An Adaptive Classification Scheme to Approximate Decision Boundaries Using Local Bayes Criteria - The "Melting Octree" Network

L. M. Encarnação, M. H. Gross
International Computer Science Institute (ICSI), Berkeley
and
Computer Graphics Center (ZGDV)
D-6100 Darmstadt, Germany
Tel.: +6151-155-232
Fax: +6151-155-299
Email: miguel@igd.fhg.de, gross@igd.fhg.de

Abstract: The following paper describes a new method to approximate the minimum error decision boundary for any supervised classification problem by means of a linear neural network consisting of simple neurons that use a local Bayes criterium and a next neighbor decision rule. The neurons can be interpreted as centroids in feature space or as a set of particles moving towards the classification boundary during training. In contrary to existing LVQ methods and RCE networks each neuron has a receptive field of an adjustable width ε and the goal of the supervised training method is completely different. Furthermore, the network is able to grow in the sense of generating new entities in order to decrease the classification error after learning.
For this purpose we initialize the network via a multidimensional octree representation of the training data set. The neurons generated during initialization only depend on the maximum number of data in a single octree cell. The learning method introduced ensures that all neurons move towards the class boundaries by checking the local Bayes criterium in their receptive field. For this process can also be interpreted as a melting away of the initial octree, we called the network "*The Melting Octree*" network.
This report first describes the algorithms used for initialization, training as well as for growing of the net. The classification performance of the algorithm is then illustrated by some examples and compared with those of a Kohonen feature Map (LVQ) and of a backpropagated multilayered perceptron.

1. Introduction

Classification and decision making belong to the most important problems of nearly any field of science /6/, /9/, /10/, /11/, /12/. Lots of methods had been developed in the past, reaching from classical statistical algorithms such as regression and maximum likelihood estimation /8/ to connectionist classifiers such as backpropagation or learning vector quantization (LVQ) /1/, /5/, /7/. According to the Bayesian decision theory, the global goal has to be to find the decision boundary, that minimizes the probability of false classification for any given classification problem /14/.

There are several methods that aim at approximating these theoretically postulated boundaries in feature space, as i.e. parametric likelihood or polynomial regression techniques as well as Restricted Coulomb Energy Networks. Furthermore, iterated gradient descent coefficient updating, as used in backpropagation networks or in Kohonen’s feature map, is also capable to approximate decision boundaries. Where multilayer perceptrons are confronted with local minima and with the problem of selecting an appropriate topology, LVQ methods, that are based on an a-priori clustering, also have only a fixed set of centroids for the next neighbor classification. In both cases, the minimum error to be reached depends strongly on the number of neurons /4/.

Besides these aspects, there have been many theoretical investigations aiming at describing the internal representation /2/, /3/ of the data in these neural networks.

Inspired by Kohonen’s LVQ classifiers and by the idea of having an easy to parallelize scheme, the goal of the research reported below was to develop a supervised trained and self-optimizing neural network, that approaches the Bayesian decision boundaries for arbitrary classification problems with simple next neighbor units. In contrast to the LVQ we use a different training scheme where the neurons are associated with receptive fields of adjustable width. These neurons are moving towards the class boundaries by checking the Bayes condition in their receptive field and stabilize themselves there. Furthermore, the network is generating neurons after training and grows in order to decrease the classification error. For the initialization of the weight vectors all neurons are precalculated by an octree tessellation. For this reason we named our network "*The Melting Octree*".

The basic idea is to train and grow a linear network with gradient descent techniques in a way that

I. the weight vector of each neuron reaches a position in feature space close to the class boundaries and thus approximates the Bayes border in its local environment.

II. the network optimizes itself by bearing child neurons according to the fractional error of each neuron after classification.

To perform this, the neurons have associated receptive fields in which they pick up data samples from a given training set and compare these with the Bayes criterium for the class boundary.

Basically, our method consists of 4 steps. These are:

A. Initialization
B. Organization and Learning
C. Growing and Optimization
D. Classification

The following description of the algorithms often uses the 2-dimensional classification problem as an example, i.e. the feature vector **x** = (x_1, x_2), but all rules can be applied to N dimensions.

The outline of this report is as follows: First, we describe briefly the basics of Bayes decision making as well as Kohonen's work we relate on. The next chapter is dedicated to network topology and the initialization by octree clustering. Then, we describe the gradient descent rules for updating the neuron weights during the learning phase and explain the growing algorithm to adapt the network automatically to a certain error quality to be reached. The last chapters document detailed results applying this method on classification problems and compare the performance with those of backpropagation and LVQ methods. Finally we give an outlook on a very difficult classification problem: The automatic extraction of brain tumors from CT and MRT data sets.

2. Basics and Pre-assumptions

2.1 General Remarks

The following chapter introduces briefly the basics of decision theory we relate on and the Kohonen LVQ algorithms, that inspired our work. In particular the Bayes theory is not explained here in detail. The interested reader may find it anywhere in the literature /8/, /14/.

2.2 Bayesian decision theory

Let be given any classification problem, where we have to distinguish M classes from a data set {X}. Furthermore, a limited training data set $\{X\}_T$ is given to learn a supervised classifier, where $\{X\}_T$ consists of subsets of all classes C_k (k = 1..M) to be distinguished:

$$\{X\}_T = \{\{X\}_{T_1}, \{X\}_{T_2}, \ldots, \{X\}_{T_M}\} \qquad (1)$$

and

$$\{X\}_{T_k} \to C_k$$

Bayesian decision theory minimizes the error probability P_E for any classification problem as in the following one-dimensional example of figure 1.

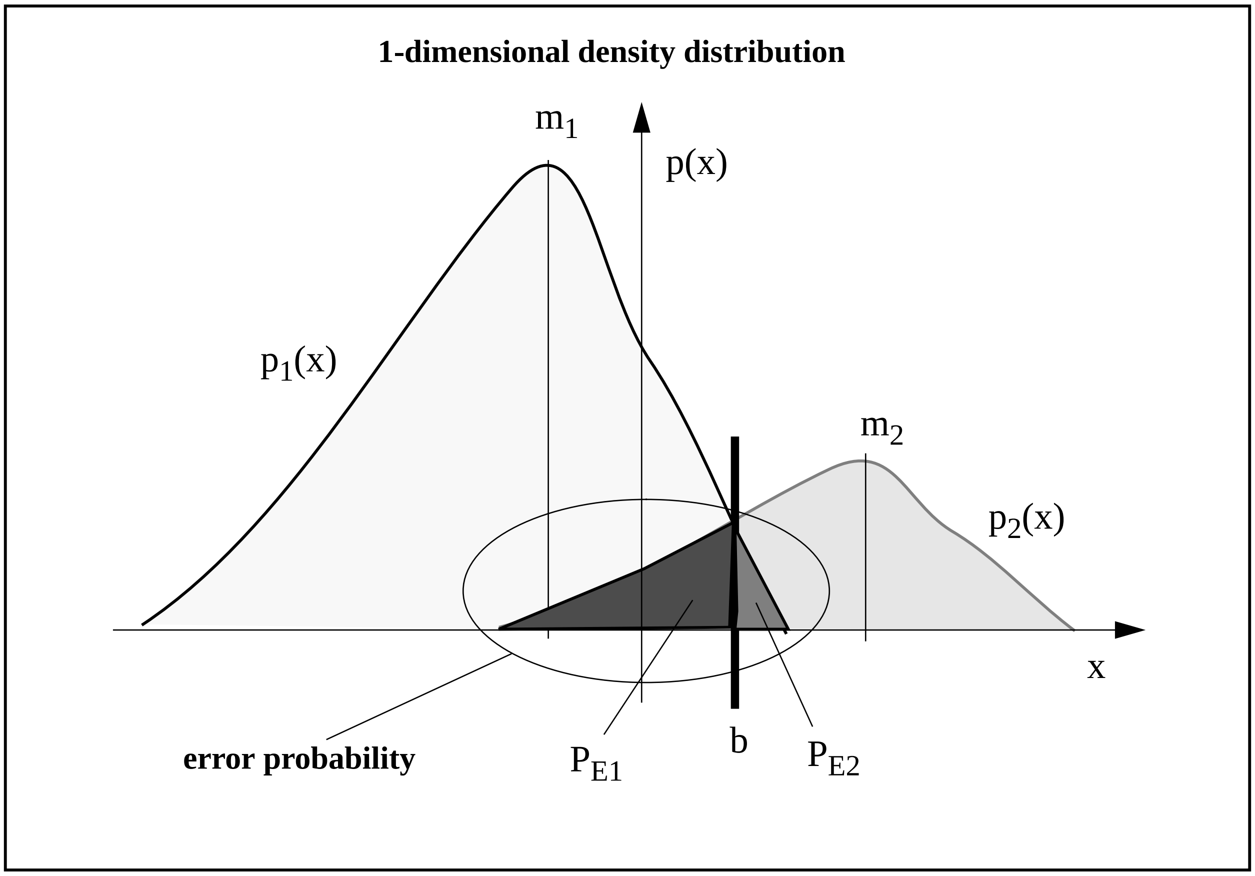


Figure 1: Minimum error decision boundary value b in the one-dimensional case.

With

$$P_{E_2} = prob\{x \in C_1 | x \rightarrow C_2\} = \int_b^\infty p_1(x)dx \qquad (2)$$

and similar

$$P_{E_1} = \int_{-\infty}^b p_2(x)dx \qquad (3)$$

we get

$$P_E = P_{E_1} + P_{E_2}$$

where $p_1(x)$ and $p_2(x)$ are the a-priori probability density functions for C_1 and C_2.

In order to minimize P_E , b has to follow the rule:

$$|p_1(x_b) - p_2(x_b)| = 0 \quad ! \tag{4}$$

That means, in general the difference of the a-priori density function of neighboring classes C_1 and C_2 to be distinguished from each other drops to zero at the so-called Bayesian border.

2.3 Kohonen’s LVQ

The Learning Vector Quantization (LVQ) method proposed by Kohonen /5/ can be considered as a postprocessing on an initial vector quantization of an arbitrary feature space. The basic idea is to move the cluster centroids close to the decision boundary in locally optimal positions for further classification. This is done via an iterated gradient descent scheme on training data sets. The updating of the coordinates of each centroid $\mathbf{m}_i$ obtained by any C-means methods is done as follows:

$$\vec{m}_i(t+1) = \vec{m}_i(t) - \alpha(t)\,[\vec{x}(t) - \vec{m}_i(t)\,] \tag{5}$$

$$\vec{m}_s(t+1) = \vec{m}_s(t) + \alpha(t)\,[\vec{x}(t) - \vec{m}_s(t)\,]$$

where $\mathbf{m}_i$ and $\mathbf{m}_s$ are the two closest centroids to **x**, and **x** and $\mathbf{m}_i$ belong to the same class, while **x** and $\mathbf{m}_s$ belong to different classes. Furthermore, $\mathbf{x} \in \{X\}_T$ and **x** falls into a window of size ω.

$$\vec{m}_{i,s}(t+1) = \vec{m}_{i,s}(t) + \sigma\alpha(t)\,[\vec{x}(t) - \vec{m}_{i,s}(t)\,] \tag{6}$$

if **x**, $\mathbf{m}_i$ and $\mathbf{m}_s$ belong to the same class. σ is a small positive constant.

$$\alpha(t) = \alpha_0\,(1 - \frac{t}{T}) \tag{7}$$

is the time dependant learning rate.

This is also illustrated in figure 2, where the general direction of the movement of the cluster positions is shown for a one-dimensional example. The positions of the centroids that are close to the actual decision boundary are shifted in a way to approach the Bayes border of the training data set.

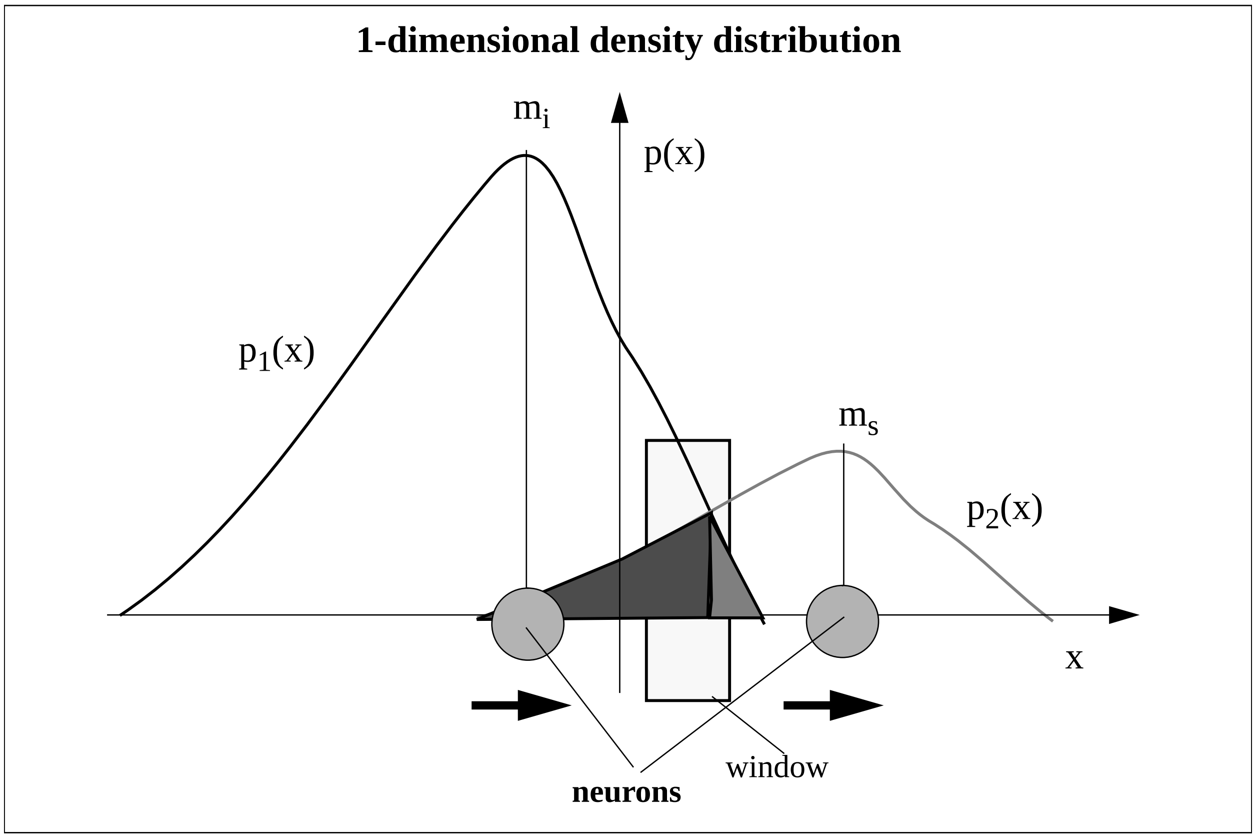


Figure 2: Illustration of the influence of the LVQ training on cluster centroids in feature space.

