Advanced Training Set Generator for Use in Self-Organizing Neural Networks

Natalia Pasturczak, Rafał Długosz

Abstract— **Success in training of artificial neural networks (ANNs) depends on properly selected training data set. The problem is not trivial, especially in situations in which the structure of data to be classified by the neural network (NN) after completing the learning phase is not known. To cope with this problem, one of the solutions is to conduct comprehensive tests of the ANN with training it with data sets that differ in the distribution of the learning patterns in the input data space. In order to prepare such sets, appropriate tools are required that introduce the ability of a broad parameterization. In the proposed work, an advanced data generator has been developed, which based on a specific variability of particular predefined parameters, generates a collection of learning data sets. The obtained sets are then used to optimize the learning process of selected algorithms of self-organizing ANNs. The NN then automatically retrieves subsequent data sets from specific location, and as a response generates reports with an assessment of the quality of the learning process. The aim of the proposed investigations is to optimize the structure of selected ANNs, so as to obtain their minimum complexity, important in case of their hardware implementation.**

I. Introduction

The aim of this work was to implement a synthetic data generator that is able to produce multidimensional numerical datasets which could be used to optimize the learning process of the ANN, especially self-organizing neural networks, often called Self Organizing Maps (SOMs).

An interest in synthetic data generation is related to the demand for large amounts of data for validation and training artificial intelligence models. The ability to generate data with predefined features allows to replicate specific data properties, simulate general trends, test less typical cases and to introduce a parameterization to particular data classes. Generating real data in a sufficient amount and variability is expensive and time-consuming, while synthetic data is quite cheap in production. Certainly, a more convenient and faster solution is to use the generator.

N. Pasturczak in with Bydgoszcz University of Science and Technology, Faculty of Telecommunication, Computer Science and Electrical Engineering ul. Kaliskiego 7, 85-796, Bydgoszcz, Poland, E-mail: nalia3486@gmail.com

R. Długosz is with Bydgoszcz University of Science and Technology, Faculty of Telecommunication, Computer Science and Electrical Engineering ul. Kaliskiego 7, 85-796, Bydgoszcz, Poland, and with Aptiv Services Poland ul. Podgórki Tynieckie 2, 30-399, Kraków, Poland E-mail: rafal.dlugosz@gmail.com and rafal.dlugosz@aptiv.com

The proposed solution allows for generation of data sets that may differ in the number of data classes, the number of learning patterns in each class, as well as the distribution of learning patterns in particular classes. In the proposed solution, the location of data classes is also one of important parameters. This allows for precise tests to be carried out to show how well the ANN is able to distinguish particular data classes from each other. The implemented tool allows once generated data classes with a specific distribution of training patterns to be appropriately placed in the input data space. As a result, particular classes may be located far from each other, but may also partially overlap with each other. Obtaining such training data under real conditions is very difficult. Particular data classes may be generated based on normal or uniform distribution of the learning patterns.

The motivation behind the presented works on the tool for generating training data are the investigations on the ability of hardware implementation of self-organizing neural networks in CMOS technology. In this case, once implemented, the architecture of the neural network in the chip cannot be modified. For this reason, at the design stage of such a network, comprehensive investigations are needed to verify the correctness of the selection of basic network parameters, such as the number of neurons for a given problem, the number of weights in particular neurons, the network topology, the type of the neighborhood function, etc.

The paper is organized as follows. Next section briefly presents the basics of classic self-organizing networks that are tested with the training sets generated with the realized tool. Selected solutions behind the proposed tool are presented in the following section. Then we present examples of resultants training data sets generated by the proposed tool. Conclusions are included in the last section.

II. Self-organizing neural networks

Self-organizing maps (SOMs) that belong to artificial neural networks and systems, may be trained using various learning rules. They include such algorithms as Winner Takes All (WTA), Winner Takes Most (WTM) proposed by Kohonen in [1], Neural Gas, a "fisherman" algorithm proposed by Lee and Verleysen in [2] with some modifications proposed in [3]. In this Section we briefly present basics of the WTM algorithm. We fo-

cus on this solution, as the remaining algorithms have evolved from this solution in different ways or are a special case of it.

