ARQII Publication
http://arqiipubl.com/ams
APPLICATIONS OF
MODELLING AND SIMULATION
eISSN 2600-8084
VOL 7, 2023, 78-84

Configurable Model for Sigmoid and Hyperbolic Tangent Functions

Khaled Salah*, Mona Safar, Mohamed Taher and Ashraf Salem

Computer and Systems Engineering Department, Ain Shams University, Cairo, Egypt

*Corresponding author: khaledsalaheldin.h@gmail.com

Submitted 05 May 2023, Revised 06 June 2023, Accepted 20 June 2023, Available online 02 July 2023.

Abstract: Recurrent neural networks (RNNs) are considered to be among the most important types of neural networks especially for the applications where processing of a sequence of data comes to place. RNNs are in general computationally expensive and need a lot of processing time and power. Therefore, there is a strong need to reduce the processing time to be able to use them in an embedded environment with limited resources. In this work, we present an accelerated field programmable gate array (FPGA) model for RNNs with an emphasis on long short-term memory neural networks (LSTMs). A new configurable block capable of calculating Tanh and Sigmoid activation functions is proposed and analyzed. The solution is based on a look-up table and additional simple math operations, which leads to a speedup of the proposed model of the neural network. Results are obtained and compared with other work by the simulation tool ISE Xillinx.

Keywords: Activation functions; Deep learning; FPGA; LSTM; Recurrent neural network.

1. INTRODUCTION

A Long Short-Term Memory (LSTM) Network is a type of recurrent neural networks (RNN) that is currently being used in many applications in place of the ordinary RNNs [1]. The key feature of the LSTM is that it is able to forget and memorize some parts of the output of the previous steps of the network. This feature helps to overcome the problems of the vanishing and exploding gradients which are present in RNNs [2]. Networks with multiple layers and multiple steps like RNNs suffer from the vanishing gradients problem [2]. In fact, when gradients smaller than unity are multiplied together and propagated throughout the whole network, these lead to an overall gradient that is approximately equal to zero and consequently the network will not be able to learn anymore. Similarly for the exploding gradients problem, when gradients larger than unity are multiplied together throughout the whole network, lead to an overall gradient that lies in the saturation region of the nonlinear activation functions [2]. Consequently, different values of input will always produce the same output and the network will not be able to learn different input patterns. LSTM solved these problems by having different controlling gates that manipulate the input and the output from previous steps as will be discussed in detail in Section 2.

Neural networks with multiple layers using linear activation functions can be reduced to a network with a single layer using another linear function. In this respect, the network functions as an artificial neural networks (ANNs) with a single layer incapable of non-trivial learning [3]. Therefore, nonlinear activation functions are essential for the functionality of deep neural networks. Choosing a specific type of activation function affects the performance of the deep neural networks [4][5]. Sigmoid and hyperbolic tangent (Tanh) functions are widely used in deep neural networks [6]. However, the calculation of these functions is computationally expensive as these functions contain the exponent term e^{-x} that is more complicated than the simple mathematical operations like the addition and multiplication functions that are simply implemented using ALUs in the FPGAs, CPU or GPU. Heterogeneous computing is also widely used where an FPGA is used for accelerating the computations and a CPU is used for pre-processing and post-processing [7].

Many improvements have been made to accelerate the LSTM neural network on FPGAs as it is considered to be one of the state-of-the-art architectures of deep learning. These improvements are categorized into two main types. The first type aims to reduce the size of the neural networks by means of quantization and binarization and the second type aims to improve the operations used to calculate the equations of the LSTMs. In our work, we focus only on the improvements that extend the second category of improvements.

Previous studies were made in the first category to improve the process of multiplication of matrices with vectors. A compression method based on block-circulant matrices for LSTMs, which aims to lower the computational complexity and memory footprint without abnormalities in computation and memory access, was introduced in [8]. Another hardware-oriented compression approach which combines structural top-k pruning, clipped gating, and multiplication-free quantization was introduced in [9]. To reduce the size of the model and the number of matrix operations, the effect of binarization was discussed

in [10], and this is achieved by two types of compression. The first type is parameter reduction by reducing the number of parameters being used in the model inference. The second type is parameter quantization by reducing the number of bits used to represent the model. In [11], sparse input encoding technique, where most columns of weight matrix are zeros and only non-zero weights are stored in memory, is combined with the improvement of matrix vector multiplication. The systolic array algorithm combined with the matrix blocking algorithm was implemented in [12], where large matrix multiplications are computed efficiently on an FPGA board.

