Some Competitive Learning Methods

Bernd Fritzke
Systems Biophysics
Institute for Neural Computation
Ruhr-Universität Bochum

Draft from April 5, 1997

(Some additions and refinements are planned for this document so it
will stay in the draft status still for a while.)
Comments are welcome.

Abstract

This report has the purpose of describing several algorithms from the literature all related to competitive learning. A uniform terminology is used for all methods. Moreover, identical examples are provided to allow a qualitative comparisons of the methods. The on-line version[1] of this document contains hyperlinks to Java implementations of several of the discussed methods.

---

[1]http://www.neuroinformatik.ruhr-uni-bochum.de/ini/VDM/research/gsn/JavaPaper/

Contents

Chapter 1

Introduction

In the area of competitive learning a rather large number of models exist which have similar goals but differ considerably in the way they work. A common goal of those algorithms is to distribute a certain number of vectors in a possibly high-dimensional space. The distribution of these vectors should reflect (in one of several possible ways) the probability distribution of the input signals which in general is not given explicitly but only through sample vectors.

In this report we review several methods related to competitive learning. A common terminology is used to make a comparison of the methods easy. Moreover, software implementations of the methods are provided allowing experiments with different data distributions and observation of the learning process. Thanks to the Java programming language the implementations run on a large number of platforms without the need of compilation or local adaptation.

The report is structured as follows: In chapter 2 the basic terminology is introduced and properties shared by all models are outlined. Chapter 3 discusses possible goals for competitive learning systems. Chapter 4 is concerned with hard competitive learning, i.e. models where only the winner for the given input signal is adapted. Chapters 5 and 6 describe soft competitive learning. These models are characterized by adapting in addition to the winner also some other units of the network. Chapter 5 is concerned with models where the network has no fixed dimensionality. Chapter 6 describes models which do have a fixed dimensionality and may be used for data visualization, since they define a mapping from the usually high-dimensional input space to the low-dimensional network structure. The last two chapters still have to be written and will contain quantitative results and a discussion.

Chapter 2

Common Properties and Notational Conventions

The models described in this report share several architectural properties which are described in this chapter. For simplicity, we will refer to any of these models as *network* even if the model does not belong to what is usually understood as "neural network".

Each network consists of a set of N units:

$$\mathcal{A} = \{c_1, c_2, \ldots, c_N\}. \tag{2.1}$$

Each unit c has an associated *reference vector*

$$\mathbf{w}_c \in \mathrm{R}^n \tag{2.2}$$

indicating its position or *receptive field center* in input space.

Between the units of the network there exists a (possibly empty) set

$$\mathcal{C} \subset \mathcal{A} \times \mathcal{A} \tag{2.3}$$

of *neighborhood connections* which are unweighted and symmetric:

$$(i, j) \in \mathcal{C} \iff (j, i) \in \mathcal{C}. \tag{2.4}$$

These connections have nothing to do with the weighted connections found, e.g., in multi-layer perceptrons (Rumelhart et al., 1986). They are used in some methods to extend the adaptation of the winner (see below) to some of its topological neighbors.

For a unit c we denote with N_c the set of its direct topological neighbors:

$$N_c = \{i \in \mathcal{A} | (c, i) \in \mathcal{C}\}. \tag{2.5}$$

The n-dimensional input signals are assumed to be generated either according to a continuous probability density function

$$p(\boldsymbol{\xi}), \boldsymbol{\xi} \in \mathrm{R}^n \tag{2.6}$$

or from a finite training data set

$$\mathcal{D} = \{\boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_M\}, \boldsymbol{\xi}_i \in \mathrm{R}^n. \tag{2.7}$$

For a given input signal $\boldsymbol{\xi}$ the *winner* $s(\boldsymbol{\xi})$ among the units in $\mathcal{A}$ is defined as the unit with the nearest reference vector

$$s(\boldsymbol{\xi}) = \arg\min_{c \in \mathcal{A}} \| \boldsymbol{\xi} - \mathbf{w}_c \|. \tag{2.8}$$

whereby $\| \cdot \|$ denotes the Euclidean vector norm. In case of a tie among several units one of them is chosen to be the winner by throwing a fair dice. In some cases we will denote the current winner simply by s (omitting the dependency on $\boldsymbol{\xi}$). If not only the winner but also the second-nearest unit or even more distant units are of interest, we denote with s_i the i-nearest unit (s_1 is the winner, s_2 is the second-nearest unit, etc.).

Two fundamental and closely related concepts from computational geometry are important to understand in this context. These are the *Voronoi Tessellation* and the *Delaunay Triangulation*:

Given a set of vectors $\mathbf{w}_1, \ldots, \mathbf{w}_N$ in R^n (see figure 2.1 a), the *Voronoi Region* V_i of a particular vector $\mathbf{w}_i$ is defined as the set of all points in R^n for which $\mathbf{w}_i$ is the nearest vector:

$$V_i = \{ \boldsymbol{\xi} \in \mathrm{R}^n | i = \arg\min_{j \in \{1, \ldots, N\}} \| \boldsymbol{\xi} - \mathbf{w}_j \| \}. \tag{2.9}$$

In order for each data point to be associated to exactly one Voronoi region we define (as previously done for the winner) that in case of a tie the corresponding point is mapped at random to one of the nearest reference vectors. Alternatively, one could postulate general positions for all data points and reference vectors in which case a tie would have zero probability.

It is known, that each Voronoi region V_i is a convex area, i.e.

$$(\boldsymbol{\xi}_1 \in V_i \wedge \boldsymbol{\xi}_2 \in V_i) \quad \Rightarrow \quad (\boldsymbol{\xi}_1 + \alpha(\boldsymbol{\xi}_2 - \boldsymbol{\xi}_1) \in V_i) \; (\forall \alpha, 0 \le \alpha \le 1). \tag{2.10}$$

The partition of R^n formed by all Voronoi polygons is called *Voronoi Tessellation* or *Dirichlet Tessellation* (see figure 2.1 b). Efficient algorithms to compute it are only known for two-dimensional data sets (Preparata and Shamos, 1990). The concept itself, however, is applicable to spaces of arbitrarily high dimensions.

If one connects all pairs of points for which the respective Voronoi regions share an edge (an $(n-1)$-dimensional hyperface for spaces of dimension n) one gets the *Delaunay Triangulation* (see figure 2.1 c). This triangulation is special among all possible triangulation in various respects. It is, e.g., the only triangulation in which the circumcircle of each triangle contains no other point from the original point set than the vertices of this triangle. Moreover, the Delaunay triangulation has been shown to be optimal for function interpolation (Omohundro, 1990). The *competitive Hebbian learning* method (see section 5.2) generates a subgraph of the Delaunay triangulation which is limited to those areas of the input space where data is found.

For convenience we define the *Voronoi Region of a unit $c, c \in \mathcal{A}$*, as the Voronoi region of its reference vector:

$$V_c = \{ \boldsymbol{\xi} \in \mathrm{R}^n | s(\boldsymbol{\xi}) = c \}. \tag{2.11}$$

In the case of a finite input data set $\mathcal{D}$ we denote for a unit c with the term *Voronoi Set* the subset $\mathcal{R}_c$ of $\mathcal{D}$ for which c is the winner (see figure 2.2):

$$\mathcal{R}_c = \{ \boldsymbol{\xi} \in \mathcal{D} | s(\boldsymbol{\xi}) = c \}. \tag{2.12}$$
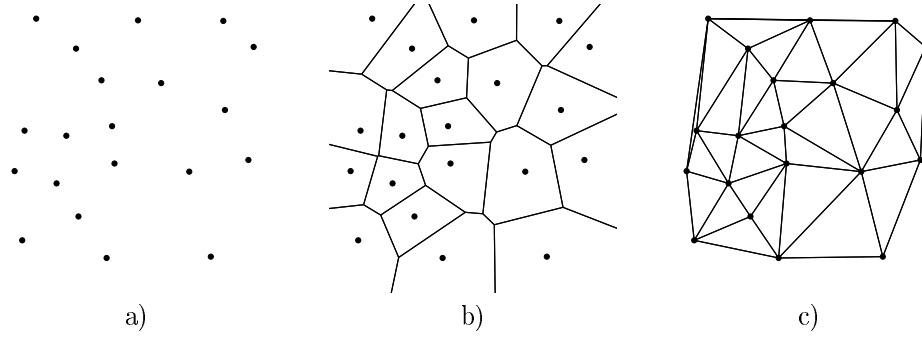
Figure 2.1: a) Point set in R^2, b) corresponding Voronoi tessellation, c) corresponding Delaunay triangulation.

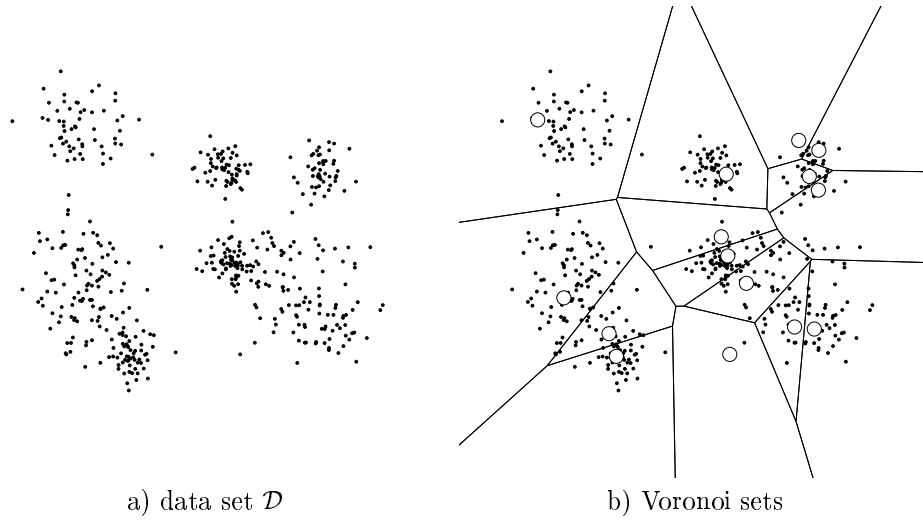a) data set $\mathcal{D}$ b) Voronoi sets

Figure 2.2: An input data set $\mathcal{D}$ is shown (a) and the partition of $\mathcal{D}$ into Voronoi sets for a particular set of reference vectors (b). Each Voronoi set contains the data points within the corresponding Voronoi field.

Chapter 3

Goals of Competitive Learning

A number of different and often mutually exclusive goals can be set for competitive learning systems. In the following some of these goals are discussed.

3.1 Error Minimization

A frequent goal is the minimization of the *expected quantization* (or *distortion*) *error*. In the case of a continuous input signal distribution $p(\boldsymbol{\xi})$ this amounts to finding values for the reference vectors $\mathbf{w}_c, c \in \mathcal{A}$ such that the error

$$\mathrm{E}(p(\boldsymbol{\xi}), \mathcal{A}) = \sum_{c \in \mathcal{A}} \int_{V_c} \|\boldsymbol{\xi} - \mathbf{w}_c\|^2 p(\boldsymbol{\xi}) d\boldsymbol{\xi} \tag{3.1}$$

is minimized (V_c is the Voronoi region of unit c).

Correspondingly, in the case of a finite data set $\mathcal{D}$ the error

$$\mathrm{E}(\mathcal{D}, \mathcal{A}) = 1/|\mathcal{D}| \sum_{c \in \mathcal{A}} \sum_{\boldsymbol{\xi} \in \mathcal{R}_c} \|\xi - \mathbf{w}_c\|^2 \tag{3.2}$$

has to be minimized with $\mathcal{R}_c$ being the Voronoi set of the unit c.

A typical application where error minimization is important is *vector quantization* (Linde et al., 1980; Gray, 1984). In vector quantization data is transmitted over limited bandwidth communication channels by transmitting for each data vector only the *index* of the nearest reference vector. The set of reference vectors (which is called *codebook* in this context) is assumed to be known both to sender and receiver. Therefore, the receiver can use the transmitted indexes to retrieve the corresponding reference vector. There is an information loss in this case which is equal to the distance of current data vector and nearest reference vector. The expectation value of this error is described by equations (3.1) and (3.2). In particular if the data distribution is clustered (contains subregions of high probability density), dramatic compression rates can be achieved with vector quantization with relatively little distortion.

3.2 Entropy Maximization

Sometimes the reference vectors should be distributed such that each reference vector has the same chance to be winner for a randomly generated input signal $\boldsymbol{\xi}$:

$$P(s(\boldsymbol{\xi}) = c) = \frac{1}{|\mathcal{A}|} \quad (\forall c \in \mathcal{A}). \tag{3.3}$$

If we interpret the generation of an input signal and the subsequent mapping onto the nearest unit in $\mathcal{A}$ as random experiment which assigns a value $x \in \mathcal{A}$ to the random variable X, then (3.3) is equivalent to maximizing the entropy

$$H(X) = - \sum_{x \in \mathcal{A}} P(x) \log(P(x)) = E(\log(\frac{1}{P(x)})), \tag{3.4}$$

with $E(\cdot)$ being the expectation operator.

If the data is generated from a continuous probability distribution $p(\boldsymbol{\xi})$, then (3.3) is equivalent to

$$\int_{V_c} p(\xi)d\xi = \frac{1}{|\mathcal{A}|} \quad (\forall c \in \mathcal{A}). \tag{3.5}$$

In the case of a finite data set $\mathcal{D}$ (3.3) corresponds to the situation where each Voronoi set $\mathcal{R}_c$ contains (up to discretization effects) the same number of data vectors:

$$\frac{|\mathcal{R}_c|}{|\mathcal{D}|} \simeq \frac{1}{|\mathcal{A}|} \quad (\forall c \in \mathcal{A}). \tag{3.6}$$

An advantage of choosing reference vectors such as to maximize entropy is the inherent robustness of the resulting system. The removal (or "failure") of any reference vector affects only a limited fraction of the data.