The disadvantages of the method explained above are:

I. Only a few cluster positions are influenced by the training. Most of them represent the class centers and those regions, where the a-priori probability density function is high. This is appropriate for non-supervised cluster analysis. For supervised classification problems, however, the goal has to be to separate the classes from each other and thus, to be sensitive at the decision boundaries. The representation of the class density function is no longer of interest.

II. This method is strongly restricted by the number of clusters C to be defined in advance. Only a fraction of this value will be taken to approximate the optimal decision boundary. There is no way to adapt automatically the number of cluster centroids to the shape of the boundary or to a certain minimum error to be reached.

Recognizing these shortcomings, the general idea was to develop a scheme being able to modify the position of each centroid towards the class boundary. Furthermore, the method should be able to grow after training, i. e. to generate new centroids close to the boundary in order to approximate its shape more precisely. Finally the classification should be performed via next neighbor decision on all neurons created during this optimization process.

The following chapter describes the algorithm, we developed, illustrates the promising results obtained and compares its classification performance with the standard LVQ algorithm, as well as with the backpropagation rule for multilayer perceptrons.

3. Network Topology and Initialization

3.1 General Remarks

This chapter introduces the network topology we use, where the octree clustering is emphasized that initializes the network. Furthermore, the neuron type and its receptive field are explained. In our approach the neurons are not connected to each other. Moreover, they can also be considered as particles in feature space that are checking the data in their local environment and moving according to the result of this calculation.

3.2 A linear network based on next neighbor units and isotrope receptive fields in feature space

Basically, we introduce the following adaptive scheme to approximate the Bayes borders using a simple neuron model, shown in figure 3. For we don’t know anything about the underlying density distribution or statistical properties of the data, we have to initialize the network by any C-means clustering technique. After this initialization of the network, each neuron represents a cluster centroid of the training data set. Thus the connection weight vector $\mathbf{m}_i = (m_1,..,m_N)$ of a neuron i (i = 1..C) holds the coordinates of the centroid associated to the neuron. C is the total number of neurons after initialization and N is the number of dimensions.

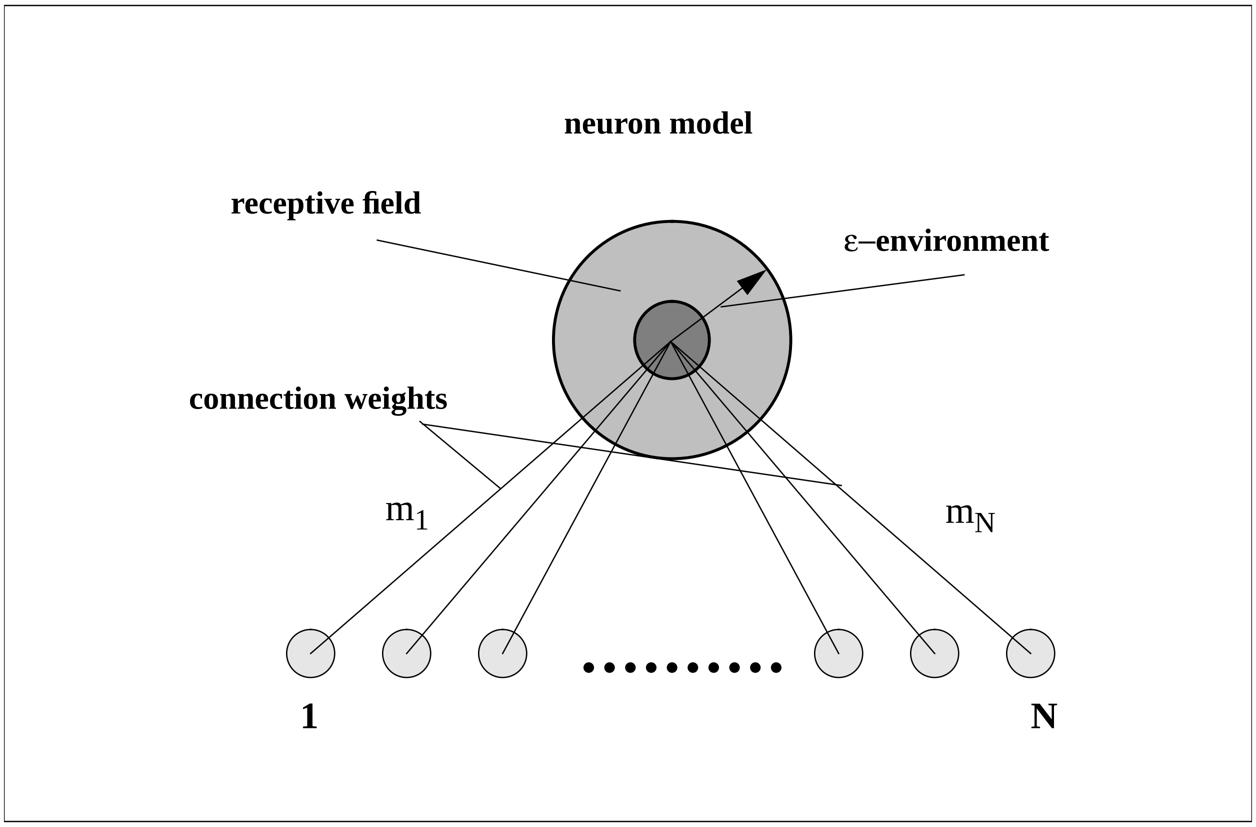


Figure 3: Simple neuron model used for the melting octree network (2-D example)

A further property of the neuron model used is its receptive field of an adjustable width ε. Any decision or training rule applied onto the neuron is based on the data in this receptive field that can be supposed as the local environment of the neuron. For a multidimensional case the receptive field becomes a hypersphere around the neuron centroid in feature space. More generally, ε can be seen as a vector describing a hyperellipsoid, $\varepsilon = (\varepsilon_1,...,\varepsilon_N)$.

The subset $\{X\}_{T,\varepsilon i}$ of the training data $\{X\}_T$ in the receptive field of the neuron $\mathbf{m}_i$ holds the following inequality:

$$\{X\}_{T\varepsilon_i} = \{\vec{x}_l | (d_{il} < \varepsilon_i), l|_1^L\} \quad (8)$$

and

$$d_{il} = \|\vec{m}_i - \vec{x}_l\| \quad (9)$$

$$L = \text{number of elements in } \{X\}_T$$

is the Euclidean distance.

For any further discussion our feature space will be considered as being normalized in the range [0..1].

Taking these types of neurons as basic units, we are able to define a linear network consisting of C neurons as shown in figure 4.

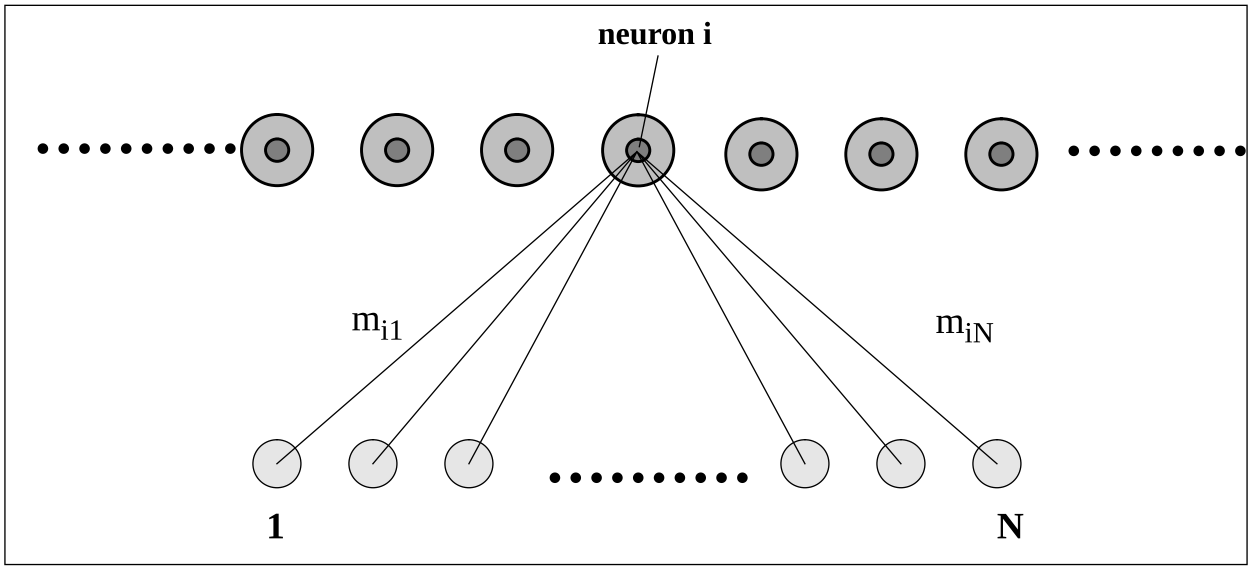


Figure 4: Linear network consisting of C neurons

In the network defined above, the neurons are not connected to each other. Moreover, they interact separately as long as their receptive fields do not overlap. According to this, the network could also be considered as a set of particles moving in feature space during the training.

3.3 Initialization by multidimensional octree tessellation and vector quantization of feature space

Usually we don’t know anything about the data distribution of the classification problem to be solved. For this reason, the initial position and the number of the neurons after the initialization have to represent the probability density function of the data p(**x**). This can be performed by any C-means method.

We propose a multidimensional octree tessellation of the training data set $\{X\}_T$ as it is indicated in figure 5 for N=2. The global criterion of clustering the data is given for each octree cell i with

$$\frac{P_i(\vec{x})}{\Delta V_i} \le \delta_{max} \qquad (10)$$

where ΔV_i is the volume element enclosed by the octree cell i, $P_i(\mathbf{x})$ is the a-priori probability of **x** in octree cell i and δ_{max} is a threshold for subdividing the cell into 2^N sons.

This induces that the amount of training data samples in each octree cell is less than an upper limit.

