Designing high performance, power-efficient, reconfigurable compute structures for specialized applications

V Shatravin and D V Shashev*

1National Research Tomsk State University, 36 Lenin ave., 634050, Tomsk, Russia

*E-mail: dshashev@mail.tsu.ru

Abstract. In this paper, a new approach to design high-performance and power-efficient computing structures are proposed for machine learning tasks. Such structures can be very useful in some specialized applications such as autonomous robots, mobile devices, smart sensors for the Internet of Things (IoT) and so on. This approach is based on the concept of reconfigurable homogeneous computing environments. Major advantages of this approach are discussed. The process of designing a set of elementary operations for such structures is described in detail using an example with a typical Feed-Forward Neural Network (FFNN) and its training module.

1. Introduction

Nowadays, systems based on machine learning algorithms and artificial neural networks (NN) are widely used in many applications. A reason for this is the ability of NNs to efficiently solve fuzzy, poorly formalized tasks, for example, object detection, speech recognition and synthesis, classification, future value prediction and many others.

One advantage of this ability in NNs is being able to use them in autonomous intelligent robotic systems for solving varying tasks, such as analyzing the environment, extracting useful data, choosing the right behaviour strategy. At the same time, such systems have severe restrictions on their hardware and software.

Another good example is the application of NNs in mobile devices and smart IoT sensors, which can significantly reduce the amount of data sent to the server. This not only saves time and power but also mitigates server load, as proposed by the concept of edge computing (EC).

However, machine learning algorithms have high power consumption and using common computing modules for such tasks outweighs its benefits. In this paper, one of the possible solutions to the problem is proposed, based on developing high-performance, power-efficient and reconfigurable computing structures, specialized for machine learning tasks.

2. Peculiar properties of using NNs in mobile autonomous systems

Complex robotic systems can require several neural networks with different architectures and parameters to solve all their tasks. These systems have severe restrictions regarding their performance in real-time, power-efficiency, reliability, minimum size and weight concerning its hardware and software[1]. Some systems require the ability to dynamically retrain the NN to achieve better results in changing environment or tasks. Smart IoT sensors, mobile devices and networks, based on concepts of the EC, face the same challenges.
An important stage in the development of a NN-based intelligent system is the training of NN i.e. computing its parameters (weights of connections and biases of neurons). It is an iterative, computationally complex process, which is connected with an architecture of the NN and underlying algorithms. The problem of training complexity is especially relevant for deep NNs, which consist of many layers and can include millions of neurons and connections [2].

There are many approaches to accelerate the training process like reducing the complexity of the NN by preserving acceptable accuracy, improving underlying algorithms, adding additional preprocessing of the training dataset, applying hardware accelerators and many others [3-5].

The solution proposed is specialized for machine learning tasks in reconfigurable computing structures. A key advantage of this structure is the ability to dynamically change the architecture and parameters of neural networks. Dynamic reconfiguration provides several important benefits:

- It is possible to implement several NNs with different architectures and parameters and its training modules in one structure. The structure acts like a finite-state machine, in which each state includes metadata about the structure, parameters of the specific NN and training unit from the certain set. The set of all possible NNs can be stored in the internal memory of the reconfigurable structure. The structure changes its state by an external or internal control signal;
- The implemented NNs can adapt to environmental changes or new tasks. Moreover, the structure can be altered using external control signals. This is a very useful feature in case the developers do not have any prior knowledge about the device’s environment and tasks;
- The reliability of the NN is increased as an unhealthy area of the NN can be restored.
- The structure is power efficient as it switches to a low power-consumption state, when available.

The concept of Reconfigurable Computing Environments (RCEs) is proposed as a fundamental idea for the design of such computing structures,

3. Reconfigurable computing environments

A computing structure consists of many similar simple functional cells-calculators (Figure 1) and interconnections between them. Each cell and connection can be independently, dynamically configured by a control signal. The cell can be tuned to implement some operation from a predefined basis. The connection can be enabled or disabled. RCEs has the form of a lattice with at least 2 axes of symmetry. Calculators placed in nodes of the lattice are interconnected with neighbours in the same way (Figure 2) [6–8]. To perform more complex computations, a multilayer RCEs can be used (Figure 3).

![Figure 1. Elementary calculator of RCEs [6]:](image)

- $y_i$ is the input from $i$-th neighbour,
- $f_i^M$ is the output to $i$-th neighbour,
- $x$ is the external input,
- “&” is the current logical operation.

The concept of RCEs is proposed due to the following reasons:

- Isotropy of RCEs, using some functionally complete basis of operations in cells theoretically allows implementing RCEs any desired function and in any region of the environment [9].
- The ability of RCEs cells and connections to be dynamically configured satisfies the previously described needs of the special computing structures for machine learning tasks.
• RCEs cells operate independently from each other, which provides excellent parallelization performance in RCEs calculations. Thus, providing high performance for machine learning algorithms [10].
• The high flexibility of RCEs and low-level customization allows implementing a very power-efficient solution.
• If any part of RCEs is damaged, the compute will be moved to an intact area of the environment. Moreover, calculations can be distributed throughout the RCEs area for better heat dissipation.
• The similarities in the modular structure of the RCEs and NN makes the implementation more viable.
• Homogeneity of RCEs promotes manufacturability of real devices in the form of VLSI circuits [11].

