sensors

MDPI

Letter

FPGA-Based Implementation of Stochastic Configuration Networks for Regression Prediction

Yunqi Gao [1,2,†], Feng Luan [1,2,*,†], Jiaqi Pan [1,2], Xu Li [3] and Yaodong He [3]

[1] School of Computer Science and Engineering, Northeastern University, Shenyang 110819, China; 20174852@stu.neu.edu.cn (Y.G.); 20174839@stu.neu.edu.cn (J.P.)
[2] Key Laboratory of Intelligent Computing in Medical Image, Ministry of Education, Northeastern University, Shenyang 110819, China
[3] The State Key Laboratory of Rolling and Automation, Northeastern University, Shenyang 110819, China; lixu@ral.neu.edu.cn (X.L.); 1870306@stu.neu.edu.cn (Y.H.)
* Correspondence: luanfeng@mail.neu.edu.cn
† These authors contributed equally to this work.

Received: 6 June 2020; Accepted: 24 July 2020; Published: 28 July 2020

Abstract: The implementation of neural network regression prediction based on digital circuits is one of the challenging problems in the field of machine learning and cognitive recognition, and it is also an effective way to relieve the pressure of the Internet in the era of intelligence. As a nonlinear network, the stochastic configuration network (SCN) is considered to be an effective method for regression prediction due to its good performance in learning and generalization. Therefore, in this paper, we adapt the SCN to regression analysis, and design and verify the field programmable gate array (FPGA) framework to implement SCN model for the first time. In addition, in order to improve the performance of the SCN model based on the FPGA, the implementation of the nonlinear activation function on the FPGA is optimized, which effectively improves the prediction accuracy while considering the utilization rate of hardware resources. Experimental results based on the simulation data set and the real data set prove that the proposed FPGA framework successfully implements the SCN regression prediction model, and the improved SCN model has higher accuracy and a more stable performance. Compared with the extreme learning machine (ELM), the prediction performance of the proposed SCN implementation model based on the FPGA for the simulation data set and the real data set is improved by 56.37% and 17.35%, respectively.

Keywords: field programmable gate array; hardware neural networks; regression prediction; stochastic configuration networks

1. Introduction

In the past few decades, the gradient-based learning method has been widely used in training neural networks, such as the backpropagation (BP) algorithm, which uses the back propagation of error to adjust the weight of the network. However, due to the improper learning step size, the convergence speed of the algorithm is very slow, and it is easy to produce a local minimum value. So, a lot of iterations are often needed to get more satisfactory accuracy. These problems have become the main bottleneck restricting its development in the application field. Therefore, improving the learning ability and generalization performance of neural network models is a challenging task. One solution is to find an appropriate architecture for the neural network model. Artificial neural networks have two important hyper-parameters, which are used to control the scale of the network: the number of layers and the number of nodes in each hidden layer. The values of these parameters must be specifically determined when configuring the network. However, there is no rule to determine the

scale of the network. In the regression prediction application, most researchers set up a series of models with different scales during the training process, and then selected the best one according to the test results [1–3]. However, this kind of method increases the training cost and time. Thus, how to determine the ideal number of hidden layer nodes before network training is an urgent problem to be solved [4,5]. In order to solve this problem, a newly developed randomized learner model, termed stochastic configuration networks (SCNs), was proposed by Wang et al. [6]. As a single-layer feedforward neural network, the SCN belongs to the random neural network class. Although the parameters of the SCN are also randomly generated, it is different from the existing randomized learning algorithms for single layer feed-forward neural networks (SLFNNs); the SCN mainly randomly assigns the input weights and biases of hidden nodes in the light of a supervisory mechanism, and the output weights are analytically evaluated in either a constructive or selective manner. Compared to other random neural networks, this random learning model is also different from the classical random vector functional link (RVFL) network. The SCN restricts the assignment of input weights and bias by introducing inequality constraints. Under the supervisory mechanism, SCN has a universal approximation property with the increase of the number of hidden nodes [7–10]. Instead of training a model with a fixed architecture, the construction process of the SCN starts with a small sized network and then adds hidden nodes incrementally until an acceptable tolerance is achieved, then solves a global least squares problem with the current learner model to find the output weights. Its advantages ar: the minimization of a convex cost that avoids the presence of local minima, good generalization performance, and a notable representation ability [6]. Compared with deep neural networks, the SCN has lower training complexity and faster learning speed.

Nowadays, random neural networks have left impressive performance in the fields of deep learning and cognitive science. Compared with being applied to computer platforms, the implementation of random neural networks on reconfigurable digital platforms, such as field programmable gate array (FPGA), shows its huge and unique advantages: First, in the neural domain where parallelism and distributed computing are inherently involved, FPGAs have increased their speed with their very high computing power [11]. Second, with the miniaturization of component manufacturing technology [12], neural networks are becoming more and more widespread in embedded applications. Third, compared to computers, hardware systems can reduce costs by decreasing power requirements and lowering the number of components [13]. Fourth, parallel and distributed architectures have a high fault tolerance rate for system components [14], and provide support for applications that require security. Also today, a large number of mobile devices are connected to the internet, and cloud computing data centers are under excessive load. The implementation of neural networks in software requires a lot of computing resources. In order to reduce the Internet load, the collaborative use of edge and cloud computing is particularly important. FPGA-based random neural networks stand at the edge-computing perspective and move part of the computational power to data collection sources, thereby reducing the network load [15]. With the surge in data volume and the constant demand for computing power, the original computing framework consisting solely of CPUs has been unable to meet the real-time requirements of the edge-computing system, while one of the greatest value of FPGAs is that they are reconfigurable, so designs can be updated at any time, even after the hardware has been deployed in the field. By virtue of this advantage and high efficiency, FPGA is widely used in many edge computing scenarios [16]. At the same time, with the development of Internet of Things (IoT), hubs should support a large number of ultra-low power network protocols, various applications workloads, and be responsible for completing authentication, encryption, and security. This kind of changing and uncertain environment is a terrible thing for ASIC or SoC, but it is easy to implement for FPGA with very high running speed and computational efficiency. Therefore, the implementation of neural networks on FPGAs has very good development prospects and application values.

Researchers have done a lot of works on the hardware implementation of random neural networks and made many achievements. In 2012, Decherchi et al. implemented the classification prediction model of extreme learning machine (ELM) [17–19] on the FPGA and achieved high-precision prediction

results [20]. In 2018, Ragusa et al. improved the hardware implementation model of the ELM classifier for resource-constrained devices, effectively balancing accuracy, and network complexity, and reducing resource utilization [21]. In 2019, Safaei et al. proposed a specialized system on chip (SoC) hardware implementation and design approach for embedded online sequential ELM (OS-ELM) classification, which has been optimized for efficiency in real-time applications [22].

