Calculation of the Sigmoid Activation Function in FPGA Using Rational Fractions

Pavlo Serhiienko[1[0000-0003-3030-0074]], Anatoliy Sergiyenko[1[0000-0001-5965-1789]], Sergii Telenyk[2[0000-0001-9202-9406]] and Grzegorz Nowakowski[2[0000-0002-3086-0947]]

[1] Igor Sikorsky Kyiv Polytechnic Institute, Kyiv, 03056, Ukraine
aser@comsys.kpi.ua
[2] Cracow University of Technology, Cracow, 31-155, Poland

Abstract. In this paper, we consider implementations of the sigmoid activation function for artificial neural network hardware systems. A rational fraction number system is proposed to calculate this function. This form of data representation offers several benefits, including increased precision compared to integers and more straightforward implementation in a field programmable gate array (FPGA) than the floating-point number system. In contemporary FPGA applications, rational fractions excel in regard to their compact hardware size, high throughput, and the ability to adjust the precision through the selection of the data width. The proposed module for calculation of the sigmoid activation function is shown to have high throughput and to occupy a relatively modest hardware volume compared to modules relying on piecewise polynomial approximation with fixed-point data.

Keywords: rational fraction, sigmoid function, FPGA, VHDL, artificial neural network.

1 Introduction

The increasing number of publications devoted to artificial neural networks (ANNs) in recent years indicates a growing interest in implementing these in hardware. The main reason for this trend is the rapid development of the element base used in digital ANN implementations. One approach to both accelerating the operations of an ANN and minimising its power consumption is to exploit its parallelism through the use of a field programmable gate array (FPGA). The speed of the artificial neurons depends heavily on the speed with which the sigmoid activation function is calculated in the nodes of the ANN. However, implementing this function in an FPGA requires significant hardware resources. The choice of approximation method for this function and its hardware implementation are crucial factors that affect the accuracy and speed of the ANN algorithm. Previous studies have explored various approximation methods for the digital implementation of nonlinear activation functions, including tabular approaches, Taylor transformation, and piecewise polynomial approximation. The most commonly used method for implementing sigmoid-type activation functions is a piecewise linear or quadratic approximation with integer data. This provides lower approximation

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_11

accuracy, which may lead to higher performance, while reducing the approximation error increases the hardware resource usage and decreases the data processing speed [13–16]. Other methods, such as rational approximation and continued fractions, provide the highest precision results; however, these involve floating point data calculations, which require large amounts of hardware resources in FPGA.

This paper proposes a new algorithm for calculating the sigmoid activation function based on a rational fraction data representation.

2 Introduction

The most common data representation in an ANN is based on the floating point, and this is explained by the features of the ANN algorithm. The ith node of a typical ANN applies a signal multiplier by the weight $\theta_{i,j}$ at its jth input. During learning by the generalised delta rule (backpropagation algorithm), the weights $\theta_{i,j}$ are updated, taking into account the difference between the available and predicted results (recognition errors), and based on the results, the gradients δ_i^l are calculated. Each gradient indicates in which direction and at what speed the weights $\theta_{i,j}$ should be changed so that they eventually reach the optimal value.

A node calculates its activation a_i as the sum of the inputs x_i multiplied by the weights $\theta_{i,j}$, which is processed by a nonlinear activation function p_i. This function must be differentiable so that network parameters can be tuned using the error backpropagation algorithm. The most common is the sigmoid activation function:

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

The weights $\theta_{i,j}$ of the nodes in the different layers are adjusted according to the chain rule. At the same time, the gradients δ_i^l, which are used to adjust the weights $\theta_{i,j}$ in the internal layers of the ANN, are calculated taking into account the gradients δ_i^{l+k}, which are propagated from all subsequent layers. The learning process involves several iterations in which the parameters $\theta_{i,j}$, δ_i^l are continuously updated until the network is optimised (for example, when the weights $\theta_{i,j}$ stop changing).

When the network is deep (more than three layers), the learning process may suffer from vanishing or exploding gradients δ_i^l depending on the choice of the activation function p_i. As a result, the weights of the initial layers cannot be properly adjusted [1].

