POLYTECHNIC UNIVERSITY
Department of Computer and Information Science

Learning vector Quantization

K. Ming Leung

Abstract: Learning vector Quantization is a NN invented by Kohonen for pattern classification. This NN combines competitive learning with supervision.

Directory

- Table of Contents
- Begin Article

Last Revision Date: April 29, 2009

Table of Contents

1. Introduction
2. LVQ Rule
3. Example 1: LVQ
4. Example 2: LVQ
5. Problems with LVQ
6. Using "Conscience" to Prohibit Dead Neurons
7. An Improved LVQ: LVQ2

1. Introduction

Learning vector Quantization (LVQ) is a neural net that combines competitive learning with supervision. It can be used for pattern classification.

A training set consisting of Q training vector - target output pairs are assumed to be given

$$\left\{\mathbf{s}^{(q)} : \mathbf{t}^{(q)}\right\}, \qquad q = 1, 2, \ldots, Q,$$

where $\mathbf{s}^{(q)}$ are N dimensional training vectors, and $\mathbf{t}^{(q)}$ are M dimensional target output vectors. M is the number of classes, and it must be smaller than Q. The target vectors are defined by

$$t_i^{(q)} = \begin{cases} 1, & \text{if, } \mathbf{s}^{(q)} \text{ belongs to class } i \\ 0, & \text{otherwise.} \end{cases}$$

The LVQ is made up of a competitive layer, which includes a competitive subnet, and a linear layer. In the first layer (not counting the input layer), each neuron is assigned to a class. Different neurons in the first layer can be assigned to the same class. Each of those

classes is then assigned to one neuron in the second layer. The number of neurons in the first layer, Q, will therefore always be at least as large as the number of neurons in the second layer, M.

In the competitive layer, neurons in the first layer learns a prototype vector which allows it to classify a region of the input space. Closeness between the input vector and any of the weight vectors is measured by the smallness of the Euclidean distance between them. A subnet is used to find the smallest element of the net input

$$\mathbf{n}^{(1)} = \begin{bmatrix} \|\mathbf{x} - \mathbf{W}^{(1)}_{.1}\| \\ \|\mathbf{x} - \mathbf{W}^{(1)}_{.2}\| \\ \cdots \\ \|\mathbf{x} - \mathbf{W}^{(1)}_{.Q}\| \end{bmatrix}$$

and set the corresponding output element to 1, indicating that the input vector belongs to the corresponding class, and set all others to 0. The action of this subnet is represented as a vector-valued vector function

$$\mathbf{a}^{(1)} = \text{compet}(\mathbf{n}^{(1)}).$$

Since some of these classes may be identical, they are really subclasses. The second layer (the linear layer) of the LVQ network is then used to combine subclasses into a single class. This is done using the $\mathbf{W}^{(2)}$ weight matrix which has element

$$w_{ij} = \begin{cases} 1, & \text{if the } i \text{ neuron belong to a subcass of } j, \\ 0, & \text{otherwise.} \end{cases}$$

Once $\mathbf{W}^{(2)}$ is set, it will not be altered.

On the other hand, the weights, $\mathbf{W}^{(1)}$, for the competitive layer have to be trained using the Kohonen LVQ rule.

2. LVQ Rule

At each iteration one of the training vector is presented to the network as input $\mathbf{x}$, and the Euclidean distance from the input vector to each of the prototype vector (forming columns of the weight matrix) is computed. The hidden neurons compete. Neuron j^* wins the competition if the Euclidean distance between $\mathbf{x}$ and the j^* prototype vector is the smallest. The j^* element of $\mathbf{a}^{(1)}$ is set to 1 while others

are set to 0. The activations $\mathbf{a}^{(1)}$ is then multiplied by $\mathbf{W}^{(2)}$ on its right to get the net input $\mathbf{n}^{(2)}$. This produces the output of the entire network $\mathbf{a}^{(2)} = \mathbf{n}^{(2)}$, since the transfer function of the output neurons is an identity function. $\mathbf{a}^{(2)}$ also has only one nonzero element k^*, indicating that the input vector belongs to class k^*.

