Self-Organizing Maps with a Single Neuron

George M. Georgiou and Kerstin Voigt

***Abstract*—Self-organization is explored with a single complex-valued quadratic neuron. The output is the complex plane. A *virtual* grid is used to provide desired outputs for each input. Experiments have shown that training is fast. A quadratic neuron with the new training algorithm has been shown to have clustering properties. Data that are in a cluster in the input space tend to cluster on the complex plane. The speed of training and operation allows for efficient high-dimensional data exploration and for real-time critical applications.**

I. Introduction

SELF-ORGANIZING maps have been extensively studied and used in many applications. [1]–[3] In their basic form, they consist of an array of neurons which could be arranged as a rectangular grid. During training each neuron is presented with each input vector $X \in \mathbb{R}^n$ in turn. There is a competitive process during which a winner neuron c is found of which its weight vector $M_c \in R^n$ most closely matches X:

$$c = \underset{i}{\operatorname{argmin}}(\|X - M_i\|). \tag{1}$$

The weights of the winner neuron c and those of the neurons within a neighborhood N_c on the neuron grid are modified according to this rule:

$$M_j(t+1) = \begin{cases} M_j(t) + \alpha(t)[X(t) - M_j(t)] & \text{if } j \in N_c(t) \\ M_j(t) & \text{if } j \notin N_c(t), \end{cases} \tag{2}$$

where t is the discrete time parameter, α is the learning rate, a small real value which decreases with time, and N_c is a set of neuron indices in the topological neighborhood around the winning neuron with index c. The size of neighborhood N_c decreases with time. After the learning process has iterated through the input set of vectors many times, the neurons on the grid are tuned to respond to different areas of the input space. In the normal operation, an input X is presented to the neurons on the grid and the winning neuron is identified as the response. Vectors in close proximity in the input space will either cause response from the same neuron or from a group of neurons within close topological proximity to each other on the grid. Thus, the two-dimensional grid provides a topologically consistent map of the input space. Training usually takes many iterations. The process is governed by many parameters, e.g the initialization of weights, the schedule of decreasing of the neighborhood N_c, the size of the grid of neurons, and the decreasing learning rate $\alpha(t)$. For example, a simple 2-D map from the input space to a 2-D grid of neurons required $70,000$ iterations.

George M. Georgiou and Kerstin Voigt are with the School of Computer Science and Engineering, California State University, San Bernardino, CA 92407, USA (email: {georgiou, kvoigt}@csusb.edu).

Given the 2-D nature of complex numbers, complex-valued neurons will be used in the introduced self-organizing map. Complex-valued neural networks is an active area of research. [4]–[10]

II. Complex-valued neurons

It is desired that we have a 2-D output space so that visualization of the data in the output space is possible. Whereas two real neurons can be used to obtain the two coordinates at the output, we chose to use a single complex neuron. This choice is motivated by the applicability to both real and complex data, and also, in the case of the quadratic neuron (below), by our ability to use results related to the field of values of the weight matrix. [11], [12] For example, if the input vectors are normalized, the output is confined to the field of values, which is useful a priori information.

A. The complex linear neuron

The complex linear neuron was introduced in [13] in the context of the LMS (Least-Mean Square) algorithm. The input is $X \in \mathbb{C}^n$ and the weight vector of the neuron is $W \in \mathbb{C}^n$. The output of the neuron is $y = W^T X$, the superscript $(\cdot)^T$ is the matrix (vector) transpose and $y \in \mathbb{C}$ is a complex scalar. The error is defined as $\varepsilon_j = d_j - y_j$, where $d_j \in \mathbb{C}$ is the desired valued for a given input vector X_j. The mean-square error is

$$E = \frac{1}{2} \sum_{j=1}^{n} \varepsilon_j \overline{\varepsilon}_j. \tag{3}$$

The gradient of the real function E with respect to W for an input vector X, omitting the subscripts, is

$$\nabla_W E = -\varepsilon \overline{X}. \tag{4}$$

The overbar signifies complex conjugation. Hence, the online learning rule that minimizes the mean-square error is the following:

$$\Delta W = \alpha \varepsilon \overline{X}. \tag{5}$$

The small real value α is the learning rate. The training of the neuron is done as in the real case: each input is presented in turn, and the weight vector is adjusted according to Equation (5).