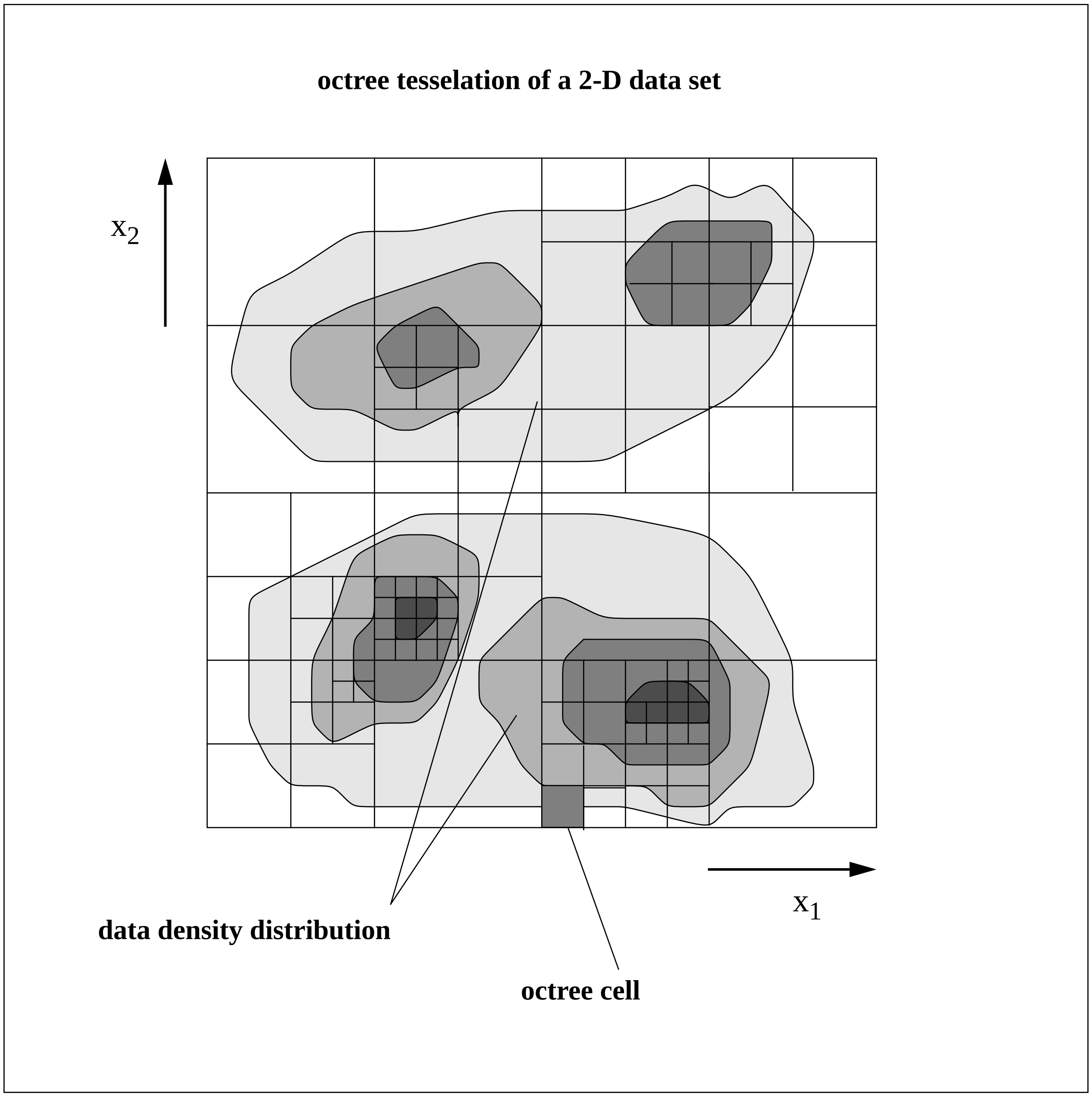

octree tesselation of a 2-D data set
x_2
x_1
data density distribution
octree cell

Figure 5 (a): Octree tessellation of the training data set and associated neuron centroids for N=2.

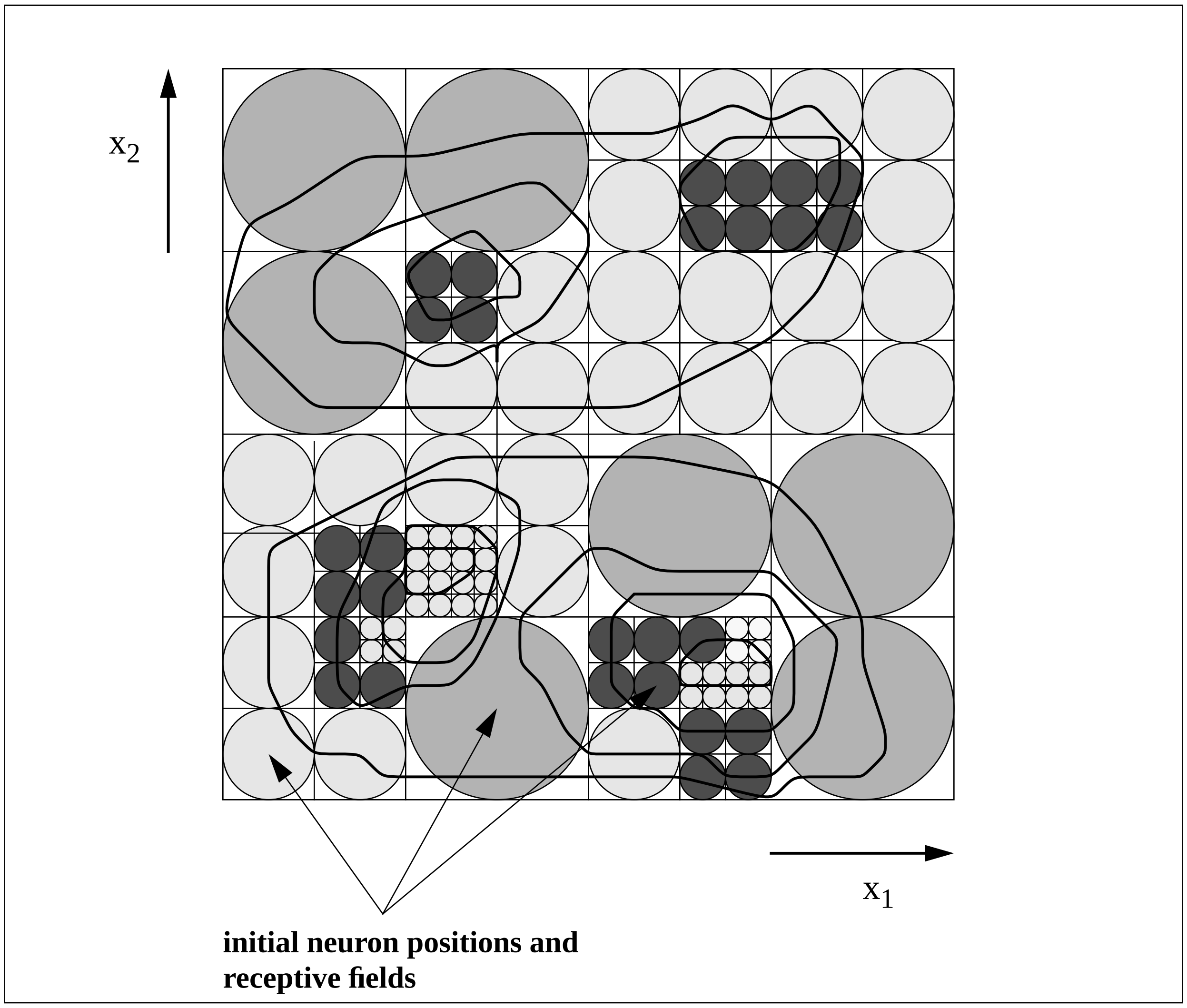


Figure 5 (b): Initial placement of the neurons.

The rule for generating the octree can be written in pseudcode as:

```
.
.
.
init_octree ();
create_first_cell ();
while (!eof) {
  read (x);
  actual_cell = root_of_octree;

  while (child (actual_cell) )                    /* search the octree branch for x  */
    actual_cell = son_cell (x, actual_cell);

  while ( delta (actual_cell) >= DELTA_MAX ){     /* divide this branch into sons if
    split_cell_into_sons (actual_cell);              data limit is reached          */
    transfer_data_into_sons (actual_cell);        /* transfer all data to the new son */
    actual_cell = son_cell (x, actual_cell);
    }

  if ( delta (actual_cell) < DELTA_MAX )
    write_x_into_actual_cell;

  }
.
.
.
```

After defining the octree, one neuron $\mathbf{m}_i$ is set up in the center of each octree cell i (figure 5 (b)). The initial weight vector of the neuron is given by the center coordinate $\mathbf{x}_{\text{cell } i}$ of the respective octree cell.

$$m_{ij} = x_{cell}(i,j) = b_{min(i,j)} + (b_{max(i,j)} - b_{min(i,j)})/2 \qquad (11)$$

$$j|_1^N \quad ; \quad i|_1^C$$

The initial width ε of the receptive field of this neuron is also given by the boundaries of the octree cell. If the boundaries of a certain cell i are given in a N-dimensional case with $b_{\min(i,j)}$ and $b_{\max(i,j)}$, then we can easily calculate $\varepsilon = (\varepsilon_1,...,\varepsilon_N)$ with

$$\varepsilon_{ij} = (b_{max(i,j)} - b_{min(i,j)})/2 \qquad (12)$$

In our scheme, we assume isotrope octree cells and isotrope receptive fields in all dimensions.

After this step, each neuron centroid **m**$_i$, its position and its associated ε_i are initialized.
For further learning and optimization of the network, we have to assign the class number C_k (k = 1..M) to each neuron. This can be done by majority voting on the training data set $\{X\}_T$ based on a next neighbor decision:

$$\{\vec{m}_i \rightarrow C_c | sum_c = max \ , \forall (\vec{x} \in \ \{X\})\} \tag{13}$$

and

$$sum_c \ = \ max\{sum_j \{\vec{x}_l | \ (\vec{x}_l \rightarrow C_j \wedge d_{il} = \ min|_i)\} \big|_j\} \tag{14}$$

$$l|_1^L \ ; \ L = \text{number of data in } \{X\}_T$$

based on the Euclidean distance d_{il} of eq. (9).

The octree representation has also advantages in data and neuron management during the different phases of the network. For instance, in order to find all the training data in the ε-environment of a neuron (in its receptive field) it is sufficient to check the data of all neighboring octree cells. We don’t have always to search through the whole training data set which strongly reduces the calculation times. This is shown in figure 6.

.

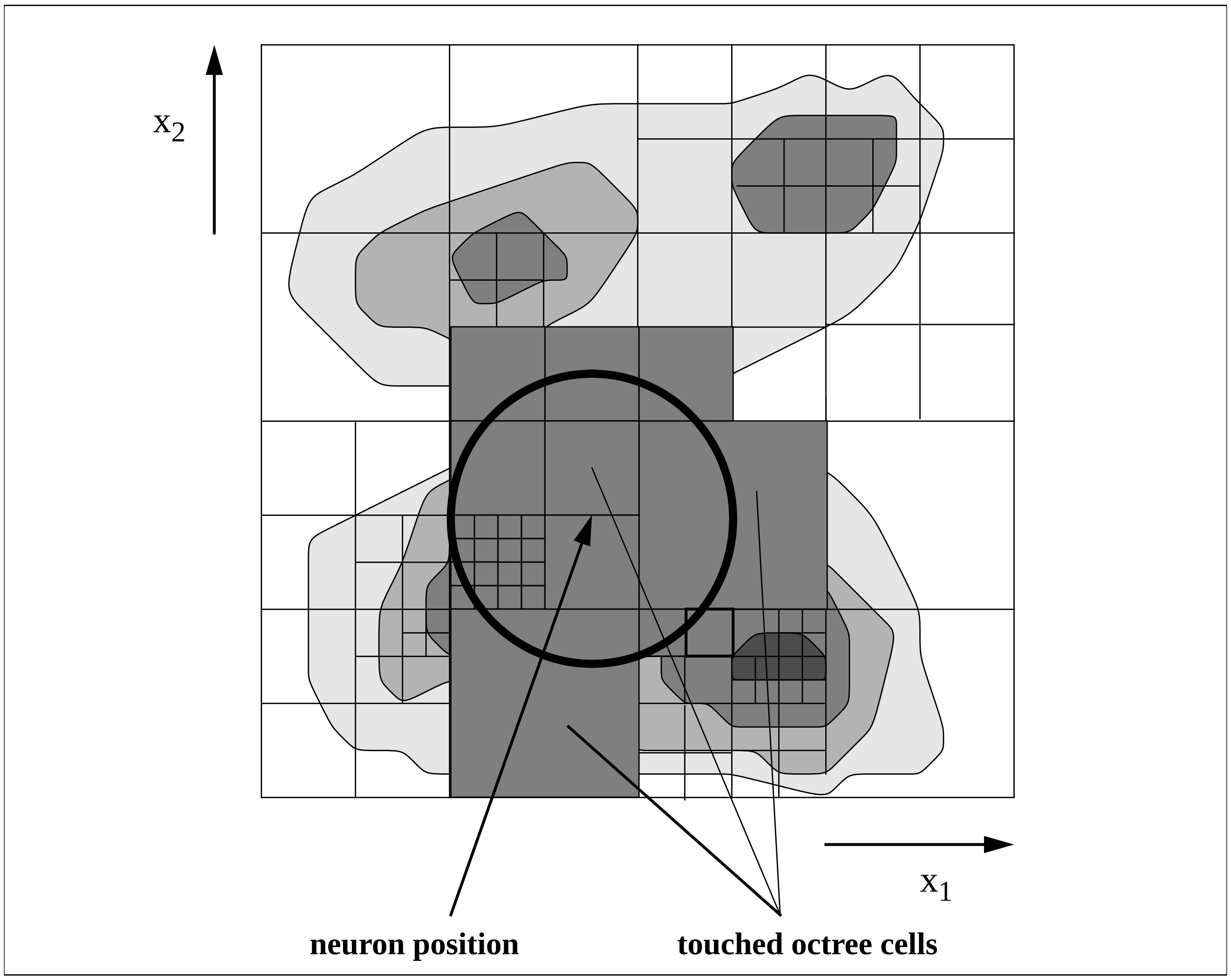


Figure 6: Checking data in the receptive field of an arbitrary neuron position. Only the data sets of the indicated cells have to be checked.

4. Organization and Learning of the network

4.1 General Remarks

After initializing the network as described above, we have now to set up a method for updating the weight vector $\mathbf{m}_i$ of each neuron in a way, that the neuron moves towards the Bayes class boundary. This can also be interpreted in the sense of a particle system, where the single particles (neurons) move towards local energy minima by iterated gradient descent techniques. The following chapter introduces the basic ideas on it and explains the updating rules in detail. In particular, collision avoidance is a general problem of particle systems. We introduce a simplified model for collision handling.

4.2 The basic concept: Moving the neurons towards the class boundaries

As emphasized above, the general goal has to be to find an updating rule for the weights $\mathbf{m}_i$ of all neurons created by the octree, in order to move them towards the class boundaries. This is illustrated for a 2-D example in figure 7. If we succeed in arranging the neurons close to the optimal class boundary, we could obtain a very sensitive classification scheme even though, we only use simple next neighbor decision rules and some of the neurons may get stuck in local minima.