To ensure both convergence of the weights and an ANN with the optimum throughput-cost ratio, the execution of an ANN usually has two stages. In the first, the ANN is trained using precise floating point calculations, meaning that the effect of vanishing or exploding gradients is minimised. In the second step, an effective data representation is selected, and the weights $\theta_{i,j}$ are truncated and rounded. The trained ANN is then used for a particular application. In this inference step, the ANN is effectively implemented in FPGA as well. Through this process, the unneeded relations in the ANN are removed, which significantly simplifies the structure configured in the FPGA (network pruning) [2]. However, a problem remains in terms of selecting both the data representation and

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_11

the sigmoid function approximation that can provide the minimum degradation in the pattern recognition effectiveness [3, 4].

A wide range of data formats can be used for an ANN implementation. To compress the huge sets of weights $\theta_{i,j}$ to provide a sufficient data range, 16-bit floating point formats are used, such as fp16 and BFloat16 [2]. Moreover, the different ANN layers need different levels of precision, and specific floating point formats of different bit widths can be used with a FPGA [5].

The results of previous studies show that the data dynamic range for an ANN is more important than the mantissa range. This fact is taken into account in the BFloat16 format proposed by Google, in which the mantissa has only 7 bits while the exponent has 8 bits [6].

A logarithmic number system is sometimes used that can also provide a wide dynamic range [7] and multiplier-free structures for the node. However, a logarithmic arithmetic unit may be more expensive than a hardware multiplier due to the significant complexity of the logarithmic function approximation.

Other investigations have shown that the weights $\theta_{i,j}$ and outputs of the activation functions in a trained convolutional ANN such as AlexNet are distributed over a relatively small range [8]. To exploit this fact, the Posit format is used in some ANNs. This format is distinguished by the variable lengths of both the mantissa and exponent, which is coded by the additional regime field [9–11].

Although both the floating point and Posit formats provide the necessary computational precision, they yield lower throughputs, higher hardware volumes, and higher power consumption than integer formats for ANNs configured in FPGA [12]. FPGA is an excellent basis for the implementation of application-specific processors that calculate integer data. The data bit width can be tuned easily in an FPGA to provide the optimum ratios for the throughput, precision to hardware volume, and power consumption [4]. When performing calculations in the respective data ranges, integers are considered as fixed point data.

The position of the point in the fixed point format depends on the scale factor of the particular data. In most convolutional ANNs, the point in the data and weights $\theta_{i,j}$ is positioned just after the sign bit [8]; however, the input data for a sigmoid-type activation function are usually limited by a value of ± 8. The point therefore divides the word into an integer and its fractional parts. In this situation, the data distribution in the set of all integers of a particular bit width is not effective. Fig. 1 illustrates the data distribution for a set of 2^8 unsigned integers. It is clear that in the case of the input data entering the sigmoid function, most of the uniformly represented data are concentrated in the range [4: 8]. This example illustrates that the integer range is used ineffectively here.

Integers are used in convolution algorithms very well, but it is hard to find an effective algorithm to calculate a nonlinear function such as the sigmoid function using integers. As a result, most of the activation functions in ANNs are calculated using a piecewise linear or second-order polynomial approximation [13–17, 28]. Most of these have data representations based on 16 bits and limited precision, with a maximum error in the range [0.0005: 0.07].

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_11

It is therefore preferable to use a data format that occupies a place between floating point and integer formats, and which provides high precision and dynamic range, high speed, and a low hardware volume for an ANN implementation as well as an effective sigmoid function calculation. The next section presents such a format.

3 Rational fractions and calculation of the sigmoid activation function

3.1 Rational fractions

A rational fraction is a numerical object that consists of an integer numerator and integer denominator, and represents a rational number. The rational fraction a/b has the characteristic that it can approximate a given transcendental number x. If $a/b < x$ and $c/d >$ x, then the fraction $(a + c)/(b + d)$, called a medianta, is nearer to x than a/b or c/d. Hence, if a set of mediantes is built, then we can approximate a number x with any precision [18].

If a noninteger number x is represented by $2n$ digits in fixed point format with error ε_1, then it can be represented by the fraction a/b with error $\varepsilon_2 = \varepsilon_1$, and the numbers a and b have no more than n digits in their representation [19].

A representation based on a fractional number has a set of advantages. Firstly, any binary fraction or fixed point datum is dependent on the binary data representation, and does not exactly represent a real number. In binary representation, a floating point number is equal to a fraction where the denominator is a power of two, and is not equal to the respective decimal fraction. For example, the number $1/9 = 1/1001_2$ is an exact fraction in any numerical system, and can be represented with an error as the decimal fraction 0.111110 or binary fraction 0.11100011100011_2.