The Kohonen rule is used to improve the weights of the hidden layer in the following way. If $\mathbf{x}$ is classified correctly, then the weight vector $\mathbf{w}^{(1)}_{\cdot j^*}$ of the winning hidden neuron is moved towards $\mathbf{x}$

$$\Delta\mathbf{w}^{(1)}_{\cdot j^*} = \alpha(\mathbf{x} - \mathbf{w}^{(1)}_{\cdot j^*}) \qquad \text{if } a^{(2)}_{k^*} = t_{k^*} = 1.$$

But if $\mathbf{x}$ is classified incorrectly, then it is clear that a wrong hidden neuron won the competition. In this case its weight $\mathbf{w}^{(1)}_{\cdot j^*}$ is moved away from $\mathbf{x}$:

$$\Delta\mathbf{w}^{(1)}_{\cdot j^*} = -\alpha(\mathbf{x} - \mathbf{w}^{(1)}_{\cdot j^*}) \qquad \text{if } a^{(2)}_{k^*} = 1 \neq t_{k^*}.$$

3. Example 1: LVQ

Consider an LVQ network having the following weights:

$$\mathbf{W}^{(1)} = \begin{bmatrix} 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 \end{bmatrix}$$

$$\mathbf{W}^{(2)} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

We want to answer the following questions:

- The total number of classes this LVQ network has.
- The total number of subclasses.
- To which class that a subclass belongs.
- The decision boundaries that separates each of the subclasses.

Since $\mathbf{W}^{(2)}$ is 5 by 3, this LVQ network has 5 subclasses and 3 classes. From the form of $\mathbf{W}^{(2)}$ it is clear that subclass 1 belongs to class 1, subclasses 2 and 3 belong to class 2, and subclasses 4 and 5 belong to class 3.