Entropy maximization and error minimization can in general not be achieved simultaneously. In particular if the data distribution is highly non-uniform both goals differ considerably. Consider, e.g., a signal distribution $p(\boldsymbol{\xi})$ where 50 percent of the input signals come from a very small (point-like) region of the input space, whereas the other fifty percent are uniformly distributed within a huge hypercube. To maximize entropy half of the reference vectors have to be positioned in each region. To minimize quantization error however, only one single vector should be positioned in the point-like region (reducing the quantization error for the signals there basically to zero) and all others should be uniformly distributed within the hypercube.

3.3 Feature Mapping

With some network architectures it is possible to map high-dimensional input signals onto a lower-dimensional structure in such a way, that some similarity relations present in the original data are still present after the mapping. This has been denoted *feature mapping* and can be useful for data visualization. A prerequisite for this is that the network used has a fixed dimensionality. This is the case, e.g., for the *self-organizing feature map* and the other methods discussed in section 6 of this report.

A related question is, how *topology-preserving* is the mapping from the input data space onto the discrete network structure, i.e. how well are similarities preserved? Several quantitative measures have been proposed to evaluate this like the topographic product (Bauer and Pawelzik, 1992) or the topographic function (Villmann et al., 1994).

3.4 Other Goals

Competitive learning methods can also be used for *density estimation*, i.e. for the generation of an estimate for the unknown probability density $p(\boldsymbol{\xi})$ of the input signals.

Another possible goal is *clustering*, where a partition of the data into subgroups or *clusters* is sought, such that the distance of data items within the same cluster (intra-cluster variance) is small and the distance of data items stemming from different clusters (inter-cluster variance) is large. Many different flavors of the clustering problem exist depending, e.g., on whether the number of clusters is pre-defined or should be a result of the clustering process. A comprehensive overview of clustering methods is given by Jain and Dubes (1988).

Combinations of competitive learning methods with *supervised learning* approaches are feasible, too. One possibility are radial basis function networks (RBFN) where competitive learning is used to position the radial centers (Moody and Darken, 1989; Fritzke, 1994b). Moreover, *local linear maps* have been combined with competitive learning methods (Walter et al., 1990; Martinetz et al., 1989, 1993; Fritzke, 1995b). In the simplest case for each Voronoi region one linear model is used to describe the input/output relationship of the data within the Voronoi region.

Chapter 4

Hard Competitive Learning

Hard competitive learning (a.k.a. winner-take-all learning) comprises methods where each input signal only determines the adaptation of one unit, the winner. Different specific methods can be obtained by performing either *batch* or *on-line* update. In batch methods (e.g. LBG) all possible input signals (which must come from a finite set in this case) are evaluated first before any adaptations are done. This is iterated a number of times. On-line methods, on the other hand (e.g. k-means), perform an update directly after each input signal. Among the on-line methods variants with constant adaptation rate can be distinguished from variants with decreasing adaptation rates of different kinds.

A general problem occurring with hard competitive learning is the possible existence of "dead units". These are units which – perhaps due to inappropriate initialization – are never winner for an input signal and, therefore, keep their position indefinitely. Those units do not contribute to whatever the networks purpose is (e.g. error minimization) and must be considered harmful since they are unused network resources. A common way to avoid dead units is to use distinct sample vectors according to $p(\boldsymbol{\xi})$ to initialize the reference vectors.

The following problem, however, remains: if the reference vectors are initialized randomly according to $p(\boldsymbol{\xi})$, then their *expected* initial local density is proportional to $p(\boldsymbol{\xi})$. This may be rather suboptimal for certain goals. For example, if the goal is error minimization and $p(\boldsymbol{\xi})$ is highly non-uniform, then it is better to undersample the regions with high probability density (i.e., use less reference vectors there than dictated by $p(\boldsymbol{\xi})$) and oversample the other regions. One possibility to adapt the distribution of the reference vectors to a specific goal is the use of local statistical measures for directing insertions and possibly also deletion of units (see sections 5.4, 6.2 and 6.3).

Another problem of hard competitive learning is that different random initializations may lead to very different results. The purely local adaptations may not be able to get the system out of the poor local minimum where it was started. One way to cope with this problem is to change the "winner-take-all" approach of hard competitive learning to the "winner-take-most" approach of *soft competitive learning*. In this case not only the winner but also some other units are adapted (see chapters 5 and 6). In general this decreases the dependency on initialization.

4.1 Batch Update: LBG

The LBG (or generalized Lloyd) algorithm (Linde et al., 1980; Forgy, 1965; Lloyd, 1957) works by repeatedly moving all reference vectors to the arithmetic mean of their Voronoi sets. The theoretical foundation for this is that it can be shown

(Gray, 1992) that a *necessary* condition for a set of reference vectors $\{\mathbf{w}_c | c \in \mathcal{A}\}$ to minimize the distortion error

$$\mathcal{E}(\mathcal{D}, \mathcal{A}) = 1/|\mathcal{D}| \sum_{c \in \mathcal{A}} \sum_{\xi \in \mathcal{R}_c} \|\xi - \mathbf{w}_c\|^2. \tag{4.1}$$

is that each reference vector $\mathbf{w}_c$ fulfills the *centroid condition*. In the case of a finite set of input signals and the use of the Euclidean distance measure the centroid condition reduces to

$$\mathbf{w}_c = \frac{1}{|R_c|} \sum_{\xi \in R_c} \xi \tag{4.2}$$

whereby R_c is the Voronoi set of unit c.

The complete LBG algorithm is the following:

1. Initialize the set $\mathcal{A}$ to contain N ($N \ll M$) units c_i

$$\mathcal{A} = \{c_1, c_2, \ldots, c_N\} \tag{4.3}$$

   with reference vectors $\mathbf{w}_{c_i} \in \mathrm{R}^n$ chosen randomly (but mutually different) from the finite data set $\mathcal{D}$.

2. Compute for each unit $c \in \mathcal{A}$ its Voronoi set $\mathcal{R}_c$.

3. Move the reference vector of each unit to the mean of its Voronoi set:

$$\mathbf{w}_c = \frac{1}{|\mathcal{R}_c|} \sum_{\xi \in \mathcal{R}_c} \xi. \tag{4.4}$$

4. If in step 3 any of the $\mathbf{w}_c$ did change, continue with step 2.

5. Return the current set of reference vectors.

The steps 2 and 3 together form a so-called *Lloyd iteration*, which is guaranteed to decrease the distortion error or leave it at least unchanged. LBG is guaranteed to converge in a finite number of Lloyd iterations to a local minimum of the distortion error function (see figure 4.1 for an example).

An extension of LBG, called LBG-U (Fritzke, 1997), is often able to improve on the local minima found by LBG. LBG-U performs non-local moves of single reference vectors which do not contribute much to error reduction (and are, therefore, not *useful*, thus the "U" in LBG-U) to locations where large quantization error does occur. Thereafter, normal LBG is used to find the nearest local minimum of the distortion error function. This is iterated as long as the LBG-generated local minima improve. LBG-U requires a finite data set, too, and is guaranteed to converge in a finite number of steps.

4.2 On-line Update: Basic Algorithm

In some situations the data set $\mathcal{D}$ is so huge that batch methods become impractical. In other cases the input data comes as a continuous stream of unlimited length which makes it completely impossible to apply batch methods. A resort is *on-line update*, which can be described as follows:

1. Initialize the set $\mathcal{A}$ to contain N units c_i

$$\mathcal{A} = \{c_1, c_2, \ldots, c_N\} \tag{4.5}$$

   with reference vectors $\mathbf{w}_{c_i} \in \mathrm{R}^n$ chosen randomly according to $p(\xi)$.
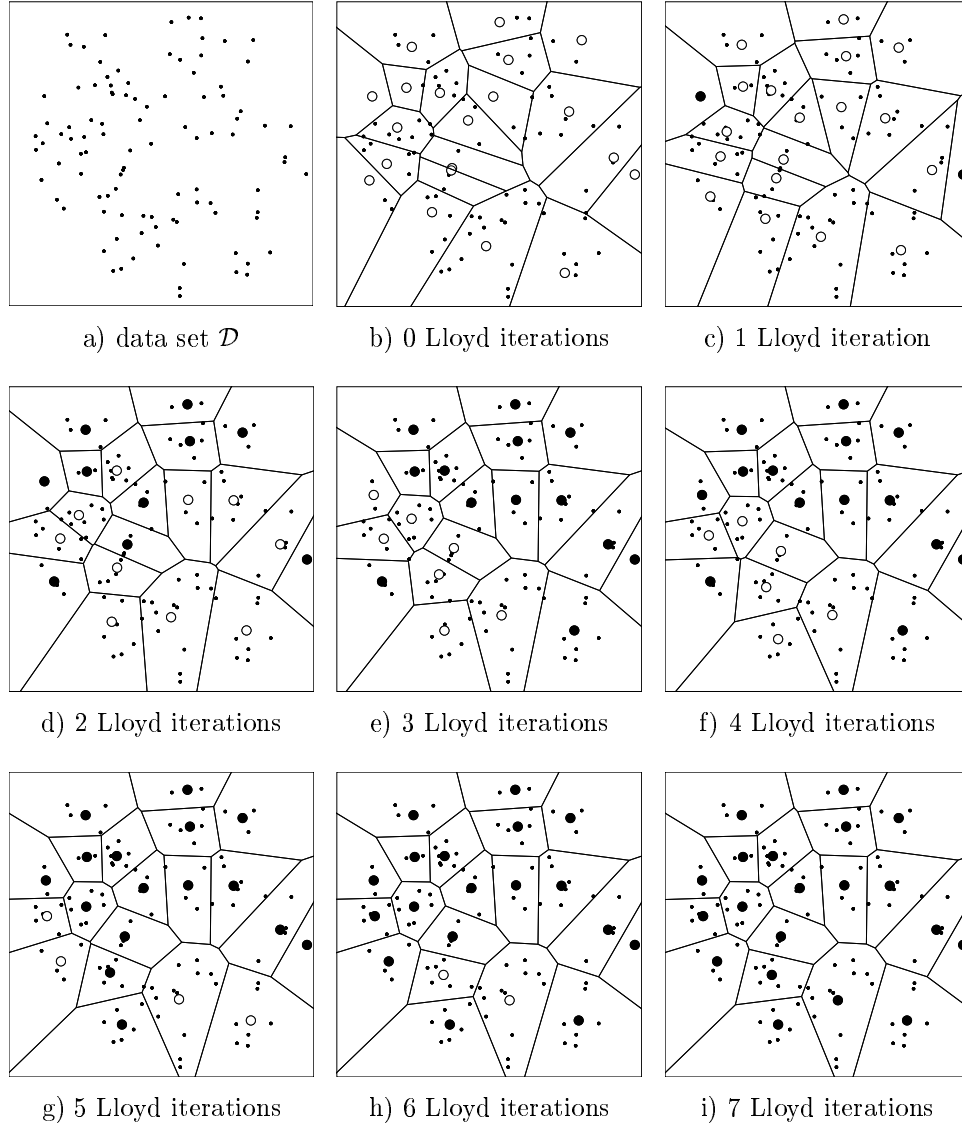
Figure 4.1: LBG simulation. a) The data set $\mathcal{D}$ consisting of 100 data items. b) 20 reference vectors have been initialized randomly from points in $\mathcal{D}$. The corresponding Voronoi tessellation is shown. c-i) The positions of the reference vectors after the indicated number of Lloyd iterations. Reference vectors which did not move during the previous Lloyd iteration are shown in black. In this simulation LBG has converged after 7 Lloyd iterations.

2. Generate at random an input signal $\boldsymbol{\xi}$ according to $p(\boldsymbol{\xi})$.

3. Determine the winner $s = s(\boldsymbol{\xi})$:

$$s(\boldsymbol{\xi}) = \arg\min_{c \in \mathcal{A}} \|\boldsymbol{\xi} - \mathbf{w}_c\|. \tag{4.6}$$

4. Adapt the reference vector of the winner towards $\boldsymbol{\xi}$:

$$\Delta \mathbf{w}_s = \epsilon\,(\boldsymbol{\xi} - \mathbf{w}_s). \tag{4.7}$$

5. Unless the maximum number of steps is reached continue with step 2.

Thereby, the *learning rate* ϵ determines the extent to which the winner is adapted towards the input signal. Depending on whether ϵ stays constant or decays over time, several different methods are possible some of which are described in the following.