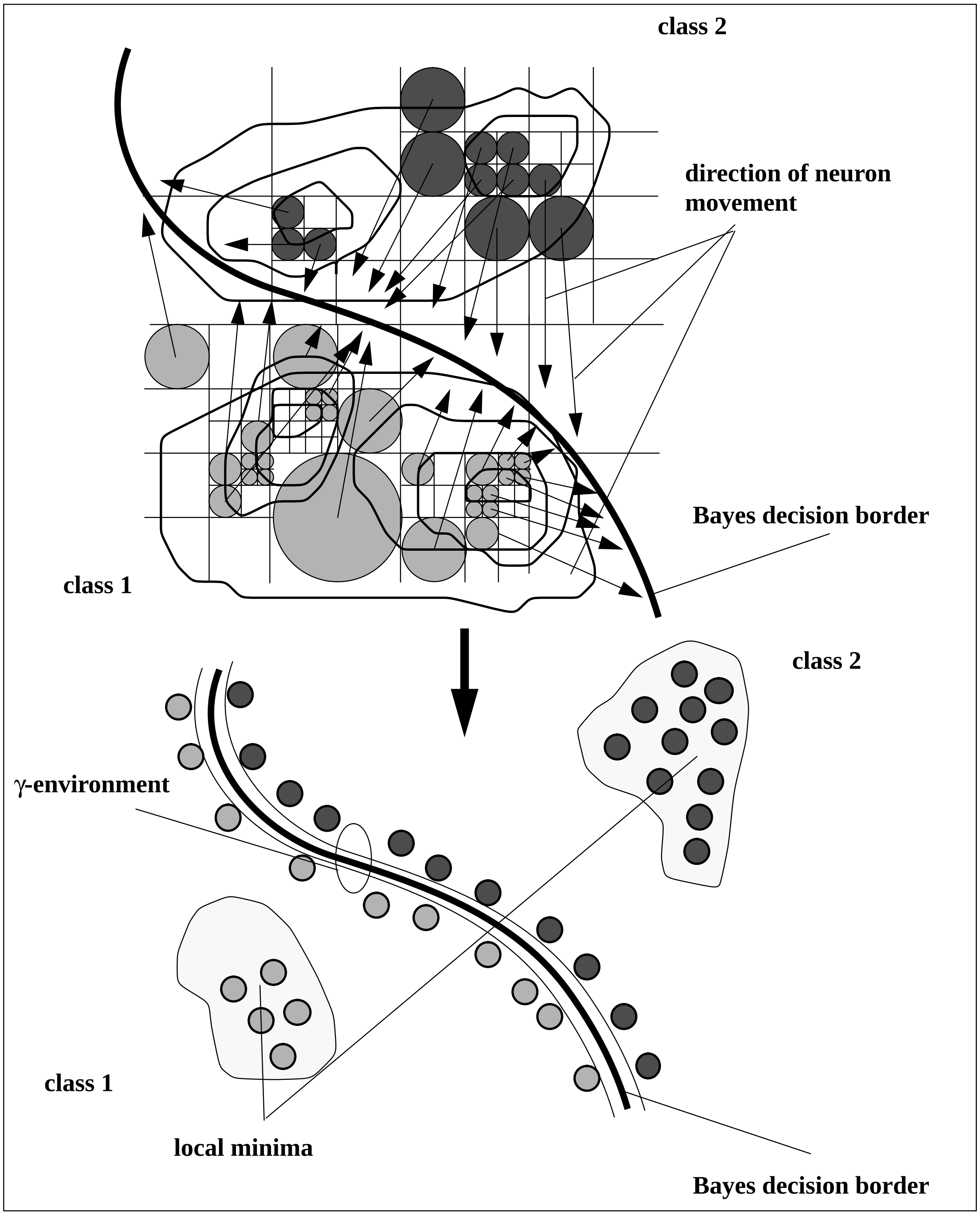


Figure 7: Moving the neurons towards the Bayes decision boundary and stop them in a γ-environment of it. During this movement, some of the neurons might get captured in local minima because of the gradient descent method specified below.

Because this dynamic process can be interpreted as some kind of "melting away" of the initial octree position of the neurons, we named our network the *"The Melting Octree"*.

In order to achieve this behavior, the neurons have to be assigned with local intelligence. The Bayes border may be of arbitrary complex shape for a given classification problem and thus it might be too difficult to set up some global criteria on the data for a single neuron. For this reason, we introduce the receptive field of a neuron as a kind of local environment, in which the Bayes criterium of eq. (4) can be checked.

This means, that the neuron $\mathbf{m}_i$ has to perform four steps on each updating cycle:

I Check the a-priori probabilites $P_{k,i}$ $(\mathbf{x} \in \{X\}_{T,\varepsilon i} \mid \mathbf{x} \rightarrow C_k)$ of the data set $\{X\}_{T,\varepsilon i}$ in its ε-environment.

II Stop close to the border in a sense of a γ-distance.

III Move one step along the gradient of the density function of the data set $\nabla p(\mathbf{x})$.

IV Detect and try to avoid a possible collision with other neurons.

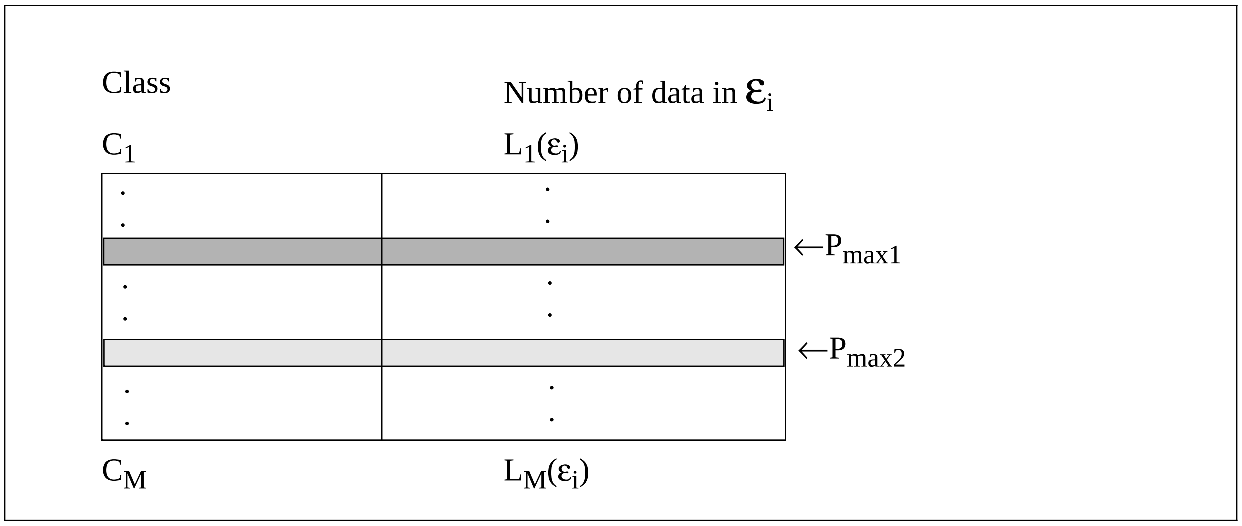
These four steps are now described in detail:

4.3 Checking the a-priori class probabilities and the local Bayes criteria

To check the a-priori class probabilities $P_{k,i}$ $(\mathbf{x} \in \{X\}_{T,\varepsilon i} \mid \mathbf{x} \rightarrow C_k)$ for all classes k of the data set $\{X\}_{T,\varepsilon i}$ in the local ε-environment of a neuron $\mathbf{m}_i$, we propose the following scheme: For any position of the neuron $\mathbf{m}_i$ in feature space, set up a table, that counts the class assignments of C_k for all $\mathbf{x} \in \{X\}_{T,\varepsilon i}$ according to table 1. If $L(\varepsilon_i)$ is the total number of data in $\{X\}_{T,\varepsilon i}$ and $L_k(\varepsilon_i)$ is the number of data belonging to class C_k, then we can use the local estimation of $P_{k,i}$ $(\mathbf{x} \in \{X\}_{T,\varepsilon i} \mid \mathbf{x} \rightarrow C_k)$.

$$P_{k,\,i}\,(\vec{x} \in \{X\}_{T\varepsilon_i}\big|\,(\vec{x} \rightarrow C_k)\,) \approx \frac{L_k(\varepsilon_i)}{L(\varepsilon_i)} \qquad (15)$$

Table 1: Estimation of the local density distributions and of the local Bayes criteria by counting the class assignments of the data in the receptive field of a neuron.



Class	Number of data in ε_i	
C_1	$L_1(\varepsilon_i)$	
.	.	
.	.	
▨	▨	$\leftarrow P_{max1}$
.	.	
.	.	
▨	▨	$\leftarrow P_{max2}$
.	.	
.	.	
C_M	$L_M(\varepsilon_i)$	

$$\left(L(\varepsilon_i) = \sum_{k=1}^{M} L_k(\varepsilon_i)\right)$$

Dividing L by the N-dimensional hypervolume covered by the receptive field which is proportional to ε^N , we can also estimate the local value of the density function $p_k(\mathbf{m}_i)$ of class C_k at position $\mathbf{m}_i$ in feature space.

$$p_k(\vec{m}_i) \sim \frac{P_{k,i}}{\varepsilon^N} \tag{16}$$

This is shown in figure 8.

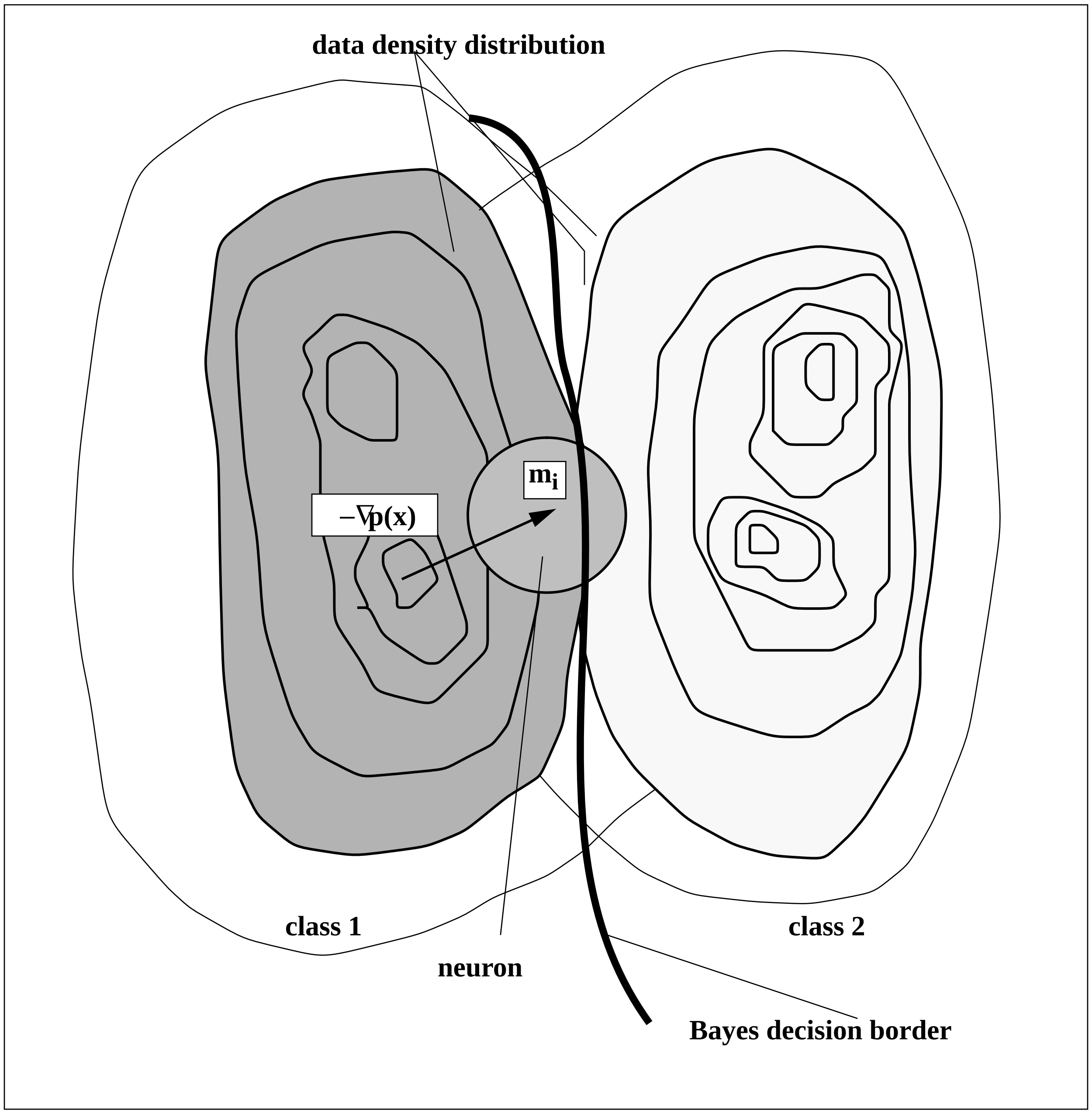


Figure 8: Checking the local a-priori probabilities $P_{k,i}$ for all classes C_k in the receptive field of a neuron $\mathbf{m}_i$. The contour lines represent the class density distributions.