Inspired by the above papers, this paper designs and completes the implementation of the SCN regression prediction model on the FPGA. The main contributions of this paper are listed below. (1) The SCN model exhibiting good performance in learning and generalization is investigated for regression prediction; this is the first time the SCN model on the FPGA has been implemented. (2) A new nonlinear activation function is proposed to optimize the FPGA implementation of the SCN model; this new activation function, unlike the previous ones, further considers the prediction accuracy and hardware resource utilization. (3) Experimental results from simulation and real data sets indicate that the proposed FPGA framework successfully implements the SCN regression prediction model. (4) The prediction performance of the proposed FPGA implementation of the SCN model is significantly improved compared with the same case studies for other implementation in the literature [20].

The rest of this paper is organized as follows. Section 2 describes the specifics of the SCN. Section 3 proposes the hardware architecture of the SCN. Section 4 proposes methods for improving and optimizing the performance of FPGA models. Section 5 verifies the designed SCN hardware prediction model on the simulation data set and the real industrial data set. Finally, the conclusion is drawn in Section 6.

2. Stochastic Configuration Networks

For a target function $f : R^d \to R^m$, suppose that an SCN model has already been built with $L-1$ hidden nodes, i.e., $f_{L-1} = \sum_{l=1}^{L-1} \beta_l \phi_l\left(\omega_l^T x + b_l\right)$ $(L = 1, 2, \ldots; f_0 = 0)$, where $\beta_l = \left[\beta_{l,1}, \beta_{l,2}, \ldots, \beta_{l,m}\right]^T$, and $\phi_l\left(\omega_l^T + b_l\right)$ is an activation function of the lth hidden node with random input weights ω_l and bias b_l. $e_{L-1}^* = f - f_{L-1} = \left[e_{L-1,1}^*, \ldots, e_{L-1,m}^*\right]$ denotes the residual error where $\left[\beta_1^*, \beta_2^*, \ldots, \beta_{L-1}^*\right] = argmin_\beta \|f - \sum_{l=1}^{L-1} \beta_l \phi_l\|$.

Given a training data set with N sample pairs $\{(x_n, y_n), n = 1, 2, \ldots, N\}$, where $x_n \in R^d$ and $y_n \in R^m$, let $X \in R^{N \times d}$ and $Y \in R^{N \times m}$ represent the input and output data matrix; $e_{L-1}(X) \in R^{N \times m}$ be the residual error matrix, where each column $e_{L-1,q}(X) = \left[e_{L-1,q}(\boldsymbol{x}_1), \ldots, e_{L-1,q}(\boldsymbol{x}_N)\right]^T \in R^N$, $q = 1, 2, \ldots, m$. Denote the output vector of the Lth hidden node ϕ_L for the input X by

$$h_L(X) = \left[\phi_L\left(\boldsymbol{\omega}_L^T x_1 + b_L\right), \ldots, \phi_L\left(\boldsymbol{\omega}_L^T x_N + b_L\right)\right]^T. \tag{1}$$

Thus, the hidden layer output matrix of f_L can be expressed as $H_L = [h_1, h_2, \ldots, h_L]$. Denoted by $\xi_{L,q} = \frac{\left(e_{L-1,q}^T(X) \cdot h_L(X)\right)^2}{h_L^T(X) \cdot h_L(X)} - (1 - r - \mu_L) e_{L-1,q}^T(X) e_{L-1,q}(X)$,

$$q = 1, 2, \ldots, m, \tag{2}$$

where $0 < r < 1$ and $\{\mu_L\}$ is a nonnegative real number sequence with $lim_{L \to +\infty} \mu_L = 0$ subjected to $\mu_L \leq (1 - r)$. The SCN algorithm firstly generates a large pool of $T_{\max}$ candidate nodes, namely $\left\{\phi_L^{(1)}, \phi_L^{(2)}, \ldots, \phi_L^{(T_{\max})}\right\}$, in varying intervals. Then, it picks up those candidate nodes whose minimal value of the set $\left\{\xi_{L,1}, \ldots, \xi_{L,m}\right\}$ is positive. Then, the candidate note ϕ_L^* with the largest value of $\xi_L = \sum_{q=1}^m \xi_{L,q}$ will be assigned as the Lth hidden node for f_L. Thus, the output weight matrix of the SCN model, $\beta = [\beta_1, \beta_2, \ldots, \beta_L]^T \in R^{L \times m}$, could be computed by the standard least squares method, that is,

$$\beta^* = \underset{\beta}{argmin}\left\|H_L\beta - Y\right\|_F^2 = H_L^{\dagger}Y, \tag{3}$$

where $H_L^{\dagger}$ is the Moore-Penrose generalized inverse of the matrix H_L, and $\|\cdot\|_F$ represents the Frobenius norm [23–25].

The construction process of the SCN starts with a small sized network, then incrementally adds hidden nodes followed by computing the output weights. This process continues until the model meets some termination criteria. The supervisory mechanism of the SCN guarantees the universal approximation property.

3. FPGA-Based Implementation of the SCN

The implementation of the SCN on the FPGA needs to balance accuracy, speed, and resource utilization. The proposed architecture should make full use of the advantages of FPGA parallel processing. Due to the fact that the SCN adopts the method of gradually increasing hidden nodes to find the optimal solution, the flexibility of the model must be fully considered in the design, so that it can make specific changes for different problems.

Figure 1 shows the overall architecture of the SCN inference prediction model based on the FPGA. The whole architecture includes three parts: the first part belongs to a parallel processing structure, and the second and third parts adopt a pipeline structure. The first part, Input Part, stores the feature vector $x = [x_1, \ldots, x_n]$, the weights $\omega_j (j = 1, \ldots, H)$ connecting the input layer to the hidden node and the bias term $b_j (j = 1, \ldots, H)$. The feature vector x adopts a signed, two-complement fixed-point representation. The binary number length of each feature vector is $a + b + 1$, where a is the number of digits representing a positive number, b is the number of digits representing a decimal, and the remaining one is used to represent a sign bit. Negative numbers are coded by inverting their absolute value and adding 1. In order to facilitate the calculation by FPGA, the weight value ω_j is specially processed and can be expressed as

$$\omega_j = sign(r_1)2^{-r_2}, \tag{4}$$

where $r_1 \in [-1, 1]$ and $r_2 \in [0, R]$ are random quantities (in the program, r_2 can take the following values: 1, 2, 3). If x is extended to $x = [x_1, \ldots, x_n, 1]$ and ω_j to $\omega_j \in R^{n+1}$, the bias term b_j can be embedded in ω_j. The second part, Neuron Part, receives the results of the parallel processing from the first part and calculates the output of the activation function. The third part, Output Part, receives the output of the second part and calculates the output neurons through serial processing. A finite-state machine controls the entire process, ensuring that the calculations of the third part are always one clock cycle ahead from the calculations of the first and second parts.