![Figure 2. RCEs [7].](image1)

![Figure 3. Multilayer RCEs [7].](image2)

Field-Programmable Gate Arrays (FPGA) can be used as a hardware platform for designing these RCEs devices. FPGA is a semiconductor device, an array of simple configurable logic elements and connections between them. The FPGA structure satisfies the basic requirements of RCEs, which allows the hardware implementation to be very close to a mathematical model.

4. Problem statement
One of the key phases in designing an RCEs is to determine a functionally complete basis of operations (the minimum set of simple operations, necessary to solve all tasks proposed to the NN) of a cell. To determine the basis, target mathematical models of a NN and its training module were chosen and decomposed into a set of operations. These operations must be reduced to elementary operations that are easy to implement in real-world applications.

As the target model of NN, FFNN with a 2–4–1 (2 input neurons, 4 hidden, single output) structure was chosen. The hidden neurons use the Rectified Linear Unit (ReLU) activation, the output uses the sigmoid activation. A model of the training unit is based on the classical supervised learning algorithm with backpropagation and the binary cross-entropy as a loss function. The models were tested on a classification task with two classes and two features and showed acceptable results.

5. Determining the basis of the training unit
At this step, we determine the set of elementary operations required by the training unit model.

A loss function is used to estimate a difference between an actual output of the NN and an expected value. A purpose of the training is to minimize the results of this function. The training unit model uses binary cross-entropy (log loss) function:

$$E = - \frac{1}{N} \sum_{i} (t_i \cdot \ln(y_i) + (1 - t_i) \cdot \ln(1 - y_i)),$$

Where,
$N$ is the number of samples in a training data set or a batch size;
t_i$ is the expected output value of NN for $i$-th sample of training set;
y_i$ is the actual output value of NN for $i$-th sample of the training set.

To minimize the loss function, the backpropagation algorithm was chosen. According to the algorithm, NN parameters are modified in the direction of the anti-gradient of the loss function. The modification of each parameter is proportional to the influence of the parameter on the result [12]. Therefore, the weights modifications can be declared as:

$$\Delta w_{ij} = -\alpha \cdot \frac{1}{N} \sum_{l} \left( \frac{\partial E}{\partial w_{ij}} \right),$$

Where,

$w_{ij}$ is the $i$-th weight of a $j$-th neuron of the output layer. Notes: the variable $j$ depends on the current layer of the neural network and is equal to the number of neurons in this layer;
$\alpha$ is the learning rate, a value of 0.01 was chosen.

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial S_j} \cdot \frac{\partial S_j}{\partial w_{ij}} = \frac{\partial E}{\partial y_j} \cdot \frac{\partial y_j}{\partial S_j} \cdot \frac{\partial S_j}{\partial w_{ij}},$$

$$S_{ij} = \sum_{f} (w_{j}'x_{fi}) + b_j,$$

Where,

$S_{ij}$ is the output of a $j$-th neuron of the output layer for an $i$-th object of the training data set;
x_{fj}$ is the $f$-th input of a $j$-th neuron;
b_j$ is the bias of a $j$-th neuron;
$K$ is the number of inputs of a $j$-th neuron.

Since RCES processes signals as a stream, it is preferable to use training batch size of 1. For the sigmoid activation, the modifications for output neurons parameters have the form:

$$\Delta w_{ij} = -\alpha \cdot (y_i - t_i) \cdot x_i,$$

$$\Delta b_j = -\alpha \cdot (y_i - t_i).$$

For the neurons of the hidden layer:

$$\frac{\partial E}{\partial S_{m}} = \sum_{h} \left( \frac{\partial E}{\partial S_{h}} \cdot \frac{\partial S_{h}}{\partial S_{m}} \right) = \frac{\partial E}{\partial S_j} \cdot \frac{\partial S_j}{\partial S_{m}},$$

$$\frac{\partial S_j}{\partial S_{m}} = \frac{\partial S_j}{\partial x_{m}} \cdot \frac{\partial x_{m}}{\partial S_{m}} = w_{mj} \cdot \frac{\partial ReLU(S_m)}{\partial S_{m}},$$

$$\frac{\partial ReLU(x)}{\partial x} = \begin{cases} 0, & x \leq 0 \\ 1, & x > 0 \end{cases},$$

Where,

$P$ is the number of neurons in the output layer, for our model $P = 1$, so $S_h = S_j$;
$S_m$ is the output of a $m$-th neuron of the hidden layer.