Secondly, the number distribution of rational fraction data is more effective for the implementation of an ANN compared to integers. This fact is illustrated in Fig. 1, where the charts represent a number n of different data samples in the range $[2^i, 2^{i+1}]$, $i = -6,\ldots,7$ for 16-bit fixed point data and rational fractions with an 8-bit numerator and denominator. From the graph, it is clear that the data are concentrated around a value of 1.0, as the data are for real ANNs.

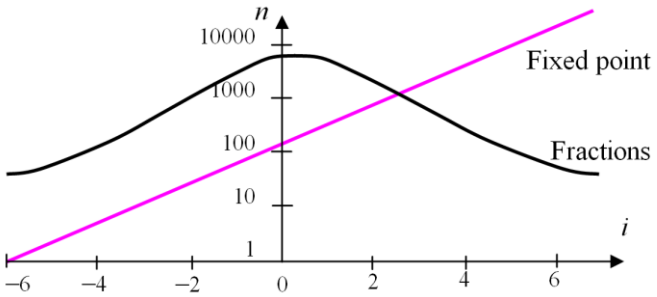

Fig. 1. Data distributions for sets of fixed point data and fractions.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_11

Thirdly, rational fractions can help us to find an approximation for an irrational or transcendental number with a given level of precision. Many elementary functions are effectively calculated using suitable rational approximation formulas. When such a function is approximated using finite continued fractions or rational approximations, the rational fraction arithmetic fits very well. Many constants and constant tables are effectively stored as rational numbers.

Finally, rational fractions provide a comparatively simple set of arithmetic operations. The multiplication a/b to c/d and division of them are equal to $ac/(bd)$ and $bd/(ac)$, respectively. Note that the division of the numerator to the denominator is not calculated. The addition operation is equal to $(ad + bc)/(bd)$. For comparison of two numbers, it is sufficient to calculate $ad - bc$ [18]. Some specific calculations are simple, such as

$$1 + \frac{a}{b} = \frac{a+b}{b}; \quad \text{or} \quad \frac{1}{1+\frac{a}{b}} = \frac{b}{a+b}, \tag{2}$$

and these can help in calculating the continued fractions.

It should also be taken into account that the numerator and denominator bit width is less than half of the bit width of the fixed-point data that provides equal precision.

Hence, the hardware complexity of a fraction adder is similar to the complexity of an integer multiplier with the same precision, and the complexity of a fraction multiplier is two times less than that of an integer multiplier.

3.2 Rational fractions in processors

Some computers in the 1970s had already the rational fraction arithmetic except floating point one. The main disadvantage of rational fractions is that the number of bits increases dramatically when operations are implemented precisely [20]. To eliminate this disadvantage, the division of the numerator and denominator by their greatest common divisor was done, as in the rational fraction processor, which was proposed in [21]. However, when floating point coprocessors became widely used, fractional number processors dropped out of use.

Later, rational fractions were built into many mathematical CAD tools such as Maple, which are implemented in PC. Such fractions are widely used for calculations with unlimited precision, for solving modern cryptographic problems, and other tasks. Languages such as PERL and Java are therefore supported by packages providing calculations with unlimited precision using rational fractions.

The development of FPGAs, which contain numerous hardware multipliers, enabled the design of application-specific processors that used rational fractions. Two processors for computing linear algebra problems were proposed in [22, 23]. Rational fractions have also been effectively used in a processor intended for autoregressive signal analysis [24].

The features of rational fractions described above were exploited in these processors. Modern FPGAs have thousands of hardware multiplier-accumulator units (MPUs) that can perform base operations with fractions, including (2). Each operation may result in

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_11

underflows in the numerator or denominator; in this situation, both the numerator and denominator are normalised by a left shift of an equal number of bits. This number of bits is equal to one after the addition operation and a maximum of $n/2$ after multiplication.

3.3 Calculation of the sigmoid function

In view of the features of the rational fraction operation, it is clear that the usual approximation methods such as piecewise approximation and power series are not effective, due to the large numbers of comparison and addition operations. In contrast, rational approximation and finite continued fractions are effectively calculated using rational fractions, as this approach requires much fewer elementary operations. We note that continued fractions frequently converge much more rapidly than power series expansions and in a much larger domain. A sigmoid activation function with a single precision floating point can be effectively calculated using only the rational Padè approximation [25].