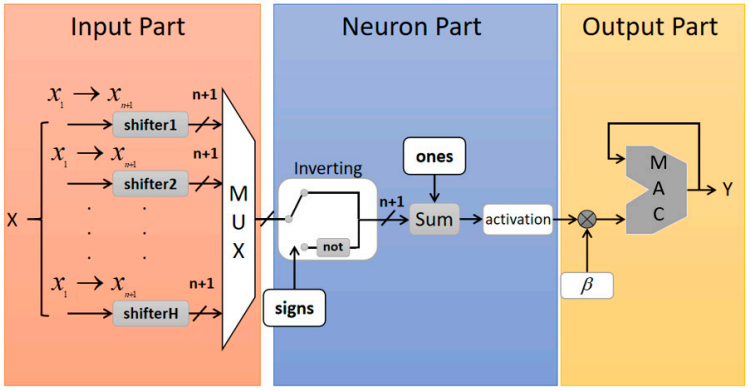

Figure 1. The architecture of FPGA-based SCN.

The specific design of each part is as follows:

(1) **Input Part:** The *Input* module stores all extended feature vectors $x_i = \left[x_{i1}, x_{i2}, \ldots, x_{i(n+1)}\right]^T \in R^{n+1}$, and the *Shifter* modules stores the absolute value of extended random weights $\omega_j \in R^{n+1}$. Due to the special processing of ω_j, the FPGA can input $(n+1) \times H$ results into the *Mux* module through parallel shift calculation. The *Mux* module outputs the calculation results in turn according to the finite state machine, and outputs $(n+1)$ items each time.

(2) **Neuron Part:** First, the *Inverting* module receives the output of the first part according to the signs of the random weights $\omega_j \in R^{n+1}$, and applies a bitwise NOT to the result item whose corresponding random weight is negative. Then, the output $(n+1)$ result and the corresponding item in the ones module are input to the *Sum* module for summation, where the *Ones* module compensates the difference "1" between the calculated result and the true value due to the bitwise NOT. Finally, the result of the *Sum* module is activated by the sigmoid function of the *activation* module to obtain the output $\varphi\left(\omega_j x + b_j\right)$ of the hidden layer. The *activation* module should be a hardware implementation of the activation function. This is an extremely critical step. The implementation and optimization of the activation function in FPGA are specifically introduced in Section 4.1.

(3) **Output Part:** The *Mac* module multiplies the output $h_j = \varphi\left(\omega_j x + b_j\right)$ of the hidden layer by the weight β_j from the hidden layer to the output layer according to the control of the state machine. The calculation results are summed by an accumulator, and then the output Y of the neural network is obtained.

4. Improvement and Optimization of FPGA-Based Model Performance

When the FPGA implements the single hidden layer neural network prediction model, the error sources are mainly the approximation degree of the nonlinear activation function and the difference between the data and the actual floating point number due to numerical coding.

4.1. Proposal of Hardware-Oriented Sigmoid Function

The sigmoid function is the most commonly used nonlinear activation function. When the SCN is implemented on FPGA, the sigmoid function (Equation (5)) is selected:

$$f(x) = \frac{1}{1+e^{-x}}. \tag{5}$$

Because factors such as accuracy and resources must be considered at the same time, the implementation of nonlinear functions on the FPGA is very complicated [26,27]. Division and exponential operations are extremely demanding operations, requiring a large amount of area resources and slow convergence, so it is not feasible to directly implement the sigmoid function on the FPGA. Polynomial approximation and Look-Up-Table (LUT) are two common methods when the accuracy and speed meet the requirements.

In terms of the approximation of the sigmoid function, researchers have made many explorations. Tommiska proposed the piecewise linear approximation of the sigmoid function in [28], which is also the traditional method of sigmoid function implementation in hardware:

$$f(x) = \begin{cases} 0, & x < -2 \\ 0.25x + 0.5, & -2 \le x \le 2 \\ 1, & x > 2 \end{cases}. \tag{6}$$

This scheme was proved to be effective in solving hardware implementation of classification problems in [20]. The approximation of the activation function has little effect on classification problems, and the results of classification problems are often determined through comparison operations. The regression problem needs to directly deal with the calculation results of the FPGA-based model, which requires that the implementation of the activation function in hardware has good similarity with

the original sigmoid function. It can be seen from Equation (6) that when $-2 \le x \le 2$, $f(x) = 0.25x + 0.5$ is the first-order Taylor expansion of the sigmoid function. Figure 2 shows that Equation (6) has an ideal approximation to the sigmoid function on $x \in [-1, 1]$, but not on $x \in [-3, -1) \cup (1, 3]$. Equation (7) gives the third-order Taylor expansion of the sigmoid function:

$$f(x) = \begin{cases} 0.167, & x < -2 \\ -0.021x^3 + 0.25x + 0.5, & -2 \le x \le 2 \\ 0.833, & 2 < x \end{cases}. \tag{7}$$

However, it is known from Figure 2 that Equation (7) does not improve the problem of Equation (6), and the approximation effect is still not ideal. Considering the use of resources, the higher-order Taylor expansion $(x \ge 5)$ the sigmoid function is no longer suitable for hardware implementation. Therefore, a combinational approximation should be found on the basis of Equation (7). A method of Piecewise second-order approximation of sigmoid function was proposed in [29] (Equation (8)):

$$f(x) = \begin{cases} -0.03125x^2 + 0.5, & -3 \le x < 0 \\ 0.03125x^2 + 0.5, & 0 \le x \le 3 \end{cases}. \tag{8}$$

Due to excessive consideration of resource utilization in [29], the proposed results are severely impaired in the approximation of the sigmoid function shown in Figure 2. The Equation (9) proposed in this paper improves the second-order approximation of Equation (8) on the basis of Equation (7), and applies it to $x \in [-3, -1) \cup (1, 3]$:

$$f(x) = \begin{cases} 0.03913x^2 + 0.2651x + 0.5, & -3 \le x < -1 \\ -0.021x^3 + 0.25x + 0.5, & -1 \le x \le 1 \\ -0.03913x^2 + 0.2651x + 0.5, & 1 < x \le 3 \end{cases}. \tag{9}$$

As shown in Figure 2, comparing with the sigmoid function, Equation (9) almost perfectly presents the sigmoid function.