Therefore, an increment of the hidden layer neurons parameters is:

$$\Delta w_{km} = -\alpha \cdot (y_i - t_i) \cdot w_{mji} \cdot \frac{\partial ReLU(S_m)}{\partial S_{m}} \cdot x_{kmi},$$

$$\Delta b_j = -\alpha \cdot (y_i - t_i) \cdot w_{mji} \cdot \frac{\partial ReLU(S_m)}{\partial S_{m}},$$
Where,

\( w_{mij} \) is the weight of connection between the \( m \)-th neuron of the hidden layer and the \( j \)-th neuron of the next (output) layer for an \( i \)-th sample of training data set;

\( x_{kmi} \) is the \( k \)-th input of the \( m \)-th neuron of the hidden layer.

According to the results obtained, the training unit uses addition and multiplication. The only exception is a derivative of the ReLU activation, which is the non-linear step function. But the step function can be efficiently implemented on hardware by the comparison operation. Besides, to modify parameters only after all related values are computed, it is suggested to use a single-cycle delay block. Therefore, the basis of operations of the training unit includes addition, multiplication, comparison and single-cycle delay.

6. Determining the basis of operations of the NN model

A common feature of NN is the predominance of addition and multiplication operations. An analysis of the chosen NN shows that the only computationally difficult operation is the sigmoid activation because it involves division and exponentiation. This problem is well-known and a lot of solutions are proposed [13, 14].

In this approach, it was decided to replace the sigmoid function by its piecewise linear approximation, as suggested in [12]. In ranges \((-\infty; -5)\) and \((5; +\infty)\) the substitute function has values 0 and 1 respectively. The range \([-5; 5]\) is separated into 10 equal intervals. On each of them the sources function is replaced by the nearest simple linear function. Tests show an approximation error of less than 0.006, which is an acceptable result for this research. The hardware implementation of this approximation will be based on adders, multipliers and comparators.

Therefore, the complete basis of operations necessary for the implementation of the proposed models of FFNN-classifier and its training module includes adders, multipliers, comparators and single-cycle delays. All of these operations are well optimized and efficient in both RCEs and hardware.

7. Conclusion

In this paper, a new approach to the development of high-performance power-efficient computing structures for machine learning tasks in specialized applications is proposed. Such structures will be useful in various fields of science and engineering, especially for standalone robots, mobile devices, smart IoT sensors, nodes of edge computing networks, because rely on machine learning algorithms in their tasks.

The proposed approach is based on the concept of the RCEs, that have significant advantages like high performance, natural parallelization, power-efficiency, flexibility, reliability and manufacturability due to the theoretical principles of architecture construction shown in Section 3.

The first step in designing the RCEs is to determine the basis of operations that describe a typical NN and its training module as an example. It is demonstrated, that implementation of the FFNN-classifier and its back propagation-based training module requires only elementary operations – addition, multiplication, comparison, single-cycle delay, which are well suited for implementation on RCEs and hardware.

Further research will be aimed at building a computational structure using the considered architectural features of the RCEs and simplifications for the mathematical model of the NN. Testing this structure will allow to evaluate the numerical characteristics and compare them with theoretical assumptions.

Acknowledgments

The reported study was funded by Grants Council of the President of the Russian Federation, project number SP-3573.2019.5.
References

[1] Carrio A, Sampedro C, Rodriguez–Ramos A and Campoy P 2017 Journal of Sensors 2017(2) 296874 DOI: 10.1155/2017/3296874

[2] Sozykin A V 2017 SUSU Proc. Series: Computational Mathematics and Computer Science 6(3) pp 28–59. DOI: 10.14529/cmse170303

[3] Zhao R, Luk W and Niu X 2017 IEEE Computer Society Annual Symposium on VLSI 2017 pp 645–50 4767520 DOI: 10.1109/ISVLSI.2017.127

[4] Bourley A, Boueri J P and Choromonski K 2017 ArXiv: Mathematics, Computer Science arXiv:1706.05683

[5] Kim M and Smaragdis P 2016 ArXiv: Mathematics, Computer Science arXiv:1601.06071

[6] Shidlovskiy V and Shashev D V 2018 IOP Conf. Series: Materials Science and Engineering 363 012028 DOI: 10.1088/1757-899X/363/1/012028

[7] Shidlovskiy S V 2006 J. Comput. Syst. Sci. Int. 45 pp 282-286

[8] Shidlovskiy S V 2016 MATEC Web of Conf. 79 01014

[9] Khoroshevsky V G 2004 Int. Conf. on Parallel Computing in Electrical Engineering (Dresden, Germany: IEEE) pp 262-267

[10] Kalyayev I A 1993 Lecture Notes in Computer Science (Berlin: Springer) 694 pp 438-451

[11] Dordopulo A, Kalyaev I, Levin I and Slasten L 2011 Lecture Notes in Computer Science (Berlin: Springer) 6874 pp 272-283

[12] Haykin S 1999 Neural Networks: A Comprehensive foundation (Singapore: Pearson Education) p 842

[13] Faiedh H, Gafsi Z and Besbes K 2001 The 13 International Conference on Microelectronics (Rabat, Morocco) pp 189–192 7360708 DOI: 10.1109/ICM.2001.997519

[14] Basterretxea K, Tarela J M and del Campo I 2004 IEEE Proc. – Circuits Devices and Systems 151(1) pp 18–24 7877947 DOI: 10.1049/ip-cds:20030607