Neural Computing and Applications (2021) 33:9467–9479
https://doi.org/10.1007/s00521-021-05706-3

ORIGINAL ARTICLE

Hardware implementation of radial-basis neural networks with Gaussian activation functions on FPGA

Volodymyr Shymkovych[1] · Sergii Telenyk[2] · Petro Kravets[1]

Received: 22 July 2019 / Accepted: 5 January 2021 / Published online: 13 March 2021

Abstract
This article introduces a method for realizing the Gaussian activation function of radial-basis (RBF) neural networks with their hardware implementation on field-programmable gaits area (FPGAs). The results of modeling of the Gaussian function on FPGA chips of different families have been presented. RBF neural networks of various topologies have been synthesized and investigated. The hardware component implemented by this algorithm is an RBF neural network with four neurons of the latent layer and one neuron with a sigmoid activation function on an FPGA using 16-bit numbers with a fixed point, which took 1193 logic matrix gate (LUTs—LookUpTable). Each hidden layer neuron of the RBF network is designed on an FPGA as a separate computing unit. The speed as a total delay of the combination scheme of the block RBF network was 101.579 ns. The implementation of the Gaussian activation functions of the hidden layer of the RBF network occupies 106 LUTs, and the speed of the Gaussian activation functions is 29.33 ns. The absolute error is ± 0.005. The Spartan 3 family of chips for modeling has been used to get these results. Modeling on chips of other series has been also introduced in the article. RBF neural networks of various topologies have been synthesized and investigated. Hardware implementation of RBF neural networks with such speed allows them to be used in real-time control systems for high-speed objects.

Keywords Approximation algorithm · RBF neural network · Gaussian activation functions · FPGA

1 Introduction

Neural network control systems represent a new high-tech direction in control theory and relate to a class of nonlinear dynamical systems [1–3]. The realization of parallel processing of input data in neural networks will transform them into high-speed devices for information processing. This property, together with the ability of neural networks to learn and approximate a wide class of continuous functions with given accuracy, makes this technology very attractive for the creation of adaptive control devices in automatic systems [4–7].

Neural networks can be used, Fig. 1, in robotics, drone control, vehicle control, pattern recognition, analysis and decision-making in the Internet of things systems, spacecraft control, military equipment and other different applications in modern technology [8–12]. In these systems, neural networks can be used to identify objects, predict the state of objects, recognition, clustering, classification, analysis of large amounts of data coming at high speed from a large number of devices and sensors and so on [11–17].

Neural networks can be used to implement various components of control systems. On the basis of neural networks, we can construct adaptive, nominal, inverse-

✉ Sergii Telenyk
stelenyk@pk.edu.pl

Volodymyr Shymkovych
v.shymkovych@kpi.ua

Petro Kravets
p.kravets@kpi.ua

[1] Department of Automation and Control in Technical Systems, National Technical University of Ukraine «Igor Sikorsky Kyiv Polytechnic Institute», 37, Prosp. Peremohy, Kiev 03056, Ukraine

[2] Department of Theoretical Electrical Engineering and Computer Science, Cracow University of Technology, Warszawska 24, 31-155, Cracow, Poland

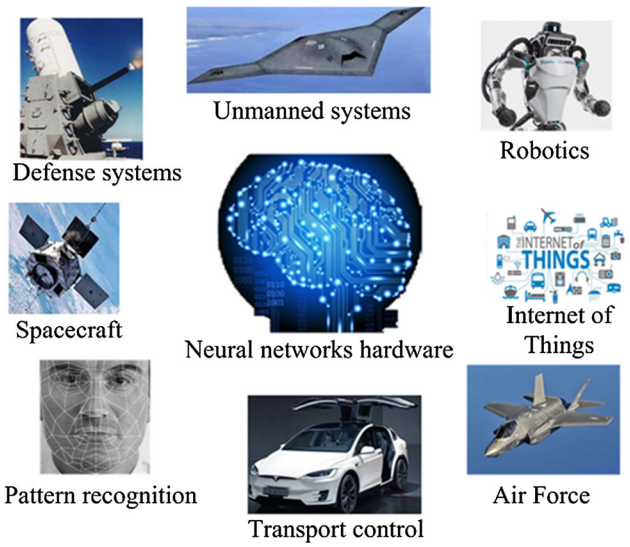

Fig. 1 Applications of neural networks hardware systems

dynamic models of control objects. Such components can perform a wide range of functions.

First, they can be used for monitoring, evaluating the control object parameters and disturbances and deviations in the control systems. Second, with their help, the optimal control function can be implemented. Third, for the purpose of predicting the state of the object, its identification can be carried out. Besides, other operations can be performed on the base of these models [12–19].

Nowadays, the main implementation method of neural network control systems is to implement the special software, using universal computers or specialized controllers built on their basis. Such implementation significantly narrows the circle of the practical application of neural network control systems, due to the considerable cost of such regulators, and makes them virtually inexpedient and inaccessible for use in control systems. Computer-based neural network regulators have limited performance and require a certain amount of time to research. Recurrence and sequence of actions of the neural network learning procedure during the implementation into the whole set of parameters do not allow to solve the problem completely in the pace with the dynamics of the control object. Although increased performance of systems with a consistent Von Neumann architecture allows the implementation of large ANNs, a large number of ANN applications require high speed, compactness, low cost and energy consumption. The only alternative to this is the introduction of parallel learning and operation of internal elements of neural networks. Such capabilities emerge with the development of digital technologies such as field-programmable gate array (FPGAs), digital signal processors (DSPs) and application-specific integrated circuit (ASICs) [20–22]. In addition, the creation of hardware description languages (HDL) has allowed engineers to develop and encode the ANN model once and use it on any hardware platform.

ANNs can be implemented on any of these devices using VHDL [22–25]. VHDL is a standard independent language, also portable and reusable. After writing the VHDL code, it can be used either to implement the circuit in the FPGA, or can be submitted for the manufacture of the ASIC chip [26]. VHDL is based on the principles of software design, but contrary to conventional computer programs that are consistent, its description is essentially parallel.

However, nonlinear multilayer neural networks have certain disadvantages. Due to the nonlinear dependence of the network outputs on the given parameters, the problem of approximation becomes one of the extreme problems. Along with the possible global minimum learning criterion, the standard learning algorithm can stop on one from many local minimum. The interconnection of the adjustment of the weight coefficients of all major elements of the network greatly increases the time of learning, which is critical for real-time systems. The neural networks with radial activation functions discussed below have no such deficiencies.