B. The complex quadratic neuron

The real quadratic neuron has been briefly discussed [14]. For input vector $X \in \mathbb{C}^n$, the scalar complex output $y = X^* A X$, where $A \in \mathbb{C}^{n \times n}$, the weight matrix. The superscript $(\cdot)^*$ denotes the conjugate transpose. The output can be written

as a summation of the individual terms that involve the components of X and A:

$$y = \sum_{j=1}^{n} \sum_{k=1}^{n} \overline{x}_j x_k a_{jk}. \tag{6}$$

Using the same mean-square error as in Equation (3), the gradient of the error E with respect to weight a_{jk} is

$$\nabla_{a_{jk}} E = -\varepsilon \overline{x}_j x_k, \tag{7}$$

or in vector format:

$$\nabla_A E = -\varepsilon \overline{X} X^T. \tag{8}$$

The gradient descent learning rule that minimizes the mean-square error is

$$\Delta A = \alpha \varepsilon \overline{X} X^T, \tag{9}$$

where α, as always, is a small real value, the learning rate.

The quadratic neuron has certain properties that other commonly used neurons, e.g. ones with sigmoid transfer functions, do not have. For example, the output can be scaled and rotated by simply multiplying the weight matrix A by $ae^{i\theta}$, where a is the scaling factor, a real scalar, and θ is the angle of rotation. If the input vectors are normalized, translation of the output y by a complex scalar z can be achieved by replacing A with $A + zI$:

$$X^*(A + zI)X = X^*AX + zX^*X = y + z. \tag{10}$$

These are desirable properties in cases where the neuron is already trained and yet transformation of the output is needed.

C. Mapping properties of the quadratic neuron

To explore the mapping properties of the quadratic neuron, a 2-dimensional data set arranged on a grid was generated. The rectangular data grid has diagonal points $(-1, -1)$ and $(1, 1)$, and a total of $21 \times 21 = 441$ data points were generated. Each of the Figures 1, 2, 3 and 4, represents the output of the quadratic neuron for same 2-dimensional data set. In each case, the weight matrix A was randomly assigned values. The figures show that the general structure of the input data is preserved in the output. In the common case, the neurons on the grid in the Kohonen self-organizing maps are randomly initialized, the structure of the input data needs to be discovered through training. The quadratic neuron starts with the advantage of having inherent topological properties.

III. The new self-organizing algorithm

Instead of having a grid of neurons where each is tuned to respond to an area of the input space, in the new self-organizing map we only have a single complex neuron. This neuron can be chosen to be either the linear one in Section II-A or the quadratic one in Section II-B. The output is observed on the complex plane which provides a continuum in 2-D as opposed to the discrete grid of neurons in the usual self-organizing maps. This map can be used for data exploration, i.e. to visually gauge the structure of the input data which is normally high dimensional and not amenable to visualization.

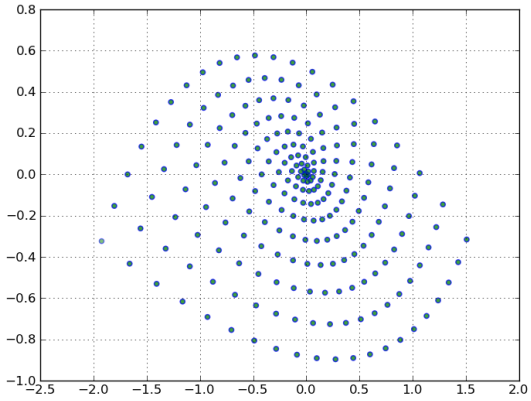

Fig. 1. The output of the grid data.

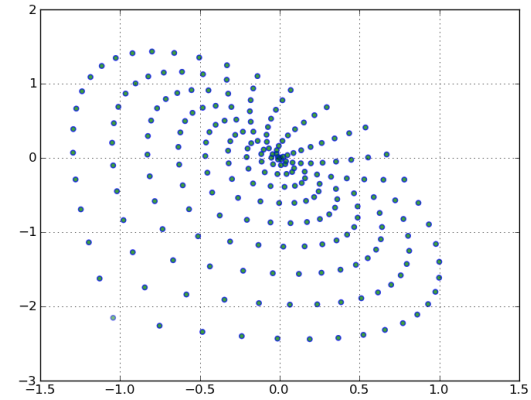

Fig. 2. The output of a grid data.