For the second category of improvements, previous studies were based on piecewise linear approximation of the Tanh function [13]. Also, second order polynomial functions approximations were used in [14], while general polynomial approximations were used in [15]. Look up table is a common implementation technique of complex mathematical functions. It is used in implementation of Tanh and Sigmoid functions on FPGAs, and modifications were made in this area by reducing the size of the lookup table. In [12], studying the characteristics of both functions was made, and the usage of comparators to check that the input is in a certain range was presented. A mixture between piecewise linear interpolation and look up tables was used in [16], where two rough linear segments are used for approximation and the difference between the actual value and the approximated value is calculated and stored in a lookup table. Whenever a new input value comes, the output value will be the value corresponding to the line segment then the stored error in the lookup table will be added to get the final output. Another approach was to get approximations of the first order derivative of the function and then the final output will be the integration of this approximated derivative [17].

In this paper, we focus our studies on improving the calculation of nonlinear functions (Sigmoid and Tanh functions) used in the LSTM networks on FPGAs by trying to simplify the complexity of these activation functions which is considered to be an extension to the improvements made in the second category of improvements. The paper is organized as follows. Section 2 provides background for LSTM neural networks. Section 3 demonstrates the methodology behind the proposed configurable block. Section 4 presents the experimental results of the implemented proposed design. Finally, Section 5 concludes the paper.

2. BACKGROUND

LSTM neural networks were first introduced in [1]. The main motivation behind LSTM was to overcome the problems of vanishing and exploding gradients that were present in the vanilla RNN [18]. There are a lot of variations of LSTM networks that came after the original version, one of them with peepholes [19] where there are connections from the memory cell to the gates. The addition of these peepholes helps the cell to control the gates. However, using peepholes in LSTM increases the complexity with minimal improvement in performance [20]. In this paper we deal with the vanilla LSTM [1][14] that keeps performing well with proposed reduction in unnecessary complexity.

The LSTM network deals with problems where processing of sequence is important, every step in a layer of the network can be presented by a LSTM cell as shown in Figure 1. The inputs to the LSTM cell at time step *t* are the input feature vector x_t as well as the output vector h_{t-1} from the previous time step $t-1$, and the output of the cell is the output vector h_t. The LSTM cell can be represented by the following six equations which show the contribution of the input feature vector to the final output. The equations also show how the cell is capable of memorizing and forgetting the input information by multiplying the features by certain weights W_s and by having multiple controlling gates [14]. These equations represent the gates that are the key difference between the LSTM and the Vanilla RNN as follows:

- Input Gate (i_t): Given by Equation (1), this gate controls the influence of the features from the input vector x_t using the weight matrix W_{ix} as well as the influence of the output from the previous time step h_{t-1}.
- Forget gate (f_t): Given by Equation (2), this gate controls the memory cell c_t in Equation (5).
- Output Gate (o_t): Given by Equation (3), this gate controls the activated output from the whole LSTM cell h_t in Equation (6).
- Candidate cell gate (g_t): Given by Equation (4), this gate controls the memory cell c_t by manipulating the input vector x_t in Equation (5).
- Memory Cell (c_t): Given by Equation (5), this is the key gate in the LSTM cell where the previous state is stored.
- Output Activation (h_t): Given by Equation (6), the output of this gate is the final output of the LSTM cell.

$$i_t = \text{Sigmoid}(W_{ix}x_t + W_{ih}h_{t-1} + b_i) \tag{1}$$

$$f_t = \text{Sigmoid}(W_{fx}x_t + W_{fh}h_{t-1} + b_f) \tag{2}$$

$$o_t = \text{Sigmoid}(W_{ox}x_t + W_{oh}h_{t-1} + b_o) \tag{3}$$

$$g_t = \text{Tanh}(W_{gx}x_t + W_{gh}h_{t-1} + b_g) \tag{4}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \tag{5}$$

$$h_t = o_t \odot \text{Tanh}(c_t) \tag{6}$$

In order to implement the LSTM cell either on FPGA or on a GPU, the hardware used should be capable of calculating these equations. There are dependencies between these equations which should be taken into consideration during the implementation of the LSTM cell. These dependencies affect the pipelining and parallelization of these equations. Figure 2 is a dependency graph which shows that some equations must be calculated first in order to obtain the correct result. Assume that each equation takes 1 clock cycle to obtain its own result. At clock cycle 1, i_t, f_t, o_t and g_t can be calculated simultaneously

without depending on any other result. At clock cycle 2, c_t can be calculated as c_t depends on i_t and f_t. Only at clock cycle 3 h_t can be calculated as it depends on c_t which will not be available until the end of the second clock cycle.