As we remember the criterium for the Bayes decision border with eq. (4), the values of the density distributions for both classes have to become equal. According to our estimation scheme, the local Bayes criterium for a neuron $\mathbf{m}_i$ can be obtained from the table. If we indicate the two largest estimates P_{max1} and P_{max2}, then the neuron has to find a position, where

$$|P_{max1} - P_{max2}| < \gamma \qquad (17)$$

and γ is a small positive constant.

That means, the neuron stops in feature space at a position, where the local estimates of the density functions hold the Bayes criterium. This is also a locally optimal position of the neuron and any learning process has to stop there.

4.4 An iterated gradient descent technique

From figure 8 we see that the neuron itself should move from its initial position towards the class borders using the gradient of the density function $\nabla P(\mathbf{x})$. This induces the following gradient descent updating scheme for learning the weights $\mathbf{m}_i$ of any neuron.

According to the illustration of figure 9 we have to distinguish five cases for the updating of a neuron position.The primary goal is just to ensure, that a neuron

moves as long as it is not beyond the Bayes border of its assigned class $C(\mathbf{m}_i)$
{
I. in the direction of the steepest descent of the density function of its own class, $-\nabla P_{C(\mathbf{m}i)}(\mathbf{x})$
II. towards the cluster centroids of any other class
},

that it moves, if it is beyond the Bayes border of its class $C(\mathbf{m}_i)$
{
III. towards the cluster centroids of its class $C(\mathbf{m}_i)$, according $\nabla P_{C(\mathbf{m}i)}(\mathbf{x})$
IV. in the direction of the steepest decent of the density function of any other class.
}

and that it stops, if
V. the local Bayes criterium of eq. (17) holds.

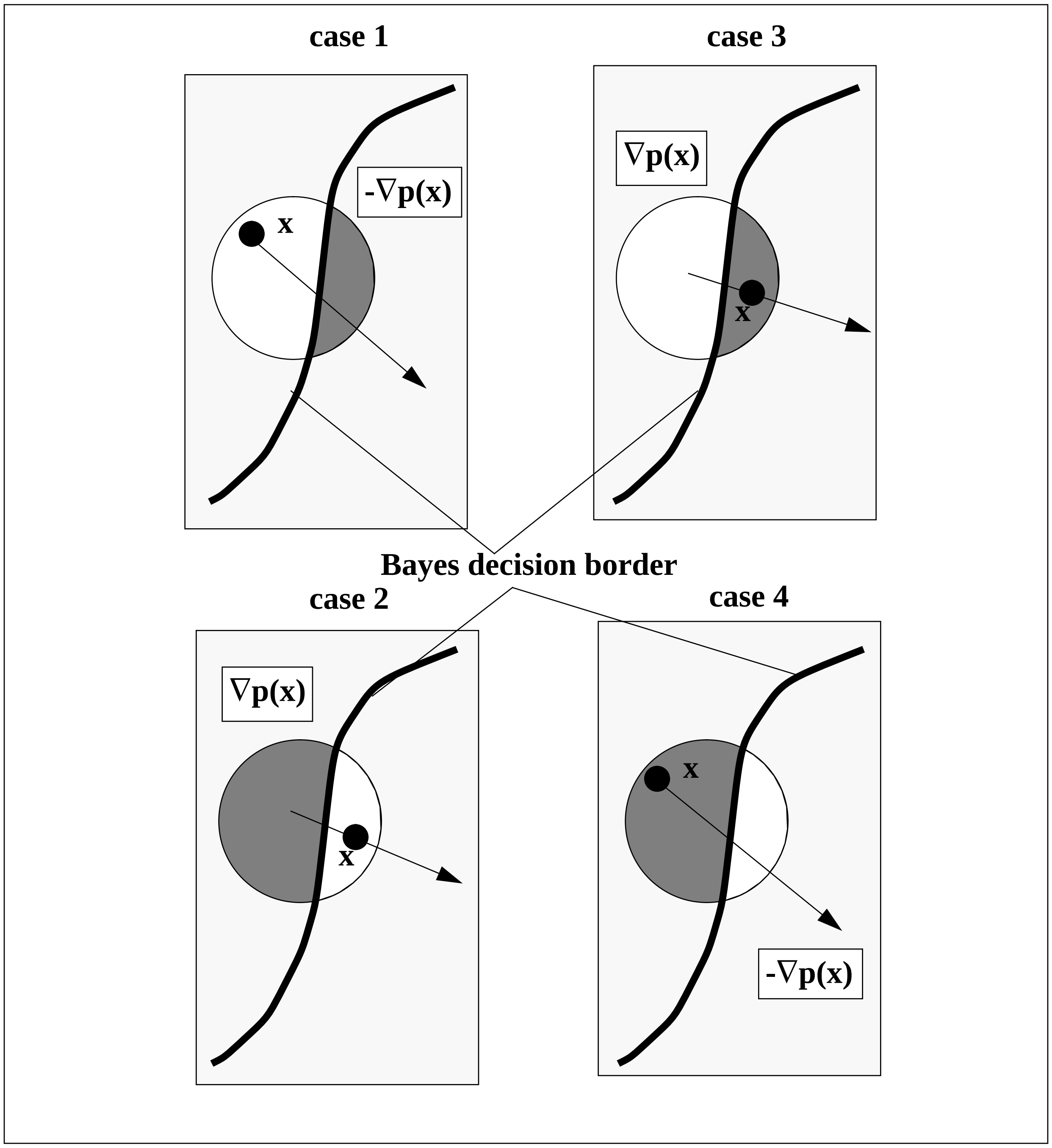


Figure 9: Updating the neuron position with gradient decent methods depending on the data condition in its receptive field.

This can be formulated as follows:
Choose randomly a sample $\mathbf{x}_l \in \{X\}_T$ and calculate the neuron $\mathbf{m}_c$ with the minimum euclidean distance

$$d_c = min(d_{i,l})|_i \tag{18}$$

and update the weight vector $\mathbf{m}_c$ of the neuron according to

Case 1:

if	$C(\mathbf{x}_l) = C(\mathbf{m}_c)$
and	$C(P_{max1}) = C(\mathbf{m}_c)$
and	$P_{max1} - P_{max2} > \gamma$

$$\vec{m}_c(t+1) = \vec{m}_c(t) - \alpha(t)\ [\vec{x}_l(t) - \vec{m}_c(t) + rand] \tag{19}$$

Case 2:

if	$C(\mathbf{x}_l) = C(\mathbf{m}_c)$
and	$C(P_{max1}) \neq C(\mathbf{m}_c)$
and	$C(P_{max2}) = C(\mathbf{m}_c)$

$$\vec{m}_c(t+1) = \vec{m}_c(t) + \alpha(t)\ [x_l(t) - \vec{m}_c(t) + rand] \tag{20}$$

Case 3:

if	$C(\mathbf{x}_l) \neq C(\mathbf{m}_c)$
and	$C(P_{max1}) = C(\mathbf{m}_c)$
and	$C(P_{max2}) = C(\mathbf{x}_l)$

$$\vec{m}_c(t+1) = \vec{m}_c(t) + \alpha(t)\ [\vec{x}_l(t) - \vec{m}_c(t) + rand] \tag{21}$$

Case 4:

if	$C(\mathbf{x}_l) \neq C(\mathbf{m}_c)$
and	$C(P_{max1}) \neq C(\mathbf{m}_c)$
and	$C(P_{max2}) = C(\mathbf{x}_l)$

$$\vec{m}_c(t+1) = \vec{m}_c(t) - \alpha(t)\ [\vec{x}_l(t) - \vec{m}_c(t) + rand] \tag{22}$$

Case 5: all other cases

$$\vec{m}_c(t+1) = \vec{m}_c(t) + \alpha(t)\, rand \tag{23}$$

α (t) holds eq. (7) and *rand* is a random number in the range [-1.0..1.0].

As this method uses a gradient descent technique, some of the neurons can get captured in local minima. The octree initialization of the neurons, however, ensures that most of them will find the local Bayesian border. For this reason, the additional noise term rand in the equations (19) - (23) can be used to allow some of them to escape from there.

In particular, the octree management of both data and neurons can help to learn more efficiently following the scheme above.

4.5 Updating the receptive field of a neuron

The scheme introduced above assumes that the local values of the density function p(**x**) are high enough to find reliable local estimations in the receptive field of the neuron. This has not always to be the case, especially when dealing with sparse data sets or when using large numbers of neurons and small values of δ_{max} for the octree.

To ensure reliable estimates, we have to update the ε_i of the receptive field after each learning step of the neuron $\mathbf{m}_i$. This could be done by using several schemes. If we assume the average number of data in $\{X\}_{T,\varepsilon i}$ to be constant and equal to δ_{max} and if we furthermore estimate the data to be equally distributed around the center $\mathbf{m}_i$, then the total number of data in $\{X\}_{T,\varepsilon i}$ is for a N-dimensional problem proportional to the hypervolume covered by ε_i . This implies, that ε_i (t+1) can be estimated by

$$\varepsilon_i(t+1) = \varepsilon_i(t) \sqrt[N]{\frac{P_i(t+1)}{P_i(t)}} \qquad (24)$$

P_i represents the a-priori probability of **x** in ε_i of neuron $\mathbf{m}_i$. This implies, that the local gradient of ∇p(**x**) is constant, as often used in integration schemes.

4.6 Collision handling

Another problem that arises with the schemes introduced above is the question of what to do if one neuron collides with another neuron of the same class during the simulation. This collision detection and avoidance is a very generic problem in any particle systems. Indeed we have to compare the neuron position after updating with the positions of all other neurons of the same class. If we detect a collision, we have to move the actual neuron away. But after this, a new collision with another neuron might happen. We would have to check it again and move the neuron once more and so forth. Depending on the updating scheme we could even get captured in endless loops.

For this reason, we cannot avoid a collision in general. But we can update the neuron's position taking into account the position of other neurons. This algorithm is shown in figure 11. Let $\mathbf{x}_l$ be the datum actually chosen for updating the next neighbored neuron $\mathbf{m}_c$, that holds this new position and collides there with a neuron $\mathbf{m}_a$. If $d_{c,a}$ is the euclidean distance between the neurons, we postulate, that a collision happens, when

$$d_{c,a} \le min(\varepsilon_c, \varepsilon_a) \qquad (25)$$

that means, the centroid of the neuron with the larger receptive field lies in the ε-environment of the neuron with the smaller one (see also figure 10).

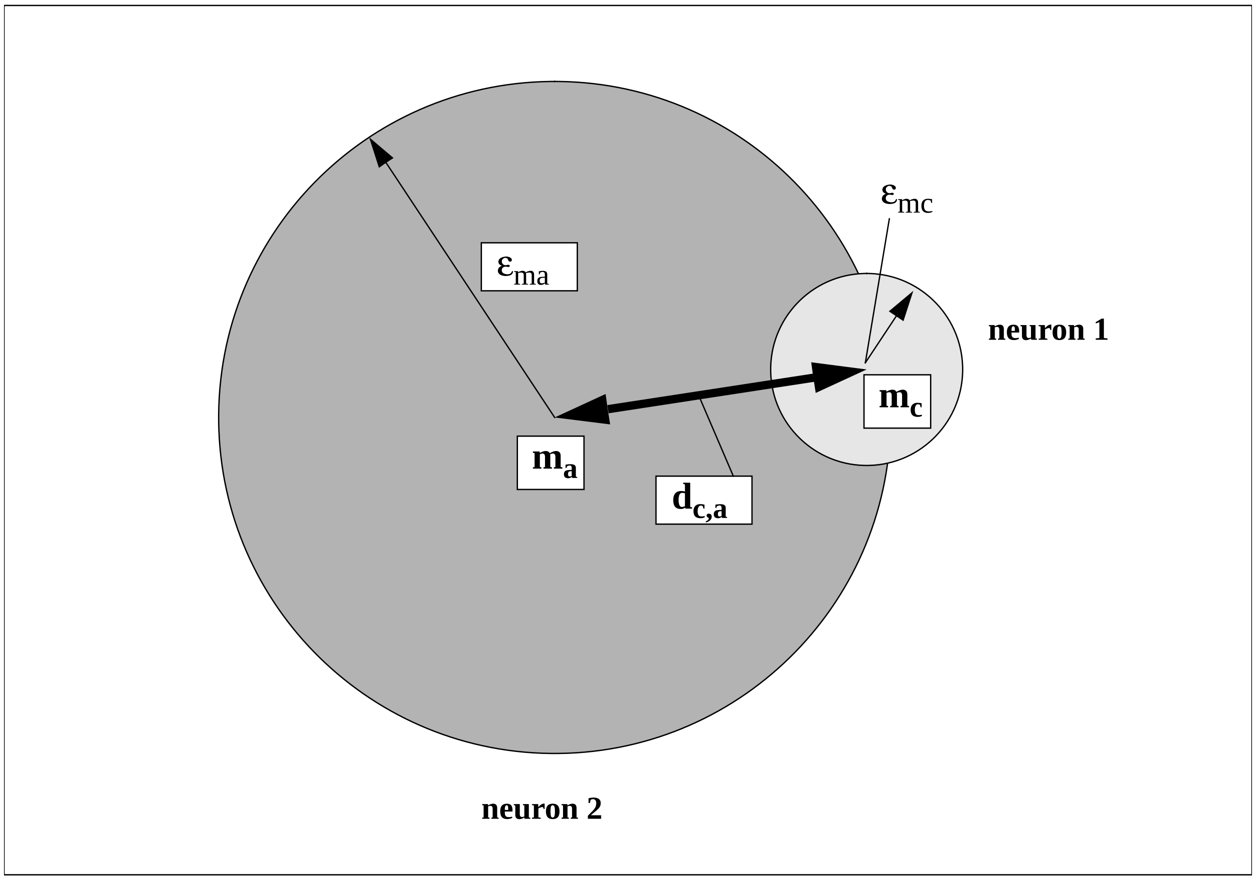


Figure 10: Definition of the collision of neurons $\mathbf{m}_a$ and $\mathbf{m}_c$.