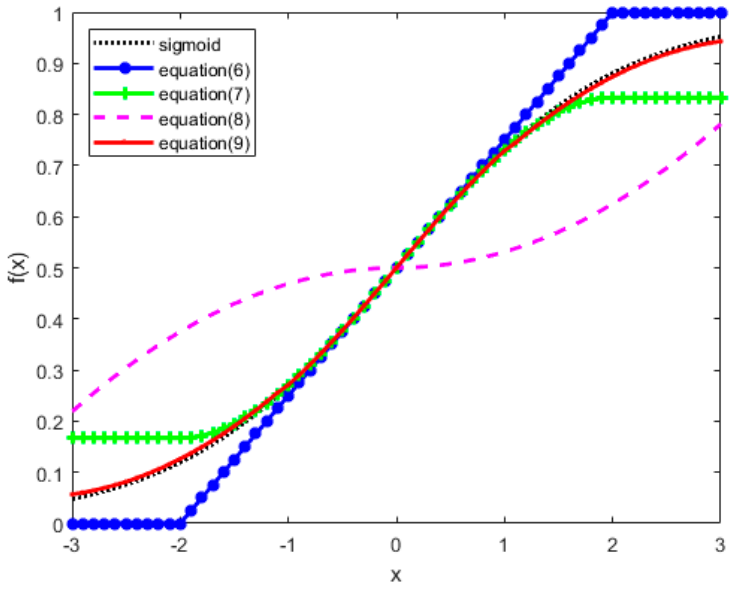

Figure 2. Comparison of different sigmoid functions.

In fact, many researchers have given different approximation methods for the implementation of the sigmoid function on FPGA. In 2012, Panicker et al. proposed a piecewise linear approximation method in [30]:

$$f(x) = \begin{cases} 0.03125|x| + 0.84375, & 2.375 \le |x| \le 3 \\ 0.125|x| + 0.625, & 1 \le |x| < 2.375 \\ 0.25|x| + 0.5, & 0 \le |x| < 1 \end{cases}. \tag{10}$$

In 2015, Khodja et al. proposed a piecewise second-order approximation method in [31]:

$$f(x) = \begin{cases} 0.0332x^2 + 0.2549x + 0.5, & -3 \le x < 0 \\ -0.0332x^2 + 0.2549x + 0.5, & 0 \le x \le 3 \end{cases}. \tag{11}$$

In 2016, Ngah et al. also proposed a piecewise second-order approximation method in [32]:

$$f(x) = \begin{cases} 0.5 \times (1 - 0.25|x|)^2, & -3 \le x < 0 \\ 1 - 0.5 \times (1 - 0.25|x|)^2, & 0 \le x \le 3 \end{cases}. \tag{12}$$

In order to qualitatively analyze the approximation of sigmoid function by Equations (6)–(12), according to the method of [29], the maximum and average errors are used to evaluate the approximation degree of the sigmoid function. If a function $f(x)$ is approximated by a function $\hat{f}(x)$ the interval $x \in (\eta_0, \eta_1)$, the average and maximum errors are obtained by uniformly sampling x on 10^6 equally spaced points in the domain of (η_0, η_1).

$$\begin{cases} Average\ Error = \frac{\sum_{i=0}^{10^6-1} \left| \hat{f}(x_i) - f(x_i) \right|}{10^6} \\ Maximum\ Error = \max\limits_{\eta_0 < x_i < \eta_1} \left| \hat{f}(x_i) - f(x_i) \right| \end{cases} \tag{13}$$

According to Equation (13), when $\eta_0 = -3$, $\eta_1 = 3$, the average and maximum errors corresponding to Equations (6)–(12) are shown in Table 1.

Table 1. The average and maximum errors of Equations (6)–(12).

	Average Error	Maximum Error
Equation (6)	0.048187	0.1192
Equation (7)	0.035147	0.11924
Equation (8)	0.1914	0.25718
Equation (9)	0.000606	0.00244
Equation (10)	0.006037	0.018941
Equation (11)	0.006228	0.013326
Equation (12)	0.008038	0.016176

As shown in Table 1, the average error between the Equation (9) proposed in this paper and the sigmoid function is less than 0.001, which best completes the approximation of the sigmoid function.

Table 2 lists the resource utilization rate (Proportion of Slice LUTS used to available Slice LUTS) of Equations (6), (7) and (9) after synthesizing on Xilinx's FPGA XC7Z020CLG400-2. As can be seen from Table 2, the resource utilization rate of Equation (9) is similar to that of Equations (6) and (7). On the regression model that requires higher calculation accuracy, the proposed Equation (9) can make the calculation result of FPGA prediction model closer to the real value, which is a better choice for balancing accuracy and resource utilization.

Table 2. Resource utilization of the sigmoid function with different approximation methods.

	Equation (6)	Equation (7)	Equation (9)
Resource utilization	0.0282%	0.0320%	0.0469%

4.2. Format of Numerical Representation on FPGA

The FPGA architecture uses the signed, two-complement fixed-point representation for numbers. The difference between the encoded number and the target value may also be one of the reasons for the error. Table 2 shows the comparison between the binary number and the target value in the 16-bit ($a = 4, b = 11$) or 21-bit ($a = 4, b = 16$) encoding mode.

Table 3 shows that the 16-bit and 21-bit encoded numbers are almost the same as the target values, proving that the results of the two different encoded numbers are basically equal. In order to save resources, the FPGA-based model uses a 16-bit coding format.

Table 3. Comparison of encoded binary numbers with target values.

Target Value: λ_{target}	Binary Number	Decimal Value for Binary Number: $\lambda_{decimal}$	Relative Error: $\frac{\lvert\lambda_{target}-\lambda_{decimal}\rvert}{\lvert\lambda_{target}\rvert}$
0.294294	0000001001011010	0.293945	0.119%
−1.086123	1111011101010000	−1.085937	0.017%
0.294294	000000100101101010110	0.294281	0.004%
−1.086123	111101110100111110100	−1.086120	0.003%

5. Experimental Results

The regression prediction model based on the FPGA is tested on simulation data set and real industrial data set. The simulation data set consists of 1200 patterns (located in 1-eigenvalue space), of which the training set contains 600 patterns and the test set contains 600 patterns. The real hot-rolled strip crown data set were collected from a 1780 mm hot strip production line of a company in Hebei Province, China. In a hot-rolled process, crown is defined as the difference of thickness between the center and a point 40 mm from the edge of the strip. For strip products, smaller crown is required, which can save materials and reduce costs. Therefore, control of strip crown is a high priority for hot-rolled production process. The crown of the strip is decided by the 3D deformation in finishing mill, it can be regarded as the reflection of the cross-sectional shape of roll gap at the outlet of finishing mill. Thus, all the factors which can affect the cross-sectional shape of roll gap at the outlet of deformation zone are the factors that affect the crown value of strip. In this paper, nine important attributes in hot-rolled strip production are selected as input variables. They are: Cooling water flow of rolling mill (%), Entrance temperature (°C), Exit temperature (°C), Strip width (m), Entrance thickness (mm), Exit thickness (mm), Bending force (kN), Rolling force (kN) and Entry profile (μm). The real hot-rolled strip crown data set contains 474 patterns (located in 9 eigenvalue space), of which the training set contains 380 patterns and the test set contains 94 patterns. All data are normalized within the range of [−1, 1]. The experimental test realizes the regression prediction of the SCN based on the FPGA. In [20], the ELM model implemented on the FPGA is applied to the detection of classification problems. We modified the FPGA-based ELM model in [20] and adopted it to the regression problem. We give the experimental results of the FPGA-based ELM model on the simulation data set and the real data set, and compare it with the proposed FPGA-based SCN model to explain the superiority of the SCN. The FPGA used in the experiment is Xilinx's FPGA XC7Z020CLG400-2.