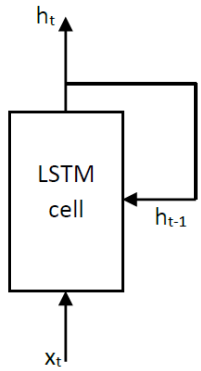

Figure 1. Representation of LSTM cell

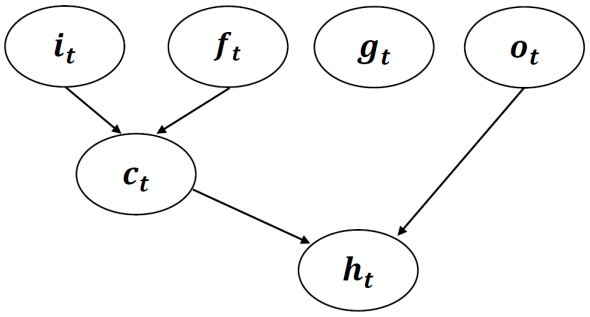

Figure 2. Dependencies of LSTM equations

3. METHODOLOGY

In this paper, we introduce the idea of having a configurable block that is capable of calculating the Sigmoid and Tanh activation functions, and this is possible due to the similarities between the two activation functions as they are both non-linear activation functions with the exponent term e^{-x}. In our design, we used floating point representation for dealing with numbers that are being used in the calculations of the deep neural network. They are very common when implementing a design on FPGAs because of their efficiencies. The most common type of floating point representation is the standard IEEE745, but at some point we want to have a fixed-point representation of numbers to be able to access the look up table memory for the Sigmoid function. Therefore, we reused already implemented blocks for conversion of numbers from the floating point to the fixed-point representation before accessing the lookup table then converting them back to floating point representation if needed. A 12-bit fixed point representation is commonly used in state-of-the-art solutions [12]. However, we found that using fixed-point (5,6) representation for the input of the Sigmoid LUT leads to unused bits where 5 bits are used to represent the integer part of the number which gives total range of [-31.984375 31.984375]. Most of this range lies in the saturation region of the Sigmoid function so we will not benefit from most of the bits. In our work we use 10-bit fixed point (3,6) representation where 1 bit is used for sign, 3 bits for the integer part and 6 bits for the decimal part. This representation gives a range [-7.984375 7.984375] with resolution 0.015625 (1/64) and total number of entries of the look up table equals to 1024 entries. The proposed representation uses less bits than fixed point representation in [12] which will save resources of the FPGA while having the same resolution which will lead to the same error.

Using smaller resolution leads to a smaller error in the calculation of the desired functions. Unfortunately, this will lead to having more bits to represent one number and as a result it will cost more to store the output values for all the input values for the desired function. Consequently, the total area of the design will increase, so we must make a trade-off between the error for the output and the total area being used.

By plotting the two activation functions on the same graph in Figure 3, we see great similarities between the two functions. Both functions are monotonic functions, and both functions saturate after different thresholds from the negative and positive directions. Starting from the mathematical definition for each activation function, we can show that both activation functions have a common mathematical structure.

$$\mathrm{Tanh}(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \quad \text{and} \quad \sigma(x) = \frac{1}{1+e^{-x}}$$

First, we notice that

$$\sigma(-x) = \frac{1}{1+e^{x}} = \frac{e^{-x}}{1+e^{-x}} = 1 - \sigma(x) \tag{7}$$

and

$$\mathrm{Tanh}(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} = 1 - \frac{2e^{-x}}{e^{x}+e^{-x}}$$