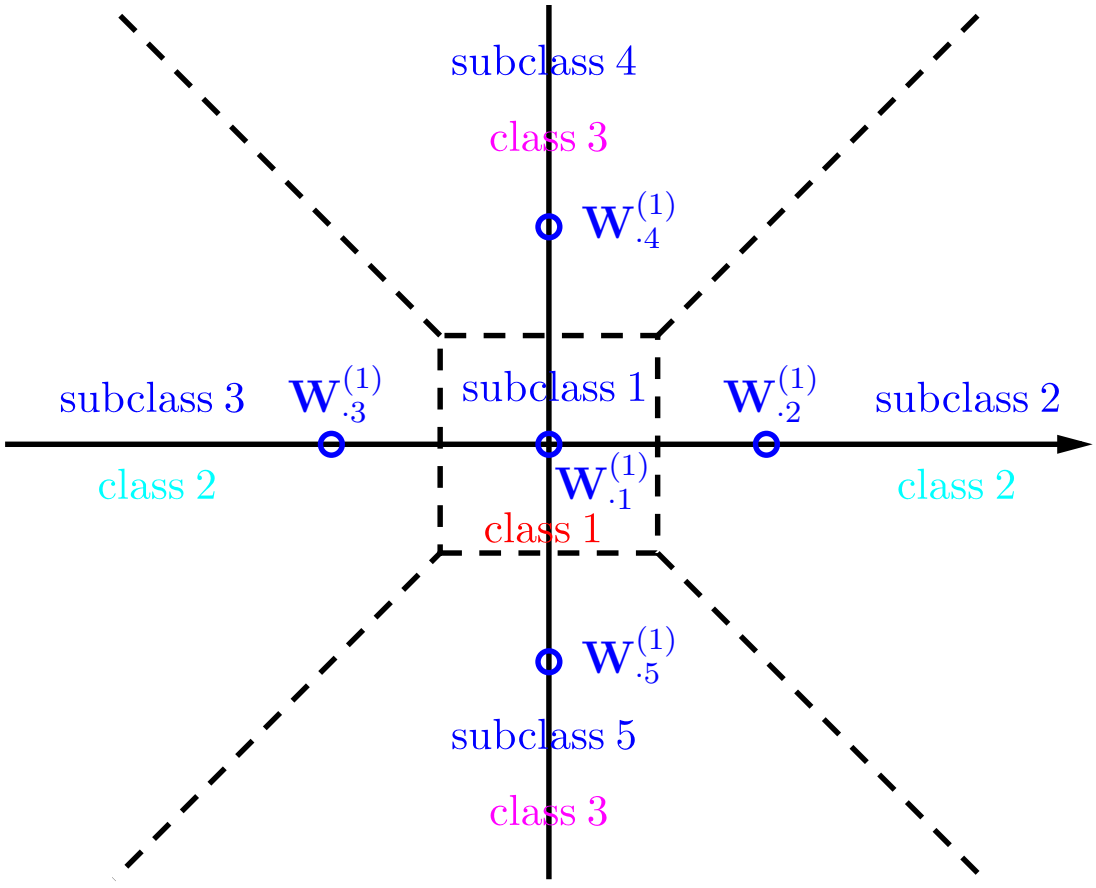

Figure 1: *Decision boundaries separating input space into subclasses.*

We assume here that we are using the square of the Euclidean distance as a measure the closeness of vectors. The weight vector for the i-th cluster unit is given by the i-th column of $\mathbf{W}^{(1)}$. All input vectors in the input vector space that are closest to the i-th weight vector will belong to subclass i. Thus we draw the following decision boundaries separating each subclass.

4. Example 2: LVQ

Suppose we are given the following training set:

class 1:

$$\mathbf{s}_1^{(1)} = \begin{bmatrix} -1 \\ -1 \end{bmatrix} \qquad \mathbf{t}^{(1)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\mathbf{s}_1^{(2)} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \qquad \mathbf{t}^{(2)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

class 2:

$$\mathbf{s}_1^{(3)} = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \qquad \mathbf{t}^{(3)} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