Figure 3 shows the functional simulation results of the input and output signals of the modules in the FPGA architecture, including: clock signal (net_clk), reset signal (net_reset), pattern selection signal (input_select), an output signal of the *Input* module (Input_out0—an eigenvalue), an output of the *Shifter* module (Shifter1_out0), an output of the *Mux* module (Mux_out0), an output of the *Inverting* module (Inverting_out0), an output of the *Sum* module (Sum_out0), an output of the *activation* module (activation_out0), output enable signal (out0_ready) of the *Mac* module which indicates that the result has been calculated and the final output of the system (out0) in the *Mac* module. Taking the signal in Figure 3 as an example, it represents the calculation process of the data of one pattern in the FPGA, where the value "10'h002" of the Signal *input_select* represents the calculation of the second pattern currently being performed. The calculation process in the FPGA given in Figure 3 has 18 clock cycles

of the Signal *net_clk*, corresponding to the 18 hidden nodes in the SCN model. The Signal *out_ready* in the figure is the flag signal for calculating the data of each pattern. The Signal *out_ready* is set low at the rising edge of the Signal *net_reset*, indicating that this pattern data calculation process starts, and the low level state is always maintained during the calculation process. Then after the calculation is completed (that is to say, after the 18 clock cycles of the Signal *net_clk*), the Signal *out_ready* is set high, which indicates that the pattern data calculation process ends. After the rising edge of the Signal *out_ready* appears, the value "16'h066B" of the Signal *out0* is the prediction result of the second pattern. The specific implementation process of the SCN on the FPGA is as follows: After being triggered by the reset signal (*net_reset*), the *Shifter* module acquires an input vector $[x_1, x_2, \ldots, x_N]$ from the *Input* module. At the rising edge of the next clock, the *Mux* module receives the result $[sh1, sh2, \ldots, shN]$ from the *Shifter* module in parallel. The *Inverting* module processes the data and outputs it according to the signs of the input weights. After another rising edge of the clock, the *Sum* module adds up the data and activates the data through the *activation* module. The *Mac* module accumulates the activated data after another rising edge of the clock. When all data accumulation is completed, the Signal out0_ready generates a rising edge, and then the final output of the system can be read out from the signal out0.

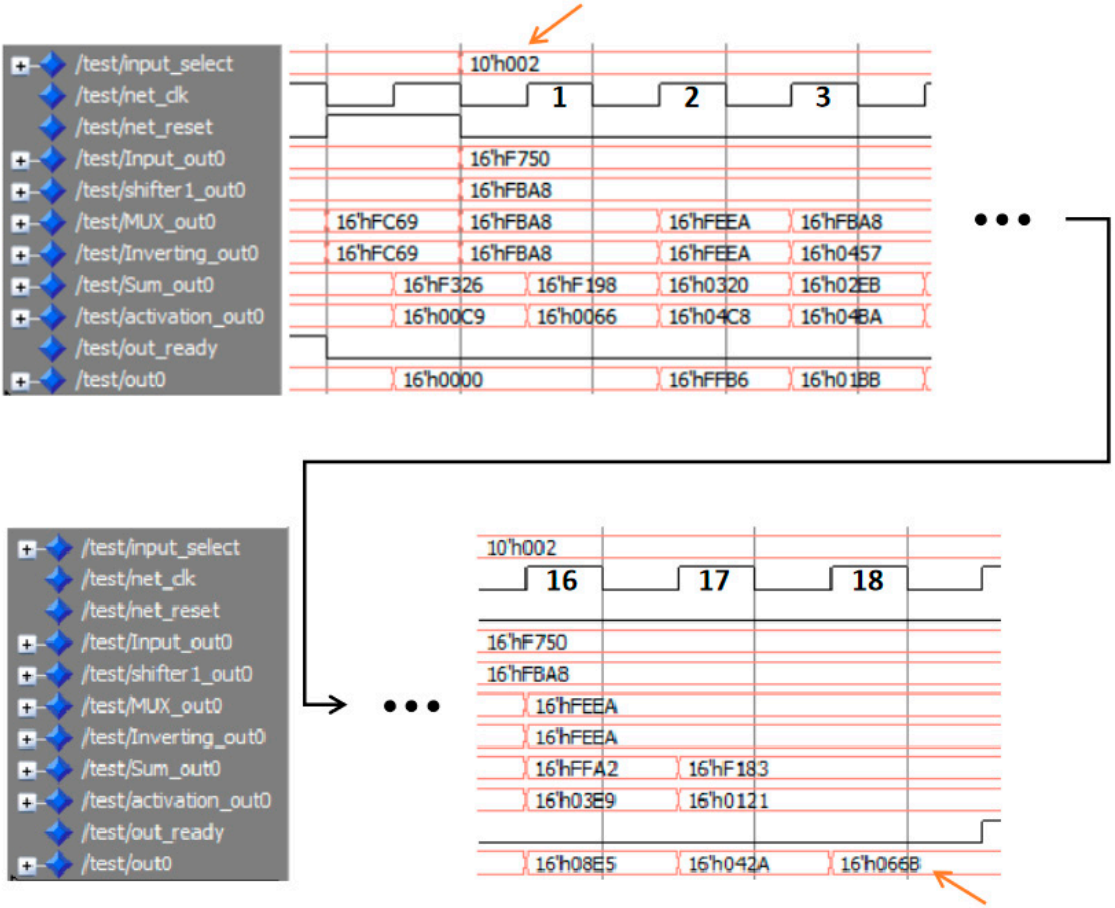

Figure 3. Functional simulation of the field programmable gate array (FPGA)-based implementation of the stochastic configuration network (SCN).

Figures 4 and 5 show the results of FPGA regression prediction model on the simulation data set and real data set. In the simulation data set, the number of hidden layer nodes of the SCN and ELM were 18 and 35 respectively, while in the real data set, the number of hidden layer nodes of the SCN and the ELM were 42 and 55 respectively. As can be seen from Figure 4, for the simulation data set, the SCN model based on Equation (7) and the ELM model can only predict the general trend of the real value, while the SCN model based on Equation (6) cannot predict the trend of the real value,

and only the SCN model based on Equation (9) can predict the real value well. As can also be seen from Figure 5, for the real data set, the SCN model based on Equation (9) has the best prediction result, and the predicted value almost completely coincides with the real value, while the prediction result of other models is relatively poor. Therefore, both the simulation data set and real data set prove that the FPGA implementation of the SCN model based on Equation (9) proposed in this paper has the best prediction performance.

