilepsy. For the wavelet function, three different nonlinear wavelet function is implemented using Taylor series and the look-up table. PSO algorithm is used for training.

In this study, implementation and training of WNN structure suggested are implemented on the base of FPGA using Particle Swarm Optimization Algorithms. As the direct implementation of the exponential statement is hard in hardware implementation of activation functions, a new mathematical activation approach is suggested in the study that is convenient for hardware implementation, differentiable and the performances of this approach and the wavelet function is compared in the software environment. Because of the high-sensitive it provides, IEEE 754 32-bit floating point number format is used.

2. Wavelet Neural Networks

In recent years, wavelet functions that have intensive processing support in time and frequency dimensions are preferred as an alternative to sigmoid functions. Wavelet Neural Networks, which uses these functions in its architecture, have high success in the solutions of nonlinear problems [25,33]. In this study, WNN architecture that is constituted of four layers is suggested (Fig. 1). The function of these layers are summarized shortly below.

Layer 1: Each input is put in the wavelet functions it has. These functions are of limited duration and they have an average value of zero. The wavelet function is derived from its mother wavelet[27] (Eq. 1).

$$\psi_{d,t} = (\frac{x-t}{d}) e^{\frac{-\| \frac{x-t}{d} \|^2}{2}} \tag{1}$$

Layer 2: Values passed through the wavelet function are multiplied by each other (Eq. 2). In the Eq. 2 represents the wavelet function index that belongs to the first input that enters to the multiplication and represents the wavelet function index that belongs to the second input.

$$\mu_{ij} = \psi_{1i} \psi_{2j} \tag{2}$$

Layer 3: This layer is named as the rule layer. The results of the multiplication operation obtained in the Layer 2 is multiplies with the results obtained in the rule layer (Eq. 3). In the Eq. 3 , the defined parameters p, q and r are named as rule parameters.

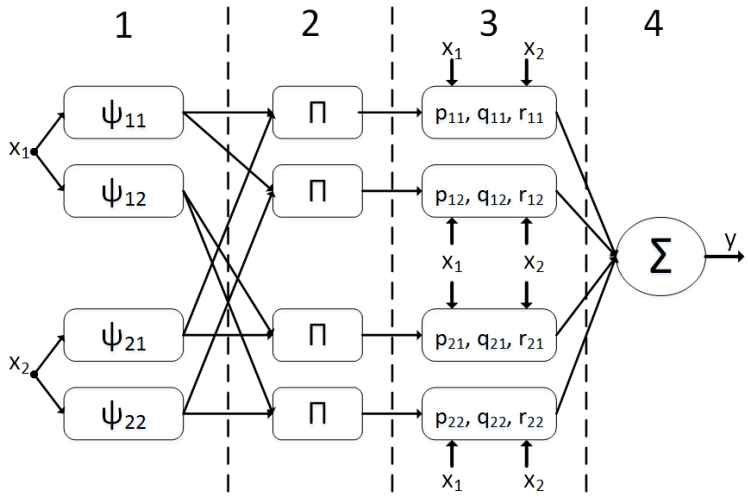


Fig. 1: Suggested WNN block architecture in this study.

$$f_{ij} = \mu_{ij}(p_{ij}x_1 + q_{ij}x_2 + r_{ij}) \tag{3}$$

Layer 4: The output value is obtained by summing up the output values obtained in the rule layer(Eq. 4)

$$y = \sum_i \sum_j f_{ij} \tag{4}$$

3. Particle Swarm Optimization Algorithm

Particle Swarm Optimization Algorithm (PSO), which is developed by the inspiration from the behaviors of birds, is a population-based random search algorithm [28]. Algorithm, in the beginning, is started by the randomly assigned individuals (particle). In each iteration, the speeds and positions of the particles are updated. Each particle in the swarm can be evaluated to be a solution candidate for the problem. particle of the swarm that has N members is defined as a vector in the Eq. 5 [29]. Each particle changes position in the search space depending on its best position till that iteration (local best, p_{lbi} , Eq. 6) and on the position of the particle that has the best position until then for the whole population (global best, p_{gbi}, Eq. 7). The speed of change in the position of the particle is named as the particle speed (v_i, Eq. 8).