RBF networks are also universal approximators and can be used for approximation of any continuous function with given accuracy [27–35]. RBF networks in their structure belong to two-layer networks that use a hidden layer with fixed nonlinear transformations of the input vector with constant weight coefficients. This layer performs a function of the static display of input variables $\mathbf{r} \in R^n$ in new variables $\mathbf{q}_1^{(l)} = \mathrm{col}\left(q_1^{(l)}, \ldots, q_m^{(l)}\right)$. The second linear output layer is the one that delivers these variables with adjustable weights $\mathbf{w}_i = \mathrm{col}\left(w_{i,1}, w_{i,2}, \ldots, w_{i,m}\right)$ so that, for example, the ith scalar output of the RBF network is written as:

$$q_i^{(2)} = q_i = F(\mathbf{r}) = \sum_{j=1}^{m} w_{i,j} f_i(\mathbf{r}) + w_0 \cdot 1, \quad i = 1, 2, \ldots, K \tag{1}$$

Structure of the RBF network with the input vector $\mathbf{r} \in R^n$ and the output vector $\mathbf{q}^{(2)} = \mathbf{q} \in R^n$ is shown in Fig. 2. Activation functions $f_j(r) : R^n \to R^1$ are selected from the class of radial-symmetric functions:

$$f_i(\mathbf{r}) = f\left(\left\|\mathbf{r} - \mathbf{a}_j^*\right\|; h\right), \ \mathbf{a}_j^* \in R^n. \tag{2}$$

They have an extremum when changing the inputs $\mathbf{r}$ only near the set values of the weight coefficients of the hidden layer neurons:

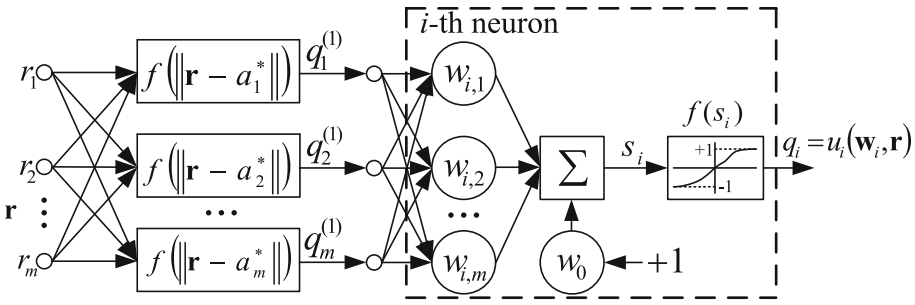

Fig. 2 Block diagram of the RBF network

$$\mathbf{w}_j^{(1)} = \mathbf{a}_j^*,\ j = 1, 2, \ldots, m;$$
$$\mathbf{w}_j^{(1)} = \mathrm{col}(w_{j,1}^{(1)}, w_{j,2}^{(1)}, \ldots, w_{j,n}^{(1)}).$$

Weight coefficients of the second layer are $w_{i,j} = w_{i,j}^{(2)}$, where parameter h is the position of the so-called 'centers of susceptibility fields.' They depend on the set of the weight coefficients values in the first layer $\mathbf{w}_j^{(1)} = \mathbf{a}_j^*$. It is here that we can form functions $f_j(\cdot)$, selecting their number m, to provide the RBF network with the required properties.

As functions $f(z)$ one of the functions is selected:

$$\exp(-z^2);\ \exp(-z);\ 1/(1+z)^2;$$
$$1/1+z^2;\ z^\alpha \log z;\ z^2 + \alpha z + \beta.$$

In practice, the first of these functions is widely used—the Gaussian functions:

$$f_j(\mathbf{r}) = \exp\left(\sum_{k=1}^{n} 0.5\sigma_{j,k}^{-2}\left\|\mathbf{r} - \mathbf{a}_j^*\right\|^2\right), \tag{3}$$

where $\mathbf{a}_j^* = \mathrm{col}\left(a_{j,1}^*, a_{j,2}^*, \ldots, a_{j,n}^*\right)$ is a numeric vector and $\sigma_{j,k}$—deviation of the Gaussian function. The specificity of the RBF network is that they learn in operating mode only to determine the weight coefficients of the output layer. The approximation error is calculated directly on the output of the network as in the multilayer neural networks, by parameters of the output layer. Therefore, learning only this layer, we eliminate the problem of a functional criterion (usually quadratic) global minimum search. In turn, it helps to accelerate the network learning process.

The paper [36] proposes a Gauss–Sigmoid neural network for learning with continuous input signals. The input signals are put into an RBF network, and then, the outputs of the RBF network are put into a sigmoid-based multilayered neural network. After learning based on backpropagation, the localized signals by the RBF network are integrated and an appropriate space for the given learning is reconstructed in the hidden layer of the sigmoid-based neural network. Once the hidden space is constructed, both the advantage of the local learning and the global generalization ability can exist together. Therefore, for further implementation, we will use the sigmoidal activation function in the initial layer.

The problem with RBF networks is to select the number of radial-basis functions required for approximation. This becomes a critical factor in the use of RBF networks in identification and control tasks where information required to determine the dimension of RBF networks is generally absent. The number of necessary radial-basic functions increases exponentially with increasing the number of input variables, with increasing n. This allows us to make the next conclusion that RBF networks can be applied only in those problems where the dimension of the input set r is limited and known (there is no such restriction in multilayer neural networks).

The rest of the paper is structured as follows, Sect. 2 reviews existing hardware implementation artificial neuron RBF network on FPGA in more detail, formulation of the problem and aim of the work, Sect. 3 provides a description of the proposed method and algorithm for hardware implementation of an artificial neuron of the RBF network on the FPGA, Sect. 4 presents the results and discussion, and finally Sect. 5 summaries the conclusions and discusses future work.

2 Formulation of the problem

As noted above, the RBF networks are universal approximators and, with slight restrictions, can be applied to approximate any continuous function. RBF networks in their structure refer to two-layer networks, which use a hidden layer with a fixed nonlinear transformation of the input vector with constant weight coefficients. This layer carries a static display of input variables $\mathbf{r} \in R^n$ in new variables $\mathbf{q}_1^{(l)} = \mathrm{col}(q_1^{(l)}, \ldots, q_m^{(l)})$.

One of the problems of hardware implementation of neural network control systems based on RBF networks is the implementation of an artificial neuron and its nonlinear activation functions using an FPGA. Existing approaches to the digital implementation of nonlinear functions use different methods of approximation, such as tabular method, Taylor series, piecewise linear approximation and

BRAM method [37–52]. Taylor series require many multiplications and therefore unacceptable for implementation in FPGA, since the multiplication block occupies a large amount of resources [38, 39, 44, 49]. So the best method for the implementation of activation functions of a nonlinear type is a piecewise linear approximation.

This paper [46] discusses the hardware implementation of the RBF type neural network, the architecture and parameters and the functional modules of the hardware-implemented neuroprocessor. An RBF type hardware-implemented network with three inputs has been developed. The neural network was implemented on an XSA-3S1000 development board. The system was designed in VHDL language, with the Xilinx WebPack program. In the paper [47], hardware-implemented artificial neural network used for trajectory control of a mobile robot is presented. As controller RBF type hardware-implemented artificial neural network in [46] has been used.

This study [48] presents a digital hardware implementation of an RBF network with learning mechanism. A high accuracy method, which combines Taylor series expansion technique and LUT technique to compute the Gaussian function in RBF network, is proposed. Computational ability for a complicated RBF network is presented by using the digital hardware implementation. For example, an RBF network with three inputs, five neurons, one output and one learning mechanism structure that the computational time only needs 500 ns in performing the function of the forward computation. Implementation is based on the Altera FPGA kit (DE2-Cyclone IV—EP4CE115F29C7).

This paper [49] presents the hardware implementation of the floating-point processor to develop the RBF function neural network for the general purpose of pattern recognition and nonlinear control. The floating-point processor is designed on a FPGA chip to execute nonlinear functions required in the parallel calculation of the back-propagation algorithm. The paper uses the 50th order in the Taylor series to approximate a nonlinear function. This approximation requires a large amount of computing resources.

In paper [50], the implementation of RBF network, which is used for image recognition, has been presented. Several implementations of the neural network on the Xilinx FPGA Virtex 4 with indicators for estimating the speed and occupied resource of the FPGA chip have been presented.

In [51], the hardware implementation on FPGA RBF neural networks using fixed-point numbers has been described. Speed RBF neural networks with hardware implementation on the FPGA and used chip resources for different formats of fixed-point numbers have been presented. The hardware implementation of the RBF neural network is made using a chip Virtex-6 xc6vcx240t-1ff1156.