4.3   Constant Learning Rate

If the learning rate is constant, i.e.

$$\epsilon = \epsilon_0, (0 < \epsilon_0 \leq 1), \tag{4.8}$$

then the value of each reference vector $\mathbf{w}_c$ represents an *exponentially decaying average* of those input signals for which the unit c has been winner. To see this, let $\boldsymbol{\xi}_1^c, \boldsymbol{\xi}_2^c, \ldots, \boldsymbol{\xi}_t^c$ be the sequence of input signals for which c is the winner. The sequence of successive values taken by $\mathbf{w}_c$ can then be written as

$$
\begin{aligned}
\mathbf{w}_c(0) &= \quad (\text{random signal according to } p(\boldsymbol{\xi})) \\[2mm]
\mathbf{w}_c(1) &= \quad \mathbf{w}_c(0) + \epsilon_0(\boldsymbol{\xi}_1^c - \mathbf{w}_c(0)) \\
&= \quad (1 - \epsilon_0)\mathbf{w}_c(0) + \epsilon_0\boldsymbol{\xi}_1^c \\[2mm]
\mathbf{w}_c(2) &= \quad (1 - \epsilon_0)\mathbf{w}_c(1) + \epsilon_0\boldsymbol{\xi}_2^c \\
&= \quad (1 - \epsilon_0)^2\mathbf{w}_c(0) + (1 - \epsilon_0)\epsilon_0\boldsymbol{\xi}_1^c + \epsilon_0\boldsymbol{\xi}_2^c \\[2mm]
&\;\;\vdots \\[2mm]
\mathbf{w}_c(t) &= \quad (1 - \epsilon_0)\mathbf{w}_c(t - 1) + \epsilon_0\boldsymbol{\xi}_t^c \\
&= \quad (1 - \epsilon_0)^t\mathbf{w}_c(0) + \epsilon_0\sum_{i=1}^{t}(1 - \epsilon_0)^{t-i}\boldsymbol{\xi}_i^c.
\end{aligned}
$$

$$(4.9)$$
$$(4.10)$$
$$(4.11)$$

From (4.8) and (4.11) it is obvious that the influence of past input signals decays exponentially fast with the number of further input signals for which c is winner (see also figure 4.2). The most recent input signal, however, always determines a fraction ϵ of the current value of $\mathbf{w}_c$. This has two consequences. First, such a system stays adaptive and is therefore in principle able to follow also non-stationary signal distribution $p(\boldsymbol{\xi})$. Second (and for the same reason), there is no convergence. Even after a large number of input signals the current input signal can cause a considerable change of the reference vector of the winner. A typical behavior of such a system in case of a stationary signal distribution is the following: the reference vectors drift from their initial positions to *quasi-stationary* positions where they
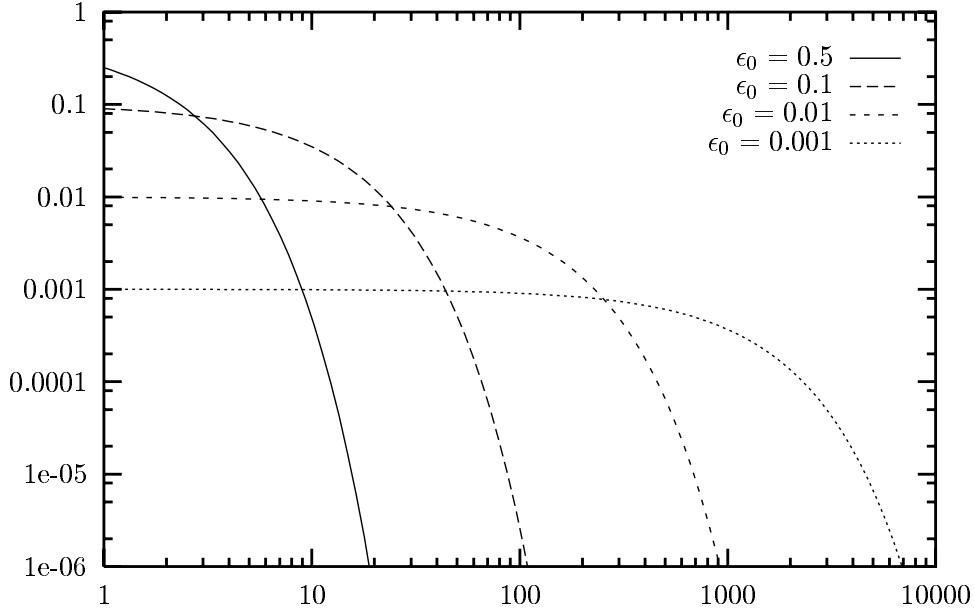
Figure 4.2: Influence of an input signal $\boldsymbol{\xi}$ on the vector of its winner s as a function of the number of following input signals for which s is winner (including $\boldsymbol{\xi}$). Results for different constant adaptation rates are shown. The respective section with the x-axis indicates how many signals are needed until the influence of $\boldsymbol{\xi}$ is below 10^{-6}. For example if the learning rate ϵ_0 is set to 0.5, about 10 additional signals (the section with the x-axis is near 11) are needed to let this happen.

start to wander around a dynamic equilibrium. Better quasi-stationary positions in terms of mean square error are achieved with smaller learning rates. In this case, however, the system also needs more adaptation steps to reach the quasi-stationary positions.

If the distribution is non-stationary then the information about the non-stationarity (how rapidly does the distribution change) can be used to set an appropriate learning rate. For rapidly changing distributions relatively large learning rates should be used and vice versa. Figure 4.3 shows some stages of a simulation for a simple ring-shaped data distribution. Figure 4.4 displays the final results after 40000 adaptation steps for three other distribution. In both cases a constant learning rate $\epsilon_0 = 0.05$ was used.

4.4 k-means

Instead of having a constant learning rate, we can also decrease it over time. A particularly interesting way of doing so is to have a separate learning rate for each unit $c \in \mathcal{A}$ and to set it according to the harmonic series:

$$\epsilon(t) = \frac{1}{t}. \tag{4.12}$$

Thereby, the time parameter t stands for the number of input signals for which this particular unit has been winner so far. This algorithm is known as k-means (MacQueen, 1967), which is a rather appropriate name, because each reference vector $\mathbf{w}_c(t)$ is always the exact arithmetic mean of the input signals $\boldsymbol{\xi}_1^c, \boldsymbol{\xi}_2^c, \ldots, \boldsymbol{\xi}_t^c$ it has been winner for so far. The sequence of successive values of $\mathbf{w}_c$ is the following:

a) 0 signals   b) 100 signals   c) 300 signals   d) 1000 signals

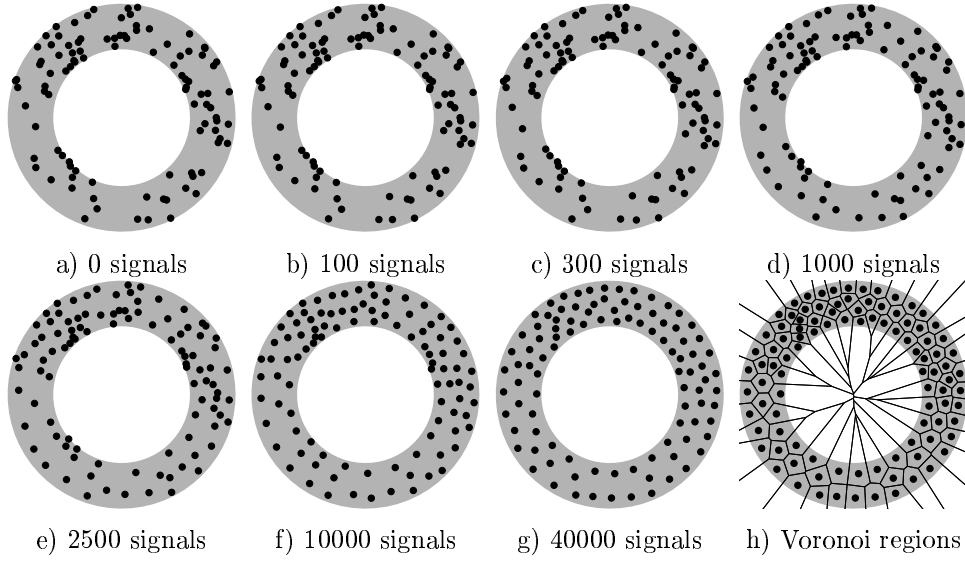e) 2500 signals   f) 10000 signals   g) 40000 signals   h) Voronoi regions

Figure 4.3: *Hard competitive learning* simulation sequence for a ring-shaped uniform probability distribution. A constant adaptation rate was used. a) Initial state. b-f) Intermediate states. g) Final state. h) Voronoi tessellation corresponding to the final state.

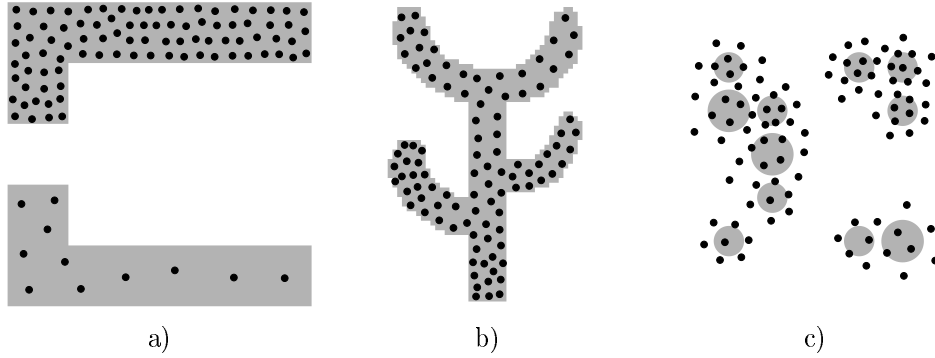a)   b)   c)

Figure 4.4: *Hard competitive learning* simulation results after 40000 input signals for three different probability distributions. A constant learning rate was used. a) This distribution is uniform within both shaded areas. The probability density, however, in the upper shaded area is 10 times as high as in the lower one. b) The distribution is uniform in the shaded area. c) In this distribution each of the 11 circles indicates the standard deviation of a Gaussian kernel which was used to generate the data. All Gaussian kernels have the same a priori probability.

$$\mathbf{w}_c(0) \quad = \quad \text{(random signal according to } p(\boldsymbol{\xi})\text{)}$$

$$
\begin{aligned}
\mathbf{w}_c(1) \quad &= \quad \mathbf{w}_c(0) + \epsilon(1)(\boldsymbol{\xi}_1^c - \mathbf{w}_c(0)) \\
&= \quad \boldsymbol{\xi}_1^c
\end{aligned}
\tag{4.13}
$$

$$
\begin{aligned}
\mathbf{w}_c(2) \quad &= \quad \mathbf{w}_c(1) + \epsilon(2)(\boldsymbol{\xi}_2^c - \mathbf{w}_c(1)) \\
&= \quad \frac{\boldsymbol{\xi}_1^c + \boldsymbol{\xi}_2^c}{2}
\end{aligned}
\tag{4.14}
$$

$$\vdots$$

$$
\begin{aligned}
\mathbf{w}_c(t) \quad &= \quad \mathbf{w}_c(t-1) + \epsilon(t)(\boldsymbol{\xi}_t^c - \mathbf{w}_c(t-1)) \\
&= \quad \frac{\boldsymbol{\xi}_1^c + \boldsymbol{\xi}_2^c + \ldots \boldsymbol{\xi}_t^c}{t}
\end{aligned}
\tag{4.15}
$$

One should note that the set of signals $\boldsymbol{\xi}_1^c, \boldsymbol{\xi}_2^c, \ldots, \boldsymbol{\xi}_t^c$ for which a particular unit c has been winner may contain elements which lie outside the current Voronoi region of c. The reason is that each adaptation of $\mathbf{w}_c$ changes the borders of the Voronoi region V_c. Therefore, although $\mathbf{w}_c(t)$ represents the arithmetic mean of the signals it has been winner for, at time t some of these signal may well lie in Voronoi regions belonging to other units.

Another important point about k-means is, that there is no strict convergence (as is present e.g. in LBG), the reason being that the sum of the harmonic series diverges:

$$\lim_{n \to \infty} \sum_{i=1}^{n} \frac{1}{i} = \infty. \tag{4.16}$$

Because of this divergence, even after a large number of input signals and correspondingly low values of the learning rate $\epsilon(t)$ arbitrarily large modifications of each input vector may occur in principal. Such large modification, however, are very improbable and in simulations where the signal distribution is stationary the reference vectors usually rather quickly take on values which are not much changed in the following. In fact, it has been shown that k-means does converge asymptotically to a configuration where each reference vector $\mathbf{w}_c$ is positioned such that it coincides with the expectation value

$$E(\boldsymbol{\xi}|\boldsymbol{\xi} \in V_c) = \int_{V_c} \boldsymbol{\xi} p(\boldsymbol{\xi}) d\boldsymbol{\xi} \tag{4.17}$$

of its Voronoi region V_c (MacQueen, 1965). One can note that (4.17) is the continuous variant of the centroid condition (4.2). Figure 4.5 shows some stages of a simulation for a simple ring-shaped data distribution. Figure 4.6 displays the final results after 40000 adaptation steps for three other distribution.

4.5   Exponentially Decaying Learning Rate

Another possibility for a decaying adaptation rate has been proposed by Ritter et al. (1991) in the context of self-organizing maps. They propose an exponential decay according to

$$\epsilon(t) = \epsilon_i (\epsilon_f / \epsilon_i)^{t/t_{\max}} \tag{4.18}$$
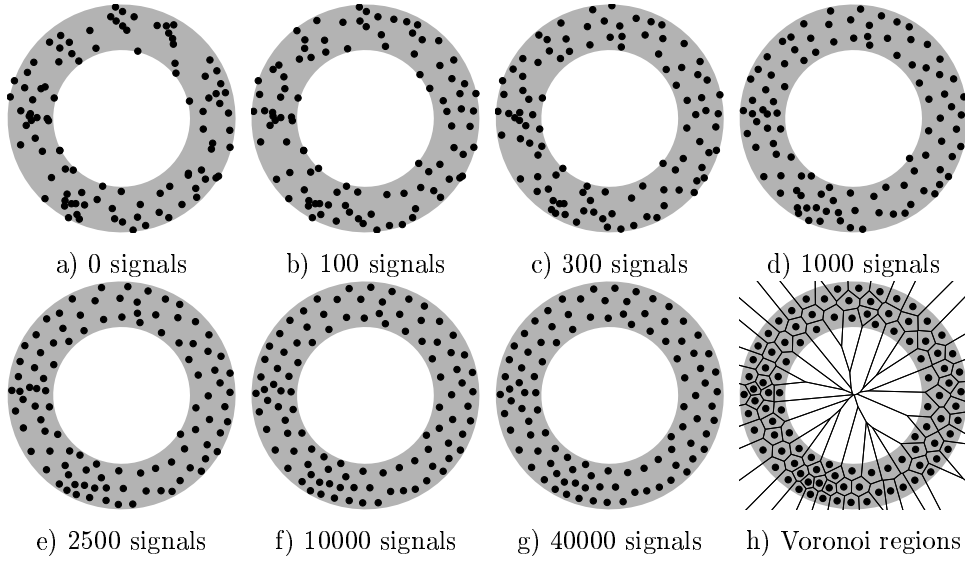
a) 0 signals   b) 100 signals   c) 300 signals   d) 1000 signals

e) 2500 signals   f) 10000 signals   g) 40000 signals   h) Voronoi regions

Figure 4.5: k-means simulation sequence for a ring-shaped uniform probability distribution. a) Initial state. b-f) Intermediate states. g) Final state. h) Voronoi tessellation corresponding to the final state. The final distribtion of the reference vectors still reflects the clusters present in the initial state (see in particular the region of higher vector density at the lower left).
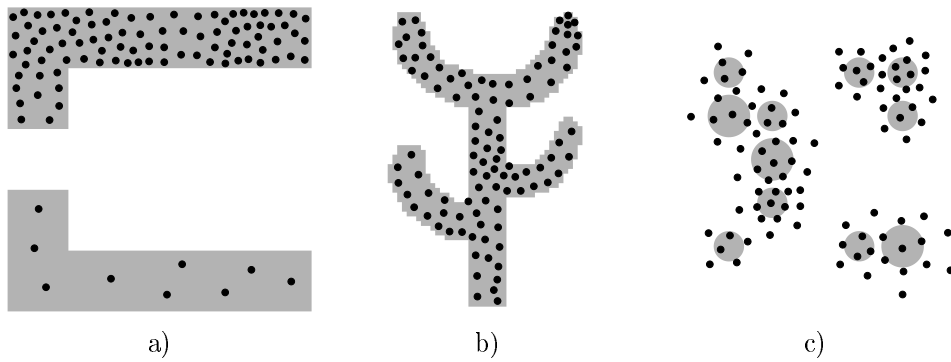

a)   b)   c)