Then we can set up the following updating rule according to figure 11. Let β be the angle between the vectors $|\mathbf{x}_l - \mathbf{m}_c|$ and $|\mathbf{m}_a - \mathbf{m}_c|$. To ensure a motion towards the class C ($\mathbf{m}_i$), we calculate

$$\cos\beta = \frac{(\vec{x}_l - \vec{m}_c)\ (\vec{m}_a - \vec{m}_c)}{\|\vec{x}_l - \vec{m}_c\|\ \|\vec{m}_a - \vec{m}_c\|} \qquad (26)$$

and distinguish

Case 1:

if $(\beta \in [0..\pi/2)$ or $\beta \in (3\pi/2..2\pi])$
and $C(\mathbf{x}_l) = C(\mathbf{m}_c)$

$$\vec{m}_c new = \vec{m}_c old + lap\,({}^n(\vec{x}_l - \vec{m}_c) + {}^n(\vec{m}_a - \vec{m}_c)) + \alpha(t) rand \quad (27)$$

Case 2:

if $(\beta \in [0..\pi/2)$ or $\beta \in (3\pi/2..2\pi])$
and $C(\mathbf{x}_l) \neq C(\mathbf{m}_c)$

$$\vec{m}_c new = \vec{m}_c old - lap\,({}^n(\vec{x}_l - \vec{m}_c) + {}^n(\vec{m}_a - \vec{m}_c)) + \alpha(t) rand \quad (28)$$

Case 3:

if $\beta \in\,]\pi/2..3\pi/2[$
and $C(\mathbf{x}_l) = C(\mathbf{m}_c)$

$$\vec{m}_c new = \vec{m}_c old + lap\,({}^n(\vec{x}_l - \vec{m}_a)) + \alpha(t) rand \quad (29)$$

Case 4:

if $\beta \in\,]\pi/2..3\pi/2[$
and $C(\mathbf{x}_l) \neq C(\mathbf{m}_c)$

$$\vec{m}_c new = \vec{m}_c old - lap\,({}^n(\vec{x}_l - \vec{m}_a)) + \alpha(t) rand \quad (30)$$

α (t) holds eq. (7), *rand* is a random number in the range [-1.0..1.0] and *lap* is a small positive value relative to $d_{a,c}$. ${}^n\mathbf{x}$ represents the normalized vector of **x**.

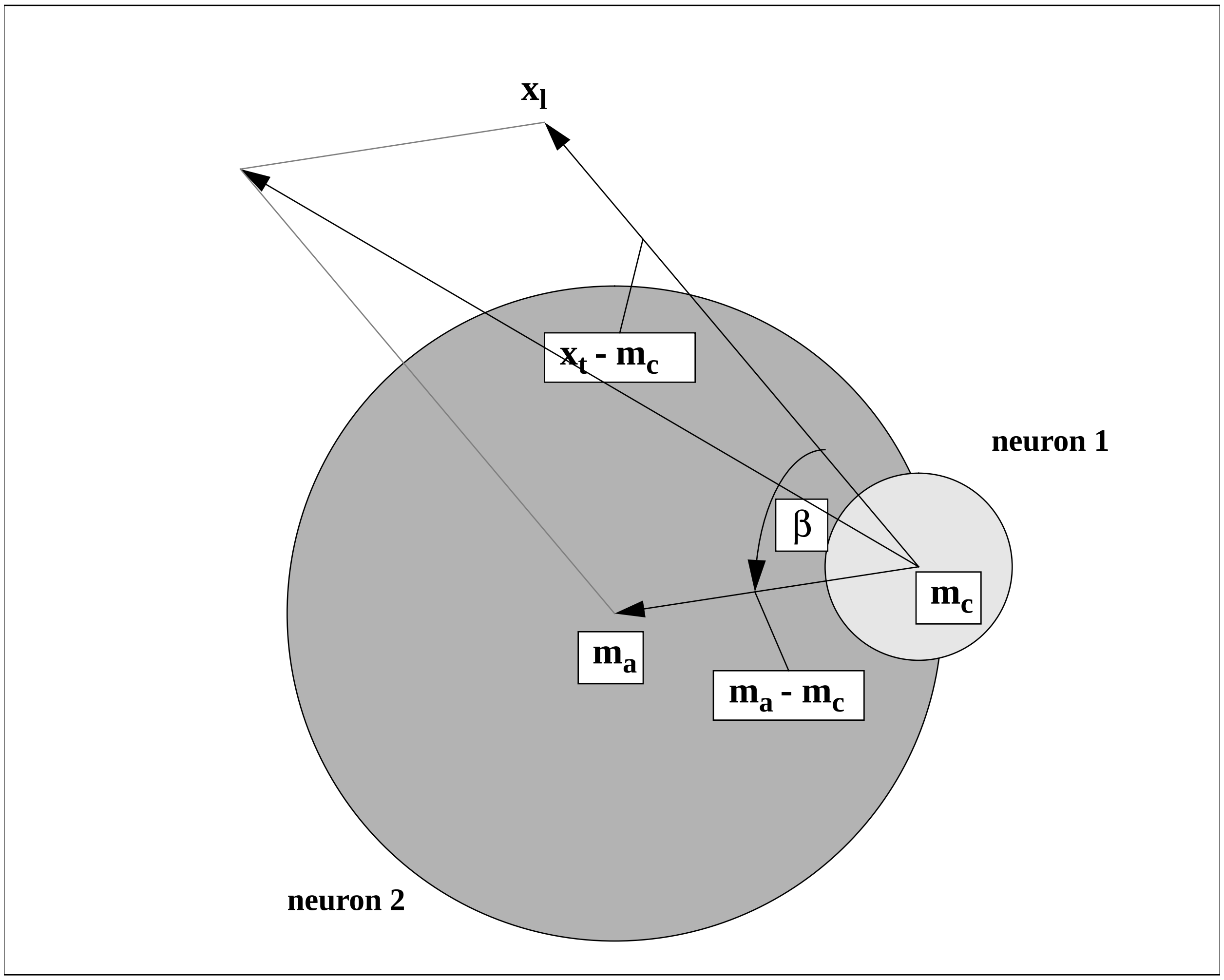


Figure 11: Collision avoidance and updating the neuron

This prevents the neurons from moving towards the centroids of classes $C_k \neq C(\mathbf{m}_c)$ and avoids a "spreading apart" of these neurons in case of a collision.

5. Growing and Optimization

5.1 General Remarks

During the learning phase of the network, the goal was to find some kind of equilibrium state, where most of the neurons arrange close to the Bayesian decision boundary satisfying local optimization criteria. When classifying any unknown data set {X} with this scheme in the sense of next neighbor decision, the class boundaries thus obtained should be not far from the real Bayes border. This is illustrated in figure 12. Depending on the underlying density distribution, some of the neurons could stay in local minima.

Also depending on the local positions of neurons of different class assignments, the actual classification boundary could deviate locally from the optimal border and could be deformed. This effect can decrease the classification accuracy by false class assignment (see also figure 12).

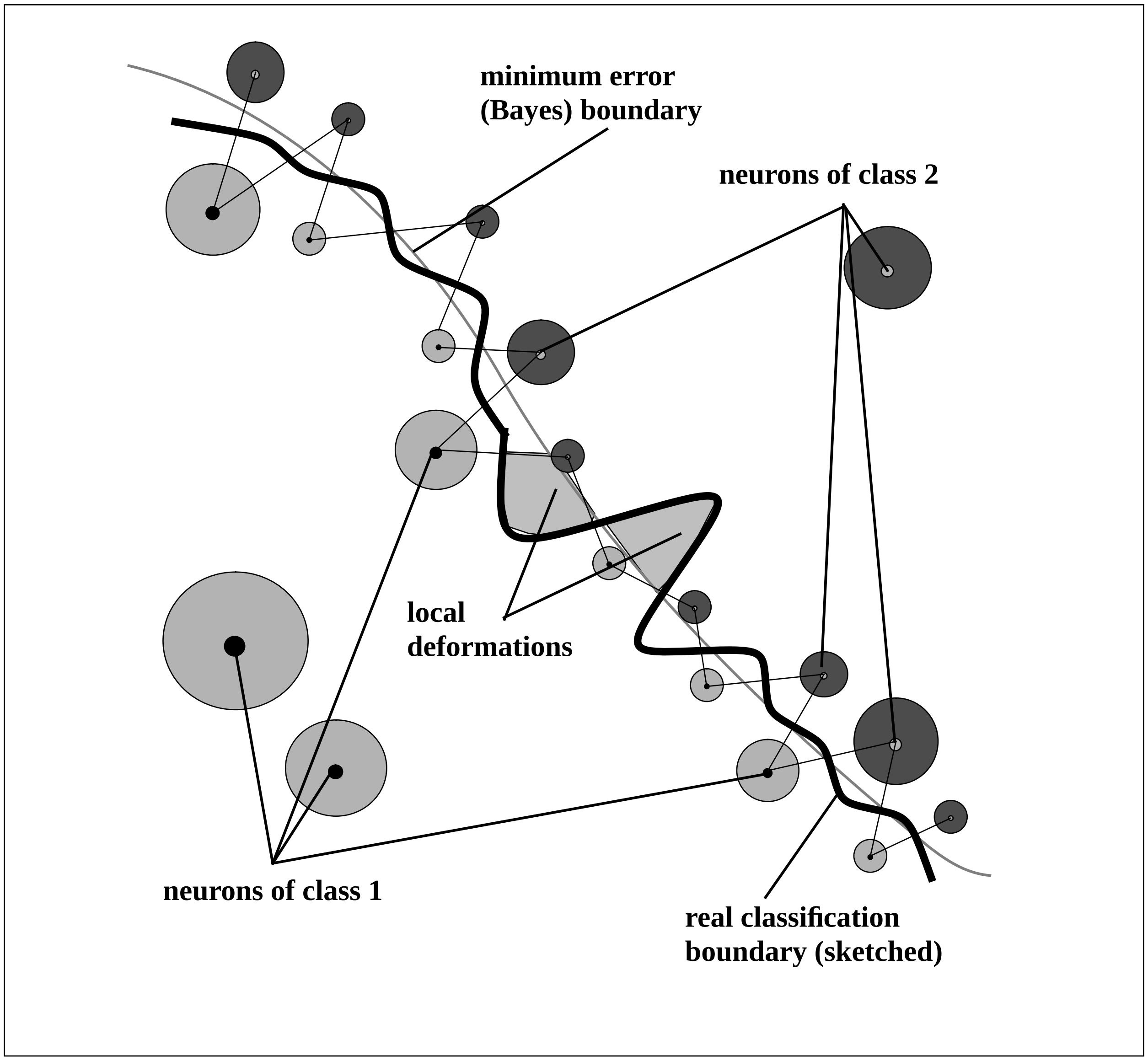


Figure 12: Real and optimal classification boundary after learning the network. Neurons that have no counterpart of the other class can lead to a strong deformation of the real class boundary.

To avoid this effect, we added a growing phase to the network, where additional neurons are generated and positioned close to the already existing ones, that have the largest fractional classification error. These neurons help to define the class border more accurately.

5.2 "Bearing" child neurons

The basic idea of the following growing concept is based on generating a child neuron from the J neurons with the largest fractional errors. For this purpose, we have first to set up a confusion matrix according to table 2 that describes the percentage of the false classified data of $\{X\}_T$ for each neuron $\mathbf{m}_i$. This can be done by reclassification of the whole data set $\{X\}_T$.

Table 2: Confusion matrix $||E||$ of the network

$$\begin{array}{c|c|ccccc} & & \multicolumn{5}{c}{\text{Neurons}} \\ & & \mathbf{m}_1 & \mathbf{m}_2 & \mathbf{m}_3 & .. & \mathbf{m}_L \\ \hline \text{Classes} & C_1 & e_{11} & e_{12} & e_{13} & .. & e_{1L} \\ & C_2 & e_{21} & e_{22} & \mathbf{e_{ef}} & .. & e_{2L} \\ & C_3 & e_{31} & e_{32} & e_{33} & .. & e_{3L} \\ & : & : & : & : & .: & : \\ & C_M & e_{M1} & e_{M2} & e_{M3} & .. & e_{ML} \end{array}$$

$e_{i,k}$ describes the number of false classified data of class k by the neuron $\mathbf{m}_i$ relative to the total number of data classified by $\mathbf{m}_i$.

If we pick now the neuron $\mathbf{m}_e$ with the largest fractional error $e_{e,f}$ in the class f, then we create a child neuron $\mathbf{m}_{ch}$ with the position

$$\vec{m}_{ch} = \vec{m}_e + rand \tag{31}$$

and the class assignment

$$\vec{m}_{ch} \rightarrow C_f \tag{32}$$

That means, the newly generated neuron belongs to the class where we had the largest fractional error and can be interpreted as a counterpart of the neuron $\mathbf{m}_e$. This is illustrated in figure 13.