$$\overrightarrow{p_i} = [p_{i1}, p_{i2}, \cdots, p_{iN}] \tag{5}$$

$$\overrightarrow{p_{lbi}} = [p_{lbi1}, p_{lbi2}, \cdots, p_{lbiN}] \tag{6}$$

$$\overrightarrow{p_{gbi}} = [p_{gbi1}, p_{gbi2}, \cdots, p_{gbiN}] \tag{7}$$

$$\overrightarrow{v_i} = [v_{i1}, v_{i2}, \cdots, v_{iN}] \tag{8}$$

In order the particle speed to be updated, there are different methods in the literature. For speed updates, Eq. 9 and Eq. 10 (it is suggested in our studies numbered [16,29,30]) are used in this study.

$$v_i(n+1) = \xi[v_i(n) + \alpha_1 r_1(p_{lbi} - p_i(n)) + \alpha_2 r_2(p_{gb} - p_i(n))] \tag{9}$$

$$v_i(n+1) = \xi[v_i(n) + \alpha_1 r_1(p_{lbi} - p_i(n)) + \alpha_2 r_2(p_{gb} - p_i(n))] + \alpha_3 \lambda(n) \tag{10}$$

In Eq. 9 and Eq. 10 α_1 and α_2 are learning constants, r_1 and r_2 are normally distributed random numbers in the range of [0-1) and ξ is the limitation factor. In the Eq. 11, the last term allows a more detailed search in the relevant space by preventing early installation of the particles to the local minimum. After this phase, the method realized using the Eq. 9 will be named as PSO and the method realized using the Eq. 10 will be named as iPSO. After determination of particle speeds, particles are updated according to the Eq. 11.

$$p_i(n+1) = p_i(n) + v_i(n+1) \tag{11}$$

4. WNN Learning Based-PSO/iPSO on FPGA

In this chapter details in relation to the FGPA based implementation of WNN and training of WNN using PSO/iPSO algorithms is presented. Optimization operations of the wavelet function parameters and the rule parameters defined in the rule later are implemented using PSO/iPSO algorithms.

Training of WNN constitutes of 5 stages (Fig 2). In the first stage, each particle's initial position and speeds are determined. In the second stage WNN output values for each particle according to the positions they are in. Using the convenience values calculated depending on the output values local bests are specified in the third stage. In the fourth stage, the global best is specified within the local bests and in the fifth stage, which is the last stage, the update operation of the positions of the particles is handled.

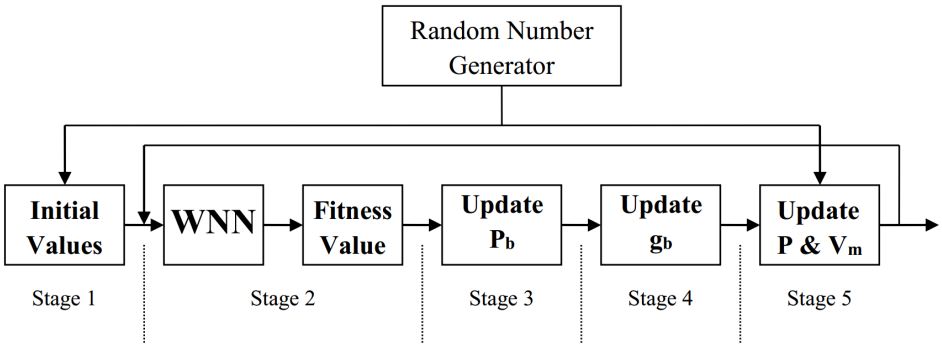


Fig. 2: Order of the implementation with 5 stages.

4.1. *Stage 1: Assigning initial parameter values*

A swarm of length NxD for N particles and D positions that will be optimized for each particle (P_RAM), local best particles (Pbest_RAM), particle velocity (Vm_RAM), D-length global best particle (gb_RAM) and N-length fitness value (En_RAM) memory units are constituted. In the study as IEEE 754 floating point number format will be used each memory unit depth is set to be 32-bit [29], [30].

Starting positions and speeds of the particles are constituted from the numbers produced in the IEEE 754 floating point number format within the range [0-1) (Eq. 12) [31]. In Eq. 12 random number generation is started with the initial (X_0) seed value. a and b are the constant numbers. c is modular operation.