In [52], the machine vision system in real time has been described, which allows you to locate faces in video sequences and verify their identity. The machine vision system is based on image processing methods and RBF neural networks. The reliability of this system has been quantified in eight video sequences. The paper describes three hardware implementations of the machine vision system in real-time embedded systems based, respectively, on the FPGA, computers with zero instruction set (ZISC) and digital signal processor (DSP) TMS320C62. The success rate of face tracking and identity verification is 92% (FPGA), 85% (ZISC) and 98.2% (DSP), respectively. For the three embedded systems, the processing speed for 288×352 images is 14 images/s, 25 images/s and 4.8 images/s, respectively.

The aim of this work is to increase the speed of RBF neural networks by developing a method of parallel hardware implementation on the FPGA, with minimal use of computing resources.

3 Method and algorithm hardware implementation artificial neuron RBF network on FPGA

In the paper [53], a general method for the implementation of nonlinear activation functions of artificial neurons has been developed and described. The algorithm has been developed for the implementation of an artificial neuron with the sigmoid activation function, and the learning of hardware neurons and neural networks implemented on the FPGA.

For the implementation of the Gaussian activation functions of RBF neural networks, the method [30, 43] has been modified. The modified developed method is as follows.

At the first stage, the study of the activation function is carried out on symmetry with respect to the axes. If the condition is fulfilled:

$$1 - f(x) = f(-x), \tag{4}$$

we can consider $f(x)$ only for positive arguments. For its negative values of the argument of the activation function, its value can be found by the formula (4). That significantly accelerates the calculation of the value of the function and reduces the occupied resource of the FPGA.

To approximate the activation function, we set the piecewise linear function at each of the intervals $(-\infty; x_1), (x_1; x_2); \ldots (x_n; +\infty)$ with the separate formula. The general description of the piecewise linear approximation of the activation function is:

$$f(x) = \begin{cases} k_0 x + b_0, \ x < x_1 \\ k_1 x + b_1, \ x_1 < x < x_2 \\ \dots \\ k_n x + b_n, \ x_n < x \end{cases}. \quad (5)$$

For each linear function, the coefficients k and b are introduced, which are written in a chip memory (RAM). Then, the calculation of value $f(x)$ is performed according to the formula (5). For the input value x of the activation function, the coefficients k and b of the formula are selected from the memory (5). If the argument of function $f(x)$ has a negative value, the final value of the function $f(x)$ is calculated by the formula (4).

One of the main problems with the hardware implementation of the RBF network is the implementation of hidden layers and nonlinear activation functions $f_i(r)$, in particular the implementation of the Gaussian function.

The Gaussian function, the graphical representation of which is shown in Fig. 3 is described by the formula:

$$f(x) = \alpha e^{-\frac{(x-\beta)^2}{2\sigma^2}}, \quad (6)$$

where α—the height of the function, β—the shift of the function along x-axis and σ is the decay rate of the function when the vector is removed from the center.

Let us consider the features of the Gaussian functions graphically represented by the curves a, b, c in Fig. 3. So, the Gaussian function, which corresponds to curve a, has a low decay rate and a biasing along the x-axis. In this case, the expression (6) becomes the following:

$$f(x) = 0.6e^{-\frac{(x+0.5)^2}{3}} \quad (7)$$

In turn, the curve b has a more elongated shape and is not shifted relative to the x-axis. The expression (6) of the Gaussian function, to which it corresponds, becomes the following:

$$f(x) = e^{-\frac{x^2}{2}} \quad (8)$$

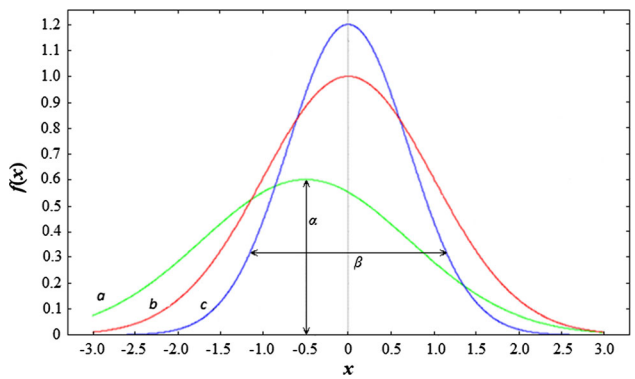

Fig. 3 Gaussian functions

Finally, the curve c has a more height and is not shifted relative to the x-axis. The expression (6) of the Gaussian function, to which it corresponds, becomes the following:

$$f(x) = 1.2e^{-\frac{x^2}{1}} \quad (9)$$

The coefficients of the formula (6), which give Gaussian function the desired properties, for example, one of the combinations of such properties is illustrated by the curves a, b, c in Fig. 3, are determined by learning the neural network.

For hardware implementation of ANN, for inputs, weights and activation functions it is necessary to take into account the format of numbers that will be used (floating point or fixed point) and the number of bits (accuracy), which can range from 16 bits, 32 bits or 64 bits, number format floating-point or fixed-point format. Increasing the accuracy, the number of bits per number significantly increases the resources used.

The format of floating-point numbers requires significant hardware resources but has high accuracy, while the format of fixed-point numbers requires minimal resources but has poorer accuracy. The minimum allowable accuracy and the minimum allowable range should be determined to optimize the hardware resource used and the computational speed without affecting the ANN performance. The accuracy of numbers with a fixed point does not depend on the number, but depends on the position of the point, what the number of numbers will be allocated to the whole part and how many to the fractional. In [40], it was shown that the need for hardware resources of networks implemented using fixed-point numbers is significantly less than when using floating-point numbers with similar accuracy and range of values. Also, fixed-point calculations provide better convergence for smaller clock cycles. For normalized input and output in the range [0, 1] for the implementation of the nonlinear activation function, a 16-bit representation of numbers with a fixed point is acceptable accuracy.

We propose the following step-by-step algorithm for hardware implementation of FPGA artificial neuron RBF network with hidden layer and Gaussian function activations. The steps of this algorithm for implementing the artificial neuron of the network are described in detail below.

Step 1. Submit the corresponding value of the input vector on the input of artificial neurons and set the variable x. Numbers with a fixed point are used for implementation. Numbers are 16-bit, 9 bits are used for the decimal part, and 7 are needed for the fractional part.

Step 2. Calculate the module from the Gaussian function argument and define the variable x':

$$x' = |x| \tag{10}$$

Step 3. Split the Gaussian activation function into linear pieces and determine the coefficients k and b of the linear equations. Piecewise linear approximation consists in finding the coefficients k and b of the equation such that all experimental points lie closest to the approximating straight line. The Gaussian function was divided into five linear segments, and the coefficients of these segments were calculated (11). The results of the piecewise linear approximation of the Gaussian function are shown in Fig. 4.

Determine the coefficients of the linear equations:

$$k, b = \begin{cases} [-0.275, 1], & \text{if } 0 \le x' < 0.6 \\ [-0.58, 1.183], & \text{if } 0.6 \le x' < 1.2 \\ [-0.48, 1.063], & \text{if } 1.2 \le x' < 1.8 \\ [-0.238, 0.628], & \text{if } 1.8 \le x' < 2.4 \\ [-0.093, 0.28], & \text{if } x' \ge 2.4 \end{cases}. \tag{11}$$

Step 4. Determine the variable f, calculate the values by the formula:

$$f(x') = k \cdot x' + b \tag{12}$$

Step 5. Submit the resulting value of the function $f(x')$ calculation on the neuron with a sigmoid activation function.

