ESANN 2011 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 27-29 April 2011, i6doc.com publ., ISBN 978-2-87419-044-5.
Available from http://www.i6doc.com/en/livre/?GCOI=28001100817300.

Fisherman learning algorithm of the SOM realized in the CMOS technology

Rafał Długosz[1], Marta Kolasa[1] and Witold Pedrycz[2]

1- Faculty of Telecommunication and Electrical Engineering
University of Technology and Life Sciences
ul. Kaliskiego 7, 85-796, Bydgoszcz, Poland

2- Department of Electrical and Computer Engineering
University of Alberta
Edmonton, AB T6G 2V4, Canada

Abstract. This study presents an idea of transistor level realization of the fisherman learning algorithm of Self-Organizing Maps (SOMs) which is described in [4]. The realization of this algorithm in hardware calls for a solution of several specific problems not present in software implementation. The main problem is related to an iterative nature of the adaptation process of the neighboring neurons positioned at particular rings surrounding the winning neuron. This makes the circuit structure of the SOM very complex. To come up with a feasible realization, we introduce some modifications to the original fisherman algorithm. Detailed simulations of the software model of the SOM show that these modifications do not have the negative impact on the learning process, and helps bring significant reduction of the circuit complexity. In consequence, a fully parallel adaptation of all neurons is possible, which makes the SOM very fast.

1 Introduction

In the competitive learning of SOMs training patterns $X(l)$ being vectors in an n-dimensional space $\Re^n$ coming from a given learning set are presented to the neural network (NN) in the random fashion. At each learning cycle (l) the network computes a distance between a given pattern and the weight vectors $W_j(l)$ of all neurons of the map. The neuron, whose weights resemble a given input pattern to the highest extent becomes a winner. Various learning algorithms of SOMs have been proposed. One of the basic algorithms is the Winner Takes Most (WTM) method, which is often referred to as the classic Kohonen's rule. The adjustment of the weights is in this case realized as follows:

$$W_j(l+1) = W_j(l) + \eta(k)G(i,j,R,d)[X(l) - W_j(l)] \tag{1}$$

where $\eta(k)$ is the possible learning rate in the k^{th} training epoch, d is the distance in neuron space, R is the range of the neighborhood, W_j is the weight vector of the particular neuron in the map, and X is a given input pattern. The neighboring neurons are adjusted at different intensities that depend on the neighborhood function (NF) $G()$ and a distance in neuron space, d, between the winning, i^{th} neuron and the corresponding neighbors. In the classical approach

ESANN 2011 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 27-29 April 2011, i6doc.com publ., ISBN 978-2-87419-044-5.
Available from http://www.i6doc.com/en/livre/?GCOI=28001100817300.

the rectangular neighborhood function (RNF) is used [3]. As stressed in the literature, better results are achieved by using Gaussian neighborhood function (GNF). A realization of the GNF in the software-based NNs is simple, but the hardware realization is very complex. For this reason, we recently proposed an efficient hardware implementation of the triangular neighborhood function (TNF), which requires only a single multiplication and shifting the bits [1].

In this paper, we consider a hardware realization of another algorithm, proposed by Lee and Verleysen in [4], called the fisherman rule. Let us recall that in the classical Kohonen's algorithm all neurons that belong to the winner's neighborhood are adapted so that the connections move towards the input pattern X, as shown in (1). In the algorithm proposed in [4], in the first iteration the winning i^{th} neuron is adapted in the same way as encountered in the classic update rule:

$$W_i(l+1) = W_i(l) + \alpha_0[X(l) - W_i(l)] \tag{2}$$

For a distance $d = 0$ the α_0 parameter is equal to the term $\eta_0(k) \cdot G()$ term present in (1). On the other hand, in the fisherman rule the neighboring neurons ($d = 1, ..., R$) are trained in an iterative fashion according to formula:

$$W_d(l+1) = W_d(l) + \alpha_d[W_{d-1}(l+1) - W_d(l)] \tag{3}$$

In the second iteration, for $d = 1$, all neurons from the first ring surrounding the winner are adapted in such a way that their weights move toward the weights of the winning neuron calculated in the first iteration. The neurons from the second ring, i.e., for $d = 2$, are in the next iteration adapted towards the updated weights of the neurons of the first ring, and so forth. A detailed comparison between different learning rules, presented in [4], show that the fisherman algorithm usually leads to better results than the classic one.

