Context-Switching Neural Node for Constrained-Space Hardware

Yassen Gorbounov[1(✉)] and Hao Chen[2]

[1] New Bulgarian University and MGU "St. Ivan Rilsky", Sofia, Bulgaria
gorbounov@gmail.com
[2] China University of Mining and Technology, Xuzhou 221000, People's Republic of China
hchen@cumt.edu.cn

Abstract. Artificial neural networks are a mathematical abstraction that models the nerve cells and their connections in the biological brain. Their use to solve a wide range of nonlinear problems in the field of approximation, machine learning, and artificial intelligence may require significant computing power. This requires the search for approaches to more efficient use of hardware resources. The article describes an early stage in the development of an optimized neural network designed for use on hardware with constrained resources, but with the ability to run parallel algorithms. Circuits that meet these requirements are the field programmable gate arrays. In the research an approach is proposed to significantly reduce the amount of logic used by contextually switching the weight matrices and the matrices of the activation functions which will result in reducing the number of network layers. This suggests the ability to create more affordable and smarter devices without compromising performance significantly.

Keywords: Artificial neural network · Contextual switching · Programmable logic

1 Introduction

Despite the high level of technological development nowadays, there is still nothing more perfect than nature in terms of the ability to recognize images and situations, decision-making under conditions of uncertainty, adaptation to changing environments, and the efficient use of energy. Unequivocal examples in this regard are, on the one hand, the ability to orient, the speed of reaction, and the path that an insect can travel in relation to its weight, and, on the other hand, the complex organizational structure of organisms such as bees and ants [22]. At the bottom of all this is the nervous system, the functioning of which people try to recreate through various mathematical approaches. One possible apparatus are the artificial neural networks [15], which are an attempt to simulate the interaction and collaboration of a large group of nerve cells. In purely mathematical terms, neural networks have been considered by [21] as a universal approximator. The ultimate goal of these models is to create a technical device capable of training and making decisions similar to the human brain.

T. Zlateva and R. Goleva (Eds.): CSECS 2022, LNICST 450, pp. 45–59, 2022.
https://doi.org/10.1007/978-3-031-17292-2_4

To solve a real problem, artificial neural networks consist of many neural models (cells), arranged in a multilayer structure. There are many different and quite complex models of neural networks [26]. Some topologies such as Convolutional Neural Networks (CNN) [13], Recursive Neural Networks (RNN) [3], Generative Adversarial Networks (GAN) [27] and many others, are aimed at processing non-digital and unstructured data such as text, image and speech. However, if analog methods of implementation [5, 24] are excluded, in their deep essence, all structures and algorithms for building artificial neural networks work with digital and structured data. Analog methods and features of different network architectures, as well as solving specific examples, are not the subject of this paper and therefore they are not discussed here. However, it is clear that the improvement of artificial neural networks increases the level of complexity of the mathematical apparatus used. This leads to a number of problems in their practical implementation, mainly related to the work with floating point arithmetic and the high-speed exchange of large volumes of data. As a result, the requirements for digital circuits in terms of their speed, reduction of energy consumption and augmenting the degree of integration are increasing. Despite the increase in the number of cores and thus in the performance of modern processors, and the proliferation of highly optimized software approaches for small devices such as TinyML [23, 28], today more and more programmable logic circuits such as Field Programmable Gate Arrays (FPGA) are used. They are characterized by high operating frequencies, the possibility of multiple reprogramming, but most of all the ability to perform parallel algorithms [4, 8, 12]. This property is architecturally inherent and is their undeniable advantage over conventional microprocessors and microcontrollers.

In recent research from Princeton University [14] a multiplexing technique called DataMUX is proposed. It allows the neural network to analyze multiple data feeds nearly simultaneously by multiplexing the inputs and the outputs. The authors conducted their experiments in the Python language. Although they claim to potentially prune the numbers of configuration parameters by more than 90 percent, the architecture of their neural network remains unchanged. The current research article takes different approach. It presents an early stage in the development of an optimized neural network designed for use on hardware with limited resources, but with the possibility of running algorithms in parallel. Among the circuits with such capabilities are the lower-end FPGAs. An approach to significantly reduce the amount of logic used is proposed, which is expected that when the artificial neural network is trained in advance, the tasks using this category of reprogrammable logic will be performed without compromising performance. This means creating more affordable and smarter devices.

The remainder of this work is organized as follows: At first in chapter “Architecture of the artificial neuron,” it is described the digital implementation of a possible basic model of an artificial neuron. Discussed are its main building block and occupied resource. Next, the proposed new method for sharing resources and reducing the total number of neurons in the neural network is presented in the chapter “Context switching approach”. Guidelines for future development and improvement are given in chapter “Future work”. The chapter “Conclusions” summarizes the results.

2 Architecture of the Artificial Neuron

In biological nerve cells, signals are exchanged through the axon with the aid of a synaptic structure that passes an electrical or chemical signal from a source neuron to a group of target neurons. A target neuron can have many inputs (dendrites) that can connect to many other neurons. The soma (cell body) contains a nucleus whose function is to sum all the inputs and to compute their overall "influence" weight. This signal is then processed by an activation function. If the result is above a given threshold the neuron fires, thus spreading the nerve signal to other neurons. This way a quite complicated network is organized. The process of learning consists of strengthening or weakening connections between neurons. This process inspired the creation of artificial neural networks, which are the basis of modern disciplines such as deep learning (DL), machine learning (ML), artificial intelligence (AI), autonomous decision-making systems, image recognition, and many applications in robotics, including also much simpler tasks such as function approximation.