The assignment operations of starting position and speed values of the particles to the memory units are realized depending on the condition of the out_flg signal. In the first condition, data produced by the random number generator P_RAM and Pbest_RAM are written at the same time. The reason of this operation is that starting position for each particle is accepted to be the best positions. With P_RAM and Pbest_RAM memory units becoming full out_flg signal value changes and data produced by the random number generator is written to Vm_RAM (Fig. 3) [29], [30].

$$X_{n+1} = (aX_n + b) \pmod{c} \tag{12}$$

4.2. *Stage 2: Hardware implementation of WNN*

In hardware implementation of WNN, first, it is necessary to designate the number format and activation function. In this study, because of the dynamism in its number representation and its precise operating capacity floating point numbers are preferred. To verify WNN implementation, an arithmetic operation library developed and coded in VHDL language in our previous studies [1],[2], [16], [29], [30] has been used for WNN's arithmetic operations on FPGA.

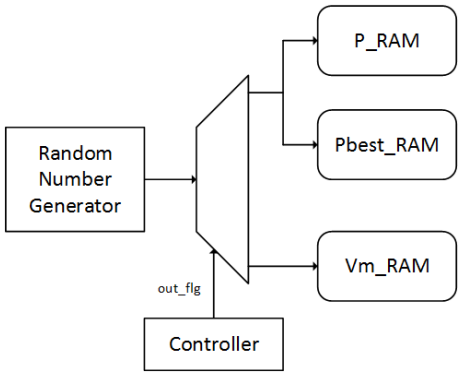


Fig. 3: Block diagram of Stage 1

In the stage of implementation of WNN, the most important process is the wavelet activation function. In the wavelet function defined in the Eq. 1 direct hardware implementation of the exponential statement is really hard. Hence in the study mathematic approach is suggested for the operation (Eq. 13). In Fig. 4, behaviors of the wavelet function defined in the Eq. 1 and the approach suggested in the Eq. 13 within the range of [-20 20] are shown.

$$Q(x) = \frac{-d(x-m)}{d^2(x-m)^2} \tag{13}$$

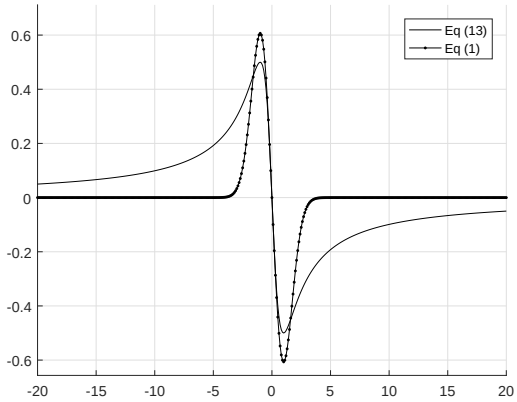


Fig. 4: Comparison of wavelet activation function approaches

4.3. *Stage 3: Determination of the local best particles*

Determination of local best particles operations is made via comparison of convenience functions obtained as a result of the particle's updated position and a particle's best local position. If the updated position is better than the local best position updated position is assigned as the best local position of the particle. In the Eq. 14 the convenience function used in the study is shown. In the equation, j denotes sample index and e_j represents error between j_{th} desired output and the actual output of WNN for j_{th} training input data.

$$E_n = \frac{1}{2} e_j^2 \tag{14}$$

4.4. *Stage 4: Determination of the global best particle*

After the convenience values are calculated for all the particles and local bests are updated in the Pb_RAM memory unit the particle in the best position is determined to be the global best particle. Namely, the position information of the particle whose convenience value is minimum in Pb_RAM is assigned to gb_RAM.

4.5. *Stage 5: Swarm updating*

P_RAM, Pbest_RAM, Vm_RAM ve gb_RAM values got from RAM at the same time got into the update process in two ways. In the first method, Vm matrix update operation is done according to the standard Eq. 9. And in the second method, Vm matrix is updated according to the Eq. 10. In the equations, ξ is 0.76, α_1 and α_2 are 2.1. P_RAM and Vm_RAM are written to the same memory units again after the parameter updates are held. After the update operations of the parameters belonging to all the particles are held the number of iterations are increased by one and step number 2 is turned back. If the number of iterations is more than the maximum generation number the operation terminated.