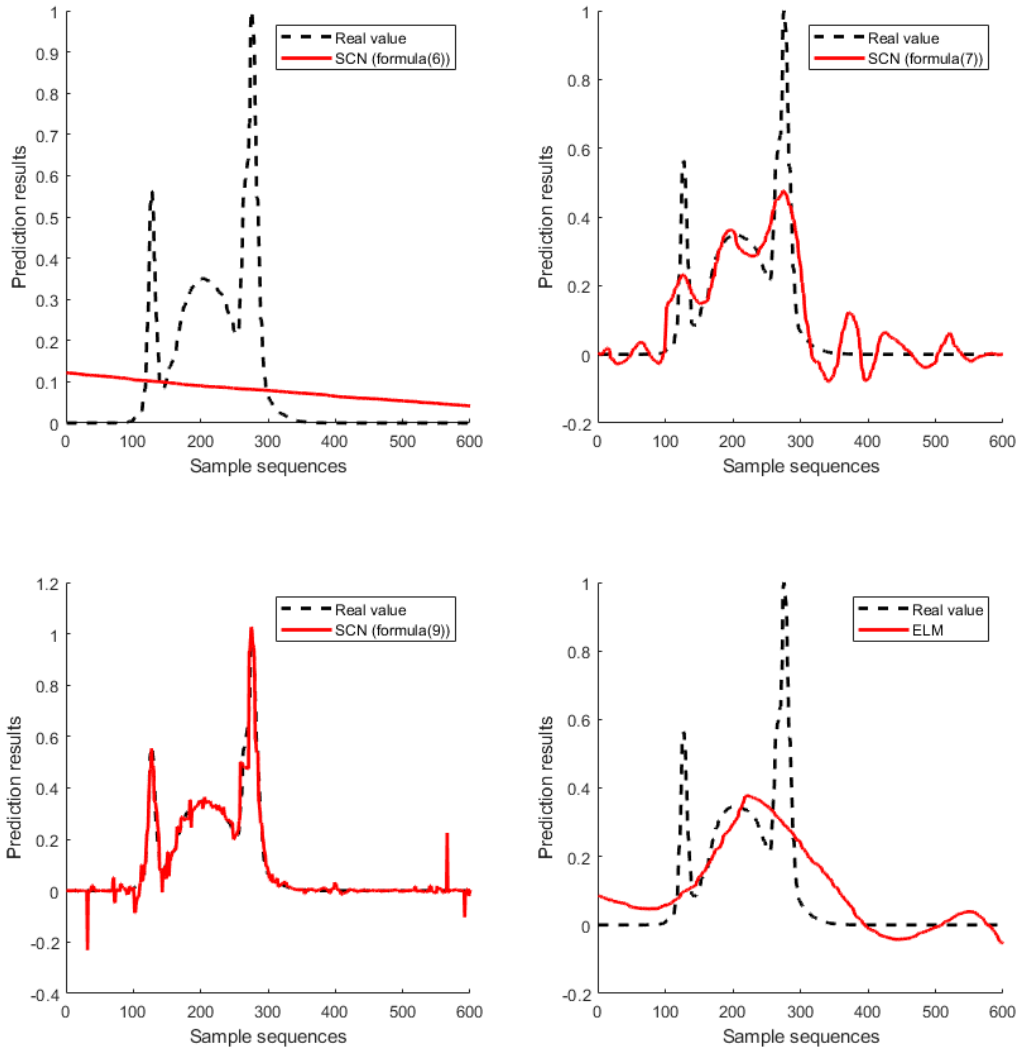

Figure 4. Comparison of SCN (Formulas (6), (7), (9)) and ELM implementation on the simulation data set.

In order to quantitatively analyze the implementation effect of the optimized SCN on the FPGA, Table 4 shows the average values of 30 groups of root-mean-square errors (RMSE) of the implementation results of FPGA-based SCNs and ELM, computer-based SCN and ELM on two data sets. Considering that FPGA is limited by hardware resources and calculation accuracy, and its calculation capability is relatively weak compared with that of a computer, Table 4 takes the prediction results of computer as benchmark, and compares the implementation accuracy of the proposed model on FPGA with that of a computer. As can be seen from Table 4, in the simulated data set, the implementation result of the SCN on the computer is worse than that of the ELM, but in the real data set, the implementation result of the SCN on the computer is better than that of the ELM. Therefore, the advantages of the SCN based on computer implementation are not obvious. Compared with the implementation of the SCN on computer, the prediction accuracy of the SCN implemented on FPGA with Equations (6) and (7) is greatly reduced. Using the sigmoid function proposed in Equation (9), the prediction accuracy of the

SCN implemented on FPGA is obviously the best, which is completely better than the ELM, especially in real data set, and is almost the same as the implementation accuracy of the SCN on the computer. The results fully demonstrate the strong generalization ability and high prediction accuracy of the FPGA-based the SCN proposed in this paper.