4 Experimental results

The hardware implementation on FPGA of the Gaussian activation functions of the hidden layer of the RBF network occupies 106 logic matrices (LUTs—Look Up Table), and the speed of the Gaussian activation functions is 29.33 ns. The detailed hardware resource results for Gaussian functions implemented on FPGA Xilinx Spartan 3—xc3s5000-4fg1156 chip shown in Table 1. Chips of this family is characterized by increased performance, a large volume of standard logical resources and specialized embedded hardware units at a relatively low cost. They are focused primarily on the use in mass-produced equipment. Chips of different families have different architecture and performance. In particular, the architecture of LUTs differs. In the chips of the Spartan 3 and Virtex-4 family, the logic cells are based on four input LUTs. Starting with the Virtex-5 and Spartan-6 series, the new chips are built on more complex six input LUTs. Detailed hardware specifications will be presented for the Spartan-3 family of chips. Also, the main characteristics, such as speed and resource used, for chips of other families will be presented.

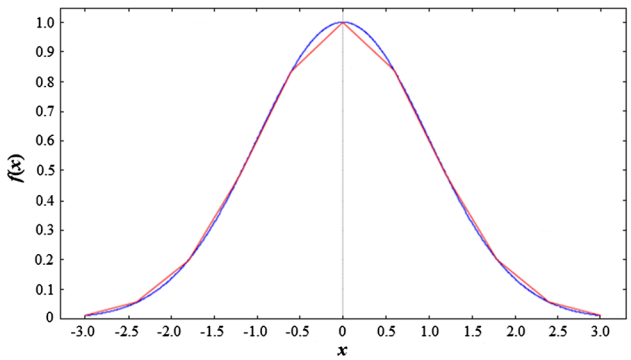

Fig. 4 Piecewise linear approximation of the Gaussian function

Figure 5 shows computational modules of the Gaussian function. The computational modules consist of a BRAM memory unit, which stores the coefficients of linear equations, a unit of calculation of the input value module (Abs), multiplication (MPL0-MPL4), addition units (ADD0-ADD4) and multiplexer (Mux).

With the help of the developed method, the Gaussian functions were modeled in FPGAs of different series. The results are shown in Table 2.

The general view of the RBF network with four hidden neurons and one neuron with a sigmoid activation function is shown in Fig. 5. Figure 6 *neur4sigm* shows a neuron with a sigmoidal activation function and with four input values × 1, × 2, × 3, × 4. For training of a neural network in hardware block of a neuron special ports are provided. Special training ports are shown in Fig. 6 hardware block of the neuron neur4sigm: *syngetw*—port for reading the value of the weight of the synapse of the neuron; *synsetw*—input port for recording the value of the weight of the synapse of the neuron; *synaddr*—input port which indicates the synapse address of the neuron from which the value is currently read or written; *synwren*—port of permission of record of value of weight in a neuron, at input value 0 record of weight is forbidden, at 1 record of weight to-allowed.

Let us take a closer look at the hardware component implemented by the developed method on the FPGA, Fig. 6. This is an RBF neural network with four intrinsic neurons with Gaussian activation functions and one neuron with a sigmoid activation function. The neuron of the output layer has four inputs, respectively, and four weights. Weighting factors, as well as coefficients of linear approximation of activation functions, are written in the BRAM memory of each neuron.

The detailed results of modeling the 4–1 network using 16-bit numbers with a fixed point on FPGA Xilinx Spartan 3—xc3s5000-4fg1156 chip are presented in Table 3. Each hidden layer neuron of the RBF network is designed on an FPGA as a separate computing unit. The speed as a total delay of the combination scheme of the block RBF network is 101.579 ns. The absolute error is ± 0.005. From these results, it can be seen that the number of LUTs increased by 63% when compared with one neuron of the sigmoid

Table 1 Hardware resource results for Gaussian functions implemented on FPGA

Logic utilization	Used		Available	Utilization (%)
Number of slices	69	Out of	33,280	1
Number of four input LUTs	106	Out of	66,560	1
Number of bonded IOBs	32	Out of	784	4
BRAM	1	Out of	104	1
Number of MULT18X18s	1	Out of	104	1
Number of GCLKs	1	Out of	8	12

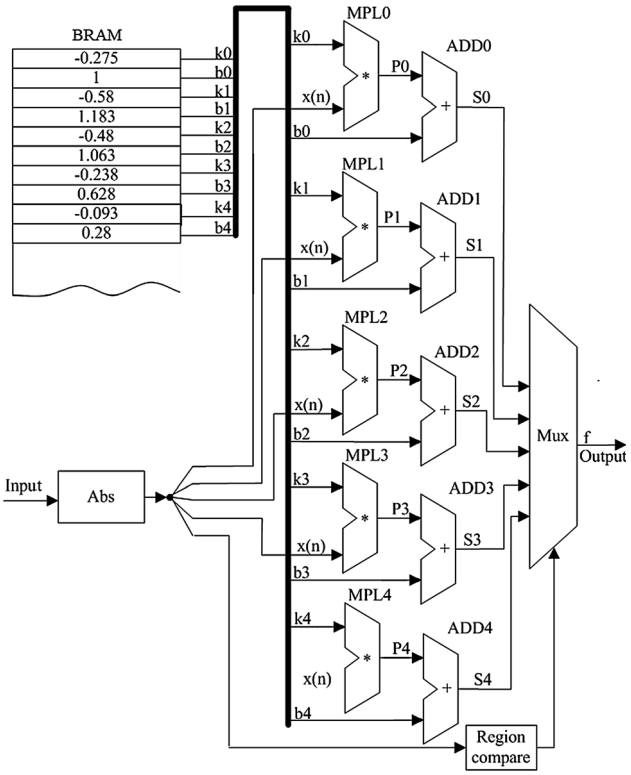

Fig. 5 Computational modules of the Gaussian function

Table 2 Results of Gaussian function modeling in FPGA of different series

FPGA		Speed (ns)	Resource of FPGA (LUT)
Series	Speed		
Xilinx Spartan 3	− 5	29.335	106
	− 4	33.968	
Xilinx Spartan 6	− 2	2.05	40
	− 3	1.86	
Virtex 5	− 2	14.968	77
	− 1	54.823	
Virtex 4	− 12	17.365	104
Xilinx Zinq	− 1	0.45	39
	− 2	0.44	
	− 3	0.42	
Virtex 7	− 1	0.45	39
	− 2	0.44	
	− 3	0.42	
Virtex 6	− 2	0.52	39
	− 1	0.56	
Kintex 7	− 1	0.45	39
	− 2	0.44	
	− 3	0.42	
Artix 7	− 1	0.52	39
	− 2	0.50	
	− 3	0.48	

activation function, while the time increased by only 50%, due to the fact that the hidden layer neurons can be calculated in parallel.

Synthesis and simulation of a neural network with four neurons of a hidden layer and one neuron with a sigmoidal activation function are performed on the software of the Xilinx ISE Design Suite 13.2 and ISE Simulator (ISIM) using the Spartan 3 family of chips. The Spartan 3 family chip has high performance, which involves the implementation of projects with system frequencies up to 326 MHz.

One of the main features of neural networks is parallel signal processing. Multilayer neural networks are a homogeneous computing environment. In the terminology of neuroinformatics, these are universal parallel computing structures designed to solve various classes of problems. Consider the basic principles of parallel signal processing on the FPGA and the concept of implementing neural networks on the FPGA.

In this regard, the method of conveyor processing is of interest, because when used, the processor can simultaneously process several instructions or data packets. To do this, the whole procedure of processing one command is divided into several stages, at the end of each of which the result is stored in temporary registers, through which the individual stages pass and switch between them.