5. Experimental Results

In the WNN architecture suggested in the study, using the approach suggested in the Eq. 13. FPGA based implementation operation with PSO and iPSO is tested experimentally via the 2-system recognition problem. Experimental tests are done on Xilinx Artix 7 xc7a100t-1csg324 using ISE Webpack 14.7 program. These problems are solved using WNN with appropriate parameters on FPGA. In this way, tuning process for WNN parameters is implied to minimize a cost function based on the error between desired output and the actual output of the WNN. For the tests, PSO and iPSO are run 2000 generation to tune WNN parameters. α_3 value for iPSO algorithm is different for each example since it depends on the training input data set as defined in [16]. Its value is given separately for each example. In the hardware implementation for both of the examples, the same 2-input and 1-output WNN architecture are used.

5.1. *Example 1: System identification 1*

The system to be identified is defined by Eq. 15 and Eq. 16 [32]. This system has been identified with a WNN with inputs of $u(k)$ and $y(k)$. The function defined in the Eq. 13 is used as the activation function in the WNN implementation. PSO and iPSO swarm size (N) is chosen as 100, λ vector is restricted to the interval of $[-2^{-12}\ 2^{-12}]$ and $\alpha_3 = 2^{-12}$.

$$y(k+1) = \frac{y(k)}{1+y^2(k)} + u^3(k) \tag{15}$$

$$u(k) = \cos(\frac{2k\pi}{100}); k = 1, 2, \cdots, 100 \tag{16}$$

In Table 1, the average error, max error, min error and standard deviation values of WNN with PSO and iPSO algorithms, obtained as a result of 100 tests using the activation functions given in the Eq. 1 and Eq. 13. As it can be seen from the table it can be said that a good generalization is done in the structure of WNN. At the same time, the activation approach suggested in the study gives better results in both of the updates. And in Fig. 5, test results and error obtained in the training of WNN, which has the activation function defined, with respectively PSO and iPSO.

Table 1: Test results belonging to the training implemented in the software environment

	PSO		iPSO	
	Eq1	Eq13	Eq3	Eq13
Mean	0.2264	0.0906	0.0822	0.0354
Max	0.3716	0.2547	0.2441	0.2005
Min	0.0561	0.0185	0.0003	0.0001
Std	0.0797	0.0627	0.0792	0.0538

In implementation operations of WNN training activation function defined in the Eq. 13 is used. In Table 2, the average, max, min and standard deviation values of the test results obtained using the parameters optimized as a result of the WNN training in the FPGA environment. As it can be seen from the table it is seen that WNN is done a good generalization as a result of the FPGA-based training realized.

In Table 3, the synthesis results belonging to the hardware implementation of WNN's training using the PSO algorithm are given.

Table 2: Results of the training held in the FPGA environment

	PSO	iPSO
Mean	0.1000	0.0377
Max	0.2325	0.2139
Min	0.0237	0.0009
Std	0.0494	0.0468

Table 3: Synthesis results belonging to the FPGA implementation of the WNN training using the PSO algorithm

Logic Utilization	Used	Available	Utilization
Number of Slice Registers	6354	126800	5%
Number of Slice LUTs	26680	63400	42%
Number of fully used LUT-FF pairs	2036	30998	6%
Number of DSP48E1s	73	240	30%
Number of bonded IOBs	34	210	16%

5.2. *Example 2: System identification 2*

As the second example, the system to be identified is defined by Eq. 17 [33]. This system has been identified with a WNN which has the same structure as the one used for Example 1. $y(k)$ and $y(k-1)$ are the inputs. PSO and iPSO swarm size (N) is chosen as 100, λ vector is restricted to the interval of $[-2^{-12}\ 2^{-12}]$ and $\alpha_3 = 2^{-12}$. In the training phase, as initial condition the inputs are set to $y(0) = 0.1$ and $y(-1) = 0.1$.

$$\begin{aligned} y(k+1) = &- 1.17059y(k-1) + 0.606861y(k) + \\ &0.679190y^2(k)y(k-1) - 0.136235y^4(k)y(k-1) + \\ &0.165646y^3(k)y(k-1) - 0.00711966y^6(k-1) + \\ &0.114719y^5(k)y(k-1) - 0.0314354y(k)y(k-1) + \\ &0.0134682y^3(k) \end{aligned} \tag{17}$$

In Table 4, average error, max error, min error and standard deviation values obtained as a result of 100 tests conducted on the WNN with the PSO and iPSO algorithms in the software environment for the Example 2 using the activation function given in the Eq. 1 and Eq. 13. As it can be seen from the table, it can be said that it has done a good generalization in the WNN structure suggested. At the same time, activation approach suggested in the study gives better results for both of the updates. And in Fig. 6, test results and error values obtained in the training of WNN, which has the activation functions defined in the Eq. 1 and Eq. 13, with

PSO and iPSO, are given.