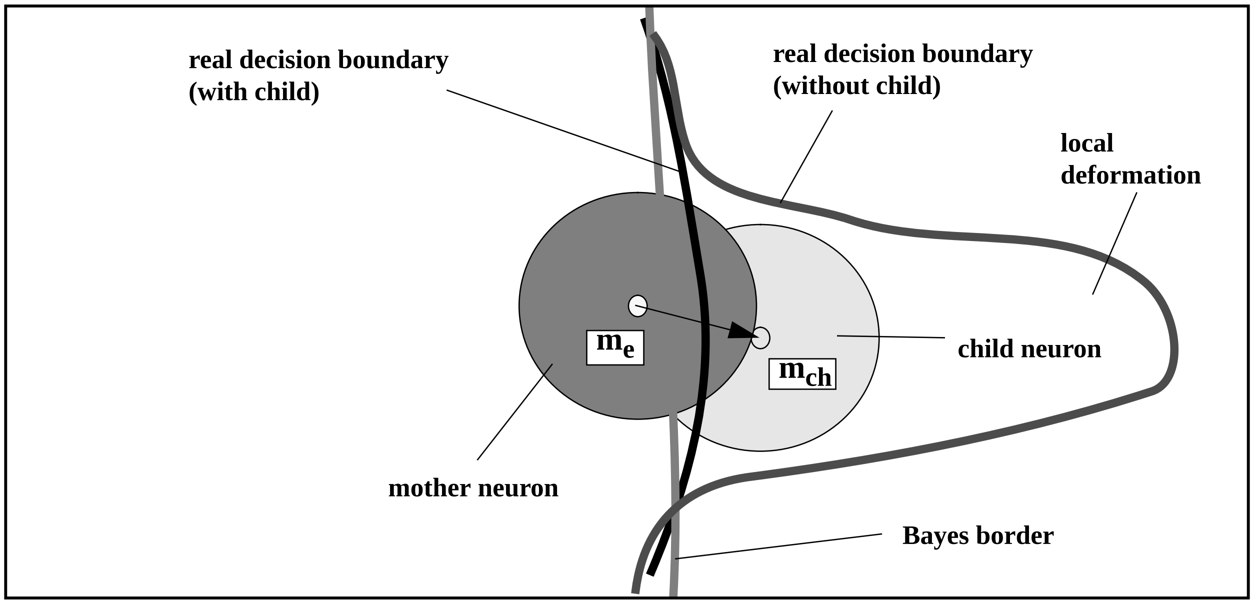

real decision boundary (with child)
real decision boundary (without child)
local deformation
m_e
m_ch
child neuron
mother neuron
Bayes border

Figure 13: The neuron with the largest fractional error is also the neuron that causes a strong local deformation in the decision boundary. This neuron creates a child of the counterclass as a counterpart to itself. In this way the error decreases and the curve becomes smoother.

5.3 Training of the child neurons

In order to ensure an optimal positioning of this neuron, we train it with data from $\{X\}_{Tf}$, exclusively belonging to the class C_f. This training is done according to the algorithm described in chapter 4.4, but only for a low number of cycles to avoid the neuron moving too much in direction of the data centroids.

A more self-stabilizing method could calculate the coordinates of $\mathbf{m}_{ch}$ directly by taking the mean of the data of class C_f falsely classified by neuron $\mathbf{m}_e$. Let $\{X\}_{Tf}$ $(\mathbf{m}_e)$ be these data, we just calculate

$$\vec{m}_{ch} = \frac{1}{L_{f,\, m_e}} \sum_{l=1}^{L_{f,\, m_e}} \vec{x}_l \quad , \quad \vec{x} \in \{X\}_{Tf} (\vec{m}_e) \tag{33}$$

This position can be interpreted as being somewhere in the center of the deformation of the classification border caused by the false classified data of neuron $\mathbf{m}_e$.

The process described above is done for the J neurons with the largest errors in the confusion matrix. It can be repeated for any number of growing cycles or until a certain error rate has been reached during reclassification. For reasons of performance, we always update the information about the 2 classes, that have the highest amount of data in the ε-environment of each neuron. One of these classes has to be the one that causes the largest error regarding the neuron and is therefore mainly responsible for the neurons reclassification error. Thus, this class becomes the class of the newly created child neuron $\mathbf{m}_{ch}$.

6. Results

6.1 General remarks

The following chapter describes simulation results obtained when applying the network described above on 2- and 3-dimensional examples. The problem to be solved by the network was a learning and classification of data samples generated by superposition of gaussian distributed data clusters. In order to faciliate a comparison of the tests, they were all proceeded until the absolute number of neurons reached the value 2000. An illustration of the development of the classification error for each test is given in figure 22. The %-values that are shown below pictures always document the remaining classification error after the phase of the algorithm decribed by the picture.

Figure 14 shows the initial data sets to be trained by the network for the 2-D case. The left picture contains a 3 class problem, the right one a four class problem.

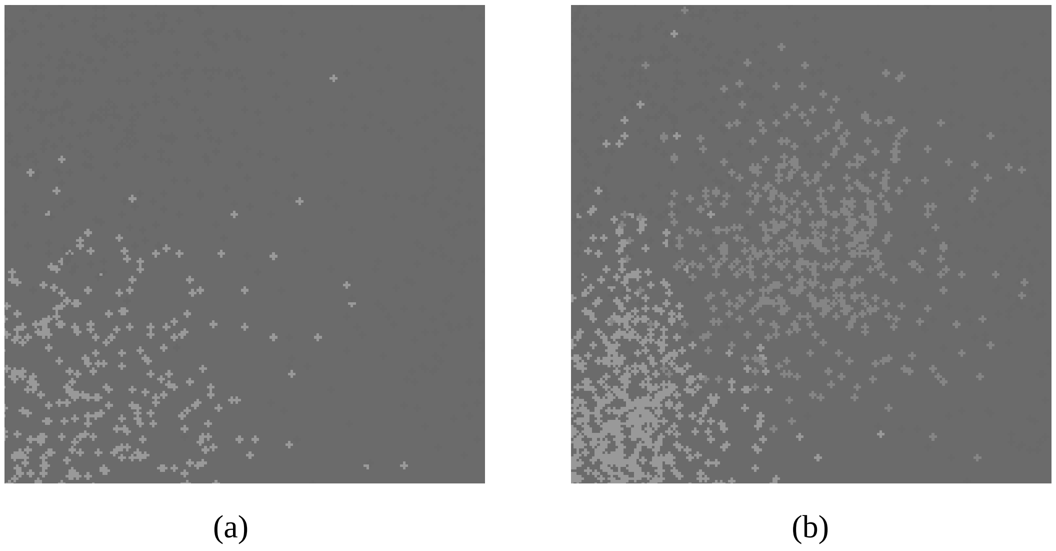


Figure 14: Data sets used to train the network (left: 3 classes, right: 4 classes)

We figured out a parameter study as well as a comparison of the network with standard LVQ and backpropagation learning.

6.2 Parameter studies

Figure 15 shows initial neuron positions (a) and the initial class assignment (b) for the 3 class problem of figure 14 (a). Furthermore the class assignment of the entire feature space is shown after initialization (c), after learning (d) and after growing. The development of the classification errors is always illustrated in the charts of figure 22. In the following case, the average data density was set to δ_{max} = 30 and 112 initial neurons had been created. A detailed parameter specification is listed in appendix B. Figures 16-17 show the same arrangement with a modified choice of δ_{max}.

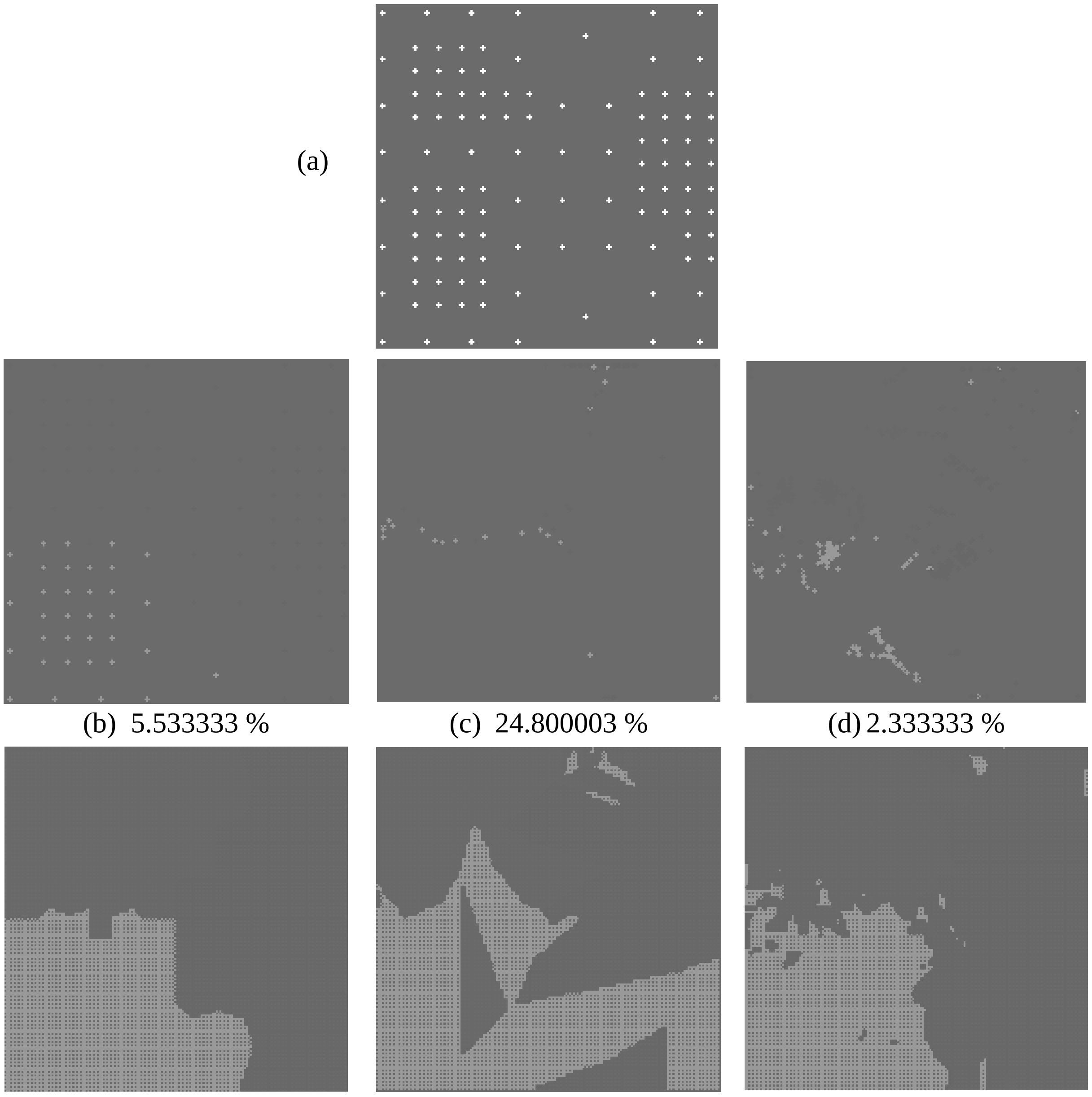


Figure 15: Neuron positions and class assignment of the entire feature space after several phases of the process (δ_{max} = 30).

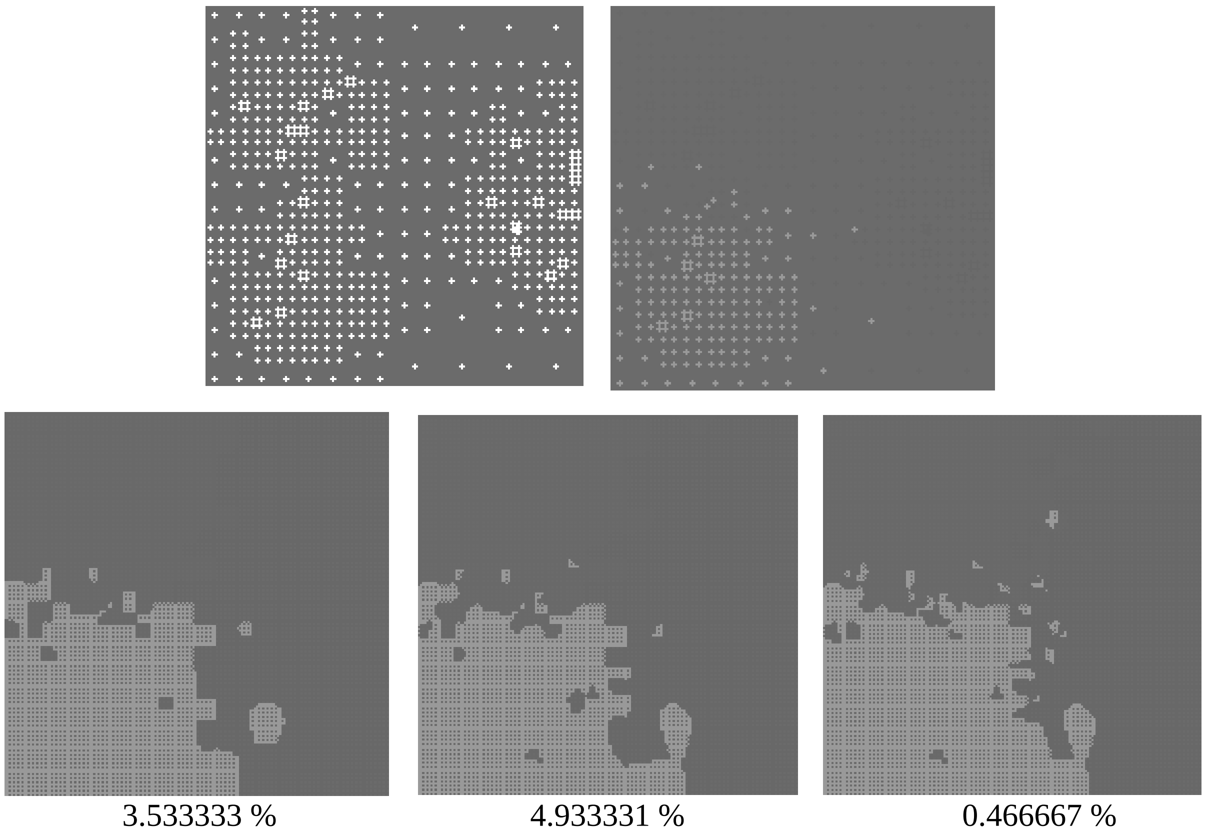


Figure 16: Neuron positions and class assignment of the entire feature space after several phases of the process ($\delta_{max} = 5$).