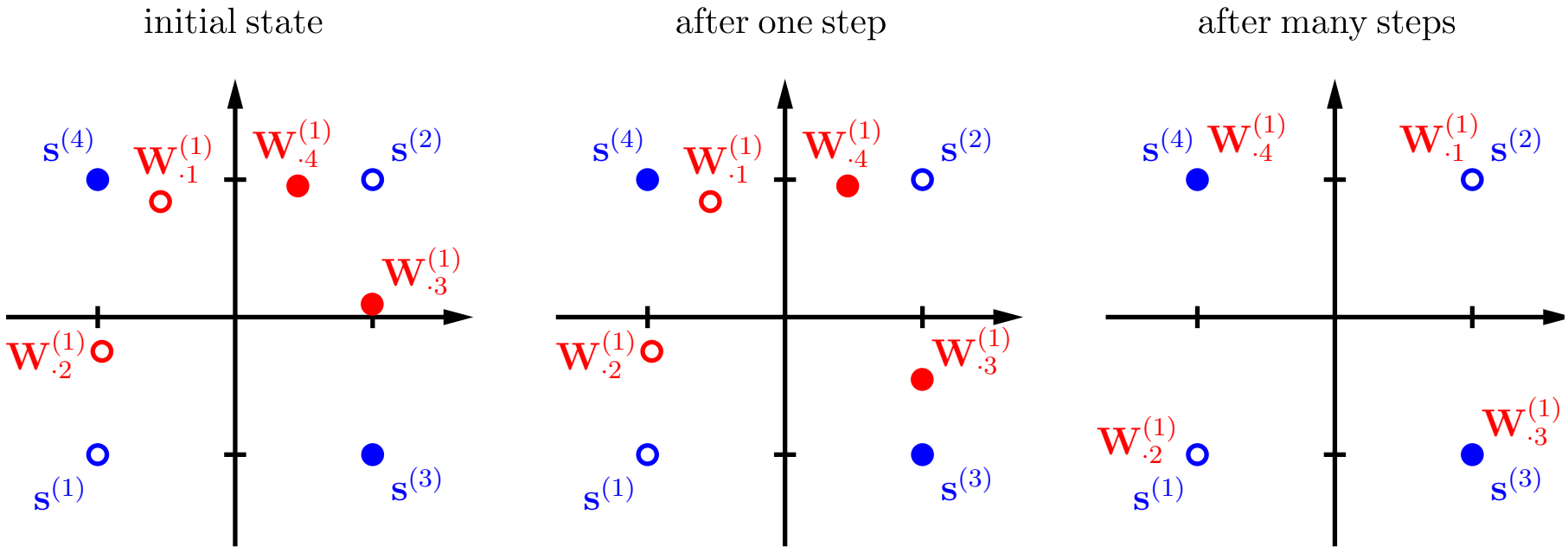

Figure 2: *LVQ after first and many iterations.*

$$\mathbf{s}^{(4)} = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \qquad \mathbf{t}^{(4)} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Since $\mathbf{s}^{(1)}$ and $\mathbf{s}^{(2)}$ belong to class 1, and $\mathbf{s}^{(3)}$ and $\mathbf{s}^{(4)}$ belong to class 2, we choose a weight matrix

$$\mathbf{W}^{(2)} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

that connects neurons 1 and 2 to output neuron 1, and connects neurons 3 and 4 to output neuron 2.

Initially each column of $\mathbf{W}^{(1)}$ is set to random values. For example,

$$\mathbf{w}_{\cdot 1}^{(1)} = \begin{bmatrix} -0.543 \\ 0.840 \end{bmatrix} \qquad \mathbf{w}_{\cdot 2}^{(1)} = \begin{bmatrix} -0.969 \\ -0.249 \end{bmatrix}$$

$$\mathbf{w}_{\cdot 3}^{(1)} = \begin{bmatrix} 0.997 \\ 0.094 \end{bmatrix} \qquad \mathbf{w}_{\cdot 4}^{(1)} = \begin{bmatrix} 0.456 \\ 0.954 \end{bmatrix}$$

A randomly chosen training vector is then presented to the network. Supposed $\mathbf{s}^{(3)}$ is picked, then we have

$$\mathbf{n}^{(1)} = \begin{bmatrix} \|\mathbf{s}^{(3)} - \mathbf{w}_{\cdot 1}^{(1)}\| \\ \|\mathbf{s}^{(3)} - \mathbf{w}_{\cdot 2}^{(1)}\| \\ \|\mathbf{s}^{(3)} - \mathbf{w}_{\cdot 3}^{(1)}\| \\ \|\mathbf{s}^{(3)} - \mathbf{w}_{\cdot 4}^{(1)}\| \end{bmatrix} = \begin{bmatrix} 2.40 \\ 2.11 \\ 1.09 \\ 2.04 \end{bmatrix}.$$

Clearly the third hidden neuron has its weight vector closest to $\mathbf{x}^{(3)}$ and is therefore the winning neuron:

$$\mathbf{a}^{(1)} = \text{compet}(\mathbf{n}^{(1)}) = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}.$$

Thus

$$\mathbf{a}^{(2)} = \mathbf{W}^{(2)}\mathbf{a}^{(1)} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

Since this is exactly $\mathbf{t}^{(3)}$, this means that $\mathbf{s}^{(3)}$ is correctly classified as belonging to class 2.

Therefore hidden weight vector $\mathbf{w}_{.3}^{(1)}$ is moved closer to $\mathbf{x}^{(3)}$:

$$\Delta\mathbf{w}_{.3}^{(1)} = \alpha(\mathbf{x}^{(3)} - \mathbf{w}_{.3}^{(1)}).$$

Thus

$$\mathbf{w}_{\cdot 3}^{(1)} = \begin{bmatrix} 0.997 \\ 0.094 \end{bmatrix} + 0.5\left(\begin{bmatrix} 1 \\ -1 \end{bmatrix} - \begin{bmatrix} 0.997 \\ 0.094 \end{bmatrix}\right) = \begin{bmatrix} 0.998 \\ -0.453 \end{bmatrix},$$

where α is chosen to be 0.5.