Table 4: Test results belonging to the training realized in the software environment for the Example 2.

	PSO		iPSO	
	Eq1	Eq13	Eq3	Eq13
Mean	1.3520	1.3160	0.3510	0.2146
Max	2.8964	2.6991	2.1512	1.2980
Min	0.0458	0.0529	0.0088	0.0061
Std	0.9339	0.9349	0.4955	0.3152

In implementation operations of WNN training activation function defined in the Eq. 13 is used. In Table 5 the average, max, min and standard deviation values of the test results obtained using the parameters optimized as a result of the WNN training in the FPGA environment. As it can be seen from the table as a result of the training realized FPGA based, WNN does a good generalization.

Table 5: Results of the training realized in the FPGA environment

	PSO	iPSO
Mean	1.2227	0.2011
Max	2.3474	0.9184
Min	0.1031	0.0085
Std	0.6368	0.2227

In Table 6, synthesis results belonging the hardware implementation of the WNN's training using the iPSO algorithm.

Logic Utilization	Used	Available	Utilization
Number of Slice Registers	6396	126800	5%
Number of Slice LUTs	27789	63400	43%
Number of fully used LUT-FF pairs	2025	30998	6%
Number of DSP48E1s	73	240	30%
Number of bonded IOBs	34	210	16%

Table 6: Synthesis results belonging to the FPGA implementation of the WNN training using the iPSO algorithm

6. Conclusion

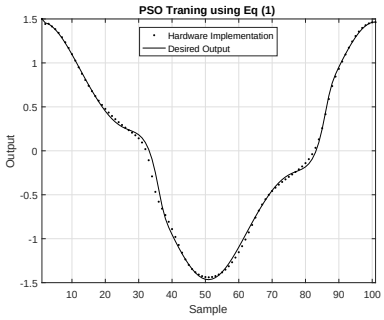
In this study FPGA-based realization of WNN architecture's learning operations of two different dynamic system recognition problems with PSO/iPSO algorithms is presented. Experimental results for both of the examples are given comparatively. Results obtained show that WNN architecture suggested in the study does a good generalization. It also shows that training held using the iPSO is more successful than the training held using the PSO algorithm. Similarly, it is seen that the wavelet activation function suggested within the context of the study has a similar success with the wavelet function presented in the literature. And the synthesis results obtained also show that system suggested is implemented on the FPGA hardware successfully.

References

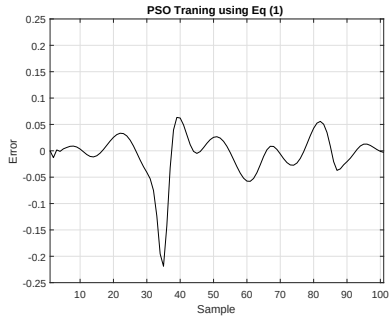
1. S. Sahin, Y. Becerikli, S. Yazici, Neural Network Implementation in Hardware Using FPGAs", *Lecture Notes in Computer Science (LNCS)*, **4234** (2006), pp. 1105–1112
2. M. A. Cavuslu, C. Karakuzu, S. Sahin, M. Yakut, Neural Network Training Based on FPGA with Floating Point Number Format and It's Performance, *Neural Computing & Application*, **20:2** (2011), pp. 195–202
3. P. S. Leea, C. S. Leea, J. H. Leeb, Development of FPGA-based digital signal processing system for radiation spectroscopy, *Radiation Measurements*, **48** (2013), pp. 12-–17
4. S. Dikmese, A. Kavak, K. Kucuk, S. Sahin, A. Tangel, H. Dincer, Digital signal processor against field programmable gate array implementations of space–code correlator beamformer for smart antennas, *IET Microwaves, Antennas & Propagation*, **4** (2010), pp. 593—599,
5. S. Kale, S. S. Shriramwar, FPGA-based Controller for a Mobile Robot, *International Journal of Computer Science and Information Security*, **3:1** (2009)
6. M. Krips, T. Lammert, A. Kummert, FPGA implementation of a neural network for a real-time handtracking system,*in Proceedings of The First IEEE International Workshop on Electronic Design, Test and Applications*, (2002), pp. 313-–317
7. H. Ossoinig, E. Reisinger, C. Steger, R. Weiss, Design and FPGA-Implementation of a Neural Networkwork, *in Proceedings of the 7th International Conference on signal Processing Applications & Technology (Boston, USA, 1996)*, pp. 939–943
8. J. Zhu, G.J. Milne, B. K. Gunther, Towards an FPGA Based Reconfigurable Computing Environment for Neural Network Implementations, *in Proceedings of the Ninth International Conference on Artificial Neural Networks (ICANN'99) IEE Conference Proceedings 470 (Edinburgh, UK ,1999)*, pp. 661–666.
9. M. A. Cavuslu, C. Karakuzu, F. Karakaya, Neural identification of dynamic systems on FPGA with improved PSO learning, *Applied Soft Computing*, **12:9** (2012),pp. 2707–2718
10. P. Ferreira, P. Ribeiro, A. Antunes, F. M. Dias, A high bit resolution FPGA implementation of a FNN with a new algorithm for the activation function, *Neurocomputing*, **71:1-3** (2006), pp. 71-77.
11. E.Won, A hardware implementation of articial neural nertworks using field programmable gate arrays, *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, **581:3** (2007), pp. 816-820.
12. D. Ferrer, R. Gonzalez, R. Fleitas, J. P. Acle, R. Canetti, NeuroFPGA - Implementing