Moreover, the known method of internal cycle's deployment in some cases can lead to a significant increase in productivity. The essence of the method is to present cyclic processing algorithms with a finite number of

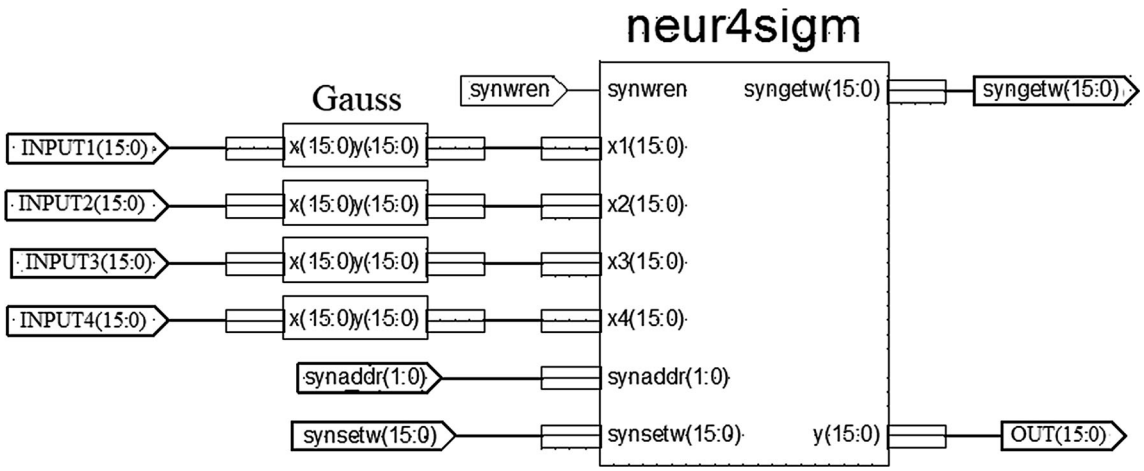

Fig. 6 RBF network hardware implemented on the FPGA

Table 3 Hardware resource results for 4–1 RBF network

Logic utilization	Used		Available	Utilization (%)
Number of slices	789	Out of	33,280	2
Number of four input LUTs	1193	Out of	66,560	2
Number of bonded IOBs	116	Out of	784	15
BRAM	1	Out of	104	1
Number of MULT18X18s	14	Out of	104	13
Number of GCLKs	1	Out of	8	12

iterations in the form of a long combinational chain, which is processed in one cycle.

In real multiprocessor systems, the processing of input instructions is done by more than one processor, which allows you to implement completely parallel processing.

In each neuron with a sigmoidal activation function, the process of multiplying the input value by the weight of the synapse is separate parallel processors. Calculating the multiplication of the input value by the weight of the synapse, finding the sum of these values and calculating the activation function work parallel on the principle of the conveyor.

In VHDL, each parallel process is represented by a separate operator *process*(*in*), when the input value *in* changes, the calculation begins in the block *process*(*in*).

VHDL code is synthesized by an automated design system Xilinx ISE Design Suite into an electrical digital circuit, which is then configured in the FPGA.

In the hardware implementation of RBF networks on the FPGA, each layer of the network works in parallel with the other, here the principle of the conveyor is used. Neurons in each layer also work in parallel on the principle of multiprocessor data processing. That is, each artificial neuron of the network is a separate processor and the processing of information in each neuron takes place simultaneously.

The artificial neurons have been configured according by the developed method in the FPGA of different series. Table 4 shows the results of this configuration.

The developed method of hardware implementation of the neuron and neural networks allows regulating the accuracy of calculations for different tasks, determining the occupied resource of the FPGA in the number of gate valves of the logical matrix LUTs.

One of the main features of the neural network is the parallel processing of signals. Multilayer neural networks represent a homogeneous computing environment. In terms of neuroinformatics, a neural network is a universal parallel computing structure, designed to solve a variety of task classes.

The hardware implementation of artificial neural networks into the FPGA means that each layer of the network can work in parallel with another. This allows using the principle of the conveyor during calculations. Neurons in each layer also work in parallel with the principle of multiprocessor data processing. That is, each artificial neural network is a separate processor and the processing of information in each neuron passes simultaneously.

Each neuron is represented as a separate block, as shown in Fig. 7, which in turn consists of several parallel processes, and the neural network is a multiprocessor system. The programming language allows you to explicitly

Table 4 Results of simulation of artificial neuron in FPGA of different series

FPGA		Speed (ns)	Resource of FPGA (LUT)
Series	Speed		
Xilinx Spartan 3	− 5	101.579	1193
	− 4	117.805	
Xilinx Spartan 6	− 2	6.916	423
	− 3	6.032	
Xilinx Zinq	− 3	0.96	420
	− 2	1.102	
	− 1	1.316	
Virtex 7	− 1	1.316	420
	− 2	1.102	
	− 3	0.96	
Virtex 6	− 2	1.209	420
	− 1	1.376	
Virtex 5	− 2	49.039	991
	− 1	57.75	
Virtex 4	− 10	70.517	1181
	− 11	61.198	
	− 12	53.903	
Kintex 7	− 3	0.96	420
	− 2	1.102	
	− 1	1.316	
Artix 7	− 1	1.588	420
	− 2	1.345	
	− 3	1.158	

specify the signals that start the process. To start the process of calculating a neuron, the input signal of this neuron is used.

Testing of hardware-implemented RBF networks was as follows:

- simulation of hardware-implemented RBF network in specialized software ISE Simulator, obtaining the result of RBF-network output;
- creation of a test model of the RBF network, Fig. 8, in the MATLAB software package with similar values of inputs and weights;
- comparison of the results of the hardware-implemented RBF network and its model in MATLAB, Table 5.

Table 5 shows the results of the research of the RBF neural networks implemented on FPGA Xilinx Spartan 3—xc3s5000-4fg1156 chip obtained by means of the developed method. The table presents the neural networks of various architectures, the resources used by them and their speed. The topology of neural networks is represented by two numbers, where the first number is the number of Gaussian functions in the input layer, and the second—number of neurons with sigmoid activation functions in the initial layer. To obtain the error of the hardware-implemented RBF network in the FPGA, a test model was collected in the software package MATLAB, Fig. 8. Testing of the neural network implemented on the FPGA was performed in specialized software ISE simulator (ISIM). System-level testing was performed with ISIM logic simulator, and such test programs also written in HDL languages. Test bench programs include simulated input signal waveforms and monitor which observe and verify the outputs of the hardware component under test. ISIM was used to perform the following types of simulations: logical verification, to ensure the hardware component produces expected results; behavioral verification, to verify logical and timing issues; post-place and route simulation, to verify behavior after placement of the hardware component within the reconfigurable logic of the FPGA.

Table 5 presents the output values of the RBF network implemented on the FPGA, and in MATLAB, with the same RBF network inputs and parameters, their difference in the column 'Error.' When configuring RBF neural networks, the Spartan 3 family chip is used. The values obtained in LUTS may vary slightly when the constants are changed, which define the synaptic weights of the neurons.

Figures 9 and 10 show the dependence of the amount of used resources on the number of neurons and the speed of the RBF neural network on the number of neurons. As it can be seen with the increase in the number of neurons in the network, the performance is almost unchanged since each neuron and computational operations in it are separate parallel processes.

The method of designing nonlinear Gaussian activation functions of an artificial neuron of the RBF neural network on an FPGA using VHDL is developed in the work. Implementation of the Gaussian function by this method takes as much of the used resource LUTs of the FPGA chip as in the works [46, 47], but allows significantly reduce memory usage from 9 to 1 bloc BRAM. That will increase the number of network neurons when using the same amount of hardware resources. That will solve more complex problems using neural networks. Indicators of the speed of hardware units with neural networks in the works [46, 47] are not specified.