Since the activation functions of both neurons are continuous functions of the input, proximity in the input space around a small neighborhood of a point will be preserved in the output. When the quadratic neuron is used, there is experimental evidence that clusters of data in the input space are preserved in the output space. This is the desired functionality of the introduced new neural structure and algorithm.

We begin the description of this new algorithm by defining a 2-D grid on the output space of the neuron, which is the complex plane. The grid, unlike the usual self-organizing maps, is devoid of neurons. An example of such grid appears in Figure 5. The intersection points of the grid serve as desired values. For a given input vector X, the output y is computed. The desired output d to be used in the learning algorithm is defined as the nearest intersection point of the grid lines. In other words, the output snaps onto the nearest corner on the grid. In practice, the snapping function snap can be defined in terms of a rounding function round, which rounds to the nearest integer and it can be found in many programming languages, such as Python. For example if one chooses a grid with grid lines spaced at a distance 0.1 in both directions, the

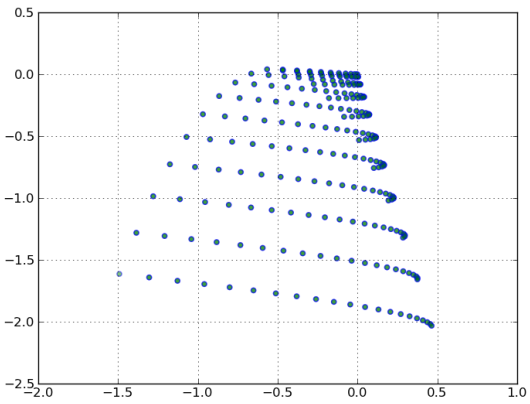

Fig. 3. The output of a grid data.

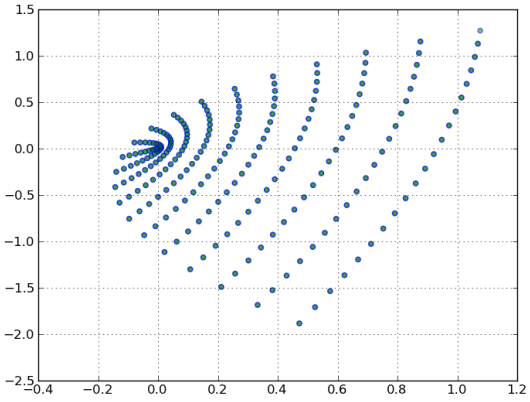

Fig. 4. The output of a grid data.

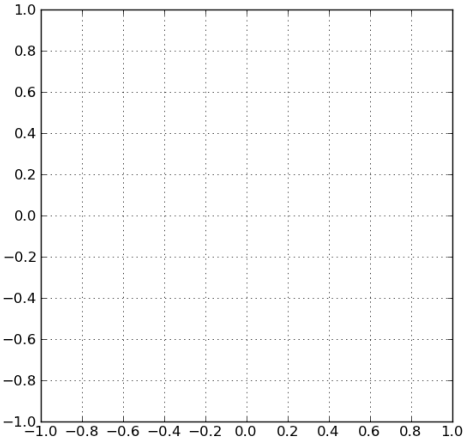

Fig. 5. The *virtual* grid on the output space.

snap function can be defined as follows:

$$\begin{aligned} d &= snap(y) \\ &= \frac{\text{round}(10\,\Re(y))}{10} + i\,\frac{\text{round}(10\,\Im(y))}{10} \\ &= (\frac{\text{round}(10\,\Re(y))}{10}, \frac{\text{round}(10\,\Im(y))}{10}), \end{aligned} \tag{11}$$

where $i = \sqrt{-1}$, and the functions $\Re$, $\Im$ return the real and the imaginary parts of their argument, respectively. Since the grid exists only implicitly through the $snap$ function, it can be called *virtual* grid. It can be limited to a specific rectangular, or square area. In that case, if the output y falls outside the grid, the corresponding value d will be a point on the boundary of the grid.