Figure 4.6: k-means simulation results after 40000 input signals for three different probability distributions (described in the caption of figure 4.4).
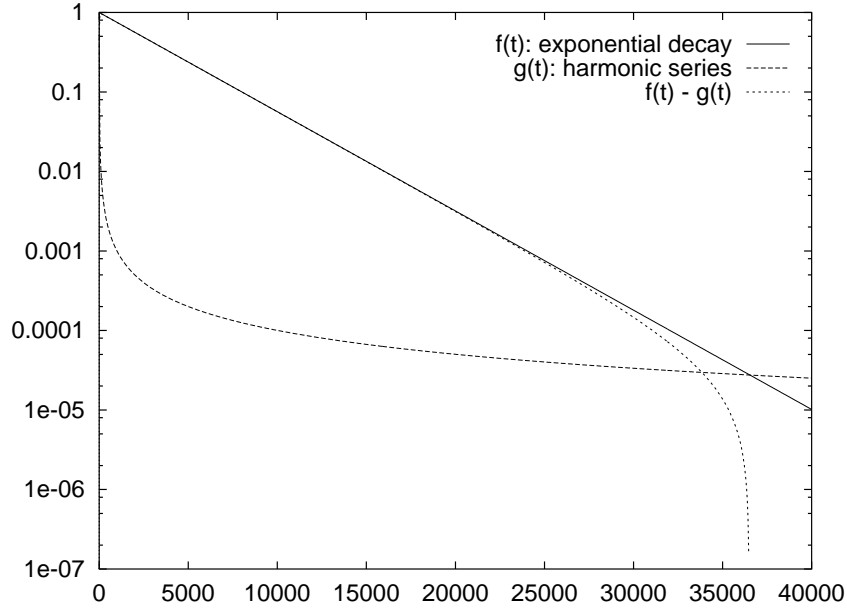
Figure 4.7: Comparison of the exponentially decaying learning function $f(t) = \epsilon_i(\epsilon_f/\epsilon_i)^{t/t_{\max}}$ and the harmonic series $g(t) = 1/t$ for a particular set of parameters ($\epsilon_i = 1.0$, $\epsilon_f = $ 1E-5, $t_{\max} = 40000$). The displayed difference between the two learning rates can be interpreted as noise which in the case of an exponentially decaying learning rate is introduced to the system and then gradually removed.

whereby ϵ_i and ϵ_f are initial and final values of the learning rate and $t_{\max}$ is the total number of adaptation steps which is taken.

In figure 4.7 this kind of learning rate is compared to the harmonic series for a specific choice of parameters. In particular at the beginning of the simulation the exponentially decaying learning rate is considerably larger than that dictated by the harmonic series. This can be interpreted as introducing noise to the system which is then gradually removed and, therefore, suggests a relationship to simulated annealing techniques (Kirkpatrick et al., 1983). Simulated annealing gives a system the ability to escape from poor local minima to which it might have been initialized. Preliminary experiments comparing k-means and hard competitive learning with a learning rate according to (4.18) indicate that the latter method is less susceptible to poor initialization and for many data distributions gives lower mean square error. Also small constant learning rates usually give better results than k-means. Only in the special case that only one reference vector exists ($|\mathcal{A}| = 1$) it is completely impossible to beat k-means on average, since in this case it realizes the optimal estimator (the mean of all samples occurred so far). These observations are in complete agreement with Darken and Moody (1990) who investigated k-means and a number of different learning rate schedules like constant learning rates and a learning rate which is the square root of the rate used by k-means ($\epsilon(t) = 1/\sqrt{(t)}$). Their results indicate that if k is larger than 1, then k-means is inferior to the other learning rate schedules. In the examples they give the difference in distortion error is up to two orders of magnitude.

Figure 4.8 shows some stages of a simulation for a simple ring-shaped data distribution. Figure 4.9 displays the final results after 40000 adaptation steps for three other distribution. The parameters used in both examples were: $\epsilon_i = 0.5$, $\epsilon_f = 0.0005$ and $t_{\max} = 40000$.
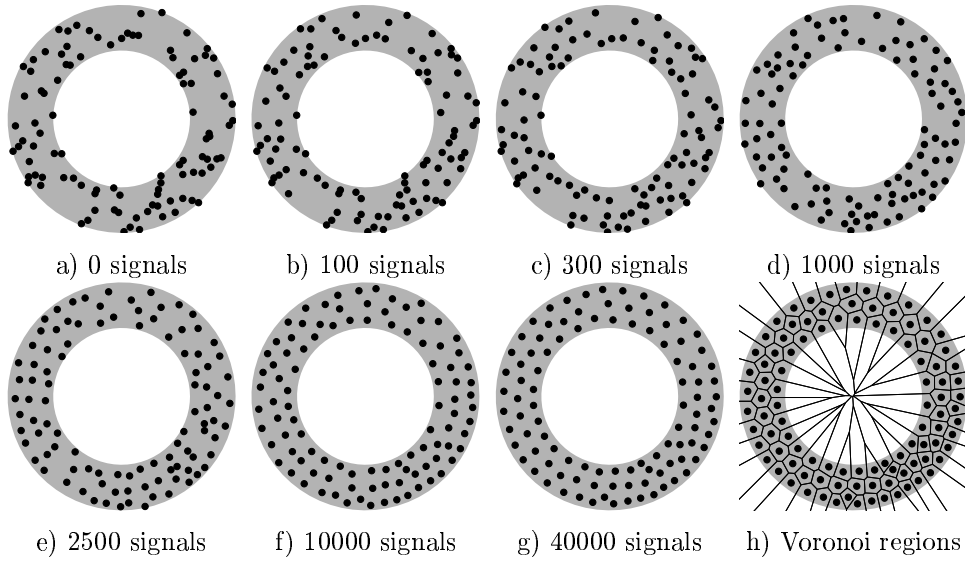
Figure 4.8: *Hard competitive learning* simulation sequence for a ring-shaped uniform probability distribution. An exponentially decaying learning rate was used. a) Initial state. b-f) Intermediate states. g) Final state. h) Voronoi tessellation corresponding to the final state.
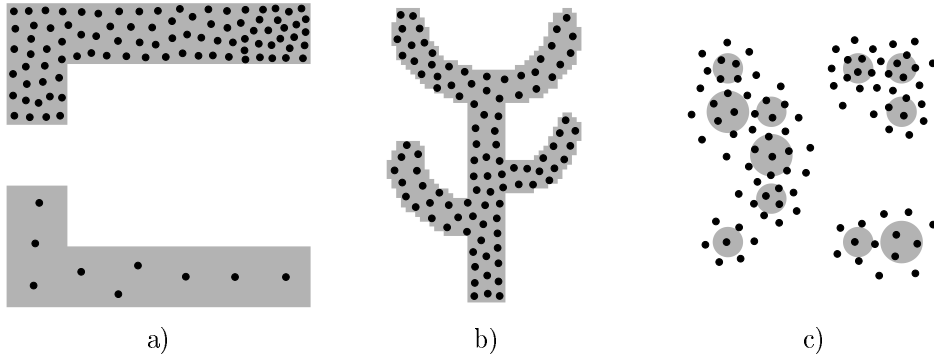

Figure 4.9: *Hard competitive learning* simulation results after 40000 input signals for three different probability distributions (described in the caption of figure 4.4). An exponentially decaying learning rate was used.

Chapter 5

Soft Competitive Learning without Fixed Network Dimensionality

In this chapter some methods from the area of soft competitive learning are described. They have in common, in contrast to the models in the following chapter, that no topology of a *fixed dimensionality* is imposed on the network. In one case there is no topology at all (*neural gas*). In other cases the dimensionality of the network depends on the *local* dimensionality of the data and may vary within the input space.

5.1 Neural Gas

The *neural gas* algorithm (Martinetz and Schulten, 1991) sorts for each input signal $\boldsymbol{\xi}$ the units of the network according to the distance of their reference vectors to $\boldsymbol{\xi}$. Based on this "rank order" a certain number of units is adapted. Both the number of adapted units and the adaptation strength are decreased according to a fixed schedule. The complete *neural gas* algorithm is the following:

1. Initialize the set $\mathcal{A}$ to contain N units c_i

$$\mathcal{A} = \{c_1, c_2, \ldots, c_N\} \tag{5.1}$$

    with reference vectors $\mathbf{w}_{c_i} \in \mathrm{R}^n$ chosen randomly according to $p(\boldsymbol{\xi})$.

    Initialize the time parameter t:
$$t = 0. \tag{5.2}$$

2. Generate at random an input signal $\boldsymbol{\xi}$ according to $p(\boldsymbol{\xi})$.

3. Order all elements of $\mathcal{A}$ according to their distance to $\boldsymbol{\xi}$, i.e., find the sequence of indices $(i_0, i_1, \ldots, i_{N-1})$ such that $\mathbf{w}_{i_0}$ is the reference vector closest to $\boldsymbol{\xi}$, $\mathbf{w}_{i_1}$ is the reference vector second-closest to $\boldsymbol{\xi}$ and $\mathbf{w}_{i_k}$, $k = 0, \ldots, N-1$ is the reference vector such that k vectors $\mathbf{w}_j$ exist with $\|\boldsymbol{\xi} - \mathbf{w}_j\| < \|\boldsymbol{\xi} - \mathbf{w}_k\|$. Following Martinetz et al. (1993) we denote with $k_i(\boldsymbol{\xi}, \mathcal{A})$ the number k associated with $\mathbf{w}_i$.

4. Adapt the reference vectors according to

$$\Delta\mathbf{w}_i = \epsilon(t) \cdot h_\lambda(k_i(\boldsymbol{\xi}, \mathcal{A})) \cdot (\boldsymbol{\xi} - \mathbf{w}_i) \tag{5.3}$$

with the following time-dependencies:

$$\lambda(t) = \lambda_i (\lambda_f / \lambda_i)^{t/t_{\max}}, \tag{5.4}$$

$$\epsilon(t) = \epsilon_i (\epsilon_f / \epsilon_i)^{t/t_{\max}}, \tag{5.5}$$

$$h_\lambda(k) = \exp(-k/\lambda(t)). \tag{5.6}$$

5. Increase the time parameter t:

$$t = t + 1. \tag{5.7}$$

6. If $t < t_{\max}$ continue with step 2

For the time-dependent parameters suitable initial values $(\lambda_i,\ \epsilon_i)$ and final values $(\lambda_f,\ \epsilon_f)$ have to be chosen. Figure 5.1 shows some stages of a simulation for a simple ring-shaped data distribution. Figure 5.2 displays the final results after 40000 adaptation steps for three other distribution. Following Martinetz et al. (1993) we used the following parameters: $\lambda_i = 10$, $\lambda_f = 0.01$, $\epsilon_i = 0.5$, $\epsilon_f = 0.005$, $t_{\max} = 40000$.

5.2 Competitive Hebbian Learning

This method (Martinetz and Schulten, 1991; Martinetz, 1993) is usually not used on its own but in conjunction with other methods (see sections 5.3 and 5.4). It is, however, instructive to study *competitive Hebbian learning* on its own. The method does not change reference vectors at all (which could be interpreted as having a zero learning rate). It only generates a number of neighborhood edges between the units of the network. It was proved by Martinetz (1993) that the so generated graph is optimally topology-preserving in a very general sense. In particular each edge of this graph belongs to the Delaunay triangulation corresponding to the given set of reference vectors. The complete *competitive Hebbian learning* algorithm is the following:

1. Initialize the set $\mathcal{A}$ to contain N units c_i

$$\mathcal{A} = \{c_1,\ c_2,\ \ldots,\ c_N\} \tag{5.8}$$

with reference vectors $\mathbf{w}_{c_i} \in \mathbb{R}^n$ chosen randomly according to $p(\boldsymbol{\xi})$.