The function in (1) can be calculated in two steps. In the first, the exponent function is calculated using the continued fraction approximation [26]

$$e^x = 1 + \frac{x}{1-}\ \frac{x}{2+}\ \frac{x}{3}\cdots \approx 1 + \frac{2x}{2 - x^2/6} \tag{3}$$

while in the second, the formula in (1) is calculated.

To evaluate rational fraction calculations with different bit widths, a package of specific functions was designed in the VHDL language so that the functions marked as '+', '-', '*' , and '/' overload the respective operations but for the rational fraction data. Functions were also designed for calculating (2) as well as functions that transferred data from the floating point format to a rational fraction and back again.

The chart in Fig. 2 was created by evaluating the sigmoid function in (1) using the approximation in (3) on the basis of 16-bit rational fractions. This gives maximum errors of ±0.12 in the range [–6:6] and ±0.002 in the range [–2:2]. It is clear from this example that a piecewise approximation would give good results. For instance, this function could be approximated by lines at the levels of ±1.0 when x is outside the range [–4:4]. However, the exponent function features provide a more effective means of improving the precision of the approximation.

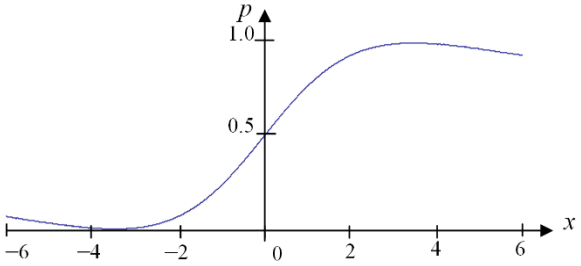

Fig. 2. Sigmoid function approximated using (3).

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_11

Consider the case where $y_0 = e^{x_0}$ is the exact value of the function in (1) for an argument x_0. Then, for any argument x which is close to x_0, we have the approximation

$$e^x = y_0\, e^z = y_0 \left(1 + \frac{z}{1 -}\, \frac{z}{2 +}\, \frac{z}{3} \ldots\right) \approx y_0 (1 + z);$$
$$z = x - x_0 \tag{4}$$

Here, x_0 is a rational fraction formed from the most significant bits of the input data x. The bits of x_0 serve as the address bits for the table, which stores the values y_0. Hence, the approximated exponent function is derived as the product of the table function $y_0(x_0)$ and the sum $1 + z$. A graph of the resulting sigmoid function when the numerator and denominator of x_0 have widths of 4 bits is shown in Fig. 3.

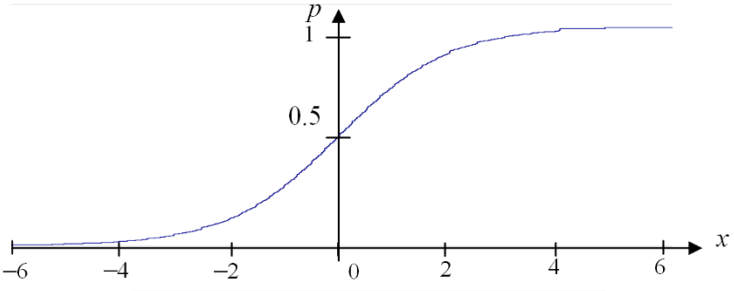

Fig. 3. Sigmoid function approximated using (4).

The resulting function gives a maximum error of –0.0088 in the range [–8:8]. This error can be reduced by increasing both the bit width of x_0 and the number of terms of the continued fraction in (4), and can be as small as necessary. When one additional term is used in (4), the maximum error is decreased to –0.0004.

In this section, we have proposed a new method for effective approximation of the sigmoid activation function based on rational fraction arithmetic, which gives moderate precision using a small number of calculations. In the next section, we present an example of a module used to calculate this function in a FPGA.