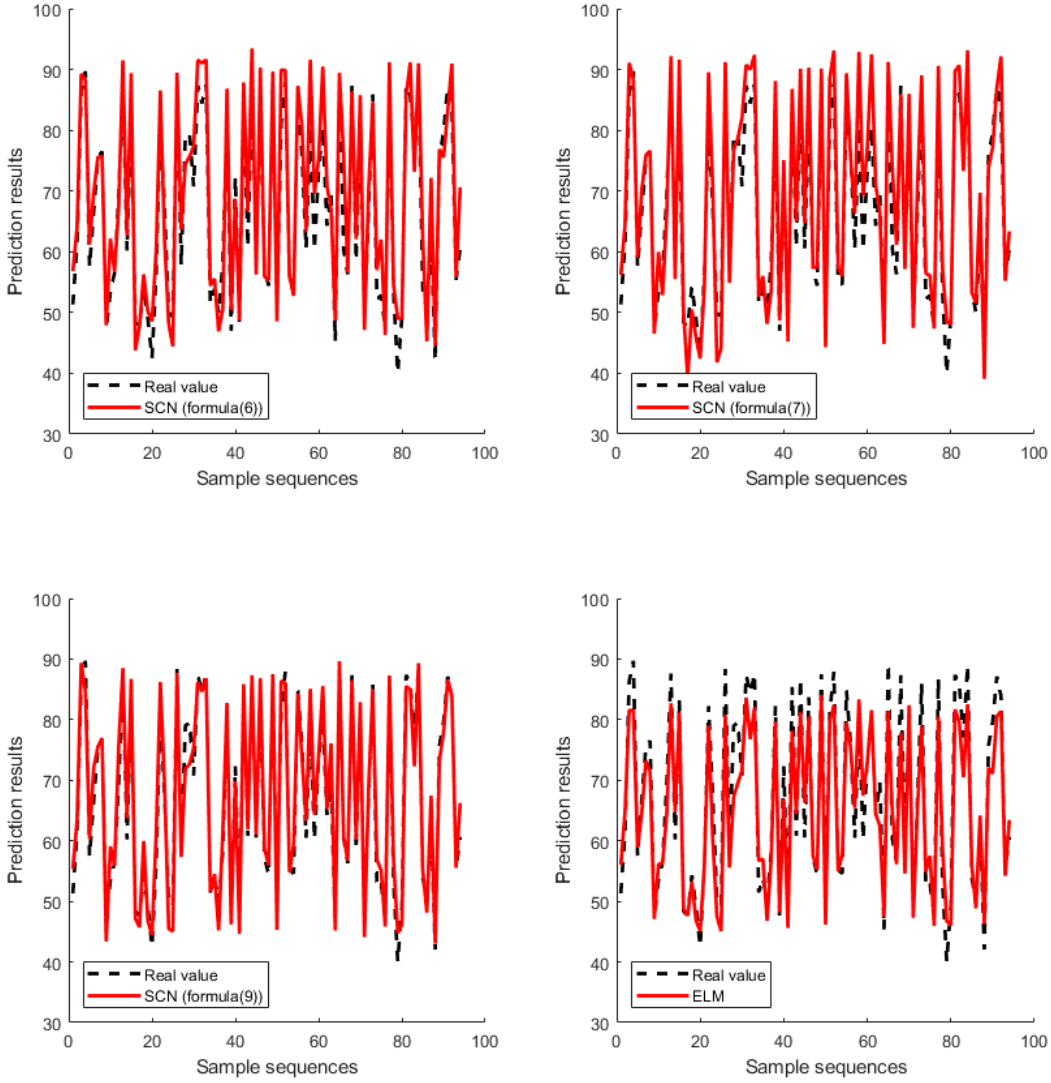

Figure 5. Comparison of SCN (Formulas (6), (7), (9)) and ELM implementation on the real data set.

Table 4. Average root-mean-square error (RMSE) of the prediction results.

	Simulation Data Set	Real Data Set
FPGA-based SCN (Equation (6))	0.1625	4.9169 μm
FPGA-based SCN (Equation (7))	0.1056	4.7567 μm
FPGA-based SCN (Equation (9))	0.0551	3.5783 μm
FPGA-based SCN (Equation (10))	0.0614	3.7821 μm
FPGA-based SCN (Equation (11))	0.0643	3.7994 μm
FPGA-based SCN (Equation (12))	0.0626	3.7187 μm
FPGA-based ELM	0.1263	4.3296 μm
Computer-based SCN	0.0150	3.5404 μm
Computer-based ELM	0.0126	4.2290 μm

Tables 5 and 6 show the resource utilization and power consumption of SCNs and ELM implemented on the FPGA. The analysis of Tables 5 and 6 shows that, whether it is the simulation data set or

the real data set, compared with the ELM, the resource utilization and power consumption of SCN implemented on FPGA is lower. However, the resource utilization and power consumption of SCNs based on different equations are basically the same. In order to analyze the running speed of the SCN model based on the FPGA, Table 7 shows the actual clock frequencies of the SCN and the ELM implemented on the two data sets for comparison. As can be seen from Table 7, the experiments on both data sets prove that the SCN implemented on FPGA runs faster than ELM. The fundamental reason is that the SCN needs fewer hidden layer nodes to reach the optimal solution, therefore, FPGA-based implementation of the SCN has lower resource utilization and power consumption and faster operation speed than the ELM.

Table 5. Resource utilization of FPGA-based prediction models.

	SCN (Equation (6))	**SCN (Equation (7))**	**SCN (Equation (9))**	**ELM**
Simulation data set	2.29%	2.30%	2.32%	2.48%
Real data set	1.65%	1.66%	1.68%	2.14%

Table 6. Power consumption of FPGA-based prediction models.

	SCN (Equation (6))	**SCN (Equation (7))**	**SCN (Equation (9))**	**ELM**
Simulation data set	0.991 W	0.991 W	0.991 W	0.993 W
Real data set	1.039 W	1.039 W	1.039 W	1.043 W

Table 7. Actual clock frequency of FPGA-based prediction models.

	SCN (Equation (9))	**ELM**
Simulation data set	43.1 MHz	31.7 MHz
Real data set	48.6 MHz	41.8 MHz

6. Conclusions

This paper adopts the SCN to regression analysis, and verifies the FPGA framework to implement the SCN model. Based on the balance between prediction accuracy and resource utilization, this paper cuts in from multiple perspectives to improve the performance of the SCN model based on the FPGA. The implementation of the activation function on the FPGA is optimized, which effectively improves the prediction accuracy while considering the utilization rate of hardware resources. Experimental results based on simulation and real data sets prove that the proposed FPGA framework successfully implements the SCN regression prediction model, and the improved SCN model has higher accuracy and more stable performance. Compared with other implementations in the literature [20], the prediction performance of the proposed the SCN implementation model based on the FPGA for simulation data set and real data set is improved by 56.37% and 17.35% respectively. In addition to the prediction accuracy, this paper also compares and analyzes the experimental results from the aspects of resource utilization rate, power consumption and running speed. The experimental results fully demonstrate the performance advantages of the FPGA-based SCN implementation architecture proposed in this paper. Some studies have shown that the SCN model is not only used for regression prediction but also can be used for recognition and detection. In the future, we can extend the FPGA-based SCN prediction model to more complex scenes. The use of FPGA-based SCN model will be of great significance for edge computing and alleviating the computing pressure of the cloud computing center.