This section discusses the working principle and building blocks of an elaborated artificial neuron. In [17–20] an attempt in that direction is already made but it has several major drawbacks. The presented neuron model is built up by using solely the schematic approach which means that arbitrary change in its function requires a significant redesign of the device and makes it hardly portable between different programmable logic families and manufacturers. This approach is especially problematic in regards to the activation function because it consumes most of the logic gates resources of the device. In addition, this model contains excessive code converter modules used for signed arithmetic operations, and it cannot be configured in terms of altering the bit width of the inputs, the outputs, or the internal bus organization. The lack of versatility limits its adaptability and applications.

The proposed device is devoid of these shortcomings and is significantly optimized in terms of performance and resources. The Verilog hardware description language (HDL) has been used for the synthesizable description of the design, and the Spartan series FPGA from Xilinx (now AMD) has been used as a target platform.

The structure of the proposed artificial neuron is given in Fig. 1. In analogy with the biological neuron, the cell nucleus is represented by the adder, which sums up the input signals xi multiplied by the weighting factors wi and some correction factor b called the bias. The obtained sum is submitted to an activation function, which sets the trigger threshold. The learning process consists in training the neural network over a training set according to different algorithms, but finally, it ends up in memorizing optimal values of the weighting factors and the bias. The weights can be either positive or negative. Negative weights reduce the value of the output, and positive weights increase it. The above said means that the hardware implementation requires signed floating-point arithmetic. FPGA circuits do not have built-in floating-point units (FPU) so they do not support the floating-point (FP) standard IEEE-754 [10] by default. In order to use the FP arithmetic, a dedicated submodule can be synthesized but it will add an unnecessary burden to the overall complexity and will negatively impact the size of the design and the speed of execution.

To overcome this the proposed design is using signed fixed-point arithmetic. This can be easily done by using the shift operation for upscaling and downscaling the FP

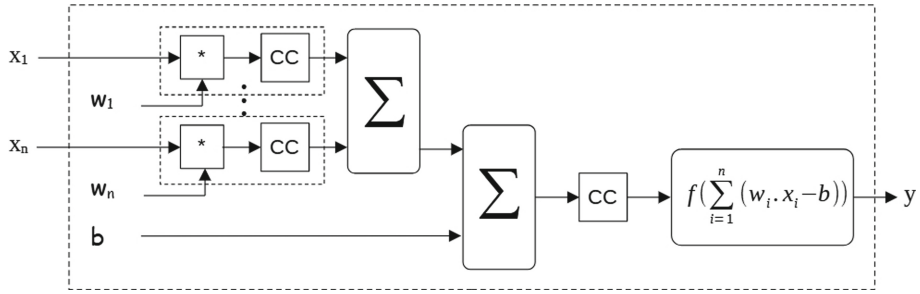


Fig. 1. Block diagram of the artificial neuron.

number with numbers multiples of the powers of two. Due to the signed numbers and to simplify the design of the adder the product from the multiplication of the input signals by the weights can be converted to two's complement code with the aid of the code converter modules (CC). Another possibility is to use a signed multiplier which will decrease the processing delay. However, the CC block should be used for the output of the adder before it is fed to the activation function due to the fact that it takes part in the addressing of the look-up table (LUT) that holds the description of that activation function. All the weights, the bias, and the activation function are stored in a Block RAM memory (BRAM). The proposed architecture follows the principles of regularity, modularity, and hierarchy [7]. That means that it is divided into subordinate modules and submodules and it is uniform, so modules can be easily reused.

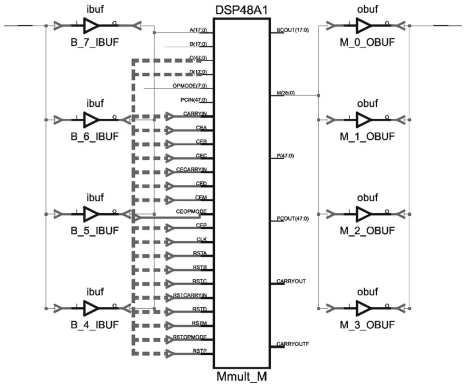
2.1 Signed Multiplier

The inputs that enter the artificial neuron need to be multiplied by the weight coefficients before they are fed to the adder. There exist many algorithms to do so such as the Dadda multiplier, Wallace tree, parallel multipliers [9], Karatsuba algorithm [6, 11], and its generalization, the Toom–Cook (Toom-3) [1] or Fourier transform, all of which can be successfully used. In the proposed neuron model the behavioral description of the signed multiplier is chosen due to its simplicity and because the Spartan-6 FPGA family has a high ratio of embedded DSP modules to logic, making it ideal for math-intensive applications. It is equipped with up to 180 DSP48A1 slices and each one contains a built-in 18 × 18 bit two's complement multiplier capable of operating at frequencies of up to 390 MHz. The Verilog description of the configurable signed multiplier together with a substantial part of the register-transfer level (RTL) view and the simulation results are shown in Fig. 2.