In all these algorithms, a set of learning patterns $X(l)$ that are vectors in an n-dimensional space $\Re^n$ is presented to the ANN in a random fashion. For each learning pattern X a new, l^{th}, learning cycle starts. It is composed of several stages. First the ANN computes a distance between a given pattern and the weights vectors $W_j(l)$ in particular neurons belonging to the ANN. The neuron, whose weights vector is the closest to a given learning pattern becomes a winner. Following steps in the learning cycle differ between particular learning algorithms. In the WTA algorithm only the winning neuron is allowed to adapt its weights. The situation is different in the WTM algorithm, in which, additionally, the neighboring neurons of the winner can adapt their weights, but with different intensity, depending on their topological distance from the winner. In this case the adaptation process is performed in accordance with the following formula:

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

The parameter $\eta(k)$ is in this formula a learning rate. It directly defines the allowable intensity of modification of the weight values. The neighboring neurons are trained with different intensities that results from the applied neighborhood function $G()$, depending on a distance $d(i,j)$, between the winning neuron, i, and a given j^{th} neuron in the SOM.

A. Neighborhood function

The neighborhood function (NF) is one of important parameters that impacts both the quality of the learning process and the hardware complexity of the implemented ANN. Kohonen in his works proposed a rectangular NF (RNF) [1], which can be expressed as follows:

$$G(R,d(i,j)) = \begin{cases} 1 & \text{if } d(i,j) \le R \\ 0 & \text{if } d(i,j) > R \end{cases} \quad (2)$$

In the formula above, R is the neighborhood size that decreases after during the learning process. In this solution, all neighboring neurons whose distance does not exceed R are trained with equal intensities. All others neurons do not change their weights. Our investigations show that the RNF is sufficient only in case of relatively small SOMs composed of 100-250 neurons. Much better results may be achieved in case of using the Gaussian NF (GNF) [4], which can be expressed as follows:

$$G(R,d(i,j)) = \exp\left(-\frac{d^2(i,j)}{2R^2}\right) \quad (3)$$

The minus sign in the equation above results from the fact that the value of the $G()$ function decreases with increasing the distance between the winning neuron and the neuron for which the function is computed.

In case of the hardware implementation, the GNF is not an optimal solution, due to large hardware complexity. It requires complex mathematical operations that include division and exponential operations [5]. For this reason, in our former works we proposed a substitution of the GNF with a triangular NF (TNF). This solution is much simpler in the hardware implementation, as it requires only a single multiplication and shifting the bits [5]. This function may be expressed as follows:

$$G(R,d(i,j)) = \begin{cases} -a(\eta_0)\cdot(R-d(i,j)) & \text{for } d(i,j) \le R \\ 0 & \text{for } d(i,j) > R \end{cases} \quad (4)$$

where $a()$ is the steepness of the triangular function and η_0 is the learning rate of the winning neuron. The values of both these parameters are reduced toward zero during the learning process. The TNF allows to reach the learning quality comparable with the GNF, while its hardware complexity is reduced by 95 % when compared to the GNF [5].

B. Topology of the SOM

All the NFs described above can be used with any network topology. The topology is yet another parameter that determines the number of neighboring neurons for each neuron in the SOM. Typical topologies include the ones with 8, 6 and 4 neighboring neurons.

In general, to summarize this part of work, various training sets are needed, additionally strongly parameterizable that allows to test various combinations of the described parameters with different number of neurons in the SOM. For this reason, it was necessary to create a tool that allows to easily generate such data sets.

III. Proposed tool for data generation

In the proposed tool, by configuring a set of parameters it is possible to generate a collection of customized training data sets. These parameters include the type of generated data distribution (normal or uniform), classes shape (e.g. square, circle, ring), data range, number of clusters, dimension of data set, data size, distance between particular classes, as well as coordinates of the positions of the central points of particular classes. Parameters are set in files so it is possible to run program multiple times and get a lot of datasets with specific properties. Of course, all parameters are optional, however, by manipulating the listed parameters it is possible to generate large amount of data with the expected features and shapes with wide range of difficulty levels to be classified by ANNs. For some shapes and distributions there are also additional parameters as radius, peak and spread of the data.