Author Contributions: Conceptualization, funding acquisition as well as supervision of this research project were done by F.L. and X.L.; the experimental investigations were carried out by Y.G. and J.P. with the support of F.L. and Y.H.; Y.G. took the lead in writing the manuscript; F.L. and X.L. revised the paper. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Key R&D Program of China (No. 2017YFB0304100), National Natural Science Foundation of China (No. 51634002), and the Fundamental Research Funds for the Central Universities (No. N180708009).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Igelnik, B.; Pao, Y.H. Stochastic choice of basis functions in adaptive function approximation and the functional-link net. *IEEE Trans. Neural Netw.* **1995**, *6*, 1320–1329. [CrossRef] [PubMed]
2. Tian, Q.; Yuan, S.J.; Qu, H.Q. Intrusion signal classification using stochastic configuration network with variable increments of hidden nodes. *Opt. Eng.* **2019**, *2*, 026105. [CrossRef]
3. Qu, H.Q.; Feng, T.L.; Zhang, Y.; Wang, Y.P. Ensemble Learning with Stochastic Configuration Network for Noisy Optical Fiber Vibration Signal Recognition. *Sensors* **2019**, *19*, 3293. [CrossRef] [PubMed]
4. Cai, K.W.; Alalibo, B.P.; Cao, W.P.; Liu, Z.; Wang, Z.Q.; Li, G.F. Hybrid Approach for Detecting and Classifying Power Quality Disturbances Based on the Variational Mode Decomposition and Deep Stochastic Configuration Network. *Energies* **2018**, *11*, 3040. [CrossRef]
5. Dai, W.; Li, D.P.; Zhou, P.; Chai, T.Y. Stochastic configuration networks with block increments for data modeling in process industries. *Inf. Sci.* **2019**, *484*, 367–386. [CrossRef]
6. Wang, D.; Li, M. Stochastic configuration networks: Fundamentals and algorithms. *IEEE Trans. Cybern.* **2017**, *47*, 3466–3479. [CrossRef]
7. Sheng, Z.; Zheng, Z.; Qu, H.; Li, W. Fiber Intrusion Signal Recognition Algorithm Based on Stochastic Configuration Network. *Laser Optoelectron. Prog.* **2019**, *14*, 140602. [CrossRef]
8. Li, W.T.; Tao, H.; Li, H.; Chen, K.Q.; Wang, J.P. Greengage grading using stochastic configuration networks and a semi-supervised feedback mechanism. *Inf. Sci.* **2019**, *488*, 1–12. [CrossRef]
9. Li, W.; Zeng, Z.Q.; Qu, H.Q.; Sun, C.B. A Novel Fiber Intrusion Signal Recognition Method for OFPS Based on SCN with Dropout. *J. Lightwave Technol.* **2019**, *20*, 5221–5230. [CrossRef]
10. Dai, W.; Li, D.P.; Chen, Q.X.; Chai, T.Y. Data driven particle size estimation of hematite grinding process using stochastic configuration network with robust technique. *J. Cent. South Univ.* **2019**, *1*, 43–62. [CrossRef]
11. Lindsey, C.; Lindblad, T. Review of hardware neural networks: A user's perspective. In Proceedings of the Third Workshop on Neural Networks: From Biology to High Energy Physics, Isola d'Elba, Italy, 26–30 September 1994; pp. 195–202.
12. Moore, G.E. Cramming more components onto integrated circuits. *Proc. IEEE* **1998**, *86*, 82–85. [CrossRef]
13. Lopes, F.F.; Ferreira, J.C.; Fernandes, M.A.C. Parallel implementation on FPGA of support vector machine using stochastic gradient descent. *Electronics* **2019**, *9*, 631. [CrossRef]
14. Mishra, J.; Saha, I. Artificial neural networks in hardware: A survey of two decades of progress. *Neurocomputing* **2010**, *74*, 239–255. [CrossRef]
15. Karras, K.; Pallis, E.; Mastorakis, G.; Nikoloudakis, Y.; Batalla, J.M.; Mavromoustakis, C.X.; Markakis, E. A Hardware Acceleration Platform for AI-Based Inference at the Edge. *Circuits Syst. Signal Process.* **2020**, *39*, 1059–1070. [CrossRef]
16. Zhu, Z.; Zhang, J.; Zhao, J.; Cao, J.; Zhao, D.; Jia, G.; Meng, Q. Hardware and Software Task-Scheduling Framework Based on CPU+FPGA Heterogeneous Architecture in Edge Computing. *IEEE Access* **2019**, *7*, 148975–148988. [CrossRef]
17. Huang, G.B.; Wang, D.H.; Lan, Y. Extreme learning machines: A survey. *Int. J. Mach. Learn. Cybern.* **2011**, *2*, 107–122.
18. Huang, G.; Huang, G.B.; Song, S.; You, K. Trends in extreme learning machines: A review. *Neural Netw.* **2015**, *61*, 32–48. [CrossRef]
19. Ouyang, T.; Wang, C.; Yu, Z.; Stach, R.; Mizaikoff, B.; Liedberg, B.; Huang, G.B.; Wang, Q.J. Quantitative Analysis of Gas Phase IR Spectra Based on Extreme Learning Machine Regression Model. *Sensors* **2019**, *19*, 5535. [CrossRef]
20. Decherchi, S.; Gastaldo, P.; Leoncini, A.; Zunino, R. Efficient digital implementation of extreme learning machines for classification. *IEEE Trans. Circuits Syst. II Express Briefs* **2012**, *59*, 496–500. [CrossRef]
21. Ragusa, E.; Gianoglio, C.; Gastaldo, P.; Zunino, R. A digital implementation of extreme learning machines for resource-constrained devices. *IEEE Trans. Circuits Syst. II Express Briefs* **2018**, *65*, 1104–1108. [CrossRef]

22. Safaei, A.; Wu, Q.M.J.; Akilan, T.; Yang, Y. System-on-a-Chip (SoC)-Based Hardware Acceleration for an Online Sequential Extreme Learning Machine (OS-ELM). *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2019**, *38*, 2127–2138. [CrossRef]
23. Wang, D.; Li, M. Robust stochastic configuration networks with kernel density estimation for uncertain data regression. *Inf. Sci.* **2017**, *412–413*, 210–222. [CrossRef]
24. Li, M.; Wang, D. 2-D stochastic configuration networks for image data analytics. *IEEE Trans. Cybern.* **2019**. [CrossRef] [PubMed]
25. Wang, D.; Cui, C. Stochastic configuration networks ensemble with heterogeneous features for large-scale data analytics. *Inf. Sci.* **2017**, *417*, 55–71. [CrossRef]
26. Bassoli, M.; Bianchi, V.; Munari, I.D. A Model-Based Design Floating-Point Accumulator. Case of Study: FPGA Implementation of a Support Vector Machine Kernel Function. *Sensors* **2020**, *20*, 1362. [CrossRef] [PubMed]
27. De Souza, A.C.D.; Fernandes, M.A.C. Parallel Fixed Point Implementation of a Radial Basis Function Network in an FPGA. *Sensors* **2014**, *14*, 18223–18243. [CrossRef]
28. Tommiska, M.T. Efficient digital implementation of the sigmoid function for reprogrammable logic. *Proc. Inst. Electr. Eng. Comput. Digit. Technol.* **2003**, *150*, 403–411. [CrossRef]
29. Zhang, M.; Vassiliadis, S.; Delgago-Frias, J.G. Sigmoid generators for neural computing using piecewise approximations. *IEEE Trans. Comput.* **1996**, *45*, 1045–1049. [CrossRef]
30. Panicker, M.; Babu, C. Efficient FPGA Implementation of Sigmoid and Bipolar Sigmoid Activation Functions for Multilayer Perceptrons. *IOSR J. Eng.* **2012**, *6*, 1352–1356. [CrossRef]
31. Khodja, D.E.; Simard, S.; Beguenan, R. Implementation of Optimized Approximate Sigmoid Function on FPGA Circuit to use in ANN for Control and Monitoring. *Control Eng. Appl. Inform.* **2015**, *2*, 64–72.
32. Ngah, S.; Baker, R.A.; Embong, A.; Razali, S. Two-steps Implementation of Sigmoid Function for Artificial Neural Network in Field Programmable Gate Array. *ARPN J. Eng. Appl. Sci.* **2016**, *7*, 4882–4888.