Table 6 presents a comparison of the results of modeling the RBF neural network on a chip implemented by the closest known analogue [51] (Analog) and implemented RBF neural network according to the developed method and algorithm in this work (Proposed). The simulation was performed on the same chip Virtex-6 xc6vcx240t-1ff1156.

As it can be seen from Table 6, the implementation of the developed method and the algorithm has a higher speed and requires less of used resource chip. The implemented

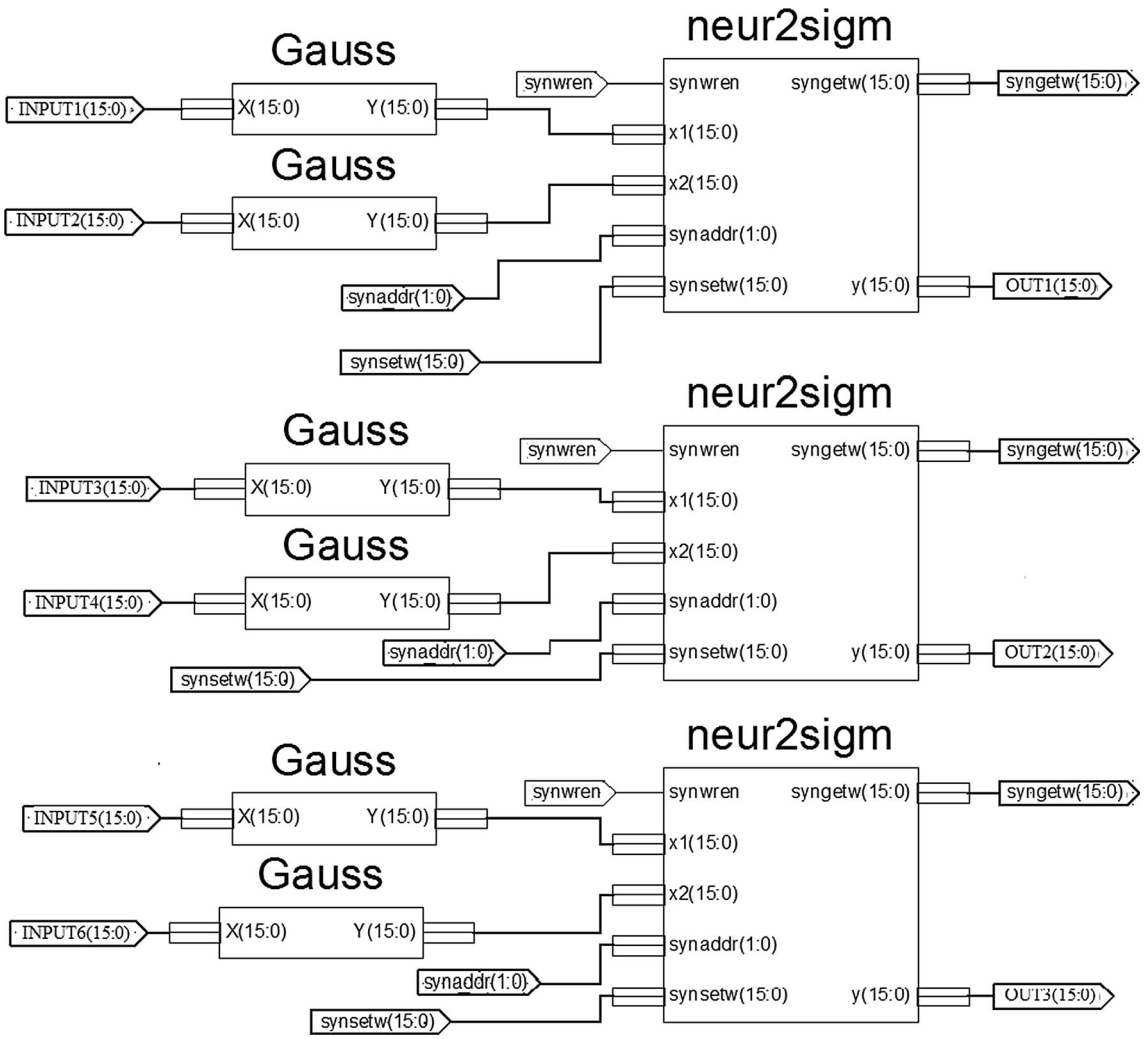

Fig. 7 RBF neural network 3–3 implemented on FPGA

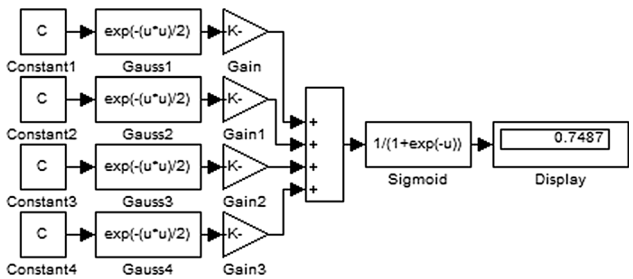

Fig. 8 Test model of RBF network 4–1 in MATLAB

neural networks in this work used nonlinear, sigmoidal functions of the source layer in contrast to the linear ones in a similar work [51].

5 Conclusions

Hardware implementation of neural networks RBF provides high-speed information processing. This allows them to be used in real-time computing systems for controlling high-speed objects. In particular, such calculators can be used to create control systems in robotics, drone control, vehicle control, pattern recognition, analysis and decision-making in the Internet of things systems, spacecraft control, military equipment, and many other different applications in modern technology. In these systems, they will help solve the problem of recognition, control, approximation and classification of received data from video cameras and sensors.

This paper presented a novel approach to increase the speed of RBF neural networks by developing a method of parallel hardware implementation on the FPGA, with minimal use of computing resources. In the future work, the hardware component of training of neural networks on

Table 5 Results of modeling of RBF networks in FPGA

Neural network	Number of Gauss functions	LUT	Speed (ns)	RBF network output		Error
				MATLAB	FPGA	
1–1	1	452	86.671	0.5638936	0.546875	0.017019
1–2	2	771	86.736	0.6257341	0.609375	0.016359
1–3	3	1090	86.761	0.6837239	0.671875	0.011849
1–4	4	1409	86.770	0.7832823	0.765625	0.017657
2–1	2	695	90.043	0.6386353	0.625	0.013635
2–2	4	1110	90.107	0.7014409	0.6875	0.013941
2–3	6	1525	90.133	0.7574761	0.75	0.007476
2–4	8	1940	90.141	0.7922394	0.765625	0.026614
3–1	3	942	93.913	0.8254911	0.796875	0.028616
3–2	6	1470	93.978	0.7574761	0.75	0.007476
3–3	9	1998	94.003	0.6837239	0.671875	0.011849
3–4	12	2526	94.012	0.5707037	0.5625	0.008204
4–1	4	1193	101.579	0.7365121	0.734375	0.002137
4–2	8	1838	101.755	0.8237374	0.796875	0.026862
4–3	12	2483	101.78	0.5770806	0.5625	0.014581
4–4	16	3128	101.789	0.6505826	0.625	0.025583
5–1	5	1445	105.123	0.9070203	0.890625	0.016395
5–2	10	2192	105.188	0.8765213	0.873749	0.002777
6–1	6	1679	107.54	0.9785327	0.981523	0.00299
6–2	12	2511	107.604	0.6329854	0.630458	0.002527
8–1	8	2156	112.968	0.5783654	0.575532	0.002833
8–2	16	3196	113.033	0.8967523	0.892632	0.00412