Initialize the connection set $\mathcal{C}$, $\mathcal{C} \subset \mathcal{A} \times \mathcal{A}$, to the empty set:

$$\mathcal{C} = \emptyset. \tag{5.9}$$

2. Generate at random an input signal $\boldsymbol{\xi}$ according to $p(\boldsymbol{\xi})$.

3. Determine units s_1 and s_2 $(s_1, s_2 \in \mathcal{A})$ such that

$$s_1 = \arg\min_{c \in \mathcal{A}} \|\boldsymbol{\xi} - \mathbf{w}_c\| \tag{5.10}$$

and

$$s_2 = \arg\min_{c \in \mathcal{A} \setminus \{s_1\}} \|\boldsymbol{\xi} - \mathbf{w}_c\|. \tag{5.11}$$

4. If a connection between s_1 and s_2 does not exist already, create it:

$$\mathcal{C} = \mathcal{C} \cup \{(s_1, s_2)\}. \tag{5.12}$$

5. Continue with step 2 unless the maximum number of signals is reached.

Figure 5.3 shows some stages of a simulation for a simple ring-shaped data distribution. Figure 5.4 displays the final results after 40000 adaptation steps for three other distribution.

a) 0 signals   b) 100 signals   c) 300 signals   d) 1000 signals

e) 2500 signals   f) 10000 signals   g) 40000 signals   h) Voronoi regions
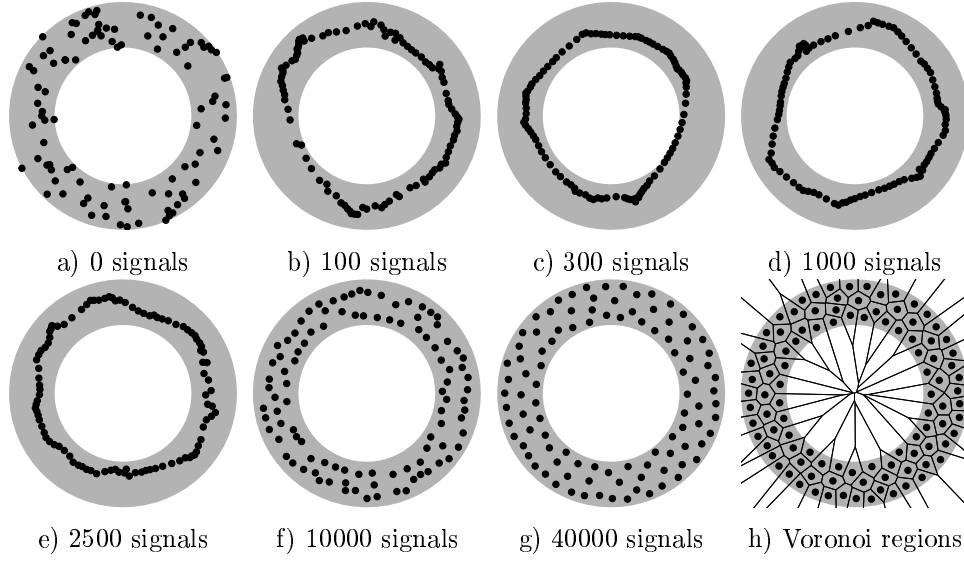
Figure 5.1: *Neural gas* simulation sequence for a ring-shaped uniform probability distribution. a) Initial state. b-f) Intermediate states. g) Final state. h) Voronoi tessellation corresponding to the final state. Initially strong neighborhood interaction leads to a clustering of the reference vectors which then relaxes until at the end a rather even distribution of reference vectors is found.

a)   b)   c)
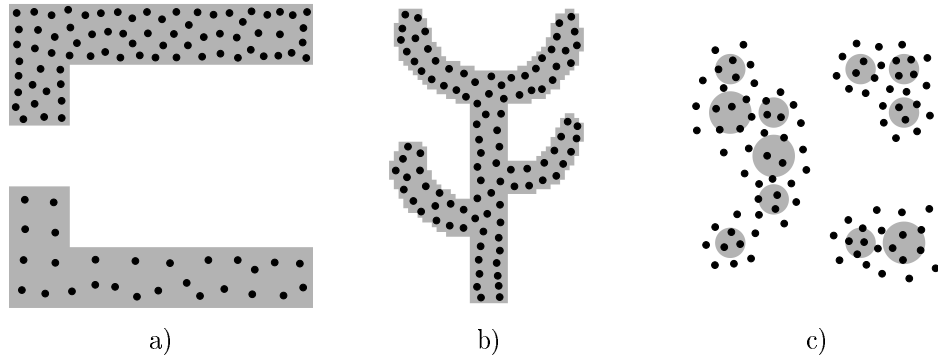
Figure 5.2: *Neural gas* simulation results after 40000 input signals for three different probability distributions (described in the caption of figure 4.4).

a) 0 signals   b) 100 signals   c) 300 signals   d) 1000 signals

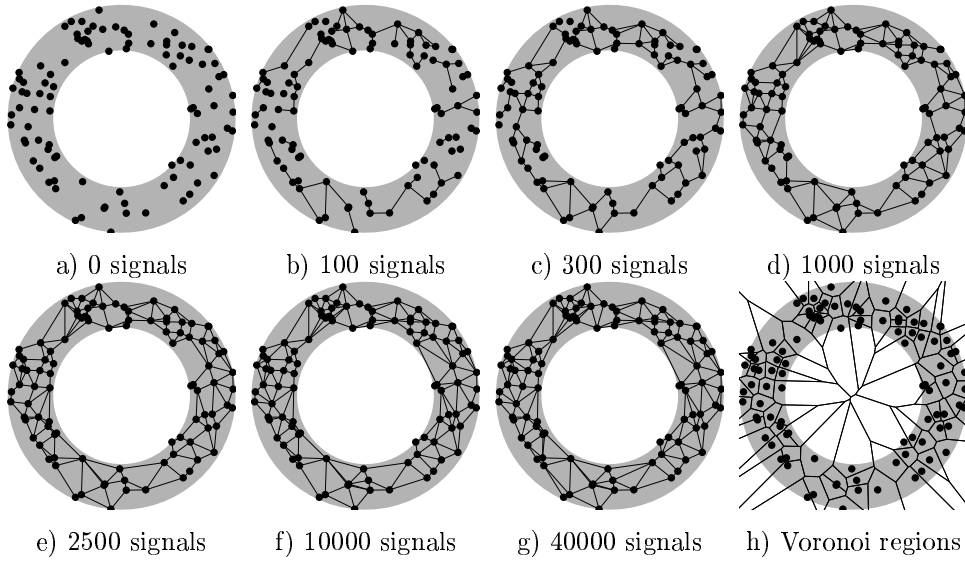e) 2500 signals   f) 10000 signals   g) 40000 signals   h) Voronoi regions

Figure 5.3: *Competitive Hebbian learning* simulation sequence for a ring-shaped uniform probability distribution. a) Initial state. b-f) Intermediate states. g) Final state. h) Voronoi tessellation corresponding to the final state. Obviously, the method is sensitive to initialization since the initial positions are always equal to the final positions.
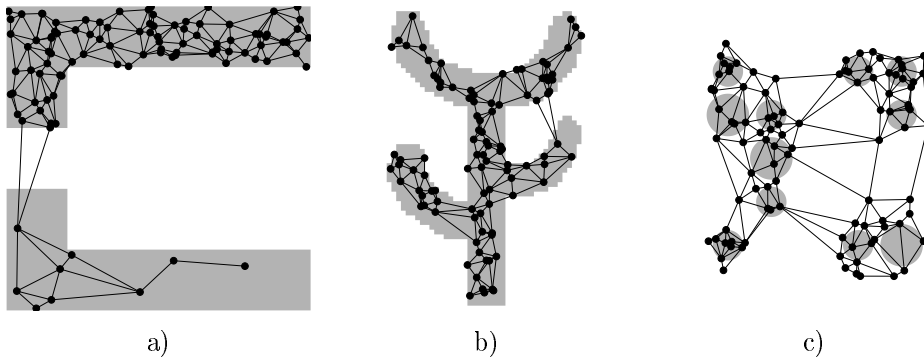

a)   b)   c)

Figure 5.4: *Competitive Hebbian learning* simulation results after 40000 input signals for three different probability distributions (described in the caption of figure 4.4).

5.3   Neural Gas plus Competitive Hebbian Learning

This method (Martinetz and Schulten, 1991, 1994) is a straight-forward superposition of *neural gas* and *competitive Hebbian learning*. It is sometimes denoted as "topology-representing networks" (Martinetz and Schulten, 1994). This term, however, is rather general and would apply also to the *growing neural gas* model described later.

At each adaptation step a connection between the winner and the second-nearest unit is created (this is *competitive Hebbian learning*). Since the reference vectors are adapted according to the *neural gas* method a mechanism is needed to remove edges which are not valid anymore. This is done by a local edge aging mechanism. The complete *neural gas with competitive Hebbian learning* algorithm is the following:

1. Initialize the set $\mathcal{A}$ to contain N units c_i

$$\mathcal{A} = \{c_1, c_2, \ldots, c_N\} \qquad (5.13)$$

   with reference vectors $\mathbf{w}_{c_i} \in \mathrm{R}^n$ chosen randomly according to $p(\boldsymbol{\xi})$.

   Initialize the connection set $\mathcal{C}$, $\mathcal{C} \subset \mathcal{A} \times \mathcal{A}$, to the empty set:

$$\mathcal{C} = \emptyset. \qquad (5.14)$$

   Initialize the time parameter t:

$$t = 0. \qquad (5.15)$$

2. Generate at random an input signal $\boldsymbol{\xi}$ according to $p(\boldsymbol{\xi})$.

3. Order all elements of $\mathcal{A}$ according to their distance to $\boldsymbol{\xi}$, i.e., find the sequence of indices $(i_0, i_1, \ldots, i_{N-1})$ such that $\mathbf{w}_{i_0}$ is the reference vector closest to $\boldsymbol{\xi}$, $\mathbf{w}_{i_1}$ is the reference vector second-closest to $\boldsymbol{\xi}$ and $\mathbf{w}_{i_k}$, $k = 0, \ldots, N-1$ is the reference vector such that k vectors $\mathbf{w}_j$ exist with $\|\boldsymbol{\xi} - \mathbf{w}_j\| < \|\boldsymbol{\xi} - \mathbf{w}_k\|$. Following Martinetz et al. (1993) we denote with $k_i(\boldsymbol{\xi}, \mathcal{A})$ the number k associated with $\mathbf{w}_i$.

4. Adapt the reference vectors according to

$$\Delta\mathbf{w}_i = \epsilon(t) \cdot h_\lambda(k_i(\boldsymbol{\xi}, \mathcal{A})) \cdot (\boldsymbol{\xi} - \mathbf{w}_i) \qquad (5.16)$$

   with the following time-dependencies:

$$\lambda(t) = \lambda_i(\lambda_f/\lambda_i)^{t/t_{\max}}, \qquad (5.17)$$

$$\epsilon(t) = \epsilon_i(\epsilon_f/\epsilon_i)^{t/t_{\max}}, \qquad (5.18)$$

$$h_\lambda(k) = \exp(-k/\lambda(t)). \qquad (5.19)$$

5. If it does not exist already, create a connection between i_0 and i_1:

$$\mathcal{C} = \mathcal{C} \cup \{(i_0, i_1)\}. \qquad (5.20)$$

   Set the age of the connection between i_0 and i_1 to zero ("refresh" the edge):

$$\mathrm{age}_{(i_0, i_1)} = 0. \qquad (5.21)$$

6. Increment the age of all edges emanating from i_0:

$$\text{age}_{(i_0,i)} = \text{age}_{(i_0,i)} + 1 \qquad (\forall i \in N_{i_0}).$$ (5.22)

Thereby, N_c is the set of direct topological neighbors of c (see equation 2.5).

7. Remove edges with an age larger than the maximal age $T(t)$ whereby

$$T(t) = T_i (T_f/T_i)^{t/t_{\max}}.$$ (5.23)

8. Increase the time parameter t:

$$t = t + 1.$$ (5.24)

9. If $t < t_{\max}$ continue with step 2.

For the time-dependent parameters suitable initial values (λ_i, ϵ_i, T_i) and final values (λ_f, ϵ_f, T_f) have to be chosen.

Figure 5.5 shows some stages of a simulation for a simple ring-shaped data distribution. Figure 5.6 displays the final results after 40000 adaptation steps for three other distribution. Following Martinetz et al. (1993) we used the following parameters: $\lambda_i = 10$, $\lambda_f = 0.01$, $\epsilon_i = 0.5$, $\epsilon_f = 0.005$, $t_{\max} = 40000$, $T_i = 20$, $T_f = 200$. The network size N was set to 100.

5.4 Growing Neural Gas

This method (Fritzke, 1994b, 1995a) is different from the previously described models since the number of units is changed (mostly increased) during the self-organization process. The growth mechanism from the earlier proposed *growing cell structures* (Fritzke, 1994a) and the topology generation of *competitive Hebbian learning* (Martinetz and Schulten, 1991) are combined to a new model. Starting with very few units new units are inserted successively. To determine where to insert new units, local error measures are gathered during the adaptation process. Each new unit is inserted near the unit which has accumulated most error. The complete *growing neural gas* algorithm is the following:

1. Initialize the set $\mathcal{A}$ to contain two units c_1 and c_2

$$\mathcal{A} = \{c_1, c_2\}$$ (5.25)

with reference vectors chosen randomly according to $p(\boldsymbol{\xi})$.

Initialize the connection set $\mathcal{C}$, $\mathcal{C} \subset \mathcal{A} \times \mathcal{A}$, to the empty set:

$$\mathcal{C} = \emptyset.$$ (5.26)

2. Generate at random an input signal $\boldsymbol{\xi}$ according to $p(\boldsymbol{\xi})$.

3. Determine the winner s_1 and the second-nearest unit s_2 ($s_1, s_2 \in \mathcal{A}$) by

$$s_1 = \arg\min_{c \in \mathcal{A}} \|\boldsymbol{\xi} - \mathbf{w}_c\|$$ (5.27)

and

$$s_2 = \arg\min_{c \in \mathcal{A} \setminus \{s_1\}} \|\boldsymbol{\xi} - \mathbf{w}_c\|.$$ (5.28)

a) 0 signals   b) 100 signals   c) 300 signals   d) 1000 signals

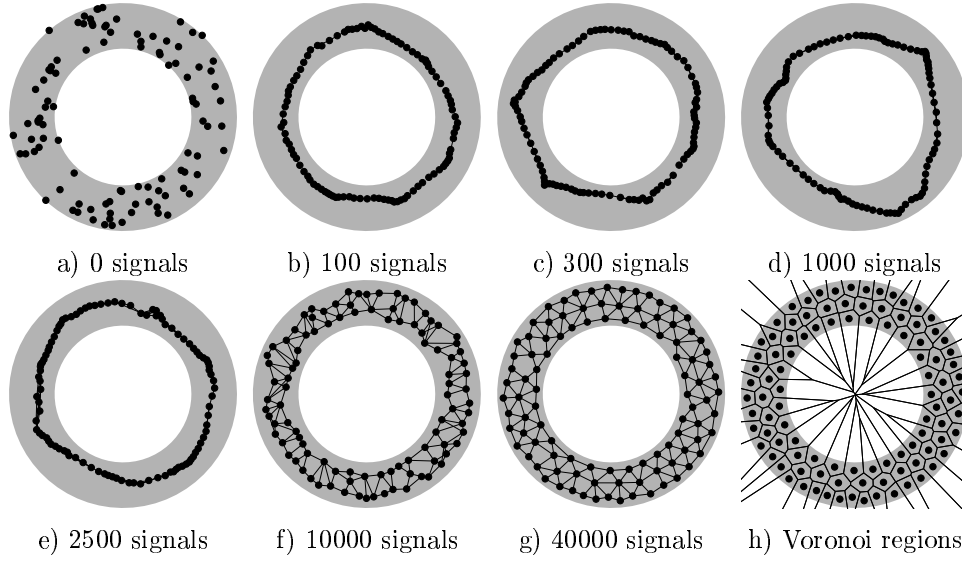e) 2500 signals   f) 10000 signals   g) 40000 signals   h) Voronoi regions

Figure 5.5: *Neural gas with competitive Hebbian learning* simulation sequence for a ring-shaped uniform probability distribution. a) Initial state. b-f) Intermediate states. g) Final state. h) Voronoi tessellation corresponding to the final state. The centers move according to the *neural gas* algorithm. Additionally, however, edges are created by *competitive Hebbian learning* and removed if they are not "refreshed" for a while.