4 Experimental results

The proposed method for approximation of the sigmoid activation function is implemented in FPGA as an IP core. The input data x and results p are represented by the 16-bit integer numerators x_n, y_n, and denominators x_d and y_d, respectively. A dataflow graph for the approximation algorithm is shown in Fig. 4. The input data are stored in the registers RXn and RXd, and are loaded and calculated in the pipeline mode. Their four most significant bits (including signs) form the value x_0, which serves as the address in the tables ROMEn and ROMEd, which store the values of the exponent coefficients $y_0 = y_{0n}/y_{0d}$.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_11

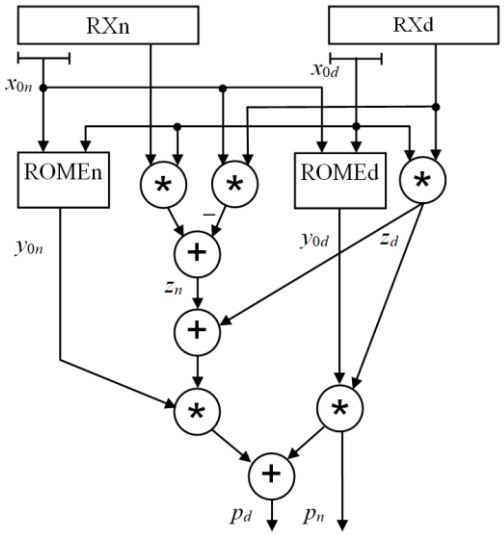

Fig. 4. Dataflow graph for calculation of the sigmoid function.

Five integer multipliers and three integer adders are used to perform the calculations shown in (1) and (4). The intermediate result z and result p are normalised by left-shifting the numerator and denominator by up to 2 bits. The module used for calculating this algorithm is described in VHDL language by the style for synthesis as the pipelined datapath. This datapath contains five pipeline register stages, and outputs the results in each clock cycle.

The parameters for the proposed module, as configured for different FPGA series, are shown in Table 1. The number of DSP blocks containing the multiplier is only two when the module is configured in a Xilinx Kintex7 FPGA.

Table 1. Parameters of the module for calculating the sigmoid function

Module	FPGA series	Hardware cost		Maximum clock frequency, MHz	Maximum error
		LUTs/ALMs	DSP blocks		
Proposed	Kintex7	381	2	238	0.0094
Proposed	Artix7	389	2	154	0.0094
Proposed	Cyclone V	151	4	117	0.0094
Tsmots [13]	Cyclone III	368	0	37	0.018
Campo [16]	Virtex6	232	6	53*	0.028
Gomar [27]	Virtex4	123	0	29*	0.087
Zhang [15]	Zed7	272	2	107	–
Li [14]	Virtex7	493	0	208	0.0078

* Data input frequency, as the algorithm is implemented in several sequential steps.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_11

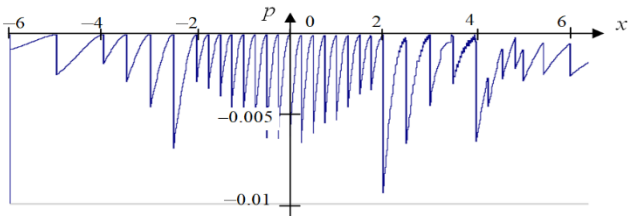

Fig. 5. Error in the sigmoid function calculation.

This is explained by the fact that the small bit multiplication in the data x0n and x0d is performed in hardware on the basis of LUTs. The maximum clock frequency in this FPGA reaches 238 MHz due to the pipelining of the calculations. When the calculations are carried out without pipelining, this frequency is reduced to 95 MHz.

Fig. 5 shows the calculation error for the module. An analysis shows that there is the opportunity to minimise the maximum error by rounding the result. Moreover, the error depends on the address bit widths of ROMEn and ROMEd, and can be decreased dramatically when these bit widths are increased. Due to the hardware consumption in Table 1, these ROMs are performed in LUTs and there is the potential to increase their volume.

The different modules that have been used to calculate the sigmoid activation function using a piecewise polynomial approximation are presented in Table 1 for comparison. We note that they all use 16-bit fixed point arithmetic, and the maximum error is derived for the input range [–8: 8]. A comparison of these different modules shows that the proposed module has the highest speed, with a moderate value for the hardware volume and a comparatively small computational error.

Our module has the advantages of increasing the computational precision and minimising the hardware volume at the cost of bit width minimisation. The other modules of the ANN system can also use rational fraction arithmetic. However, in other situations, the proposed module has to be agreed upon both with the input and output floating point or integer data by attaching not complex wrapping hardware. The hardware attached to the output must have a division unit that calculates $p = p_n/p_d$.