The locations of the training vectors and their corresponding prototype vectors are shown in the figure before and after this single step.

After many iterations cycling through the training set, $\mathbf{w}_{\cdot 3}^{(1)}$ moves close to $\mathbf{s}^{(2)}$ and $\mathbf{w}_{\cdot 3}^{(2)}$ moves close to $\mathbf{s}^{(1)}$. These are the weight vectors for class 1. At the same time, $\mathbf{w}_{\cdot 3}^{(3)}$ moves close to $\mathbf{s}^{(3)}$ and $\mathbf{w}_{\cdot 3}^{(4)}$ moves close to $\mathbf{s}^{(4)}$. These are the weight vectors for class 2.

5. Problems with LVQ

The LVQ network work well for many problems, but it suffers from the following two limitations.

- As with competitive layers, sometimes a hidden neuron can have initial weights that prevent it from ever winning any competi-

tion. This results in what is known as a dead neural that never does anything useful. This problem can be solved using a "conscience" mechanism where a neuron that wins a lot of times will attempt to let others to win.

- Depending on how the initial weight vectors are arranged, a neuron's weight vector may need to travel through a region of a class that it doesn't represent, to get to a region that it does represent. Because the weights of such a neuron will be repelled by vectors in the region it has to cross, it may never be able to accomplish. Thus it may never properly classify the region to which it is being attracted.

This second problem can be solve by applying the following modification to the Kohonen rule, resulting in an algorithm known as LVQ2.

6. Using "Conscience" to Prohibit Dead Neurons

Neurons that are too far from input vectors to ever win the competition can be given a chance by using adaptive biases that get more negative each time a neuron wins the competition. The result is that

neurons that win very often start to "feel guilty" until other neurons have a chance to win.

A typical learning rule for the bias b_i of neuron i is

$$b_i^{\text{new}} = \begin{cases} 0.9\, b_i^{\text{old}}, & \text{if} \quad i \neq i^* \\ b_i^{\text{old}} - 0.2, & \text{if} \quad i = i^* \end{cases}$$

To see the effect of "conscience", we consider the following example of a SOM network. We assume that we are given three training vectors:

$$\mathbf{s}_1^{(1)} = \begin{bmatrix} -1 & 0 \end{bmatrix}$$

$$\mathbf{s}_1^{(2)} = \begin{bmatrix} 0 & 1 \end{bmatrix}$$

$$\mathbf{s}_1^{(3)} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$$

The exemplar vectors are stored in the columns of the weight ma-

trix

$$\mathbf{W} = \begin{bmatrix} 0 & -\frac{2}{\sqrt{5}} & -\frac{1}{\sqrt{5}} \\ -1 & \frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} \end{bmatrix}$$

If we ignore the biases, it is clear that no matter which order the training vectors are presented to the network for training, the first cluster unit has no way of ever wining the competition and move closer to one of the training vectors. It is a dead neuron.

On the other hand, if biases are used and the above conscience rule to update them, even though they may initially have zero values, eventually the first cluster unit will have a chance to win and move closer to one of the training vectors.

For example, if we use a learning rate of 0.5 and the above conscience rule to update the biases, we find that it takes 18 cycles through the training set before the first cluster unit wins and move closer to one of the training vectors.

After 29 cycles through the training set, the exemplar vectors are close enough to their corresponding clusters (each containing only a single point).