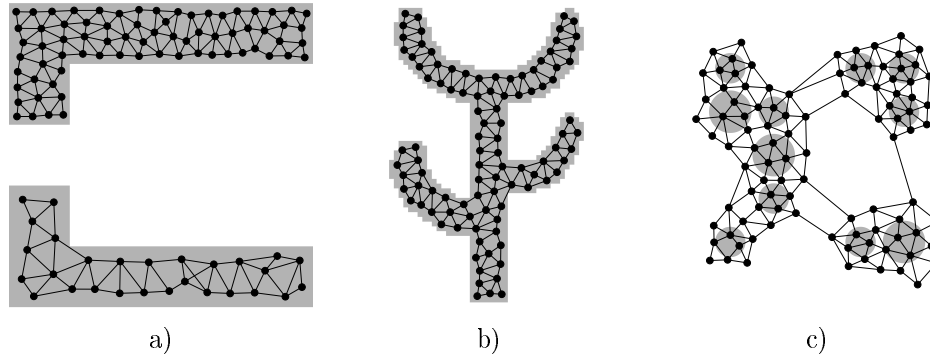a)   b)   c)

Figure 5.6: *Neural gas with competitive Hebbian learning* simulation results after 40000 input signals for three different probability distributions (described in the caption of figure 4.4).

4. If a connection between s_1 and s_2 does not exist already, create it:

$$\mathcal{C} = \mathcal{C} \cup \{(s_1, s_2)\}. \tag{5.29}$$

   Set the age of the connection between s_1 and s_2 to zero ("refresh" the edge):

$$\text{age}_{(s_1,s_2)} = 0. \tag{5.30}$$

5. Add the squared distance between the input signal and the winner to a local error variable:

$$\Delta \mathrm{E}_{s_1} = \|\boldsymbol{\xi} - \mathbf{w}_{s_1}\|^2. \tag{5.31}$$

6. Adapt the reference vectors of the winner and its direct topological neighbors by fractions ϵ_b and ϵ_n, respectively, of the total distance to the input signal:

$$\Delta \mathbf{w}_{s_1} = \epsilon_b(\boldsymbol{\xi} - \mathbf{w}_{s_1}) \tag{5.32}$$
$$\Delta \mathbf{w}_i = \epsilon_n(\boldsymbol{\xi} - \mathbf{w}_i) \qquad (\forall i \in N_{s_1}). \tag{5.33}$$

   Thereby N_{s_1} (see equation 2.5) is the set of direct topological neighbors of s_1.

7. Increment the age of all edges emanating from s_1:

$$\text{age}_{(s_1,i)} = \text{age}_{(s_1,i)} + 1 \qquad (\forall i \in N_{s_1}). \tag{5.34}$$

8. Remove edges with an age larger than a_{max}. If this results in units having no more emanating edges, remove those units as well.

9. If the number of input signals generated so far is an integer multiple of a parameter λ, insert a new unit as follows:

   - Determine the unit q with the maximum accumulated error:

$$q = \arg \max_{c \in \mathcal{A}} \mathrm{E}_c. \tag{5.35}$$

   - Determine among the neighbors of q the unit f with the maximum accumulated error:

$$f = \arg \max_{c \in N_q} \mathrm{E}_c. \tag{5.36}$$

   - Add a new unit r to the network and interpolate its reference vector from q and f.

$$\mathcal{A} = \mathcal{A} \cup \{r\}, \qquad \mathbf{w}_r = (\mathbf{w}_q + \mathbf{w}_f)/2. \tag{5.37}$$

   - Insert edges connecting the new unit r with units q and f, and remove the original edge between q and f:

$$\mathcal{C} = \mathcal{C} \cup \{(r,q), (r,f)\}, \qquad \mathcal{C} = \mathcal{C} \setminus \{(q,f)\} \tag{5.38}$$

   - Decrease the error variables of q and f by a fraction α:

$$\Delta \mathrm{E}_q = -\alpha \mathrm{E}_q, \qquad \Delta \mathrm{E}_f = -\alpha \mathrm{E}_f. \tag{5.39}$$

   - Interpolate the error variable of r from q and f:

$$\mathrm{E}_r = (\mathrm{E}_q + \mathrm{E}_f)/2. \tag{5.40}$$

10. Decrease the error variables of all units:

$$\Delta \mathrm{E}_c = -\beta \, \mathrm{E}_c \qquad (\forall c \in \mathcal{A}). \tag{5.41}$$

11. If a stopping criterion (e.g., net size or some performance measure) is not yet fulfilled continue with step 2.

Figure 5.7 shows some stages of a simulation for a simple ring-shaped data distribution. Figure 5.8 displays the final results after 40000 adaptation steps for three other distribution. The parameters used in both simulations were: $\lambda = 300$, $\epsilon_b = 0.05$, $\epsilon_n = 0.0006$, $\alpha = 0.5$, $\beta = 0.0005$, $a_{max} = 88$.
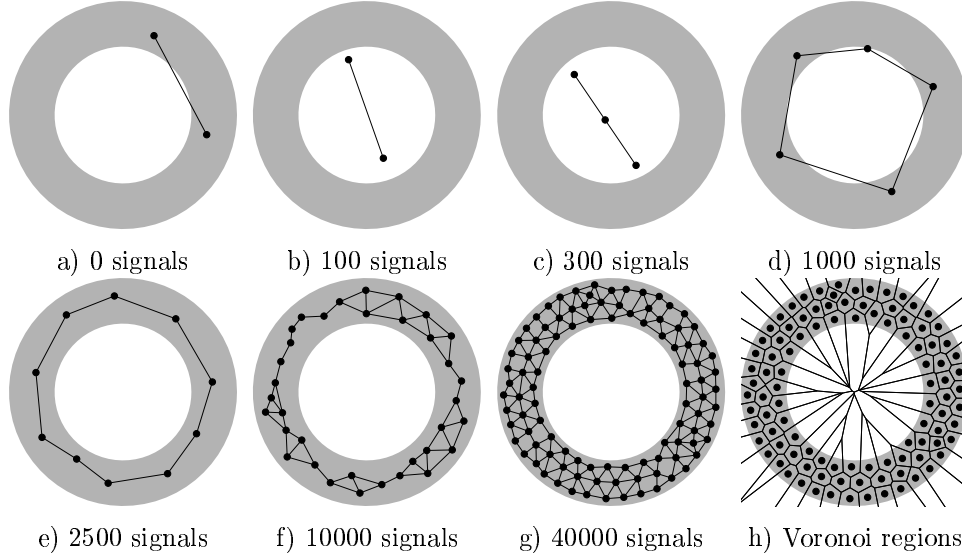
Figure 5.7: *Growing neural gas* simulation sequence for a ring-shaped uniform probability distribution. a) Initial state. b-f) Intermediate states. g) Final state. h) Voronoi tessellation corresponding to the final state. The maximal network size was set to 100.
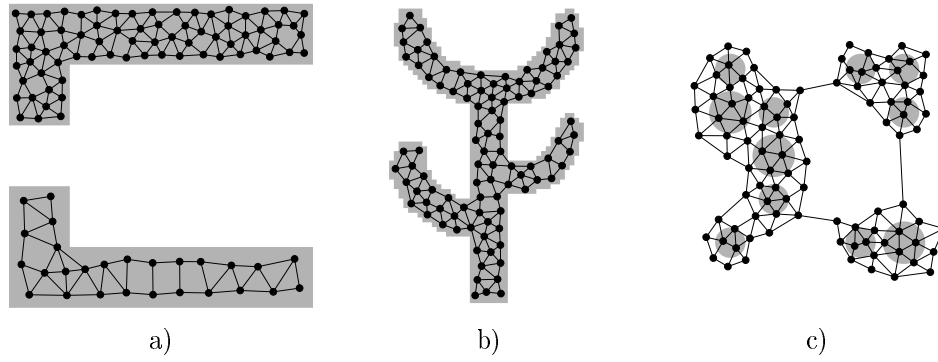

Figure 5.8: *Growing neural gas* simulation results after 40000 input signals for three different probability distributions (described in the caption of figure 4.4).

5.5   Other Methods

Several other models without a fixed network dimensionality are known. DeSieno (1988) proposed a method where frequent winners get a "bad conscience" for winning so often and, therefore, add a penalty term to the distance from the input signal. This leads eventually to a situation where each unit wins approximately equally often (entropy maximization).

Kangas et al. (1990) proposed to use the minimum spanning tree among the units as neighborhood topology to eliminate the *a priori* choice for a topology in some models.

Some other methods have been proposed.

Chapter 6

Soft Competitive Learning with Fixed Network Dimensionality

In this chapter methods from the area of soft competitive learning are described which have a network of a fixed dimensionality k which has to be chosen in advance. One advantage of a fixed network dimensionality is that such a network defines a mapping from the n-dimensional input space (with n being arbitrarily large) to the k-dimensional structure. This makes it possible to get a low-dimensional representation of the data which may be used for visualization purposes.

6.1 Self-organizing Feature Map

This model stems from Kohonen (1982) and builds upon earlier work of Willshaw and von der Malsburg (1976). The model is similar to the (much later developed) *neural gas* model (see 5.1) since a decaying neighborhood range and adaptation strength are used. An important difference, however, is the topology which is constrained to be a two-dimensional grid (a_{ij}) and does not change during self-organization.

The distance on this grid is used to determine how strongly a unit $r = a_{km}$ is adapted when the unit $s = a_{ij}$ is the winner. The distance measure is the L_1-norm (a.k.a. "Manhattan distance"):

$$d_1(r,s) = |i - k| + |j - m| \qquad \text{for } r = a_{km} \text{ and } s = a_{ij}. \tag{6.1}$$

Ritter et al. (1991) propose to use the following function to define the relative strength of adaptation for an arbitrary unit r in the network (given that s is the winner):

$$h_{rs} = \exp(\frac{-d_1(r,s)^2}{2\sigma^2}). \tag{6.2}$$

Thereby, the standard deviation σ of the Gaussian is varied according to

$$\sigma(t) = \sigma_i(\sigma_f/\sigma_i)^{t/t_{\max}} \tag{6.3}$$

for a suitable initial value σ_i and a final value σ_f. The complete *self-organizing feature map* algorithm is the following:

1. Initialize the set $\mathcal{A}$ to contain $N = N_1 \cdot N_2$ units c_i

$$\mathcal{A} = \{c_1, c_2, \ldots, c_N\} \tag{6.4}$$

with reference vectors $\mathbf{w}_{c_i} \in \mathrm{R}^n$ chosen randomly according to $p(\boldsymbol{\xi})$.

Initialize the connection set $\mathcal{C}$ to form a rectangular $N_1 \times N_2$ grid.

Initialize the time parameter t:

$$t = 0. \tag{6.5}$$

2. Generate at random an input signal $\boldsymbol{\xi}$ according to $p(\boldsymbol{\xi})$.

3. Determine the winner $s(\boldsymbol{\xi}) = s$:

$$s(\boldsymbol{\xi}) = \arg \min_{c \in \mathcal{A}} \|\boldsymbol{\xi} - \mathbf{w}_c\|. \tag{6.6}$$

4. Adapt each unit r according to

$$\Delta \mathbf{w}_r = \epsilon(t) \, h_{rs}(\boldsymbol{\xi} - \mathbf{w}_r) \tag{6.7}$$

whereby

$$\sigma(t) = \sigma_i (\sigma_f / \sigma_i)^{t/t_{\max}} \tag{6.8}$$

and

$$\epsilon(t) = \epsilon_i (\epsilon_f / \epsilon_i)^{t/t_{\max}}. \tag{6.9}$$

5. Increase the time parameter t:

$$t = t + 1. \tag{6.10}$$

6. If $t < t_{\max}$ continue with step 2.

Figure 6.1 shows some stages of a simulation for a simple ring-shaped data distribution. Figure 6.2 displays the final results after 40000 adaptation steps for three other distribution. The parameters were $\sigma_i = 3.0$, $\sigma_f = 0.1$, $\epsilon_i = 0.5$, $\epsilon_f = 0.005$, $t_{\max} = 10000$ and $N_1 = N_2 = 10$.

6.2   Growing Cell Structures

This model (Fritzke, 1994a) is rather similar to the *growing neural gas* model[1]. The main difference is that the network topology is constrained to consist of k-dimensional simplices whereby k is some positive integer chosen in advance. The basic building block and also the initial configuration of each network is a k-dimensional simplex. This is, e.g., a line for k=1, a triangle for k=2, and a tetrahedron for k=3.

For a given network configuration a number of adaptation steps are used to update the reference vectors of the nodes and to gather local error information at each node.

This error information is used to decide where to insert new nodes. A new node is always inserted by splitting the *longest edge* emanating from the node q with maximum accumulated error. In doing this, additional edges are inserted such that the resulting structure consists exclusively of k-dimensional simplices again.