Thus,

$$\mathrm{Tanh}(x) = 1 - \frac{2}{1+e^{2x}} = 1 - 2\sigma(-2x) \tag{8}$$

From Equations (7) and (8), we finally conclude that,

$$\mathrm{Tanh}(x) = 2\,\sigma(2x) - 1$$

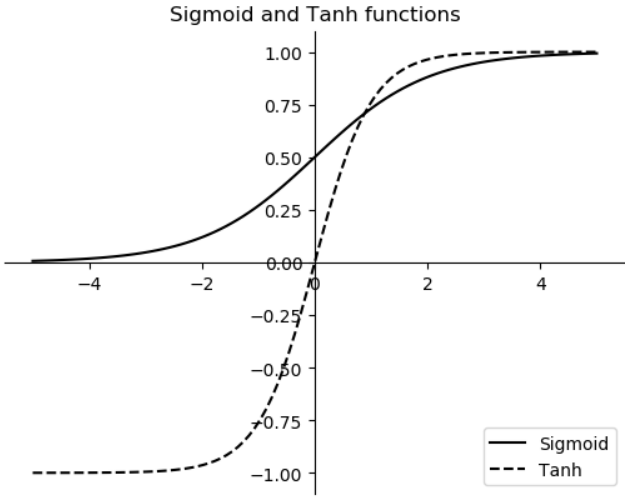

Figure 3. Sigmoid and Tanh functions plot

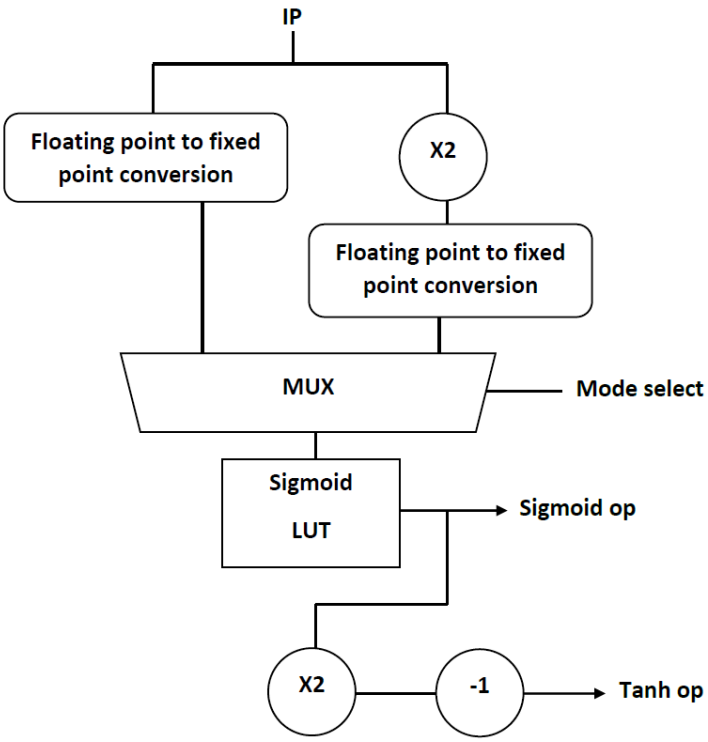

Figure 4. Configurable block for calculation of Sigmoid and Tanh functions

The final formula shows that, for the same input value it is possible to get the value of the Tanh function after doing some simple processing on the output of the Sigmoid function. As a result, we can have a common look up table for saving the values of the Sigmoid function for different inputs and be able to reconstruct the values for the Tanh function as well as shown in Figure 4. Both functions saturate at a certain point. Consequently, we can save the values for the lookup table for a certain range only as follows:

$$\text{Sigmoid}(x) = \begin{cases} 0 & x < -6 \\ 1 & x > 6 \\ LUT_{sigmoid}(x) & -6 < x < 6 \end{cases}$$

$$\text{Tanh}(x) = \begin{cases} -1 & x < -3 \\ 1 & x > 3 \\ 2(LUT_{sigmoid}(2x)) - 1, & -3 < x < 3 \end{cases}$$