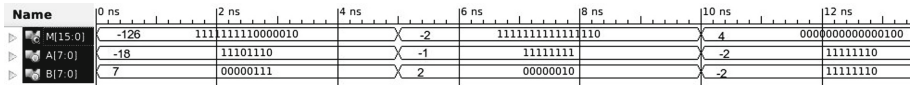
The module is designed to be parameterizable so that the width of the operands can be configured by changing the parameter N.

```
module smult #(parameter N = 8)
    (   input signed [N-1:0] A,
        input signed [N-1:0] B,
        output signed [2*N-1:0] M
    );
    assign M = A * B;
endmodule
```

(a)



(b)



c)

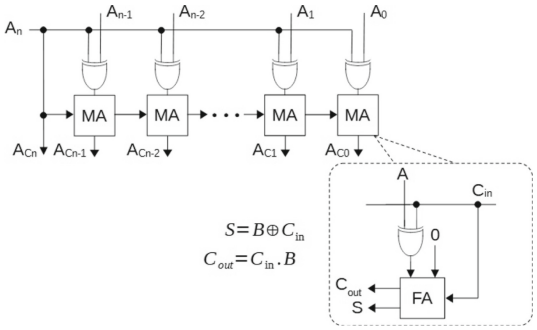
Fig. 2. Verilog description (a), partial RTL view showing the DSP48A1 slice after synthesis (b), and simulation (c) of the work of the signed multiplier.

2.2 Code Converter

The output of the second adder of the neuron has to be converted in two's complement form. This is done on the basis of a modified adder (MA) (see Fig. 3).

Cin	B	A	Cout	S
0	0 → **0**	0	0	0
0	0 → **0**	1	0	1
0	1 → **0**	0	0	0
0	1 → **0**	1	0	1
1	0 → **0**	0	0	1
1	0 → **0**	1	1	0
1	1 → **0**	0	0	1
1	1 → **0**	1	1	0

(a)



(b)

Fig. 3. Truth table of the modified adder (a), and the block diagram of the code converter.

The design of the code converter is implemented by modifying a simple full adder where one of the operands is set firmly to 0, and the carry-in bit is set to logic 1. The conversion from a positive to a negative number and vice versa is done in two's complement form by using one and the same algorithm, which consists of two main steps: 1) inverting all of the bits to get the one's complement code, and 2) adding a one

to the inverted result. The sign is presented with the left-most bit (MSB) which is a 0 for positive numbers and a 1 for negative ones. The exclusive OR gate in the figure serves the purpose of a controllable inverter. The negation of the bits or addition of a 1 is done by the left-most bit (MSB) of the number to be converted which is *An*.

2.3 Adding Device

Besides the built-in functional modules of the FPGA there are many architectures to build a binary adder such as the Carry Skip Adder, Carry Save Adder and Carry Select Adder (CSA), Pre-Fix Adder (PFA), Pipelined Parallel Adder (PPA), etc. that are well described in the literature [2, 7, 9]. Here a Carry-Lookahead Adder (CLA) has been implemented because its performance improves for larger bit-widths while preserving simplicity. The architecture of the adder is divided into groups of four bits and provides a way to determine the carry-out signal as soon as it becomes known to the group. For this purpose, two signals are generated (1):

$$\begin{aligned} g(A, B) &= A.B \\ p(A, B) &= A + B \end{aligned} \tag{1}$$

The "*generate*" (g) appears if the addition of two bits leads to a carry, regardless of whether the input *Cin* is set. The "*propagate*" (p) appears if the addition of two bits results in a carry when the *Cin* input is set but does not result in a carry if the *Cin* input is not set. Therefore, the carry out of a group can be obtained by (2):

$$C_{out(i)} = A_i.B_i + (A_i + B_i).C_{in(i)} \tag{2}$$

There are two adders in the artificial neuron (see Fig. 1). The block diagram of that portion of the circuit is shown in Fig. 4.

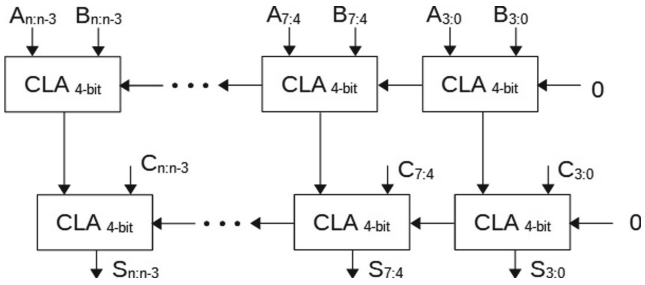


Fig. 4. Block diagram of the 3-input adder based on CLA architecture.

The inputs A and B are in two's complement form and connect to the outputs of the multipliers. The input C connects with the bias which is converted before it is stored in the memory. The output of the adder S is also in two's complement form.