The *growing cell structures* learning procedure is described in the following:

---

[1]Compared to the original *growing cell structures* algorithm described by Fritzke (1994a) slight changes and simplifications have been done regarding the re-distribution of accumulated error. Moreover, the discussion of removal of units has been left out completely for sake of brevity.

a) 0 signals   b) 100 signals   c) 300 signals   d) 1000 signals

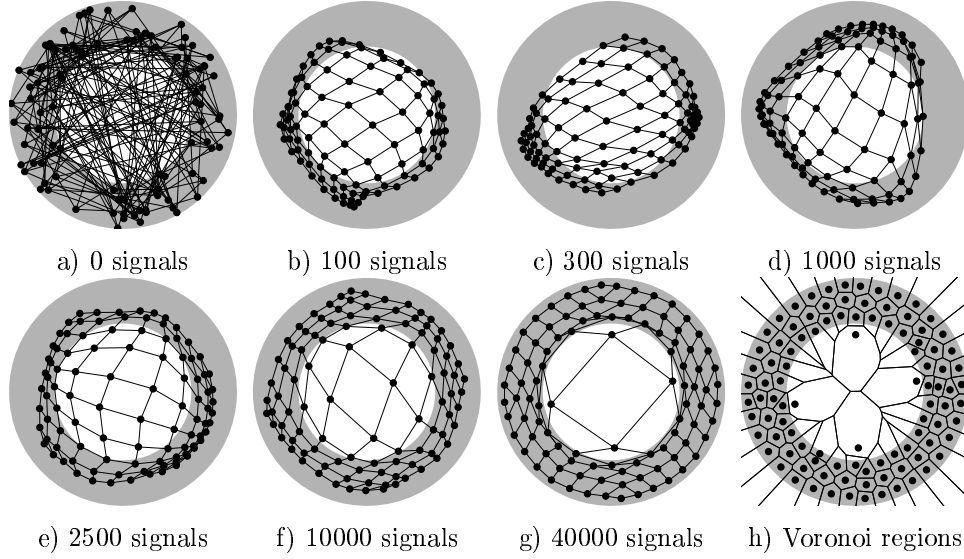e) 2500 signals   f) 10000 signals   g) 40000 signals   h) Voronoi regions

Figure 6.1: *Self-organizing feature map* simulation sequence for a ring-shaped uniform probability distribution. a) Initial state. b-f) Intermediate states. g) Final state. h) Voronoi tessellation corresponding to the final state. Large adaptation rates in the beginning as well as a large neighborhood range cause strong initial adaptations which decrease towards the end.

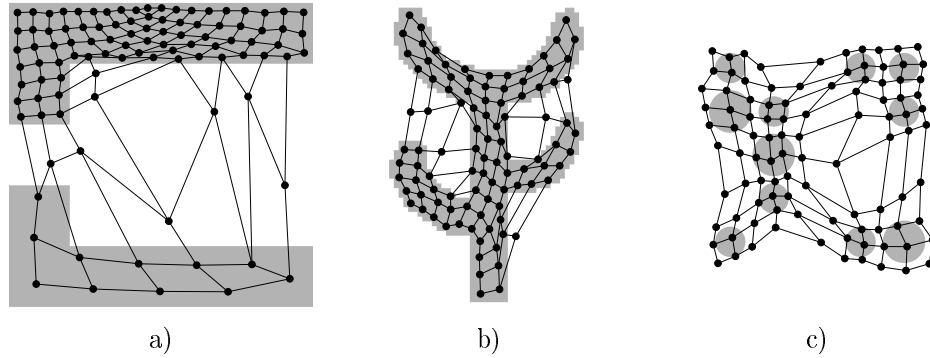a)   b)   c)

Figure 6.2: *Self-organizing feature map* simulation results after 40000 input signals for three different probability distributions (described in the caption of figure 4.4).

1. Choose a network dimensionality k.

   Initialize the set $\mathcal{A}$ to contain $k+1$ units c_i

   $$\mathcal{A} = \{c_1, \, c_2, \, \ldots, \, c_{k+1}\} \qquad (6.11)$$

   with reference vectors $\mathbf{w}_{c_i} \in \mathrm{R}^n$ chosen randomly according to $p(\boldsymbol{\xi})$.

   Initialize the connection set $\mathcal{C}$, $\mathcal{C} \subset \mathcal{A} \times \mathcal{A}$ such that each unit is connected to each other unit, i.e., such that the network has the topology of a k-dimensional simplex.

2. Generate at random an input signal $\boldsymbol{\xi}$ according to $p(\boldsymbol{\xi})$.

3. Determine the winner s:

   $$s(\boldsymbol{\xi}) = \arg\,\min_{c \in \mathcal{A}} \|\boldsymbol{\xi} - \mathbf{w}_c\|. \qquad (6.12)$$

4. Add the squared distance[2] between the input signal and the winner unit s to a local error variable E_s:

   $$\Delta \mathrm{E}_s = \|\boldsymbol{\xi} - \mathbf{w}_s\|^2. \qquad (6.13)$$

5. Adapt the reference vectors of s and its direct topological neighbors towards ξ by fractions ϵ_b and ϵ_n, respectively, of the total distance:

   $$\begin{aligned}
   \Delta \mathbf{w}_s &= \epsilon_b (\boldsymbol{\xi} - \mathbf{w}_s) & (6.14) \\
   \Delta \mathbf{w}_i &= \epsilon_n (\boldsymbol{\xi} - \mathbf{w}_i) & (\forall i \in N_s). & (6.15)
   \end{aligned}$$

   Thereby, we denote with N_s the set of direct topological neighbors of s.

6. If the number of input signals generated so far is an integer multiple of a parameter λ, insert a new unit as follows:

   - Determine the unit q with the maximum accumulated error:

     $$q = \arg\,\max_{c \in \mathcal{A}} E_c. \qquad (6.16)$$

   - Insert a new unit r by splitting the longest edge emanating from q, say an edge leading to a unit f. Insert the connections (q, r) and (r, f) and remove the original connection (q, f). To re-build the structure such that it again consists only of k-dimensional simplices, the new unit r is also connected with all common neighbors of q and f, i.e., with all units in the set $N_q \cap N_f$.

   - Interpolate the reference vector of r from the reference vectors of q and f:

     $$\mathbf{w}_r = (\mathbf{w}_q + \mathbf{w}_f)/2. \qquad (6.17)$$

   - Decrease the error variables of all neighbors of r by a fraction which depends on the number of neighbors of r:

     $$\Delta \mathrm{E}_i = -\frac{\alpha}{|N_r|} \mathrm{E}_i \qquad (\forall i \in N_r). \qquad (6.18)$$

---

[2]Depending on the problem at hand also other local measures are possible, e.g. the *number* of input signals for which a particular unit is the winner or even the positioning error of a robot arm controlled by the network. The local measure should generally be something which one is interested to reduce and which is likely to be reduced by the insertion of new units.

- Set the error variable of the new unit r to the mean value of its neighbors:

$$E_r = \frac{1}{|N_r|} \sum_{i \in N_r} E_i. \tag{6.19}$$

7. Decrease the error variables of all units:

$$\Delta E_c = -\beta\, E_c \qquad (\forall c \in \mathcal{A}). \tag{6.20}$$

8. If a stopping criterion (e.g., net size or some performance measure) is not yet fulfilled continue with step 2.

Figure 6.3 shows some stages of a simulation for a simple ring-shaped data distribution. Figure 6.4 displays the final results after 40000 adaptation steps for three other distribution. The parameters used in both simulations were: $\alpha = 1.0$, $\varepsilon_b = 0.06$, $\varepsilon_n = 0.002$, $\beta = 0.0005$, $\lambda = 200$.

6.3 Growing Grid

Growing grid is another incremental network. The basic principles used also in *growing cell structures* and *growing neural gas* are applied with some modifications to a rectangular grid. Alternatively, *growing grid* can be seen as an incremental variant of the *self-organizing feature map*.

The model has two distinct phases, a *growth phase* and a *fine-tuning phase*. During the growth phase a rectangular network is built up starting from a minimal size by inserting complete rows and columns until the desired size is reached or until a performance criterion is met. Only constant parameters are used in this phase. In the fine-tuning phase the size of the network is not changed anymore and a decaying learning rate is used to find good final values for the reference vectors.

As for the self-organizing map, the network structure is a two-dimensional grid (a_{ij}). This grid is initially set to 2×2 structure. Again, the distance on the grid is used to determine how strongly a unit $r = a_{km}$ is adapted when the unit $s = a_{ij}$ is the winner. The distance measure used is the L_1-norm

$$d_1(r,s) = |i - k| + |j - m| \qquad \text{for } r = a_{km} \text{ and } s = a_{ij}. \tag{6.21}$$

Also the function used to determine the adaptation strength for a unit r given that s is the winner is the same as for the *self-organizing feature map*:

$$h_{rs} = \exp\left(\frac{-d_1(r,s)^2}{2\sigma^2}\right). \tag{6.22}$$

The width parameter σ, however, remains constant throughout the whole simulation. It is chosen relatively small compared to the values usually used at the beginning for the *self-organizing feature map*. One can note that as the *growing grid* network grows, the *fraction* of all units which is adapted together with the winner decreases. This is also the case in the *self-organizing feature map* but is achieved there with a constant network size and a decreasing neighborhood width. The complete *growing grid* algorithm is the following:

Growth Phase

1. Set the initial network width and height:

$$N_1 = 2, \quad N_2 = 2. \tag{6.23}$$

a) 0 signals   b) 100 signals   c) 300 signals   d) 1000 signals

e) 2500 signals   f) 10000 signals   g) 40000 signals   h) Voronoi regions
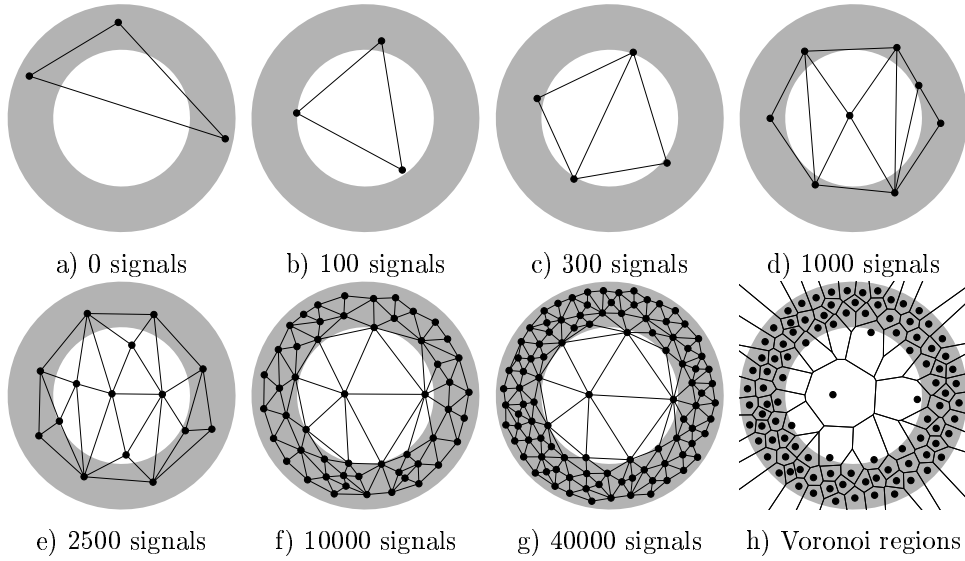
Figure 6.3: *Growing cell structures* simulation sequence for a ring-shaped uniform probability distribution. a) Initial state. b-f) Intermediate states. g) Final state. h) Voronoi tessellation corresponding to the final state. Per construction the network structure always consists of hypertetrahedrons (triangles in this case).
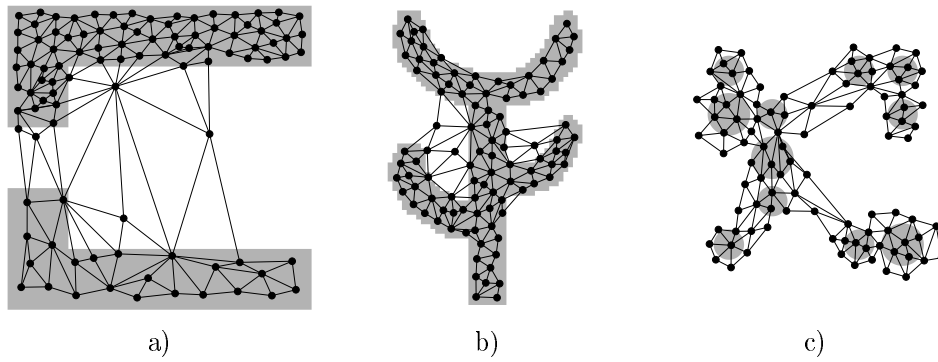

a)   b)   c)

Figure 6.4: *Growing cell structures* simulation results after 40000 input signals for three different probability distributions (described in the caption of figure 4.4).

Initialize the set $\mathcal{A}$ to contain $N = N_1 \cdot N_2$ units c_i

$$\mathcal{A} = \{c_1, c_2, \ldots, c_N\} \tag{6.24}$$

with reference vectors $\mathbf{w}_{c_i} \in \mathrm{R}^n$ chosen randomly according to $p(\boldsymbol{\xi})$.

Initialize the connection set $\mathcal{C}$ to form a rectangular $N_1 \times N_2$ grid.

Initialize the time parameter t:

$$t = 0. \tag{6.25}$$

2. Generate at random an input signal $\boldsymbol{\xi}$ according to $p(\boldsymbol{\xi})$.

3. Determine the winner $s(\boldsymbol{\xi}) = s$:

$$s(\boldsymbol{\xi}) = \arg\min_{c \in \mathcal{A}} \|\boldsymbol{\xi} - \mathbf{w}_c\|. \tag{6.26}$$

4. Increase a local counter variable of the winner:

$$\tau_s = \tau_s + 1. \tag{6.27}$$

5. Increase the time parameter t:

$$t = t + 1. \tag{6.28}$$

6. Adapt each unit r according to

$$\Delta\mathbf{w}_r = \epsilon(t)\, h_{rs}(\boldsymbol{\xi} - \mathbf{w}_r) \tag{6.29}$$

whereby

$$\epsilon(t) = \epsilon_0. \tag{6.30}$$

7. If the number of input signals generated for the current network size reaches a multiple λ_g of this network size, i.e., if

$$\lambda_g \cdot N_1 \cdot N_2 = t \tag{6.31}$$

then do the following:

- Determine the unit q with the largest value of τ:

$$q = \arg\max_{c \in \mathcal{A}} \tau_c. \tag{6.32}$$

- Determine the direct neighbor f of q with the most distant reference vector:

$$f = \arg\max_{c \in N_q} \|\mathbf{w}_q - \mathbf{w}_c\|. \tag{6.33}$$

- Depending on the relative position of q and f continue with one of the two following cases:

  Case 1: q and f are in the same *row* of the grid, i.e.

$$q = a_{i,j} \text{ and } (f = a_{i,j+1} \text{ or } f = a_{i,j-1}). \tag{6.34}$$

  Do the following:
  - Insert a new column with N_1 units between the columns of q and f.
  - Interpolate the reference vectors of the new units from the reference vectors of their respective direct neigbors in the same row.