5 Conclusion

A rational fraction number system has the advantage of providing higher precision than integers, and its FPGA implementation is simpler than that of a floating number system. The main advantages of using rational fractions in a modern FPGA implementation are small hardware volume, high throughput, and the possibility of regulating the precision by selecting the data width. It has been shown here that this data representation helps in designing effective modules for implementation of the sigmoid activation function. Our module for calculating the sigmoid activation function is shown to have high throughput and low hardware volume in comparison with modules based on a piecewise polynomial approximation using fixed point data. Future work on the use of rational fractions will focus on the implementation of an ANN system as a whole.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_11

Acknowledgments. Funding: This research was funded by the Faculty of Electrical and Computer Engineering, Cracow University of Technology, and the Ministry of Science and Higher Education, Republic of Poland (grant no. E-1/2024).

References

1. Russell S., Norvig P.: Artificial Intelligence: A Modern Approach, 4th Edition. Pearson (2022)
2. Young Kim J-Y.: Chapter Five - FPGA based neural network accelerators. In: S. Kim, G. C. Deka (eds) Advances in Computers, vol. 122, pp. 135–165, Elsevier (2021), https://doi.org/10.1016/bs.adcom.2020.11.002
3. Bailey B., Machine Learning's Growing Divide. Semiconductor Engineering (2018), https://semiengineering.com/machine-learnings-growing-divide, last accessed 2024/04/19
4. Mahajan, R., Sakhare, D. Gadgil R.: Review of Artificial Intelligence Applications and Architectures. In: Anuradha D. Thakare, Sheetal Umesh Bhandari (eds), pp. 25–34. Artificial Intelligence Applications and Reconfigurable Architectures, Wiley Online Library (2023). https://doi.org/10.1002/9781119857891.ch2
5. Floating-Point Operator v7.1 PG060, Xilinx (2020), https://docs.amd.com/v/u/en-US/pg060-floating-point, last accessed 2024/04/19
6. Lai, L., Suda, N., Chandra, V.: Deep convolutional neural network inference with floating-point weights and fixed-point activations, arXiv:1703.03073v1, pp. 1–10, Computer Science: Machine Learning (2017), https://doi.org/10.48550/arXiv.1703.03073
7. Miyashita D., Lee E. H., Murmann B.: Convolutional neural networks using logarithmic data representation, arXiv:1603.01025v2, Computer Science: Neural and Evolutionary Computing (2016), https://doi.org/10.48550/arXiv.1603.01025
8. Zhang, H., Subbian, D., Lakshminarayanan, G., Ko, S-B.: Application-Specific and Reconfigurable AI Accelerator. In: Mishra, A., Cha, J., Park, H., Kim, S. (eds) Artificial Intelligence and Hardware Accelerators, pp. 183–223. Springer, Cham (2023), https://doi.org/10.1007/978-3-031-22170-5_7
9. Johnson J.: Rethinking floating point for deep learning, arXiv:1811.01721v1, Computer Science: Numerical Analysis (2018), https://doi.org/10.48550/arXiv.1811.01721
10. Carmichael Z., Langroudi H. F., Khazanov C., Lillie J., Gustafson J. L. and Kudithipudi D.: Deep Positron: A Deep Neural Network Using the Posit Number System, Design, Automation & Test in Europe Conference & Exhibition (DATE), Florence, Italy, pp. 1421–1426, IEEE (2019), https://doi.org/10.23919/DATE.2019.8715262
11. Raposo G., Tomás P., Roma N.: PositNN: Training Deep Neural Networks with Mixed Low-Precision Posit, ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 7908-7912, Toronto, ON, Canada (2021), https://doi.org/10.1109/ICASSP39728.2021.9413919
12. Nechi, A., Groth, L., Mulhem, S., Merchant F., Buchty, R., Berekovic, M.: FPGA-based Deep Learning Inference Accelerators: Where Are We Standing?, vol. 16, issue 4, Article No.: 60, pp 1–32, ACM Transactions on Reconfigurable Technology and Systems (2023), https://doi.org/10.1145/3613963
13. Tsmots, I., Skorokhoda, O., Rabyk V.: Hardware Implementation of Sigmoid Activation Functions using FPGA, pp. 34-38IEEE 15th International Conference on the Experience of Designing and Application of CAD Systems (CADSM), Polyana, Ukraine (2019), https://doi.org/10.1109/CADSM.2019.8779253