2.4 Activation Function

The activation function is the output stage of the neuron and determines whether it fires or not. It adds nonlinearity to the neuron's function, otherwise, it would be no different than a linear regression model. Non-linear activation functions allow for backpropagation as the derivative is related to the input and thus it makes it possible to have better solution prediction based on the weights of the inputs. There is a great variety of activation functions depending on the purpose of the neural network. Some of the most common types are briefly summarized in Fig. 5.

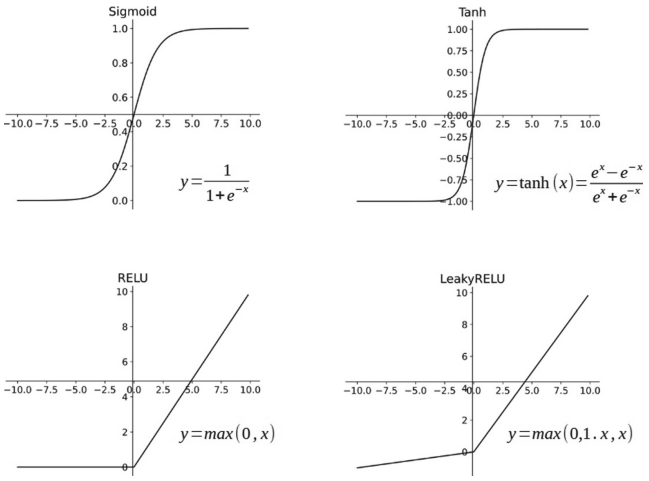


Fig. 5. Some of the most common activation functions.

The sigmoid activation function is differentiable and provides a smooth gradient. It is suitable for predicting probabilities. The sigmoid outputs values between 0 and 1 based on real value on the input. The hyperbolic tangent function (tanh) is similar to the sigmoid but its output is in the range of -1 to 1. The hyperbolic tangent output is centered across the zero. This function is mostly used in the hidden layers of the neural network. The Rectified Linear Unit (ReLU) activation function allows for back-propagation while keeping computational efficiency because only some neurons are activated. The Leaky ReLU improves the ReLU function by adding a small positive slope in the negative area. It enables backpropagation for negative input values.

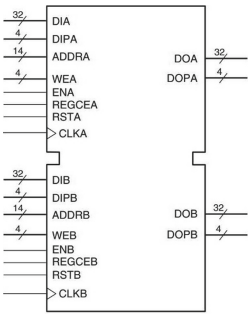
The computation of the corresponding activation function in real-time may pose a challenge as it can consume significant logic resources. It is much more convenient to compute it off-line in advance and next fill-in a memory array. The Verilog code snippet that implements the reading from the BlockRAM memory is shown in Fig. 6.

A single BlockRAM in the Spartan-6 FPGA can store up to 18K bits of data, and the FPGA has a total of 268 blocks. The memory can be used in either single-port or dual-port mode having independent bit-widths in two-port configuration [25]. That amount of memory and the proposed Verilog code allows for storing more than one look-up table into a single BlockRAM. The left term [dataWidth-1:0] describes the memory width and

```
reg [dataWidth-1 : 0] mem [2**xWidth-1 : 0];
reg [xWidth-1 : 0] out;

initial
  begin
    $readmemb("actfunc.mif",mem);
  end
```

(a)



(b)

Fig. 6. Verilog description for reading from the BlockRAM memory (a), and RTL view of the instantiated RAMB16BWER memory module.

the right term [2**xWidth-1:0] is the memory depth. Using some simple arithmetic, it is possible to calculate the separate memory spaces allocated for each LUT. The input file "actfunc.mif" is an ASCII text memory initialization file (MIF) that specifies the initial content of a memory block. It can be generated in advance by using software tools such as a Matlab or Python script.

2.5 Synthesized Device

The RTL view of the synthesized artificial neuron model is shown in Fig. 7. The SADDER block denotes the signed adder since it sums the weighed inputs and the BADDER is the one that adds the bias which value is taken from a look-up table. The SIGMOID module contains the activation function look-up table.

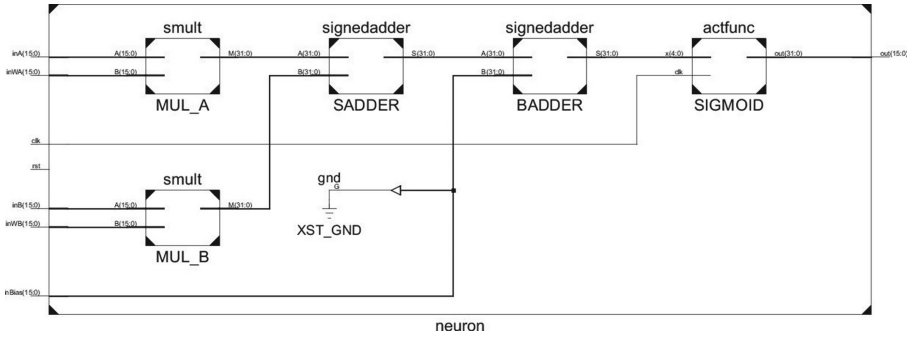


Fig. 7. RTL view of the synthesized artificial neuron model.

After compiling the design, the device usage statistics is generated (see Table 1).

Table 1. Device usage statistics.