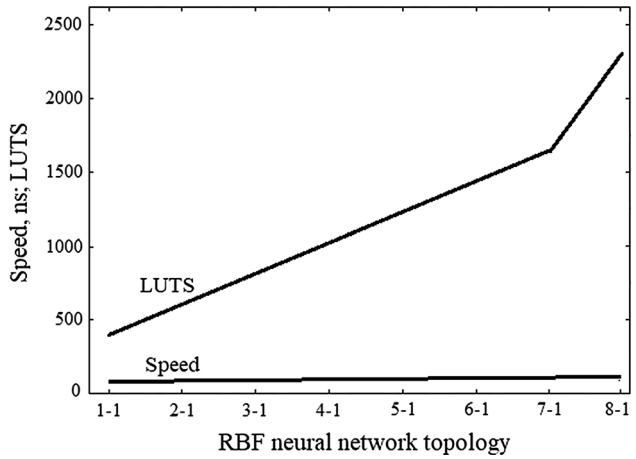

Fig. 9 Dependence of the amount of used resource on the number of neurons and the speed of the RBF neural network on the number of neurons

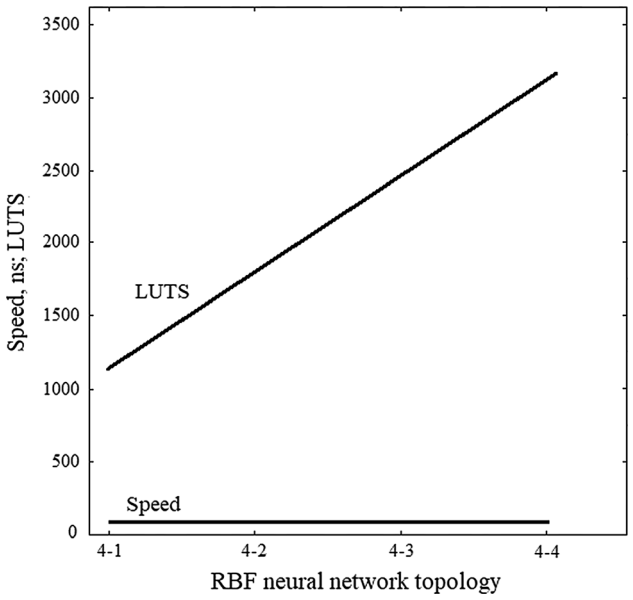

Fig. 10 Dependence of the amount of used resource on the number of neurons and the speed of the RBF neural network on the number of neurons

FPGA will be developed and investigated. A model of the current neurocontroller of the high-speed object control system in real time will be developed. The central element of the controller will be the FPGA. The controller will include regulators based on the neural networks developed in this work and components of their training.

Table 6 Comparative results of modeling of neural networks in FPGA

Neural network	LUT		Speed (ns)	
	Analog [51]	Proposed	Analog [51]	Proposed
2–1	1170	257	10	1.3
4–1	2370	420	10	1.37

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Sigeru O, Khalid MB, Rubiyah Y (1996) Neuro-control and its applications. Springer-Verlag, London. https://doi.org/10.1007/978-1-4471-3058-1
2. Dreyfus G (2005) Neural networks: methodology and applications. Springer-Verlag, Berlin. https://doi.org/10.1007/3-540-28847-3
3. Edelen AL, Biedron SG, Chase BE, Edstrom D, Milton SV, Stabile P (2016) Neural networks for modeling and control of particle accelerators. IEEE Trans Nucl Sci 63:878–897. https://doi.org/10.1109/TNS.2016.2543203
4. Melchert F, Bani G, Seiffert U, Biehl M (2019) Adaptive basis functions for prototype-based classification of functional data. Neural Comput Appl. https://doi.org/10.1007/s00521-019-04299-2
5. Gonçalves S, Cortez P, Moro S (2020) A deep learning classifier for sentence classification in biomedical and computer science abstracts. Neural Comput Appl 32:6793–6807. https://doi.org/10.1007/s00521-019-04334-2
6. Passalis N, Tefas A (2020) Continuous drone control using deep reinforcement learning for frontal view person shooting. Neural Comput Appl 32:4227–4238. https://doi.org/10.1007/s00521-019-04330-6
7. Korniyenko BY, Osipa LV (2018) Identification of the granulation process in the fluidized bed. ARPN J Eng Appl Sci 13:4365–4370
8. Hong Q, Li Y, Wang X (2019) Memristive continuous Hopfield neural network circuit for image restoration. Neural Comput Appl. https://doi.org/10.1007/s00521-019-04305-7
9. N. Sundararajan, P. Saratchandran, Yan Li (2002) Fully tuned radial basis function neural networks for flight control. Springer, US. P. 158. https://doi.org/https://doi.org/10.1007/978-1-4757-5286-1
10. Chen M, Ge SS, Voon B, How E (2010) Robust adaptive neural network control for a class of uncertain MIMO nonlinear systems with input nonlinearities. IEEE Trans Neural Netw 21:796–812. https://doi.org/10.1109/TNN.2010.2042611
11. Zhao Z, Zheng P, Xu S, Wu X (2019) Object detection with deep learning: a review. IEEE Trans Neural Netw Learn Syst 30:3212–3232. https://doi.org/10.1109/TNNLS.2018.2876865
12. Lawrynczuk M (2014) Computationally efficient model predictive control algorithms: a neural network approach. Springer, Berlin. https://doi.org/10.1007/978-3-319-04229-9
13. Krestinskaya O, James AP, Chua LO (2020) Neuromemristive circuits for edge computing: a review. IEEE Trans Neural Netw Learn Syst 31:4–23. https://doi.org/10.1109/TNNLS.2019.2899262
14. Shao L, Zhu F, Li X (2015) Transfer learning for visual categorization: a survey. IEEE Trans Neural Netw Learn Syst 26:1019–1034. https://doi.org/10.1109/TNNLS.2014.2330900
15. Bouvier M, Valentian A, Mesquida T, Rummens F, Reyboz M, Vianello E, Beigne E (2019) Spiking neural networks hardware implementations and challenges: a survey. ACM J Emerg Technol Comput Syst 15:22. https://doi.org/10.1145/3041033
16. Lawrence S, Burns I, Back A, Tsoi AC, Giles CL (2012) Neural network classification and prior class probabilities. In: Montavon G, Orr GB, Müller KR (eds) Neural networks: tricks of the trade. Lecture notes in computer science, vol 7700. Springer, Berlin. https://doi.org/10.1007/978-3-642-35289-8_19
17. Alfaro-Ponce M, Arguelles-Cruz A, Chairez I (2014) Adaptive identifier for uncertain complex nonlinear system based on continuous neural network. IEEE Trans Neural Netw Learn Syst 25(3):483–494. https://doi.org/10.1109/TNNLS.2013.2275959
18. Brassai S, Enachescu C, Losonczi L (2012) RBF network for mobile robot sonar based localization and environment modeling. In: 13th International conference on optimization of electrical and electronic equipment (OPTIM), pp 1499–1504. https://doi.org/10.1109/OPTIM.2012.6231939
19. Lee G, Kim S, Jung S (2008) Hardware implementation of a RBF neural network controller with a DSP 2812 and an FPGA for controlling nonlinear systems. Int Conf Smart Manuf Appl. https://doi.org/10.1109/ICSMA.2008.4505634
20. Omondi A, Rajapakse J (2006) FPGA implementations of neural networks. Springer, Dordrecht. https://doi.org/10.1007/0-387-28487-7
21. Zairi H, Talha MK, Meddah K, Slimane SO (2020) FPGA-based system for artificial neural network arrhythmia classification. Neural Comput Appl 32:4105–4120. https://doi.org/10.1007/s00521-019-04081-4
22. Misra J, Saha I (2010) Artificial neural networks in hardware: a survey of two decades of progress. Neurocomputing 74(1–3):239–255. https://doi.org/10.1016/j.neucom.2010.03.021
23. Baptista D, Abreu S, Freitas F, Vasconcelos R, Morgado-Dias F (2013) A survey of software and hardware use in artificial neural networks. Neural Comput Appl 23:591–599. https://doi.org/10.1007/s00521-013-1406-y
24. Zou X, Wang L, Tang Y, Liu Y, Zhan S, Tao F (2018) Parallel design of intelligent optimization algorithm based on FPGA. Int J Adv Manuf Technol 94:3399–3412. https://doi.org/10.1007/s00170-017-1447-y
25. Alfaro-Ponce M, Chairez I, Etienne-Cummings R (2019) Automatic detection of electrocardiographic arrhythmias by parallel continuous neural networks implemented in FPGA. Neural Comput Appl 31:363–375. https://doi.org/10.1007/s00521-017-3051-3