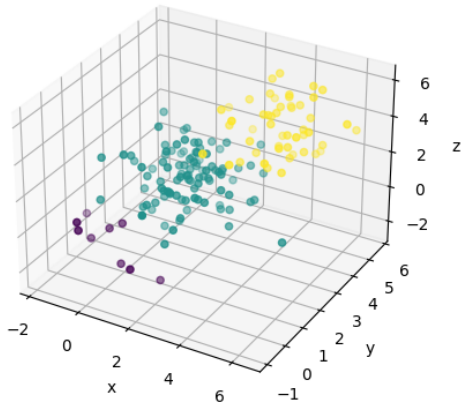

Fig. 1. Visualization of a synthetically generated dataset from normal distribution

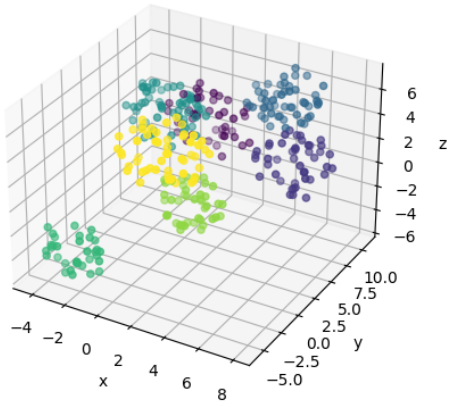

Fig. 2. Visualization of a synthetically generated dataset in a shape of sphere with 7 centers.

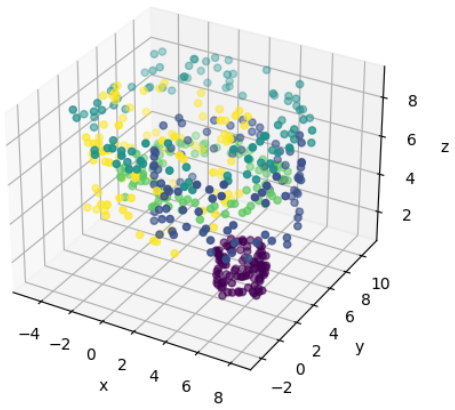

Fig. 3. Visualization I of a synthetically generated dataset in a shape of a hollow cylinder with 5 centers.

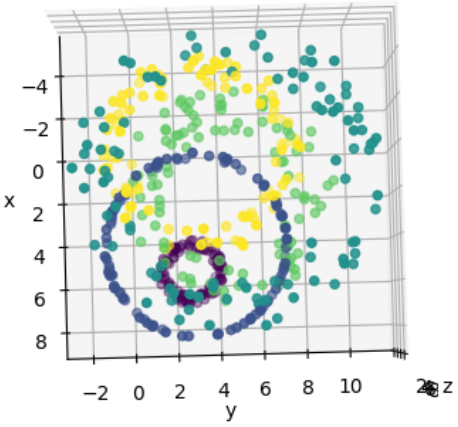

Fig. 4. Visualization II of a synthetically generated dataset in a shape of a hollow cylinder with 5 centers.

To generate clusters that have particular shapes as (hyper)sphere or hollow (hyper)cylinder additional modifications are used.

Random points in a multi-dimensional sphere are uniformly generated based on "'Random Points in an n-Dimensional Hypersphere" [6]. In this case, first of all, random function generates normal distributed points. Then the incomplete gamma function is used to map generated points radially within given radius and given sphere center, so points are uniformly distributed.

In order to generate hollow n-dimensional hollow cylinder two values are needed: r_{inner} and r_{outer}. These two parameters may be customized or randomly generated. The r_{inner} parameter is the length of the line from the center to its inner edge and analogously the r_{outer} parameter is the length of the line from the center to its outer edge. The randomly generated value from 0.1 to 1 is multiplied by the distance between the outer and the inner radius, as follows:

$$r = \sqrt{\text{rand}(0,1) \cdot (r_{\text{outer}}^2 - r_{\text{inner}}^2) + r_{\text{inner}}^2} \qquad (5)$$

The Cartesian coordinates x and y are then converted into the spherical ones:

$$\theta = \text{rand}(0,1) \cdot 2 \cdot \pi \qquad (6)$$

As a result, two coordinates of points (learning patterns) are generated:

$$\text{point} = (\text{center}_x + r \cdot \cos(\theta), \text{center}_y + r \cdot \sin(\theta)) \qquad (7)$$

If the points are produced in more than two dimensions then the remaining points' coordinates are generated using the uniform distribution without the conversion.

IV. Implementation and results

In this section we present selected results obtained by the use of the developed tool for learning data generation. Data can be generated in any space. Here we have limited our analysis to three dimensions, to allow for better visualization of the results. Selected examples

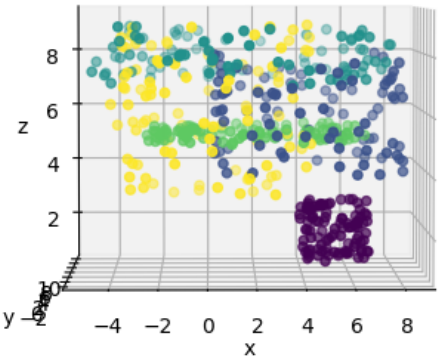

Fig. 5. Visualization III of a synthetically generated dataset in a shape of a hollow cylinder with 5 centers.