Device utilization summary	Target Device: xc6slx75t-3fgg676		
Logic utilization	Used	Available	Utilization
Number of slice registers	5	93296	0.01%
Number of slice LUTs	149	46648	0.32%
Number of fully used LUT-FF pairs	5	149	3.36%
Number of bonded IOBs	97	348	27.87%
Number of BUFG/BUFGCTRLs	1	16	6.25%
Number of DSP48A1s	2	132	1.52%

Based on the elaborated design a simple neural network for applications such as color recognition can be built. An example of a suitable structure is shown in Fig. 8.

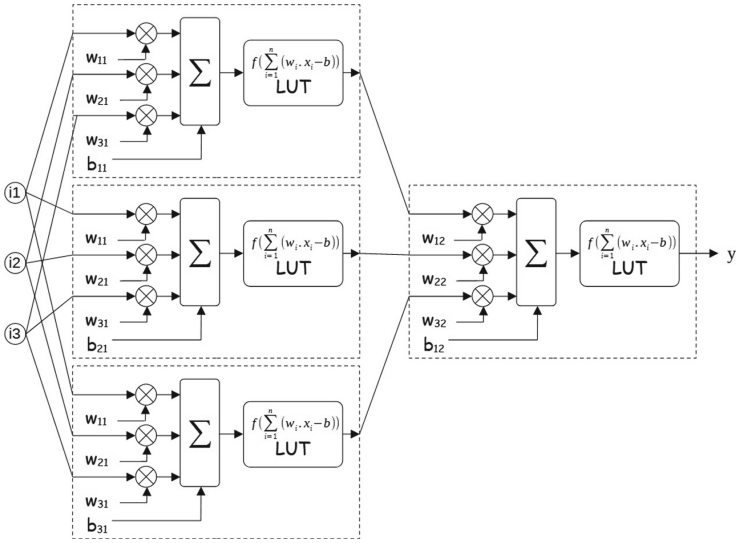


Fig. 8. Example of a three-layer artificial neural network with four neurons.

The processing effort of this simple three-layer neural network is 4 neurons, 12 signed multiplications, 12 additions (using 4 four-input adders), and 4 sigmoid activation functions. The first layer in fact represents the inputs and has no associated weights. It can be easily deduced that the processing effort will rise dramatically for a real neural network. Just to visualize the idea, a network with 76 neurons is shown in Fig. 9. This network has 5 layers with 20 neurons in each hidden layer and 16 neurons in the output layer. The number of multiplications can be calculated using (3):

$$N_{mult} = \sum_{i=1}^{N-1} (N_i . N_{i+1}) \tag{3}$$

In this equation N is the number of layers and Ni is the corresponding layer. In this case the network has 1280 multiplications and 76 activation functions. Obviously, there exist many more complicated structures. For the implementation of such a structure a bigger FPGA is to be used which means higher price, and higher power consumption.

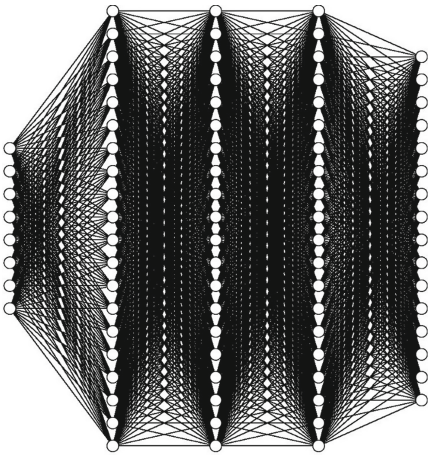


Fig. 9. Example of a five-layer artificial neural network with 76 neurons (generated with [16]).

3 Context Switching Approach

From the topology of the artificial neural network described in the previous section it can be seen that the network consists of a large amount of repeated structures – neurons, weights and biases (in the sense of repeating amounts of memory), and activation functions. Having a look at the network in Fig. 10 the mathematical description of the inputs of each layer can be derived (4–6).

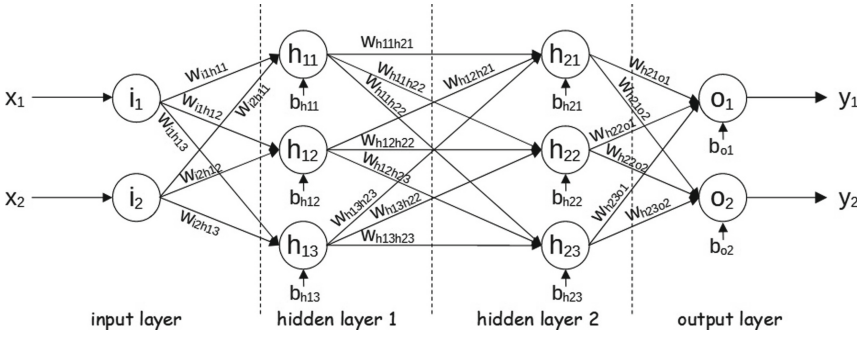


Fig. 10. Description of the connections in a four-layer neural network.