26. Nurmi J (2007) Processor design. System-on-chip computing for ASICs and FPGAs. Springer, Dordrecht. https://doi.org/10.1007/978-1-4020-5530-0
27. Nelles O (2013) Nonlinear system identification: from classical approaches to neural networks and fuzzy models. Springer, Berlin. https://doi.org/10.1007/978-3-662-04323-3
28. Omatu S (2017) Classification of mixed odors using a layered neural network. Int J Comput 16:41–48
29. Xiaofang Y, Yaonan W, Sun Wei Wu, Lianghong, (2010) RBF networks-based adaptive inverse model control system for electronic throttle. IEEE Trans Control Syst Technol 18:750–756. https://doi.org/10.1109/TCST.2009.2026397
30. Baghaee HR, Mirsalim M, Gharehpetian GB (2016) Power calculation using RBF neural networks to improve power sharing of hierarchical control scheme in multi-DER microgrids. IEEE J Emerg Sel Top Power Electron 4(4):1217–1225. https://doi.org/10.1109/JESTPE.2016.2581762
31. Kravets P, Shymkovych V (2020) Hardware Implementation Neural Network Controller on FPGA for Stability Ball on the Platform. In: Hu Z, Petoukhov S, Dychka I, He M (eds) Advances in computer science for engineering and education II. ICCSEEA 2019. Advances in intelligent systems and computing, vol 938. Springer, Cham. https://doi.org/10.1007/978-3-030-16621-2_23
32. Chin CS, Ji Xi, Woo WL, Kwee TJ, Yang W (2019) Modified multiple generalized regression neural network models using fuzzy C-means with principal component analysis for noise prediction of offshore platform. Neural Comput Appl 31:1127–1142. https://doi.org/10.1007/s00521-017-3143-0
33. Agarwal V, Bhanot S (2018) Radial basis function neural network-based face recognition using firefly algorithm. Neural Comput Appl 30:2643–2660. https://doi.org/10.1007/s00521-017-2874-2
34. Li T, Duan S, Liu J, Wang L (2018) An improved design of RBF neural network control algorithm based on spintronic memristor crossbar array. Neural Comput Appl 30:1939–1946. https://doi.org/10.1007/s00521-016-2715-8
35. Kung Y-S, Than H, Chuang T-Y (2018) FPGA-realization of a self-tuning PID controller for X-Y table with RBF neural network identification. Microsyst Technol 24:243–253. https://doi.org/10.1007/s00542-016-3248-x
36. Katsunari Shibata, Koji Ito (1999) Gauss-sigmoid neural network. In: IJCNN'99. International joint conference on neural networks. Proceedings. Washington, DC, USA, vol 2, pp 1203–1208. https://doi.org/10.1109/IJCNN.1999.831131
37. Lachowicz S, Pfleiderer H-J (2008) Fast evaluation of nonlinear functions using FPGAs. Adv Radio Sci 6:233–237. https://doi.org/10.5194/ars-6-233-2008
38. Fan Z-C, Hwang W-J (2013) Efficient VLSI architecture for training radial basis function networks. Sensors 13:3848–3877. https://doi.org/10.3390/s130303848
39. Jokar E, Abolfathi H, Ahmadi A (2019) A novel nonlinear function evaluation approach for efficient FPGA mapping of neuron and synaptic plasticity models. IEEE Trans Biomed Circuits Syst 13:454–469. https://doi.org/10.1109/TBCAS.2019.2900943
40. Goel K, Arun U, Sinha AK (2014) An efficient hardwired realization of embedded neural controller on system-on-programmable-chip (SOPC). Int J Eng Res Technol (IJERT) 3(1):276–284
41. Brassai ST, Dávid L, Bakó L (2004) Hardware implementation of CMAC-based artificial network with process control application. Timisoara, Transaction on Electronics and communication, Scientific bulletin of the "Politehnica" University of Timisoara, pp 209–213
42. Brassai ST, Bakó L (2007) Hardware implementation of CMAC type neural network on FPGA for command surface approximation. Acta Polytechnica Hungarica 4(3):5–16
43. Yang Z, Qian J (2010) Hardware implementation of RBF neural network on FPGA coprocessor. In: Zhu R, Zhang Y, Liu B, Liu C (eds) Information computing and applications. ICICA 2010. Communications in computer and information science, vol 105. Springer, Berlin. https://doi.org/10.1007/978-3-642-16336-4_55
44. Jhang J-Y, Tang K-H, Huang C-K, Lin C-J, Young K-Y (2018) FPGA implementation of a functional neuro-fuzzy network for nonlinear system control. Electronics 7:145. https://doi.org/10.3390/electronics7080145
45. Polat O, Yıldırım T (2010) FPGA implementation of a general regression neural network an embedded pattern classification system. Digital Signal Process 20:881–886. https://doi.org/10.1016/j.dsp.2009.10.013
46. Brassai ST, Bakó L, Pana G, Dan S (2008) Neural control based on RBF network implemented on FPGA. In: 11th International conference on optimization of electrical and electronic equipment, vol 2008, pp 41–46. https://doi.org/10.1109/OPTIM.2008.4602496
47. Brassai S, Bakó L (2009) Visual trajectory control of a mobile robot using FPGA implemented neural network. Pollack Periodica 4(3):129–142. https://doi.org/10.1556/pollack.4.2009.3.12
48. Thanh NP et al (2016) Digital hardware implementation of a radial basis function neural network. Comput Electr Eng 53:106–121. https://doi.org/10.1016/j.compeleceng.2015.11.017
49. Y. Ji, F. Ran, C. Ma and D. J. Lilja (2015) A hardware implementation of a radial basis function neural network using stochastic logic. 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE). Grenoble. pp. 880–883
50. Lucas L, Acosta N (2011) Hardware radial basis function neural network automatic generation. JCS&T 11:15–20
51. de Souza ACD, Fernandes MAC (2014) Parallel fixed point implementation of a radial basis function network in an FPGA. Sensors 14:18223–18243. https://doi.org/10.3390/s141018223
52. Yang F, Paindavoine M (2003) Implementation of an RBF neural network on embedded systems: real-time face tracking and identity verification. IEEE Trans Neural Netw 14:1162–1175. https://doi.org/10.1109/TNN.2003.816035
53. Kravets PI, Shimkovich VN, Ferens DA (2015) Method and algoritms of implementation on PLIS the activation function for artifical neuron chains. Èlektron Model 37(4):63–74