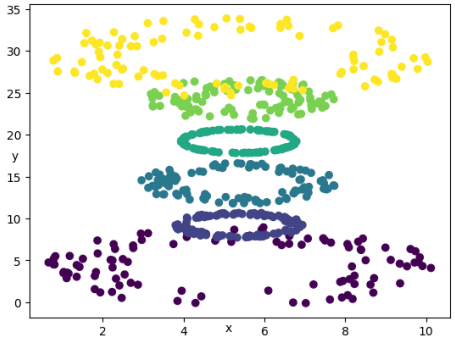

Fig. 6. Visualization of a synthetically generated dataset in a shape of hollow cylinders with 6 centers.

are shown in Figs. 1 to 6. The implemented tool allows to generate any number of data classes. For a better visualization, we have limited the number of classes to small values. Each of them is additionally marked with a different color.

Fig. 1 presents a visualization of a synthetically generated data set from the normal distribution. In this example, learning data were generated in the 3-dimensional data space. Since the centers of the classes are located close to each other, the three visible clusters are overlapping. In this example, different numbers of the learning patterns are in each class. One of them is much smaller than the remaining two, so it is harder to distinguish all three clusters properly. Such data are usually particularly difficult to distinguish by a neural network.

Fig. 2 shows a visualization of a synthetically generated data set. In this case, seven spherical-shaped data classes were generated. Data were generated in the 3-dimensional data space. Radii of particular classes are random values, varying from 2 to 3. The coordinates of the centers of particular classes are also random values. Some clusters are visibly separated from each other but there are overlapping.

Figs. 3, 4 and 5 show a visualization of a synthetically generated data set in the shape of five hollow cylinders with different radii. In this case, data were again generated in the 3-dimensional data space. Radii of the inner and outer parts of the cylinders were in this case specified, differing for each data cluster. Distances between clusters were not set. In this example, the clusters are overlapping.

Fig. 6 provides a visualization of a synthetically generated data set in the shape of six hollow cylinders with different radii. Data are generated in the two-dimensional space. The distance between the sets of points was set to five for the y axis. The radii are random and different for each hollow cylinder. Hundred patterns were generated for each cluster.

V. Conclusions

In this paper, we presented a tool whose task is to generate training data sets for self-organizing artificial neural networks. The tool allows to modify a number of various parameters that in turn allows to generate a different number of data classes that differ in specific properties. Particular features may be random, but it is also possible to predefine their values. Thanks to this, it is possible to obtain data classes of different shapes, different scattering of the positions of particular learning patterns around a given shape, different distances between data classes, etc. This tool was built to support the verification of neural networks implemented in specialized chips, at the stage of their design and selection of their parameters.

The introduced parameterization, that enables a step change of particular features of data classes, allows for comprehensive verification of the behavior of the ANNs. The data sets obtained in this way can be used in comparative tests of various learning algorithms of self-organizing neural networks.

References

[1] T. Kohonen, *Self-Organizing Maps*, 3 rd edition, Berlin, Springer, 2001.

[2] J. Lee and M. Verleysen, "Self-organizing maps with recursive neighborhood adaptation," *Neural Networks*, vol. 15, no. 8-9, pp. 993–1003, 2002.

[3] R. Dlugosz, M. Kolasa, and W. Pedrycz, "Fisherman learning algorithm of the SOM realized in the CMOS technology," in *Proc. ESANN 2011 – 18th European Symp. Artificial Neural Networks, Computational Intelligence and Machine Learning*, Bruges, Belgium, Apr. 2011, pp. 171–176.

[4] R. Dlugosz, M. Kolasa, W. Pedrycz, and M. Szulc, "Parallel programmable asynchronous neighborhood mechanism for kohonen SOM implemented in CMOS technology," *IEEE Transactions on Neural Networks*, vol. 22 (12), pp. 2091–2104, December 2011.

[5] M. Kolasa, R. Dlugosz, W. Pedrycz, and M. Szulc, "A programmable triangular neighborhood function for a kohonen self-organizing map implemented on chip," *Neural Networks*, vol. 25, pp. 146–160, January 2012.

[6] Roger Stafford, "Random Points in an n-Dimensional Hypersphere", MATLAB Central File Exchange. 12/23/05