$$\begin{pmatrix} h_{11} \\ h_{12} \\ h_{13} \end{pmatrix} = \begin{pmatrix} w_{i1\mathrm{h}11} & w_{i2\mathrm{h}11} \\ w_{i1\mathrm{h}12} & w_{i2\mathrm{h}12} \\ w_{i1\mathrm{h}13} & w_{i2\mathrm{h}13} \end{pmatrix} \begin{pmatrix} i_1 \\ i_2 \end{pmatrix} + \begin{pmatrix} b_{h11} \\ b_{h12} \\ b_{h13} \end{pmatrix} = \begin{pmatrix} w_{i1\mathrm{h}11} \cdot i_1 + w_{i2\mathrm{h}11} \cdot i_2 + b_{h11} \\ w_{i1\mathrm{h}12} \cdot i_1 + w_{i2\mathrm{h}12} \cdot i_2 + b_{h12} \\ w_{i1\mathrm{h}13} \cdot i_1 + w_{i2\mathrm{h}13} \cdot i_2 + b_{h13} \end{pmatrix} \tag{4}$$

$$\begin{pmatrix} h_{21} \\ h_{22} \\ h_{23} \end{pmatrix} = \begin{pmatrix} w_{h11\mathrm{h}21} & w_{h12\mathrm{h}21} & w_{h13\mathrm{h}21} \\ w_{h11\mathrm{h}22} & w_{h12\mathrm{h}22} & w_{h13\mathrm{h}22} \\ w_{h11\mathrm{h}23} & w_{h12\mathrm{h}23} & w_{h13\mathrm{h}23} \end{pmatrix} \begin{pmatrix} h_{11} \\ h_{12} \\ h_{13} \end{pmatrix} + \begin{pmatrix} b_{h21} \\ b_{h22} \\ b_{h23} \end{pmatrix} = \begin{pmatrix} w_{h11\mathrm{h}21} \cdot h_{11} + w_{h12\mathrm{h}21} \cdot h_{12} + w_{h13\mathrm{h}21} \cdot h_{13} + b_{h21} \\ w_{h11\mathrm{h}22} \cdot h_{11} + w_{h12\mathrm{h}22} \cdot h_{12} + w_{h13\mathrm{h}22} \cdot h_{13} + b_{h22} \\ w_{h11\mathrm{h}23} \cdot h_{11} + w_{h12\mathrm{h}23} \cdot h_{12} + w_{h13\mathrm{h}23} \cdot h_{13} + b_{h23} \end{pmatrix} \tag{5}$$

$$\begin{pmatrix} o_1 \\ o_2 \end{pmatrix} = \begin{pmatrix} w_{h21\mathrm{o}1} & w_{h22\mathrm{o}1} & w_{h23o1} \\ w_{h21o2} & w_{h22\mathrm{o}2} & w_{h23o2} \end{pmatrix} \begin{pmatrix} h_{21} \\ h_{22} \\ h_{23} \end{pmatrix} + \begin{pmatrix} b_{o1} \\ b_{02} \end{pmatrix} = \begin{pmatrix} w_{h21\mathrm{o}1}.h_{21} + w_{h22\mathrm{o}1}.h_{22} + w_{h23\mathrm{o}1}.h_{23} + b_{\mathrm{o}1} \\ w_{h21\mathrm{o}2}.h_{21} + w_{h22o2}.h_{22} + w_{h23\mathrm{o}2}.h_{23} + b_{\mathrm{o}2} \end{pmatrix} \tag{6}$$

With letters *i, h, w*, and *o* they are denoted the number of the neuron, and the ingoing and outgoing signals related with that neuron, all following from the figure. Letter *b* stands for the bias.

If the layer with the most neurons and, accordingly, the largest multiple of weight matrices is chosen, then a single structure could be obtained that can be multiplied as many times as the number of layers. For the layers with a smaller number of neurons, the weighting coefficients for the missing neurons will be set to 0. The network gets the unified type shown in Fig. 11.

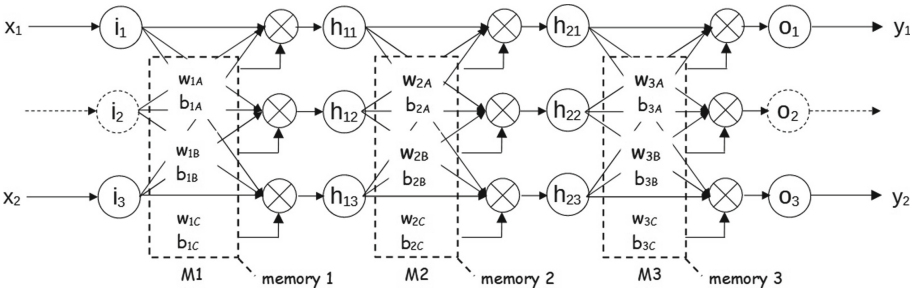


Fig. 11. Unified multi-layer structure of the neural network.

In the newly obtained structure, M1, M2, and M3 denote the matrices that unite all the weighted inputs for the neurons in each layer according to the right-hand side of

Eqs. (4–6). Taking this into consideration a single layer of the neural network can be represented as in Fig. 12.