– Adjust the variable for the number of columns:

$$N_2 = N_2 + 1. \tag{6.35}$$

Case 2: q and f are in the same *column* of the grid, i.e.

$$q = a_{i,j} \text{ and } (f = a_{i+1,j} \text{ or } f = a_{i-1,j}). \tag{6.36}$$

Do the following:
– Insert a new row with N_2 units between the rows of q and f.
– Interpolate the reference vectors of the new units from the reference vectors of their respective direct neigbors in the same columns.
– Adjust the variable for the number of rows:

$$N_1 = N_1 + 1. \tag{6.37}$$

• reset all local counter values:

$$\tau_c = 0 \quad (\forall c \in \mathcal{A}). \tag{6.38}$$

• reset the time parameter:
$$t = 0. \tag{6.39}$$

8. If the desired network size is not yet achieved, i.e. if

$$N_1 \cdot N_2 < N_{\min}, \tag{6.40}$$

then continue with step 2.

Fine-tuning Phase

9. Generate at random an input signal $\boldsymbol{\xi}$ according to $p(\boldsymbol{\xi})$.

10. Determine the winner $s(\boldsymbol{\xi}) = s$:

$$s(\boldsymbol{\xi}) = \arg \min_{c \in \mathcal{A}} \|\boldsymbol{\xi} - \mathbf{w}_c\|. \tag{6.41}$$

11. Adapt each unit r according to

$$\Delta \mathbf{w}_r = \epsilon(t) \, h_{rs} (\boldsymbol{\xi} - \mathbf{w}_r) \tag{6.42}$$

whereby
$$\epsilon(t) = \epsilon_0 (\epsilon_1 / \epsilon_0)^{t/t_{\max}} \tag{6.43}$$

with
$$t_{\max} = N_1 \cdot N_2 \cdot \lambda_f. \tag{6.44}$$

12. If $t < t_{\max}$ continue with step 9.

Figure 6.5 shows some stages of a simulation for a simple ring-shaped data distribution. Figure 6.6 displays the final results after 40000 adaptation steps for three other distribution. The parameters used for the growth phase were: $\lambda_g = 30$, $\sigma = 0.7, \epsilon_0 = 0.005$. The parameters for the fine-tuning phase were: σ and ϵ_0 unchanged, $\epsilon_1 = 0.005$, $\lambda_f = 100$, $N_{\min} = 100$.

If one compares the *growing grid* algorithm with the other incremental methods *growing cell structures* and *growing neural gas* then a difference (apart from the topology) is that no counter variables are redistributed when new units are inserted. Instead, all τ-values are set to zero after a row or column has been inserted. This
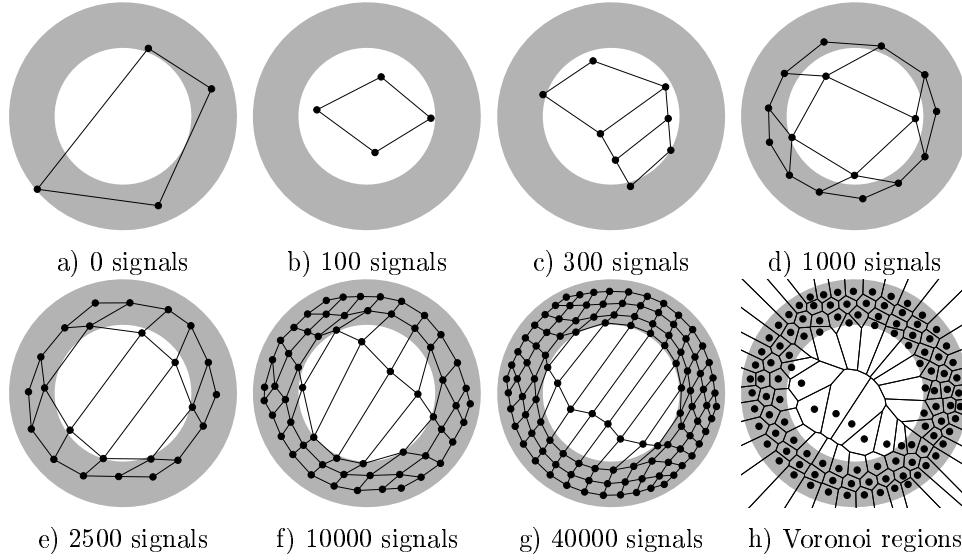
Figure 6.5: *Growing grid* simulation sequence for a ring-shaped uniform probability distribution. a) Initial state. b-f) Intermediate states. g) Final state. h) Voronoi tessellation corresponding to the final state.
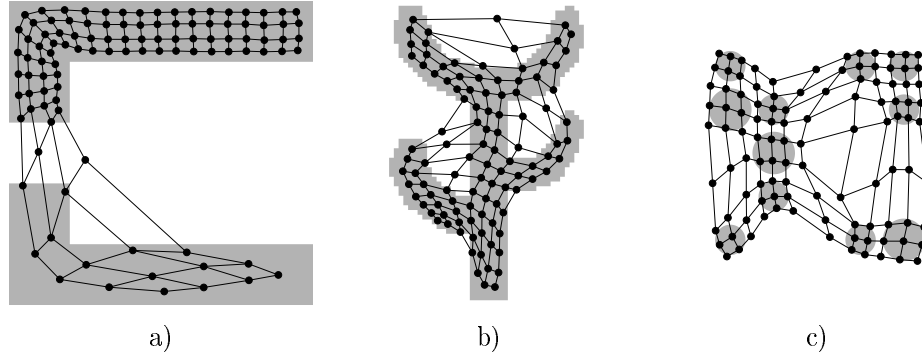

Figure 6.6: *Growing grid* simulation results after 40000 input signals for three different probability distributions (described in the caption of figure 4.4). One can note that in a) the chosen topology (4×26) has a rather extreme height/width ratio which matches well the distribution at hand. Depending on initial conditions however, also other topologies occur in simulations for this distribution. b),c) Also these topologies deviate from the square shape usually given to self-organizing maps. For the cactus a (7×15) and for the mixture distribution a (9×12) topology was automatically selected by the algorithm.

means, that all statistical information about winning frequencies is discarded after an insertion. Therefore, to gather enough statistical evidence where to insert new units the next time, the number of adaptation steps *per insertion step* must be proportional to the network size (see equation 6.31). This simplifies the algorithm but increases the computational complexity. The same could in principle be done with *growing neural gas* and *growing cell structures* effectively eliminating the need to re-distribute accumulated information after insertions at the price of increased computational complexity.

The parameter σ which governs the neighborhood range has the function of a regularizer. If it is set to large values, then neighboring units are forced to have rather similar reference vectors and the layout of the network (when projected to input space) will appear very regular but not so well adapted to the underlying data distribution $p(\boldsymbol{\xi})$. Smaller values for σ give the units more possibilities to adapt independently from each other. As σ is set more and more to zero the *growing grid* algorithm (apart from the insertions) approaches hard competitive learning.

Similar to the *self-organizing feature map* the *growing grid* algorithm can easily be applied to network structures of other dimensions than two. Actually useful, however, seem only the cases of one- and three-dimensional networks since networks of higher dimensionality can not be visualized easily.

6.4   Other Methods

A number of other methods with a fixed dimensionality exist. Bauer and Villmann (1995) proposed a method which develops a hypercubical grid. In contrast to the *growing grid* method their algorithm automatically determines a suitable dimensionality for the grid.

Blackmore and Miikkulainen (1992) let a irregular network grow on positions in the plane which are restricted to lie on a two-dimensional grid. Rodrigues and Almeida (1990) increased the speed of the normal *self-organizing feature map* by developing an interpolation method which symmetrically increases the number of units in the network by interpolation. Their method is reported to give a considerable speed-up but is not able to choose, e.g., different dimensions for width and height of the grid as the approach of Bauer and Villmann (1995) or the *growing grid*. Further approaches have been proposed, e.g. by Jokusch (1990) and Xu (1990).

Chapter 7

Quantitative Results (t.b.d.)

Chapter 8

Discussion (t.b.d.)

Bibliography

H.-U. Bauer and K. Pawelzik. Quantifying the neighborhood preservation of self-organizing feature maps. *IEEE Transactions on Neural Networks*, 3(4):570–579, 1992.

H.-U. Bauer and T. Villmann. Growing a hypercubical output space in a self-organizing feature map. Tr-95-030, International Computer Science Institute, Berkeley, 1995.

J. Blackmore and R. Miikkulainen. Incremental grid growing: encoding high-dimensional structure into a two-dimensional feature map. TR AI92-192, University of Texas at Austin, Austin, TX, 1992.

C. Darken and J. Moody. Fast adaptive k-means clustering: Some empirical results. In *Proc. IJCNN*, volume II, pages 233–238. IEEE Neural Networks Council, 1990.

D. DeSieno. Adding a conscience to competitive learning. In *IEEE International Conference on Neural Networks*, volume 1, pages 117–124, New York, 1988. (San Diego 1988) IEEE.

E. Forgy. Cluster analysis of multivariate data: efficiency vs. interpretanility of classifications. *Biometrics*, 21:768, 1965. abstract.

B. Fritzke. Growing cell structures – a self-organizing network for unsupervised and supervised learning. *Neural Networks*, 7(9):1441–1460, 1994a.

B. Fritzke. Fast learning with incremental RBF networks. *Neural Processing Letters*, 1(1):2–5, 1994b.

B. Fritzke. A growing neural gas network learns topologies. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems 7*, pages 625–632. MIT Press, Cambridge MA, 1995a.

B. Fritzke. Incremental learning of local linear mappings. In F. Fogelman and P. Gallinari, editors, *ICANN'95: International Conference on Artificial Neural Networks*, pages 217–222, Paris, France, 1995b. EC2 & Cie.

B. Fritzke. The LBG-U method for vector quantization - an improvement over LBG inspired from neural networks. *Neural Processing Letters*, 5(1), 1997.

R. M. Gray. Vector quantization. *IEEE ASSP Magazine*, 1:4–29, 1984.

R. M. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic Press, 1992.

A. K. Jain and R. C. Dubes. *Algorithms for clustering data*. Prentice Hall, 1988.

S. Jokusch. A neural network which adapts its structure to a given set of patterns. In R. Eckmiller, G. Hartmann, and G. Hauske, editors, *Parallel Processing in Neural Systems and Computers*, pages 169–172. Elsevier Science Publishers B.V., 1990.

J. A. Kangas, T. Kohonen, and T. Laaksonen. Variants of self-organizing maps. *IEEE Transactions on Neural Networks*, 1(1):93–99, 1990.

S. Kirkpatrick, C. D. G. Jr., , and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220, 1983.

T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982.

Y. Linde, A. Buzo, and R. M. Gray. An algorithm for vector quantizer design. *IEEE Transactions on Communication*, COM-28:84–95, 1980.

S. P. Lloyd. Least squares quantization in pcm. technical note, Bell Laboratories, 1957. published in 1982 in *IEEE Transactions on Information Theory*.

J. MacQueen. On convergence of k-means and partitions with minimum average variance. *Ann. Math. Statist.*, 36:1084, 1965. abstract.

J. MacQueen. Some methods for classification and analysis of multivariate observations. volume 1 of *Proceedings of the Fifth Berkeley Symposium on Mathematical statistics and probability*, pages 281–297, Berkeley, 1967. University of California Press.

T. M. Martinetz. Competitive Hebbian learning rule forms perfectly topology preserving maps. In *ICANN'93: International Conference on Artificial Neural Networks*, pages 427–434, Amsterdam, 1993. Springer.

T. M. Martinetz, S. G. Berkovich, and K. J. Schulten. Neural-gas network for vector quantization and its application to time-series prediction. *IEEE Transactions on Neural Networks*, 4(4):558–569, 1993.

T. M. Martinetz, H. J. Ritter, and K. J. Schulten. 3D-neural-network for learning visuomotor-coordination of a robot arm. In *International Joint Conference on Neural Networks*, pages II.351–356, Washington DC, 1989.

T. M. Martinetz and K. J. Schulten. A "neural-gas" network learns topologies. In T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas, editors, *Artificial Neural Networks*, pages 397–402. North-Holland, Amsterdam, 1991.

T. M. Martinetz and K. J. Schulten. Topology representing networks. *Neural Networks*, 7(3):507–522, 1994.

J. E. Moody and C. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1:281–294, 1989.

S. M. Omohundro. The Delaunay triangulation and function learning. Tr-90-001, International Computer Science Institute, Berkeley, 1990.

F. P. Preparata and M. I. Shamos. *Computational geometry*. Springer, New York, 1990.

H. J. Ritter, T. M. Martinetz, and K. J. Schulten. *Neuronale Netze*. Addison-Wesley, München, 1991.

J. S. Rodrigues and L. B. Almeida. Improving the learning speed in topological maps of patterns. In *Proceedings of INNC*, pages 813–816, Paris, 1990.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In R. D. E. and J. L. McClelland, editors, *Parallel Distributed Processing*, volume 1, pages 318–362. MIT Press, Cambridge, 1986.

T. Villmann, R. Der, M. Herrmann, and T. Martinetz. Topology presevation in self-organizing feature maps: exact definition and measurement. *IEEE TNN*, 1994. submitted.

J. Walter, H. J. Ritter, and K. J. Schulten. Non-linear prediction with self-organizing maps. In *International Joint Conference on Neural Networks*, pages I.589–594, San Diego, 1990.

D. J. Willshaw and C. von der Malsburg. How patterned neural connections can be set up by self-organization. In *Proceedings of the Royal Society London*, volume B194, pages 431–445, 1976.

L. Xu. Adding learning expectation into the learning procedure of self-organizing maps. *Int. Journal of Neural Systems*, 1(3):269–283, 1990.