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_11

14. Li, Z., Sui, B., Xing, Z., Wang, Q.: FPGA Implementation for the Sigmoid with Piecewise Linear Fitting Method Based on Curvature Analysis, vol. 11, issue 9, Electronics (2022), https://doi.org/10.3390/electronics11091365
15. Zhang, L.: Implementation of Fixed-point Neuron Models with Threshold, Ramp and Sigmoid Activation Functions, 4th International Conference on Mechanics and Mechatronics Research, vol. 224, IOP Publishing IOP Conf. Series: Materials Science and Engineering (2017), https://doi.org/10.1088/1757-899X/224/1/012054
16. Del Campo, I., Finker, R., Echanobe, J., Basterretxea, K.: Controlled Accuracy Approximation of Sigmoid Function for Efficient FPGA-based Implementation of Artificial Neurons, vol. 49(25), pp. 1598-1600, Electronics Letters (2013), https://doi.org/10.1049/el.2013.3098
17. Laudani, A., Lozito, G. M., Fulginei, F. R, Salvini, A.: On Training Efficiency and Computational Costs of a Feed Forward Neural Network: A Review, vol. 2015, 13 pages, Computational Intelligence and Neuroscience (2015), https://doi.org/10.1155/2015/818243
18. Kornerup, P., David W. Matula, D. W.: Finite Precision Number Systems and Arithmetic, Cambridge University Press (2010), https://doi.org/10.1017/CBO9780511778568
19. Hintchin A.Y. Continued Fractions: Moscow, Nauka, 3-d Ed (1978) (in russian)
20. Horn, B. K. P. Rational arithmetic for minicomputers, Vol. 8, No. 2, pp. 171–176, Software Practice and Experience (1978)
21. Irvin M. J., Smith D. R.: A rational arithmetic processor, pp. 241-244, Proc. 5-th Symp. Comput. Arithmetic (1981)
22. Maslennikow, O., Lepekha, V., Sergyienko, A.: FPGA Implementation of the Conjugate Gradient Method. In: Wyrzykowski, R., Dongarra, J., Meyer, N., Waśniewski, J. (eds) Parallel Processing and Applied Mathematics, Lecture Notes in Computer Science, vol. 3911. Springer, Berlin, Heidelberg (2006), https://doi.org/10.1007/11752578_63
23. Maslennikow, O., Lepekha, V., Sergiyenko, A., Tomas, A., Wyrzykowski, R.: Parallel Implementation of Cholesky LL T-Algorithm in FPGA-Based Processor. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds) Parallel Processing and Applied Mathematics, Lecture Notes in Computer Science, vol. 4967, Springer, Berlin, Heidelberg (2008), https://doi.org/10.1007/978-3-540-68111-3_15
24. Sergiyenko, A., Maslennikow, O., Ratuszniak, P., Maslennikowa, N., Tomas, A.: Application Specific Processors for the Autoregressive Signal Analysis. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds) Parallel Processing and Applied Mathematics, Lecture Notes in Computer Science, vol. 6067, Springer, Berlin, Heidelberg (2010), https://doi.org/10.1007/978-3-642-14390-8_9
25. Hajduk, Z.: High accuracy FPGA activation function implementation for neural networks, vol. 247, issue C, pp. 59–61, Neurocomputing (2017), https://doi.org/10.1016/j.neucom.2017.03.044
26. Roy, R., Olver, F. W. J.: Elementary Functions, In: NIST Handbook of Mathematical Functions, Cambridge Univ. Press (2010)
27. Gomar, S., Mirhassani, M., Ahmadi M.: Precise Digital Implementations of Hyperbolic Tanh and Sigmoid Function, 50th Asilomar Conference on Signals, Systems and Computers, pp. 1586-1589, Pacific Grove, USA (2016), https://doi.org/10.1109/ACSSC.2016.7869646
28. Moroz, L., Samotyy, V., Gepner, P., Węgrzyn, M., Nowakowski, G.: Power Function Algorithms Implemented in Microcontrollers and FPGAs, vol. 12, issue 16, Electronics (2023), https://doi.org/10.3390/electronics12163399

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_11