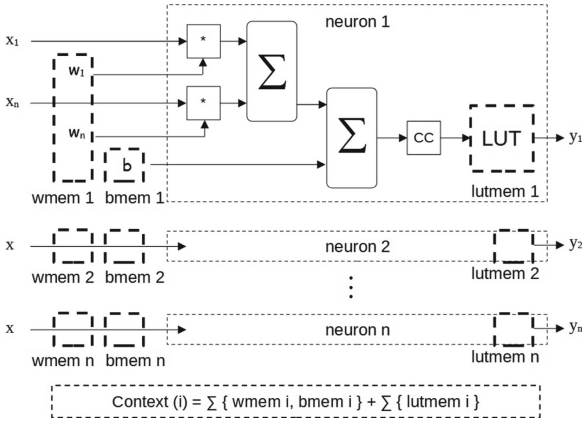


Fig. 12. Memories in the structure of a layer.

It can be seen that there are three types of memories in this structure –the weights (*wmem*), the biases (*bmem*), and the activation functions (*lutmem*). In fact, for a given network the processing elements (the neurons) have one and the same computational function, and only their configuration parameters (weights and biases) change. Moreover, the layers are connected in series so that the inputs of each layer depend on the outputs of the previous layer. In a real network, this serial dependence allows for pipelining the computation chain. It should be noted, however, that the number of layers is not that high compared to the number of neurons in a layer. Therefore, a neural network can be constructed with a single layer by simply changing the memory (context) and switching it over time. This concept is depicted in Fig. 13.

The proposed structure contains only a single computation (neuronal) layer. In this case, the "layers" differ in time, not in space. The outputs of the layer are processed by the activation functions for each neuron. They are stored in a single BlockRAM memory but occupy different memory sub-spaces. The content of the activation functions LUT can be the same or can be different for each layer depending on the type of the neural network which means that another memory sub-space is to be selected. Next, the result is stored in a register file whose purpose is to serve as a buffer for the next stage (in time). For all the layers except the last one, the buffered result is fed to the inputs of the same neuron column with the only difference that they are multiplied by weights and biases matrices from the new context – time $t + 1$. The outputs of the last layer become the outputs of the network, which is made possible with the aid of a demultiplexer. The context switching over time and the selection of the demultiplexer channels is controlled by the context switching finite state machine (FSM) whose function is very close to that of a counter. Since in a trivial neural network the inputs of each layer depend on the results from the processing of the previous one, it should be concluded that the time

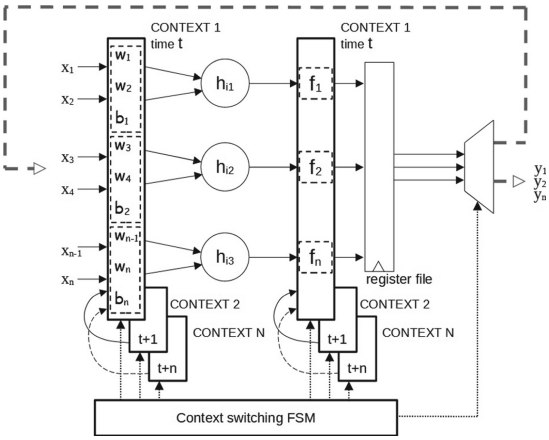


Fig. 13. Context switching organization of a single-layer neural network.

penalty, i.e. the increase in network latency will not increase significantly. This is a matter of a large volume of experiments, which are a subject of future work.

4 Future Work

The proposed new organizational approach to the structure of neural networks provides a very wide field for scientific research. The subject of future work is the construction of a multilayer neural network and the conduct of experimental comparisons of network performance, built in a classical way and through the proposed single-layer contextually switched topology. It is expected that the quality of the new network will be comparable in terms of time and it will be much better in terms of hardware resources used compared to other artificial neural networks. In addition, as future work, it is proposed to extend the presented approach in the direction of a time-multiplexed network built with only one neuron.

5 Conclusion

A new approach for building a high-performance neural network structure with only a single layer of neurons was proposed in the research article. It is based on a switching context method that employs the sharing of resources and leads to reducing the total number of neurons in the neural network. Although in an early stage, the proposed method is viable and opens a wide field for scientific research. This method will allow the development of optimized neural networks for use on hardware with constrained resources but without compromising with the computational performance.

Acknowledgments. This article is written in relation to contracts between the China University of Mining and Technology, New Bulgarian University, and the University of Mining and Geology "St. Ivan Rilski" on the subjects "Study of the control elements of a switched reluctance motor"

(MEMF-170/09.05.2022), “Joint Research and Development of key technologies for autonomous control systems”, and “Construction of International Joint Laboratory for new energy power generation and electric vehicles”.