Artificial Neural Networks on Programmable Logic Devices, *Proceedings of Design, Automation and Test in Europe Conference and Exhibition*,**3** (2004), pp. 218–223.
13. N. Chalhoub, F. Muller, M. Auguin, FPGA-based generic neural network architecture, *Industrial Embedded Systems, International Symposium on; Antibes Juan-Les-Pins,France*, 2006, pp. 1-4.
14. E. Z. Mohammed, H. K. Ali, Hardware Implementation of Artificial Neural Network Using Field Programmable Gate Array, *International Journal of Computer Theory and Engineering*,**5:5**, 2013.
15. N. Nedjah, R. M. D. Silva, L. M. M. Mourelle, M. V. C.D. Silva, Dynamic MAC-based architecture of artificial neural networks suitable for hardware implementation on FPGAs, *Neurocomputing*, **72:10-12** (2009), pp. 2171–2179.
16. M. A. Cavuslu, C. Karakuzu, S. Sahin, Neural Networkwork Hardware Implementation Using FPGA, *in ISEECE 2006 3rd International Symposium on Electrical, Electronic and Computer Engineering Symposium Proceedings, Nicosia, TRNC* (2006), pp. 287–290.
17. D. L. Elliot, A Better Activation Function for Artificial Neural Networks, *Technical Research Report T.R. 93-8, Institute for Systems Research, University of Maryland*, (1993).
18. M. Stepanova, F. Lin, V. C. L. Lin, A Hopfield Neural Classifier and Its FPGA Implementation for Identification of Symmetrically Structured DNA Motifs,*Journal of VLSI Signal Processing Systems*, **48:3** (2007), pp. 239–254.
19. J. Lázaro, J. Arias, A. Astarloa, U. Bidarte, A. Zuloaga, Hardware architecture for a general regression neural network coprocessor, *Neurocomputing*, **71:1-3** (2007), pp. 78-87.
20. M. Boubaker, M. Akil, K. B. Khalifa, T. Grandpierre, M. Bedoui, Implementation of an LVQ neural network with a variable size: algorithmic specification, architectural exploration and optimized implementation on FPGA devices, *Neural Computing & Applications*, **19:2** (2009), Number 2, pp. 283-297.
21. A.W. Savich, M. Moussa, S. Areibi, The Impact of Arithmetic Representation on Implementing MLP-BP on FPGAs: A Study, *IEEE Transactions on Neural Networks*, **18:1**(2007), pp. 240–256.
22. A. Farmahini-Farahani, S. M. Fakhraie, S. Safari, Scalable Architecture for on-Chip Neural Network Training using Swarm Intelligence, *Proc. of the Design, Automation and Test in Europe Conf. (DATE'08), Munich, Germany*, 2008, pp. 1340-1345.
23. C. J. Lin, H. M. Tsai, FPGA implementation of a wavelet neural network with particle swarm optimization learning, *Mathematical and Computer Modelling*, **47:9-10** (2008), pp. 982–996.
24. R. Harikumar, M. Balasubramani, S. Saravanan, FPGA Implementation of Wavelet Neural Network for Epilepsy Detection, *International Journal of Engineering and Innovative Technology*, **2:9** (2003).
25. Q. Zhang, A. Benveniste, Wavelet Networks, *IEEE Trans. On Neural Networks*, **3:6** (1992), pp. 889-898.
26. Y. Chen, J. Dong, B. Yang, Y. Zhang, A Local Linear Wavelet Neural Network, *Fifth World Congress on Intelligent Control and Automation,. WCICA*, 2004.
27. R.H. Abiyev, O. Kaynak, Identification and control of dynamic plants using fuzzy wavelet neural Networks, *in: IEEE Multi-conference on Systems and Control,CE press*, 2008.
28. J. Kennedy, R. C. Eberhart, Particle Swarm Optimization, *Proc, IEEE Int. Conf. Neural Network IV*, 1995, pp. 1942-1948.
29. M. A. Cavuslu, C. Karakuzu, S. Sahin, Parçacık Sürü Optimizasyonu Algoritması