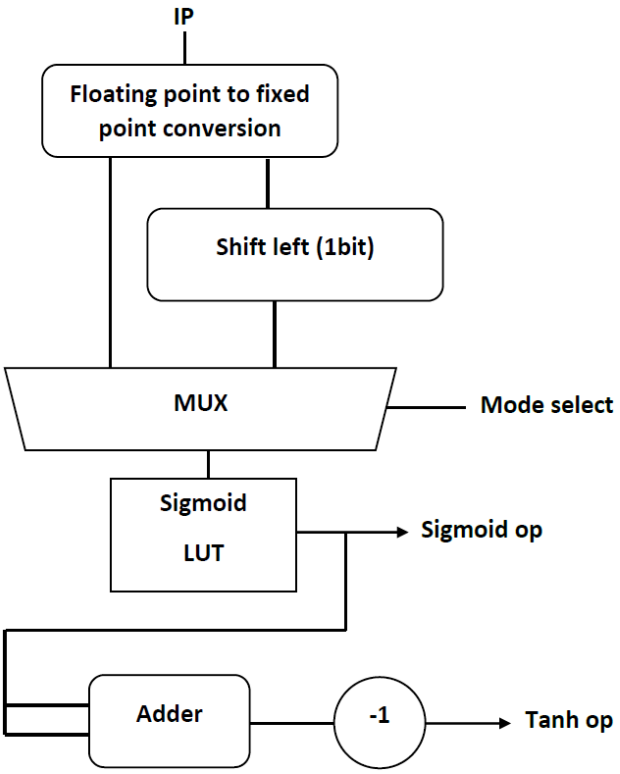

Figure 5. The configurable block after removal of multipliers

Still there is an inconvenience with this design. The usage of the multipliers in the design introduces more complexity to the design and makes it inefficient by using more area after the synthesis of the design and by consuming more time doing these calculations. As a result, we need to optimize this design to make it more efficient. Since both multipliers multiply an input value with a constant, we can replace the multipliers with adders. This can be formulated in a mathematical relation: $Y = 2 * X$ is equivalent to $Y = X + X$. We also notice that multiplying a binary number in the fixed-point representation by 2 is equivalent to shifting the number by one bit. Therefore, one multiplier is replaced by an adder where the two inputs of the adder are the same input in the floating-point format, and the other multiplier is replaced by shifting the number in the fixed-point format by one bit. The improved design is shown in Figure 5 after the replacement of the multipliers. The mode select bit in the design is used to select between the inputs for the LUT and based on the value of this bit, the output will be the Sigmoid or Tanh function. This block is a configurable block to calculate Sigmoid or Tanh function by using this select bit.

4. EXPERIMENTAL RESULTS AND VALIDATION

The main goal is to prove that the proposed configurable block consumes less resources after the synthesis of the design to make sure that that the proposed model has a smaller area than ordinary models. Having a smaller area will increase the capability of integrating more models on the same FPGA which will reduce the total cost and will save time. In our experiments, we used the simulation tool ISE Xillinx which is widely used in the synthesis of VHDL and Verilog designs. In order to compare between results, we choose a fixed evaluation board and in order to isolate the effect of having different boards, we chose the board from Zync family which is xc7z010-3clg400 in the comparison between the different four designs: the configurable block, the configurable block after removal of multipliers, the usage of polynomial approximations [12][15] and the design with dual LUTs (one for Sigmoid function and the other for the Tanh function)[12]. These are shown in Table 1.

The results show that the proposed configurable block reduces the number of the used FFs and LUTs which saves a lot of area of the FPGA to be used for other calculations. The latency in the proposed block is less than the latency when using polynomial approximations. It should be mentioned here that for specific applications, the configurable block cannot be used for calculating Sigmoid and Tanh functions at the same time. This can be overcome by taking into consideration the dependencies of the output of the activation functions for different neurons in the neural network. Tables 1 and 2 demonstrate the enhancement obtained by using an adder and a shift instead of multipliers, as expected.

Table 1. Used resources in the proposed design

Design	Proposed configurable block	Proposed configurable block with no multipliers	Polynomial approximations [12][15]	Dual LUTs [12]
Number of Slice Registers FF	1052	589	2036 + 1382	1429 + 891
Number of Slice LUTs	2501	1620	4287 + 3003	2224 + 1608
Latency (clk)	25	20	max (22,16)	max (15,10)
Resources reduction	6.16X	10.3X	1X	3.37X

Table 2. Used DSPs and the delay in the proposed design

Design	Proposed configurable block	Proposed configurable block with no multipliers	Polynomial approximations [12][15]	Dual LUTs [12]
Number of DSP48E1s	8	2	8	6
Max delay (ns)	1.361	1.343	1.746	1.612
Speedup	1.28X	1.3X	1X	1.08X

5. CONCLUSION

Deep neural networks such as recurrent neural networks contain a lot of heavy calculations of nonlinear functions such as the Sigmoid and the Tanh functions. Having a design with many of these calculations can be expensive and some FPGAs cannot be a good fit for a large-scale application. This work deals with LSTMs and their implementation on FPGA for their acceleration, in particular the implementation of Tanh and Sigmoid activation functions by means of a configurable block that uses a lookup table and additional simple math operations. The results of having this configurable block have been presented and analyzed showing how the area of the design implementation can be saved on the FPGA board by reduction in the number of slice registers used by 3.94X and reduction in the number of slice LUTs used by 2.36X, which leads to an overall area reduction by 6.3X according to Table 1. Results in Table II show that the proposed design when removing multipliers achieves a 1.3X speedup compared with the polynomial approximations and a 1.2X speedup compared with dual LUTs. The number of DSPs used are reduced significantly when multipliers are removed, as expected.