In case of the software realization, in which weights of particular neurons are calculated sequentially, both algorithms come with a comparable computational complexity, so there is a sound rationale behind using the fisherman rule in many cases. On the other hand, the fisherman concept is significantly more complex in hardware realization, as the described iterative adaptation sequence has to be controlled by an additional multiphase clock. Furthermore, the control clock signals have to be distributed on the chip using additional paths. The iterative nature of the second algorithm is a source of another disadvantage. The adaptation of neurons in each following ring can be undertaken only after the adaptation in the preceding ring has been completed. This is the source of a delay that significantly slows down the adaptation phase. For the comparison, in case of our hardware realization of the classic algorithm adaptation in all neurons is performed fully in parallel [1, 2]. Eventually this is the reason why in all reported transistor-level realizations of SOMs, the classic algorithm is being used. To overcome the described problems and to make the realization of the fisherman algorithm more efficient we modified the update formula as follows:

$$W_d(l+1) = W_d(l) + \alpha_d[W_{d-1}(l) - W_d(l)] \tag{4}$$

ESANN 2011 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 27-29 April 2011, i6doc.com publ., ISBN 978-2-87419-044-5.
Available from http://www.i6doc.com/en/livre/?GCOI=28001100817300.

In comparison with the original algorithm, in which each successive ring of neighbors uses the weights $W_{d-1}(l+1)$ i.e. adapted in a given learning cycle, in the modified algorithm, here we use the weights $W_{d-1}(l)$ from the previous cycle. As a result, the neurons at each following ring do not need to wait until the adaptation at the preceding ring has been completed. This makes the overall adaptation process approximately R times faster, where R is the neighborhood range in a given epoch. For small values of the $\eta_0(k) \cdot G()$ term (that is a typical case for larger values of l) and when the GNF or the TNF is being used, this modification does not have the negative impact on the learning algorithm.

2 System level investigations

To determine the influence of the training algorithm as well as the NF on the learning process we completed a series of simulations using the software model of the NN. The comprehensive simulations have been carried out for the map sizes varying in-between 8x8 and 32x32 neurons and for different values of the initial neighborhood size, $R_{\max}$. In hardware implementation, this parameter $R_{\max}$ exhibits the main influence on the circuit complexity [1]. In what follows, we report on some selected results which can be regarded as being representative to the overall suite of experiments.

The learning process has been assessed using five criteria described in [4]. They allow for thorough evaluation of the quality of the vector quantization, and the topographic mapping. The quantization quality is assessed using two measures. One of them is the quantization error defined as follows:

$$Q_{\text{err}} = \frac{1}{m} \sum_{j=1}^{m} \sqrt{\sum_{l=1}^{n} (x_{j,l} - w_{i,l})^2} \tag{5}$$

In this formula, m is a total number of the input patterns, x denote particular elements of a given learning pattern X, and i stands for the winning neuron. A second measure used to assess the quantization quality is a percentage of dead neurons (PDN), which tells us about the ratio of inactive (dead) neurons versus the total number of neurons.

The quality of the topographic mapping is assessed using three measures. The first one is the Topographic Error defined as follows:

$$E_{\text{T1}} = 1 - \frac{1}{m} \sum_{h=1}^{m} l\left(X_h\right) \tag{6}$$

This is one of the performance measures proposed by Kohonen [3]. The value of $l(X_h)$ equals 1 when for a given pattern X two neurons whose weight vectors resemble this pattern to the highest extent are also direct neighbors in the map. Otherwise the value of $l(X_h)$ equals 0. In the best case $E_{\text{T1}} = 0$.

The remaining two measures of the quality of the topographic mapping do not require the knowledge of the input data. In the second criterion, in the first

ESANN 2011 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 27-29 April 2011, i6doc.com publ., ISBN 978-2-87419-044-5.
Available from http://www.i6doc.com/en/livre/?GCOI=28001100817300.