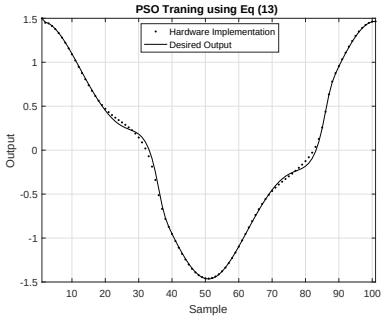
ile Yapay Sinir Ağı Eğitiminin FPGA Üzerinde Donanımsal Gerçeklenmesi, *Politeknik Dergisi*, **13:2** (2010), pp. 83–92.
30. C. Karakuzu, F. Karakaya, M. A. Cavuslu, FPGA Implementation of Neuro-fuzzy System with Improved PSO Learning, *Neural Networks*, **79** (2016), pp. 128–140.
31. M. Brysbaert, Algorithms for randomness in the behavioral sciences: A tutorial, *Behavior Research Methods, Instruments & Computers*, **23:1** (1991), pp. 45-60.
32. K. S. Narendra, K. Parthasarathy, Identification and control of dynamical systems using neural networks, *IEEE Transactions on Neural Networks*, **1:1** (1990), pp. 4-27.
33. S. Chen, S. A. Billings, Neural networks for nonlinear dynamic system modelling and identification, *Int. J. Control*, **56:2** (1992), pp. 319-346.