After randomly initializing the weights of neuron, training proceeds as follows:

1) Present input vector X to the neuron and compute the output y.
2) Compute $d = \text{snap}(y)$, the nearest intersection of grid lines. This is the desired output for input X.
3) Update the weights of the neuron using Equation (5) or (9), depending on which neuron is used.

In each iteration, the above steps are carried out for each of the input vectors in the data set. The process is repeated for a number of iterations. Label information of the data is not used at any time. The learning rate could be decreasing with time, as is the case with the Kohonen self-organizing maps, or there could be a maximum number of iterations, or there could be a given minimum mean-square error over all inputs that, when reached, stops the process. Normally the choice rests on experimentation and experience.

The basic mechanism by which clustering is achieved with the new algorithm could be explained as follows. A cluster in the input space will map to a similar region in the output space due to the continuity of the activation function. Other data vectors, not belonging to the cluster, may map to the same region. Since the cluster data are more numerous, they will cause the weights of the neuron to move so that their outputs remain close together in the same region of the grid. At the same time, the less numerous data will not have as much influence on the weights and will move away from the cluster in the output space.

IV. Results

A. The Iris data set

Clustering tests were conducted on the well known iris data set. There are 150 4-dimensional (real) data vectors, which belong to three different classes. [15] In these experiments, the quadratic neuron was used with virtual grid lines spaced at 0.01. The grid onto which the output was allowed to snap was limited to in the area of a square of size 2 centered at the origin. A point outside the rectangle was snapping on a point on the near boundary of the grid. For readability, grid lines in the figures are spaced at a larger distance. Run 1 and Run 2 are representative of numerous runs, in which the general tendency to clearly form the three clusters was observed.

For both runs, the real and imaginary parts of the entries in the weight matrix were initialized with random values in the interval $[-0.01, 0.01]$. The learning rate was set to $\alpha = 0.001$. The output for the data is color coded based on the class to which it corresponds. For Run 1, the initial output is shown in Figure 6 and after 10 iterations through the data set in Figure 7. The corresponding figures for Run 2 are Figure 8 and 9.

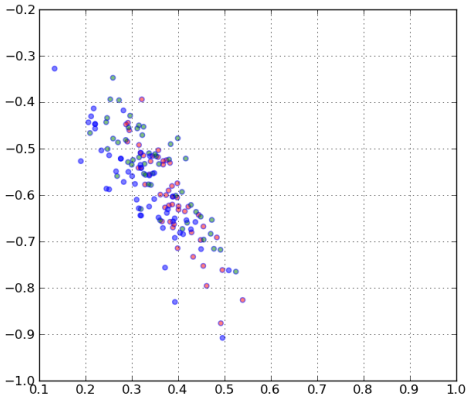

Fig. 6. Run 1: The initial distribution of the iris data.

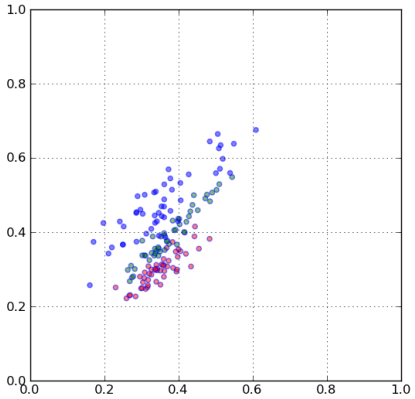

Fig. 8. Run 2: The initial distribution of the iris data.

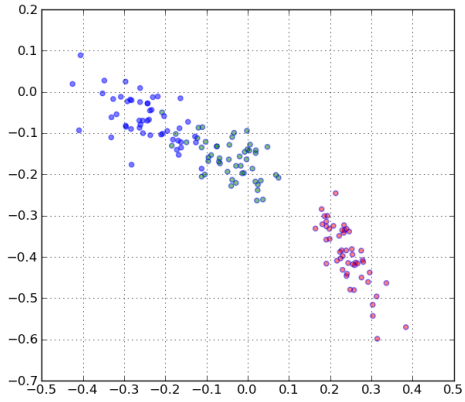

Fig. 7. Run 1: After 10 iterations the clusters are clearly formed.