step, we calculate the Euclidean distances between the weights of an r^{th} neuron and the weights of all other neurons. In the second step, we check if all p direct neighbors of neuron r are also the nearest in the sense of the Euclidean distance computed in the feature space. To express this requirement in a formal manner, let us assume that neuron r has $p = |N(r)|$ direct neighbors, where p depends on type of the topology of the map. The value of the denominator in (7) is decreased, according to the reduced number of the neighbors in this case. Let us also assume that function $g(r)$ returns the value equal to the number of the direct neighbors that are also the closest to neuron r in the feature space. As a result, the E_{T2} criterion for P neurons in the map can be defined as follows:

$$E_{\text{T2}} = \frac{1}{P} \sum_{r=1}^{P} \frac{g(r)}{|N(r)|} \tag{7}$$

The optimal value of E_{T2} is equal to 1. In the third criterion, around each neuron r we construct a neighborhood in the feature space (Euclidean neighborhood) defined as a sphere with the radius:

$$R(r) = \max_{s \in N(r)} ||W_r - W_s|| \tag{8}$$

where W_r are the weights of a given neurons r, number of W_s are the weights of its s^{th} direct neighbors. Then we count the number of neurons, which are not the closest neighbors of neuron r, but are located inside $R(r)$. The E_{T3} criterion, with the optimal value equal to 0, is defined as follows:

$$E_{\text{T3}} = \frac{1}{P} \sum_{r=1}^{P} | \{s | s \neq r, s \notin N(r), ||W_r - W_s|| < R(r)\} | \tag{9}$$

The simulation results that illustrate the quality of the learning process completed on the basis of the five criteria described above are shown in Figure 1. Data are divided into P classes (centers), where P is equal to the number of neurons in the map. Each center is represented by an equal number of the learning patterns. The centers are placed regularly in the input data space. To achieve comparable results for different map sizes the input space is fitted to input data. For example, for the 8x8 map the inputs are in the range of 0 to 1, while for the map of size 16x16 in the range of 0 to 2, and so forth. As a result, in all cases the smallest value of the Q_{err} equals 16.2e-3, while the optimal values of remaining parameters (PDN / ET1 / ET2 / ET3) equal 0/0/1/0, respectively. The results reported in Figure 1 show that for particular map sizes and particular NFs the best results are achievable for different values of $R_{\max}$, usually small even for large maps. Note that the results for the modified algorithm are comparable or in some cases even better that those obtained for the original algorithm. This is visible in Fig. 1 (a, f) i.e., for small values of the learning rate η, as expected. This conclusion is important from the hardware realization point of view.

ESANN 2011 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 27-29 April 2011, i6doc.com publ., ISBN 978-2-87419-044-5. Available from http://www.i6doc.com/en/livre/?GCOI=28001100817300.