DECLARATION OF CONFLICTING INTERESTS

The authors declare no potential conflicts of interest with respect to the research and publication of this article.

FUNDING

The authors receive no financial support for the research, authorship, and publication of this article.

REFERENCES

[1] S. Hochreiter and J. Schmidhuber, Long short-term memory, *Neural Computation*, 9(8), 1997, 1735-1780.
[2] Zachary Chase Lipton, A critical review of recurrent neural networks for sequence learning, arXiv: 1506.00019, 2015.
[3] Shiv Ram Dubey, Satish Kumar Singh and Bidyut Baran Chaudhuri, Activation functions in deep learning: A comprehensive survey and benchmark, *Neurocomputing*, 503, 2022, 92-108.
[4] A. D. Rasamoelina, I. Cík, P. Sincak, M. Mach and L. Hruška, A large-scale study of activation functions in modern deep neural network architectures for efficient convergence, *Inteligencia Artificial*, 25(70), 2022, 95-109.
[5] A. D. Jagtap and G. E. Karniadakis, How important are activation functions in regression and classification? A survey, performance comparison,and future directions, arXiv: 2209.02681v6, 2022.
[6] Tomasz Szandała, Review and comparison of commonly used activation functions for deep neural networks, *Bio-inspired Neurocomputing*, 2021, 203-224.
[7] B. Liang, S. Wang, Y. Huang, Y. Liu and L. Ma, F-LSTM: FPGA-based heterogeneous computing framework for deploying LSTM-based algorithms, *Electronics*, 12, 2023, 1139.
[8] S. Wang, Z. Li, C. Ding, B. Yuan, Q. Qiu, Y. Wang and Y. Liang, C-LSTM: Enabling efficient LSTM using structured compression techniques on FPGAs, *ACM/SIGDA International Symposium on Field-programmable Gate Arrays*, California, USA, 2018.
[9] W. Meiqi, W. Zhisheng, L. Jinming, L. Jun and W. Zhongfeng, E-LSTM: An efficient hardware architecture for long short-term memory, *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(2), 2019, 280-291.
[10] T. Mealey and T. M. Taha, Accelerating inference in long short-term memory neural networks, *IEEE National Aerospace and Electronics Conference*, Dayton, USA, 2018, 382-390.
[11] S. Wang, P. Lin, R. Hu, H. Wang, J. He, Q. Huang and S. Chang, Acceleration of LSTM with structured pruning method on FPGA, *IEEE Access*, 7, 2019, 62930-62937.
[12] J. He, D. He, Y. Yang, J. Liu, J. Yang and S. Wang, An LSTM acceleration engine for FPGAs based on caffe framework, *IEEE 5th International Conference on Computer and Communications (ICCC)*, Chengdu, China, 2019.

[13] K. Basterretxea, J. M. Tarela and I. Del Campo, Approximation of sigmoid function and the derivative for hardware implementation of artificial neurons, *IEE Proceedings - Circuits, Devices and Systems*, 151(1), 2004, 18-24.
[14] K. Chen, L. Huang, M. Li, X. Zeng and Y. Fan, A compact and configurable long short-term memory neural network hardware architecture, *25th IEEE International Conference on Image Processing (ICIP)*, Athens, Greece, 2018, 4168-4172.
[15] J. C. Ferreira and J. Fonseca, An FPGA implementation of a long short-term memory neural network, *IEEE International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, Cancun, Mexico, 2016, 1-8.
[16] A. H. Namin, K. Leboeuf, R. Muscedere, H. Wu and M. Ahmadi, Efficient hardware implementation of the hyperbolic tangent sigmoid function, *IEEE International Symposium on Circuits and Systems*, Taipei, Taiwan, 2009.
[17] C. W. Lin and J. S. Wang, A digital circuit design of hyperbolic tangent sigmoid function for neural networks, *Proceedings of IEEE International Symposium on Circuits and Systems*, Seattle, USA, 2008, 856-859.
[18] J. Schmidhuber, Deep learning in neural networks: An overview, *Neural Networks*, 61, 2015, 85-117.
[19] F. A. Gers and J. Schmidhuber, Recurrent nets that time and count, *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Network (IJCN-N)*, Como, Italy, III, 2000, 189-194.
[20] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink and J. Schmidhuber, LSTM: A search space odyssey, *IEEE Transactions on Neural Networks and Learning Systems*, 28(10), 2017, 2222-2232.