References

1. Bodrato, M.: Towards optimal Toom-Cook multiplication for univariate and multivariate polynomials in characteristic 2 and 0. In: Carlet, C., Sunar, B. (eds.) Arithmetic of Finite Fields, vol. 4547, pp. 116–133. Springer, Cham (2007). https://doi.org/10.1007/978-3-540-73074-3_10
2. Chen, Y., Xie, X., Song, L., Chen, F., Tang, T.: A survey of accelerator architectures for deep neural networks. Engineering **6**(3), 264–274 (2020). ISSN: 2095-8099, https://doi.org/10.1016/j.eng.2020.01.007
3. Chinea, A.: Understanding the principles of recursive neural networks: a generative approach to tackle model complexity. In: Alippi, C., Polycarpou, M., Panayiotou, C., Ellinas, G. (eds.) Artificial Neural Networks, vol. 5768, pp. 952–963. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04274-4_98
4. Cieszewski, R., Linczuk, M., Pozniak, K., Romaniuk, R.: Review of parallel computing methods and tools for FPGA technology. Proc. SPIE – Int. Soc. Opt. Eng. **8903**(1), 890321 (2013). https://doi.org/10.1117/12.2035385
5. Draghici, S.: Neural networks in analog hardware - design and implementation issues. Int. J. Neural Syst. **10**(01), 19–42 (2000). https://doi.org/10.1142/S0129065700000041
6. Fields WAIFI 2007 Proceedings, Madrid, Spain, pp. 116–133, LNCS 4547 (2007)
7. Harris, D., Harris, S.: Digital Design and Computer Architecture, 2nd edn. Morgan Kaufmann, Elsevier (2013). ISBN 978-0-12-394424-5
8. Hassanein, A., El-Abd, M., Damaj, I., Rehman, H.: Parallel hardware implementation of the brain storm optimization algorithm using FPGAs. Microprocess. Microsyst. **74**, 103005 (2020). https://doi.org/10.1016/j.micpro.2020.103005
9. Hennessy, J., Patterson, D.: Computer Architecture: A Quantitative Approach. Morgan Kaufmann, Elsevier (2012). ISBN: 978-8178672663
10. IEEE Std 754-2019, IEEE Computer Society 2019. IEEE Standard for Floating-Point Arithmetic IEEE STD 754-2019, pp. 1–84 (2019). ISBN 978-1-5044-5924-2
11. Karatsuba, A.: The complexity of computations. In: Proceedings of the Steklov Institute of Mathematics, vol. 211, pp. 169–183 (1995). Translation from Trudy Mat. Inst. Steklova, pp. 186–202
12. Kastner, R., Matai, J., Neuendorffer, S.: Parallel Programming for FPGAs, The HLS Book. arxiv e-prints, https://doi.org/10.48550/arXiv.1805.03648 (2022)
13. Khan, A., Sohail, A., Zahoora, U., Qureshi, A.S.: A survey of the recent architectures of deep convolutional neural networks. Artif. Intell. Rev. **53**(8), 5455–5516 (2020). https://doi.org/10.1007/s10462-020-09825-6
14. Murahari, V., Carlos, J., Yang, R., Narasimhan, K.: DataMUX: Data Multiplexing for Neural Networks (2022). https://doi.org/10.48550/arXiv.2202.09318
15. Mutihac, R.: Mathematical modeling of artificial neural networks. In: Dopico, J., Calle, J., Sierra, A. (eds.) Encyclopedia of Artificial Intelligence, pp. 1056–1063 (2019). ISBN: 13: 9781599048499, https://doi.org/10.4018/978-1-59904-849-9.ch156
16. NN-SVG. https://alexlenail.me/NN-SVG/index.html. Accessed 30 Apr 2022
17. Pavlitov, K., Gorbounov, Y.: Multiplier based on the Xilinx Spartan II programmable logic family. E+E J. Sofia (2004). ISSN 0861-4717

18. Pavlitov, K., Gorbounov, Y.: Programmable logic in electromechanics. Technical University of Sofia (2007). ISBN 978-954-438-645-0
19. Pavlitov, K., Gorbounov, Y.: TanSig non-linear converter based on Xilinx's Spar-tan II programmable logic family. E+E J. 3–4, Sofia (2005). ISSN 0861-4717
20. Pavlitov, K.: Application of programmable logic circuits for implementation of artificial neural networks. E+E J. 9–10, Sofia 33–38 (2007). ISSN: 0861-4717
21. Poggio, T., Girosi, F.: Networks for approximation and learning. Proc. IEEE **78**(9), 1481–1497 (1990)
22. Quevillon, L., Hanks, E., Bansal, S., et al.: Social, spatial, and temporal organization in a complex insect society. Nat. Sci. Rep. **5**, 13393 (2015). https://doi.org/10.1038/srep13393
23. Ray, P.: A review on TinyML: state-of-the-art and prospects. J. King Saud Univ. – Comput. Inf. Sci. **34**(4), 1595–1623 (2022). https://doi.org/10.1016/j.jksuci.2021.11.019
24. Siegelmann, H., Sontag, E.: Analog computation via neural networks. Theor. Comput. Sci. **131**(2), 331–360 (1994). https://doi.org/10.1016/0304-3975(94)90178-3
25. Spartan-6 FPGA Block RAM Resources User Guide, UG383 (v1.5), 8 July 2011. www.xilinx.com. Accessed 01 Mar 2022
26. Van Veen, F., Leijnen, S.: The Neural Network Zoo, The Asimov Institute (2019). https://www.asimovinstitute.org/neural-network-zoo/. Accessed 01 Mar 2022
27. Wang, Z., She, Q., Ward, T.: Generative adversarial networks in computer vision: a survey and taxonomy. ACM Comput. Surv. **54**, 1–38(2020). ISSN: 0360-0300
28. Warden, P., Situnayake, D.: TinyML. O'Reilly Media, Sebastopol (2019). ISBN 9781492052043