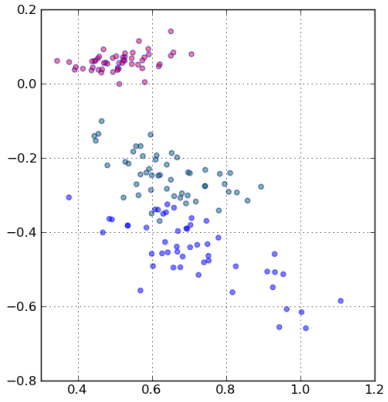

Fig. 9. Run 2: After 10 iterations the clusters are clearly formed.

B. An artificial data set

To further test the algorithm, a data set was generated in $\mathbb{R}^6 \subset \mathbb{C}^6$ that had two classes. Each class had 50 points (vectors). The classes were not overlapping, i.e. they were linearly separable, and each was uniformly distributed in a unit sphere. Of interest in this experiment was the effect of the learning rate in the separation process of the two clusters and also the convergence of the algorithm. Figure 10 shows the initial distribution of points of two classes at the output of the quadratic neuron. As it can be seen in the figure, the two classes are initially thoroughly mixed together. The new self-organizing algorithm was applied as described in the three experiments below In each case, a different learning rate α scheme was used. In these experiments, an infinite virtual grid was used. In other words, the desired output was simply the output of the $\mathrm{snap}(\cdot)$ function; it was not limited in a finite rectangular area. The virtual grid defined via the $\mathrm{snap}(\cdot)$ function had distance between grid lines, horizontal and vertical, 0.1. In all cases, the online version of the learning algorithm was used as opposed to the batch method.

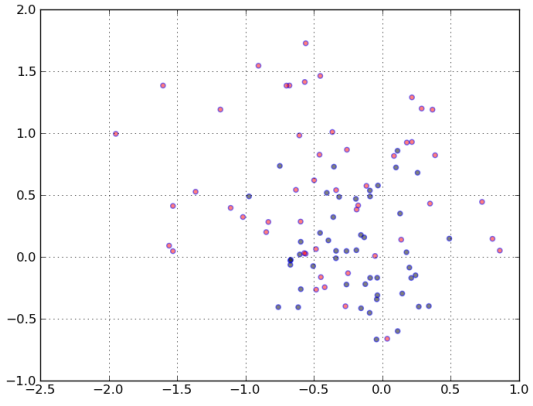

Fig. 10. The initial distribution of the data.

1) Learning rate α = 0.01: When the learning rate $\alpha = 0.01$ was used, it was visually observed that the two classes were already separated (linearly) at the output in about 45 iterations through the data set. Continuing to run the algo-

rithm through 250 iterations, the two classes were moving around, changing configurations. Their relative position was only slowly changing and the two classes remained separated through the end of the process. Figure 11 shows the final configuration of the points, after 250 iterations. We note that the points were spread out in a larger area, e.g. about 30 units in the y-direction, than likely they would have been if the grid was limited, say, to the unit square around the origin. Figure 12 shows the error (Equation (3)) against the iterations.

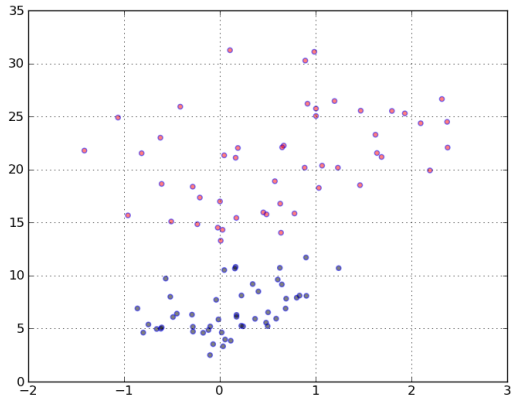

Fig. 11. The final distribution of the data.