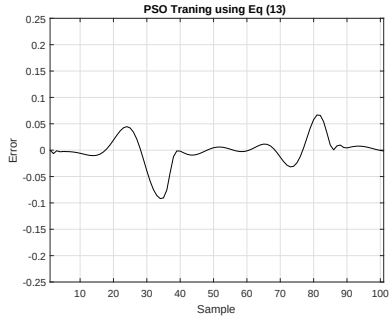
(a) PSO training output using Eq1



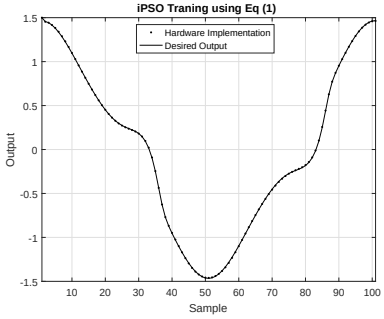
(b) PSO training error using Eq1



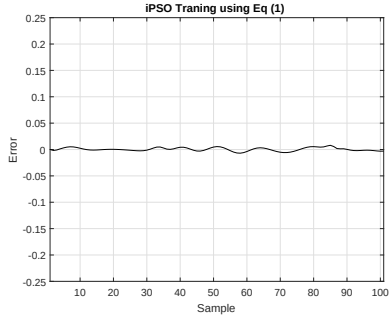
(c) PSO training output using Eq13



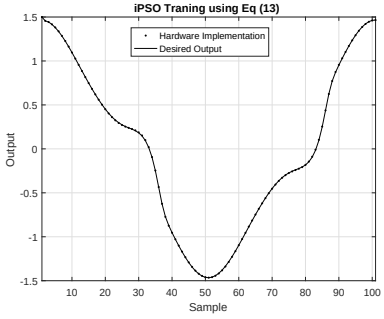
(d) PSO training error using Eq13



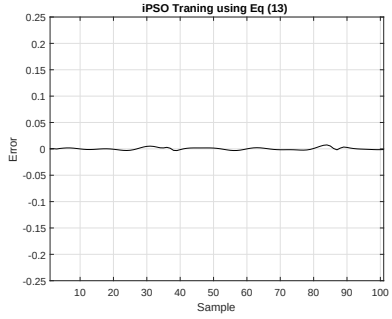
(e) iPSO training output using Eq1



(f) iPSO training error using Eq1

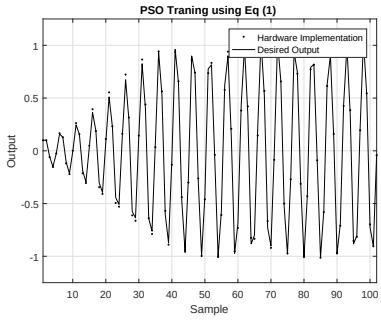


(g) iPSO training using Eq13

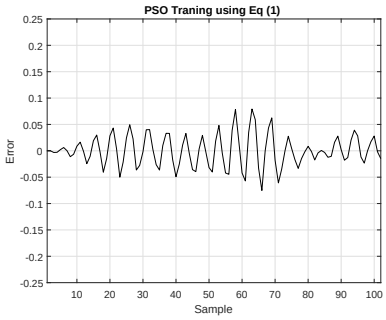


(h) iPSO training error using Eq13

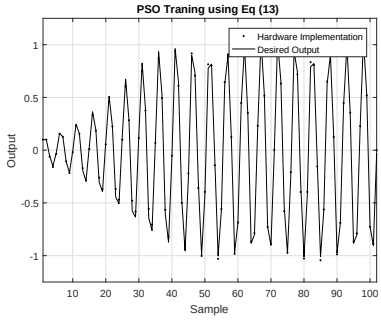
Fig. 5: Output values and errors graphs of WNN, learned with PSO and iPSO in the software environment for the Example 1



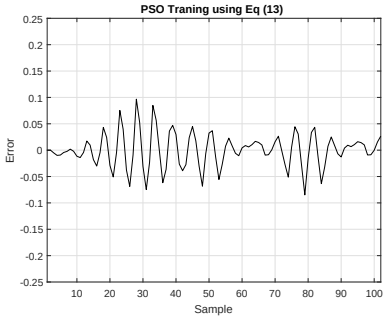
(a) PSO training output using Eq1



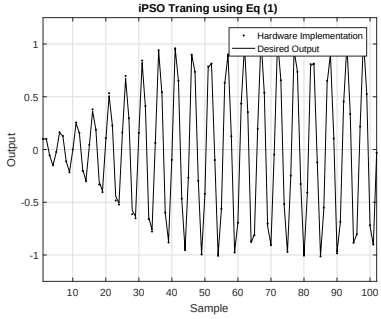
(b) PSO training error using Eq1



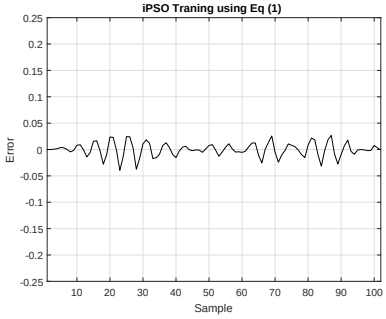
(c) PSO training output using Eq13



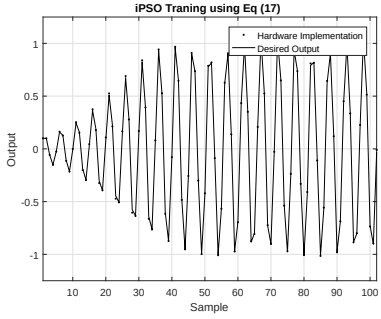
(d) PSO training error using Eq13



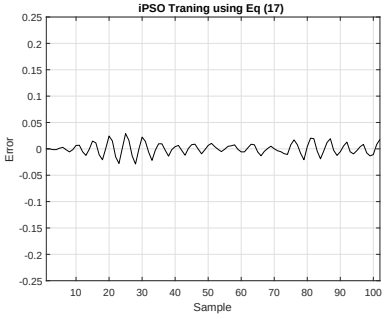
(e) iPSO training output using Eq1



(f) iPSO training error using Eq1



(g) iPSO training using Eq13



(h) iPSO training error using Eq13

Fig. 6: Output values and errors graphs of the WNN trained with PSO and iPSO in the software environment for the Example 2