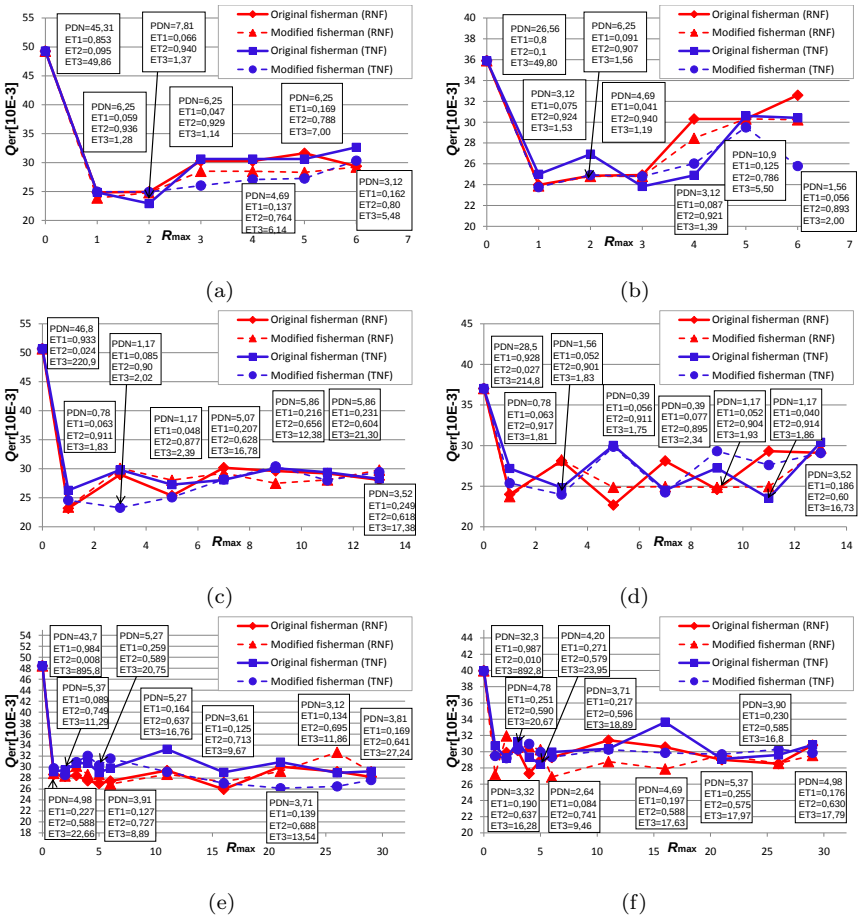

Fig. 1: Simulation results for: (a,b) 8x8, (c,d) 16x16, (e,f) 32x32 neurons in the map, for (a,c,e) $\eta = 0.9$, (b, d, f) $\eta = 0.3$ at the beginning of the learning process

3 Hardware implementation

In this section, we only briefly present the idea of the proposed hardware realization of the fisherman algorithm. The new solution is based on a fully parallel SOM, described earlier in [1, 2]. In the proposed circuit, particular weight vectors $W_j(l)$ are transferred from one ring of neighbors to the other in the same way, as the signal R in our previous circuit [2]. The block diagrams of the original, as well as the modified fisherman algorithms realized in hardware are shown in Figure 2. The ADF and the ADFM blocks are responsible for the adaptation in both algorithms, according to (3) and (4), respectively. In the first case (top diagram) the multiphase clock {ck0, clk1, clk2, ...} is used to control the

ESANN 2011 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 27-29 April 2011, i6doc.com publ., ISBN 978-2-87419-044-5.
Available from http://www.i6doc.com/en/livre/?GCOI=28001100817300.

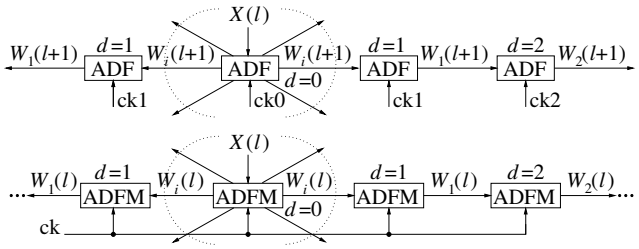

Fig. 2: Block diagram of the SOM realized in hardware with: (top) the original fisherman algorithm and (bottom) the modified algorithm.

adaptation sequence as described earlier. The duration of a single clock phase should be sufficient to enable adaptation of all weights of particular neurons in a given ring. In the CMOS 0.18 μm technology, this time equals approximately $n \cdot 5$ ns, where n is the number of the network inputs. As a result, the overall adaptation process in a single cycle takes $R \cdot n \cdot 5$ ns. In the second (bottom) case all neighbors are adapted in the same time (the ck clock phase), and the overall adaptation process in all neurons takes only $n \cdot 5$ ns.

4 Conclusions

A very fast hardware realization of the fisherman learning algorithm of the SOM has been proposed. To make such realization feasible, we have introduced some modifications to the original algorithm. The simulations performed with the software model of the SOM show that both in the original and the modified algorithms the learning quality is comparable. The modified algorithm allows for significant reduction of the circuit complexity, as well as a fully parallel adaptation of all neighboring neurons. As a result, a single learning cycle takes only 100 ns in the CMOS 0.18μm technology. As a result, the SOM with the fisherman learning rule is able to achieve the throughput of even 10 million patterns per second, independently on the size of the map. In a nutshell, this result implies that such NN can be even thousand times faster than the SOM implemented on a standard PC, while consuming 50-100 times less energy.

References

[1] R. Długosz, M. Kolasa, "CMOS, Programmable, Asynchronous Neighborhood Mechanism For WTM Kohonen Neural Network", 15$^{\text{th}}$ *International Conference Mixed Design of Integrated Circuits and Systems* (MIXDES), Poznań, Poland, June 2008, pp. 197–201

[2] R. Długosz, M. Kolasa, W. Pedrycz, ''Programmable Triangular Neighborhood Functions of Kohonen Self-Organizing Maps Realized in CMOS Technology", *European Symposium on Artificial Neural Networks* (ESANN), Bruges, Belgium,2010, pp.529-534

[3] T. Kohonen, *Self-Organizing Maps*, third ed. Springer, Berlin, 2001

[4] J. A. Lee, M. Verleysen, "Self-Organizing Maps with Recursive Neighborhood Adaptation", *Neural Networks*, Vol. 15, Issues 8-9, October-November 2002, pp. 993–1003