We see that there is continuous fluctuation, as opposed to the prototypical LMS error curve that monotonically decreases with the iterations, and the weight vector (matrix in the current case) converges to a value. (Of course, instability is possible for the LMS algorithm, e.g. if the learning rate is large.) In this case, even though the weight matrix does not converge to a value and the points in the output move around, if the algorithm were to stop at any point after 45 iterations, the result would be acceptable. In general, with every iteration, points change position, and hence there is a possibility, which increases with larger α, that their output will snap on a different point on the virtual grid. This injects non-linearity and non-monotonicity into the process, not unlike what happens in the Kohonen self-organizing maps when a point, from one iteration to the next, causes another neuron to be the winner. Low error does not necessarily imply that the two classes are separated. It simply implies, that the points are nearer to their corresponding desired outputs on the virtual grid.

2) Learning rate α = 0.001: When the learning rate was $\alpha = 0.001$, it proved to be very small, and there was very little movement of points in the output. The initial distribution of points in output (Figure 10) remained almost identical to the final one, after 250 iterations (not shown). Figure 13 shows the plot of the error against the iterations. Even though the error decreases, the configuration of the points does not change in any substantial way from the initial output, before any iterations of the algorithm.

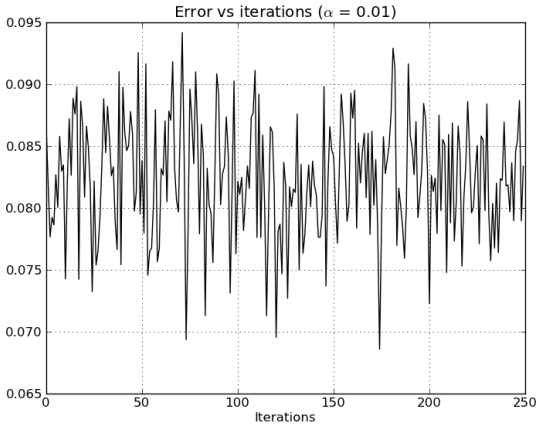

Fig. 12. The error fluctuates. However, after 45 iteration the two classes remained separated

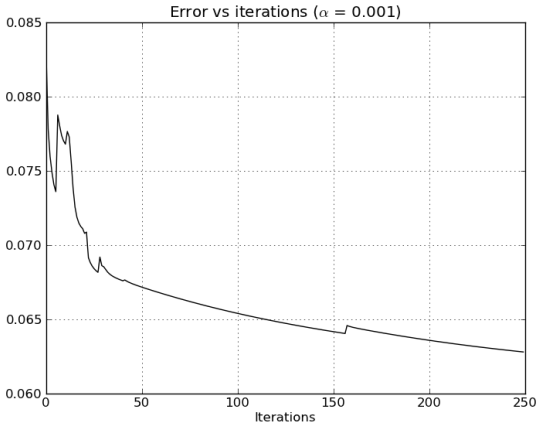

Fig. 13. Although the error decreases, the configuration of the points in the output essentially remains the same as the initial one.

3) Decaying learning rate: The learning rate α was initially 0.01 and then was decreased according to this formula:

$$\alpha_n = 0.01 * e^{-\frac{n(n+3)}{k}}, \tag{12}$$

where n is the iteration and k is constant. This formula and the constant $k = 10,000$ were found by experimentation: if the decay was too fast, the clusters did not have the opportunity to form, and the points settled in a fixed position. If the decay was too slow, the weights were changing and the points were moving around. The ideal decay would be for the clusters to have the opportunity to separate, and then become fixed in position. Figure 14 shows the error initially erratically fluctuating, and after about 150 iterations it becomes smoother. The two clusters settled in essentially the final configuration (iteration 250) from iteration 100. (Figure 15)

C. The linear neuron

In experiments with the linear neuron using the iris data, similar clustering behavior was observed. It was observed that

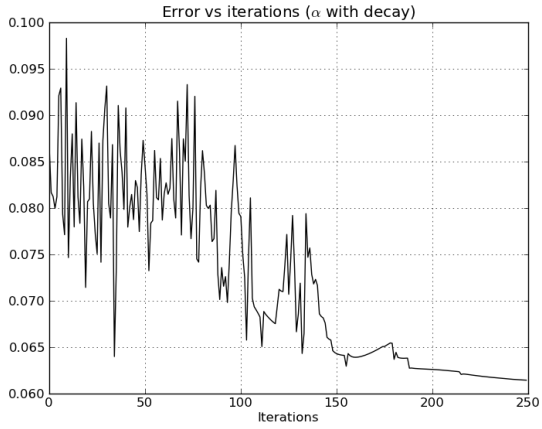

Fig. 14. The error with decaying learning rate.