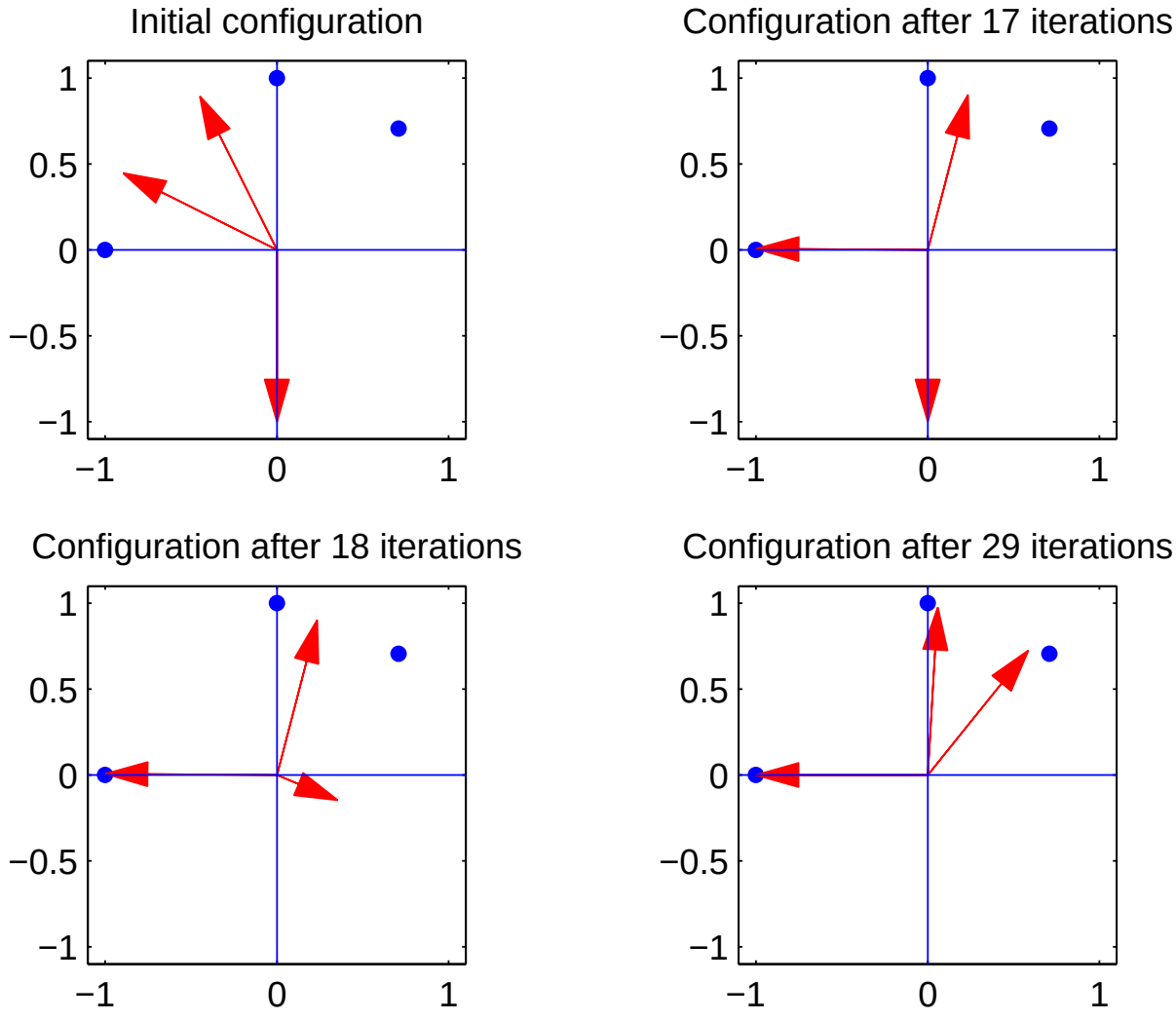

Figure 3: *SOM with conscience.*

7. An Improved LVQ: LVQ2

According to LVQ2, if the winning neuron in the hidden layer incorrectly classifies the current input, its weight vector is moved away from the current input vector, as before.

However, the weight vector of the closest neuron to the input vector that does correctly classify it is moved toward the input vector.

Thus when the network correctly classifies an input vector, the weight vector of only one neuron is moved toward the input vector. However, if the input vector is incorrectly classified, the weight vectors of two neurons are updated, one weight vector is moved away from the input vector, and the other one is moved towards the input vector.

References

[1] See Chapter 14 in M. T. Hagan, H. B. Demuth, and M. Beale, "Neural Networks Design", PWS Publishing, 1996.