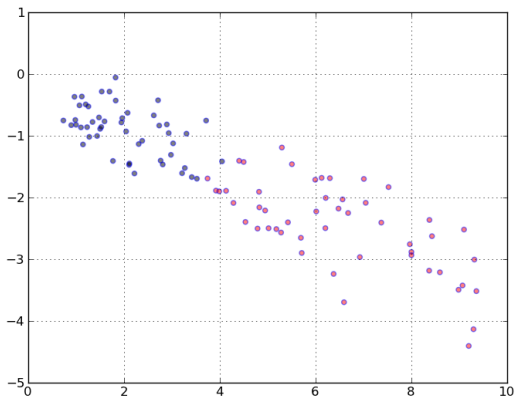

Fig. 15. The final configuration of the clusters with decaying learning rate.

on many occasions, in the initial output a class was already separable from the other two classes. Nevertheless, applying the algorithm showed that in general there is a tendency for the formed clusters to remain intact. More experiments are needed to compare the linear neuron with the quadratic one.

V. Conclusion

By introducing the concept of the virtual grid, where input vectors define their own desired outputs, self-organization has been shown to be possible with a single complex neuron.The same algorithm can be easily modified using two real neurons. Evenly distributed data on a 2-D array did not cause clustering behavior in the output. The new fast and economical method, involving only one neuron, promises to be useful in the exploration of high-dimensional data, and in applications where self-organizing maps are typically used. It will be particularly suitable for real-time and other time-critical applications. Given that training with the *virtual* grid is a local technique, one may speculate whether analogous self-organizing mechanisms exist in biological systems.

VI. Bibliography

References

[1] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biological Cybernetics*, vol. 43, 1982, reprinted in [16].

[2] ——, *Self-Organization and Associative Memory*, 3rd ed. Berlin: Springer-Verlag, 1989.

[3] ——, "The self-organising map," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464–1480, 1990.

[4] A. Hirose, Ed., *Complex-Valued Neural Networks: Theories and Applications*. World Scientific Publishing, 2003.

[5] A. Hirose, *Complex-Valued Neural Networks*, ser. Studies in Computational Intelligence. Springer, 2006, vol. 32.

[6] ——, *Complex-valued Neural networks*. Saiensu-sha, 2005, in Japanese.

[7] T. Nitta, *Complex-valued Neural Networks: Utilizing High-dimensional Parameters*, 1st ed. Information Science Reference, 1 2009.

[8] I. N. Aizenberg, *Complex-Valued Neural Networks with Multi-Valued Neurons*, ser. Studies in Computational Intelligence. Springer, 2011, vol. 353.

[9] S. Suresh, N. Sundararajan, and R. Savitha, *Supervised Learning with Complex-valued Neural Networks*, ser. Studies in Computational Intelligence. Springer, 2013, vol. 421.

[10] Z. Chen, S. Haykin, J. J. Eggermont, and S. Becker, *Correlative Learning: A Basis for Brain and Adaptive Systems (Adaptive and Learning Systems for Signal Processing, Communications and Control Series)*, 1st ed. Wiley-Interscience, 10 2007.

[11] K. E. Gustafson and D. K. Rao, *Numerical Range: The Field of Values of Linear Operators and Matrices (Universitext)*, 1st ed. Springer, 11 1996.

[12] R. A. Horn and C. R. Johnson, *Topics in Matrix Analysis*. Cambridge University Press, 5 1991.

[13] B. Widrow, J. McCool, and M. Ball, "The complex lms algorithm," *Proceedings of the IEEE*, vol. 63, no. 4, pp. 719–720, 1975.

[14] G. Georgiou, "Exact interpolation and learning in quadratic neural networks," in *IJCNN '06. International Joint Conference on Neural Networks*, 2006, pp. 230–234.

[15] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of Eugenics*, vol. 7, no. II, pp. 179–188, 1936.

[16] J. Anderson and E. Rosenfeld, Eds., *Neurocomputing: Foundations of Research*. Cambridge: MIT Press, 1988.