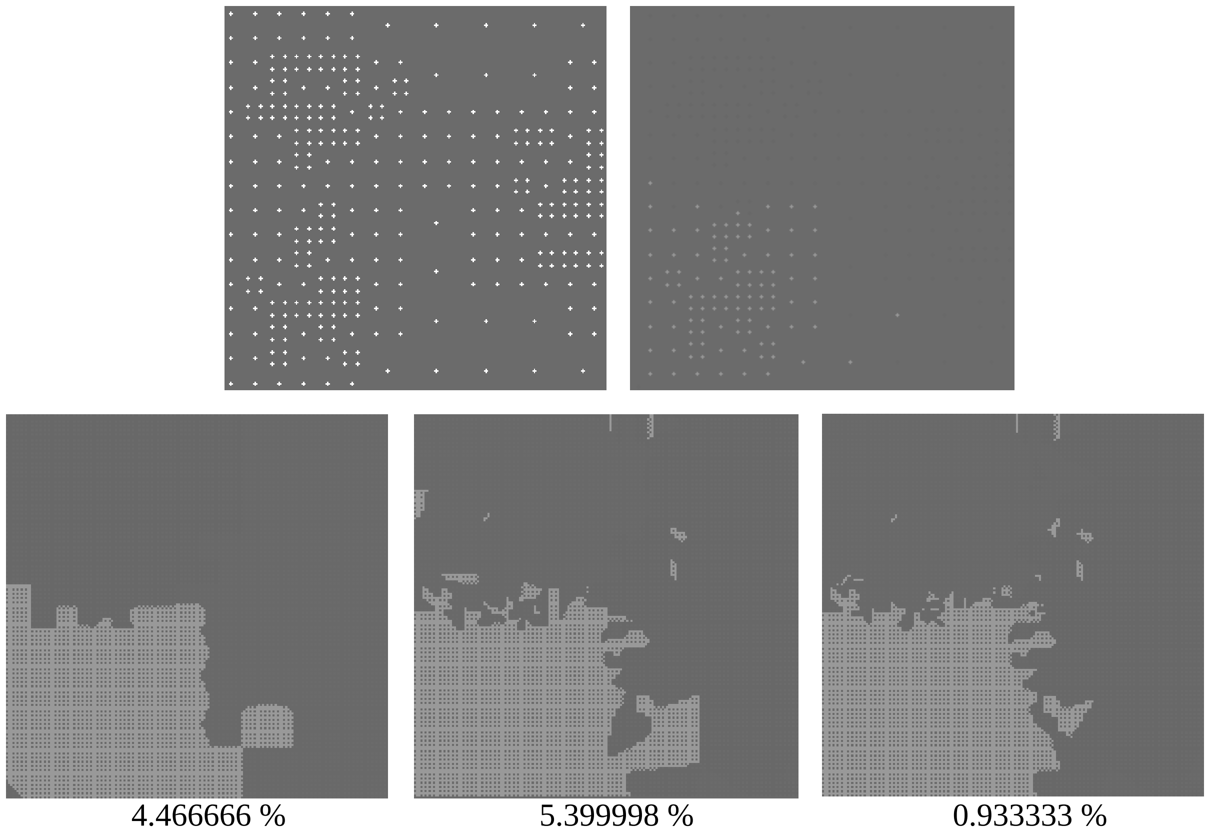


Figure 17: Neuron positions and class assignment of the entire feature space after several phases of the process ($\delta_{max} = 10$).

Figure 18 illustrates the process of learning and growing for $\delta_{max} = 20$. We recognize that the initial quadtree arrangement of the neurons is melting away in a certain way and that the neurons are arranging close to the class boundaries. During the growing phase, the differences of the neuron positions become smaller and smaller and the increasing number cannot be deviated anymore from the chart with its limited resolution.

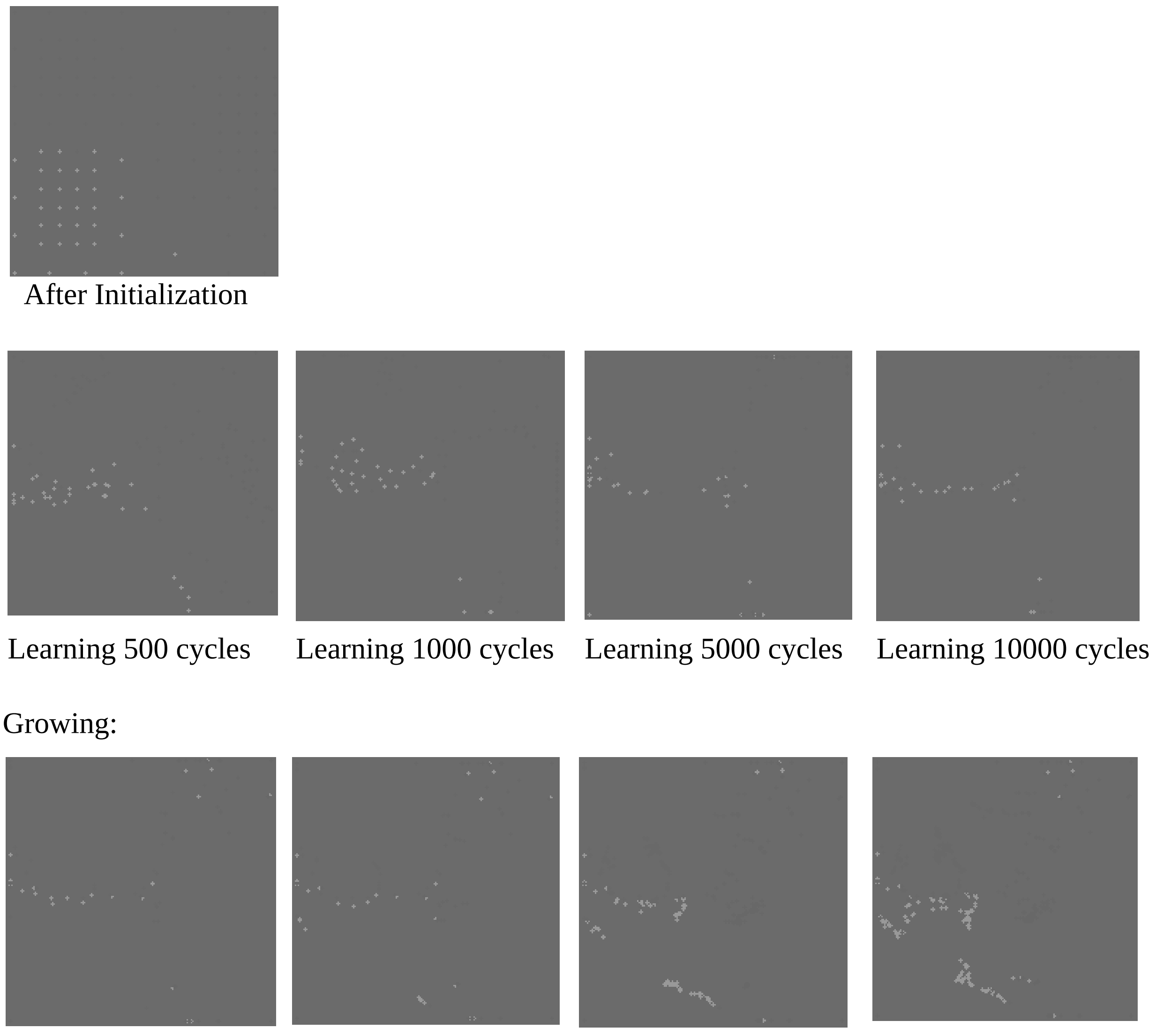
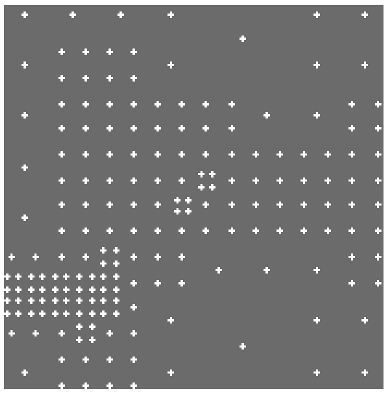
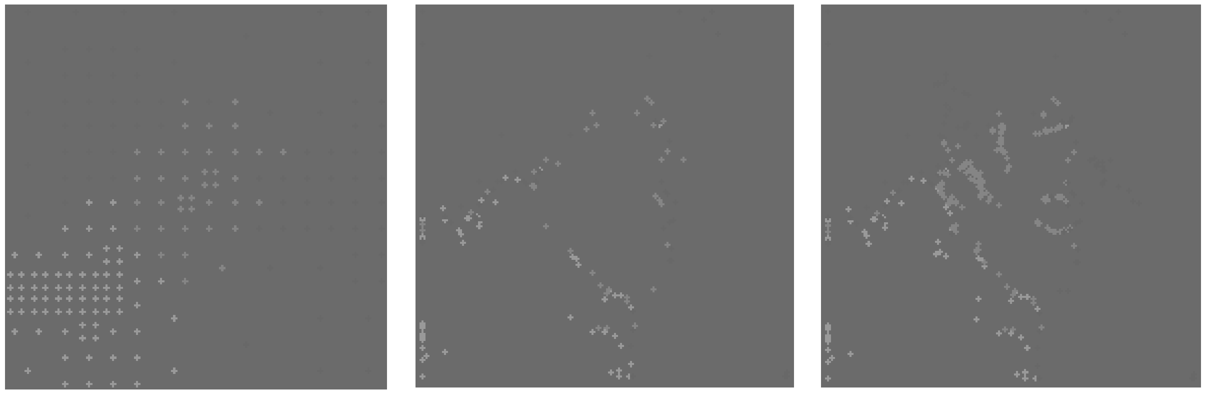


Figure 18: Neuron positioning during several phases of the algorithm. The quadtree is "melting" away with increasing number of learning cycles.

Figure 19 shows initial neuron positions (a) and the initial class assignment (b) for the 4 class problem of figure 14 (b). Furthermore the class assignment of the entire feature space is shown after initialization (c), after learning (d) and after growing. Again, development of the classification errors is illustrated in charts of figure 22. In the following case, the average data density was set to $\delta_{max} = 30$ and 193 initial neurons had been created. A detailed parameter specification is listed in the appendix B. Figures 20-21 show the same arrangement with a modified choice of δ_{max}.

(a)





(b) 8.439996 % (c) 25.520000 % (d) 7.119995 %

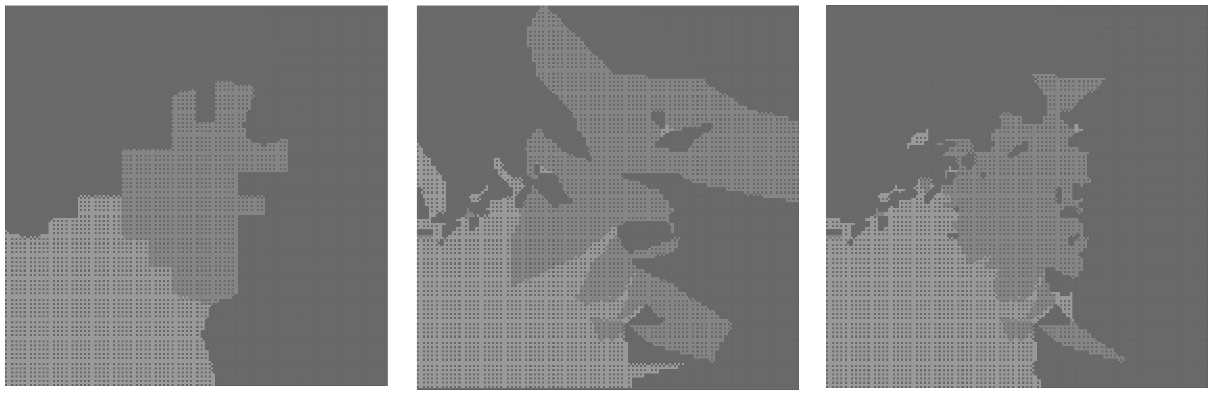


Figure 19: Neuron positions and class assignment of the entire feature space after several phases of the process (δ_{max} = 30).

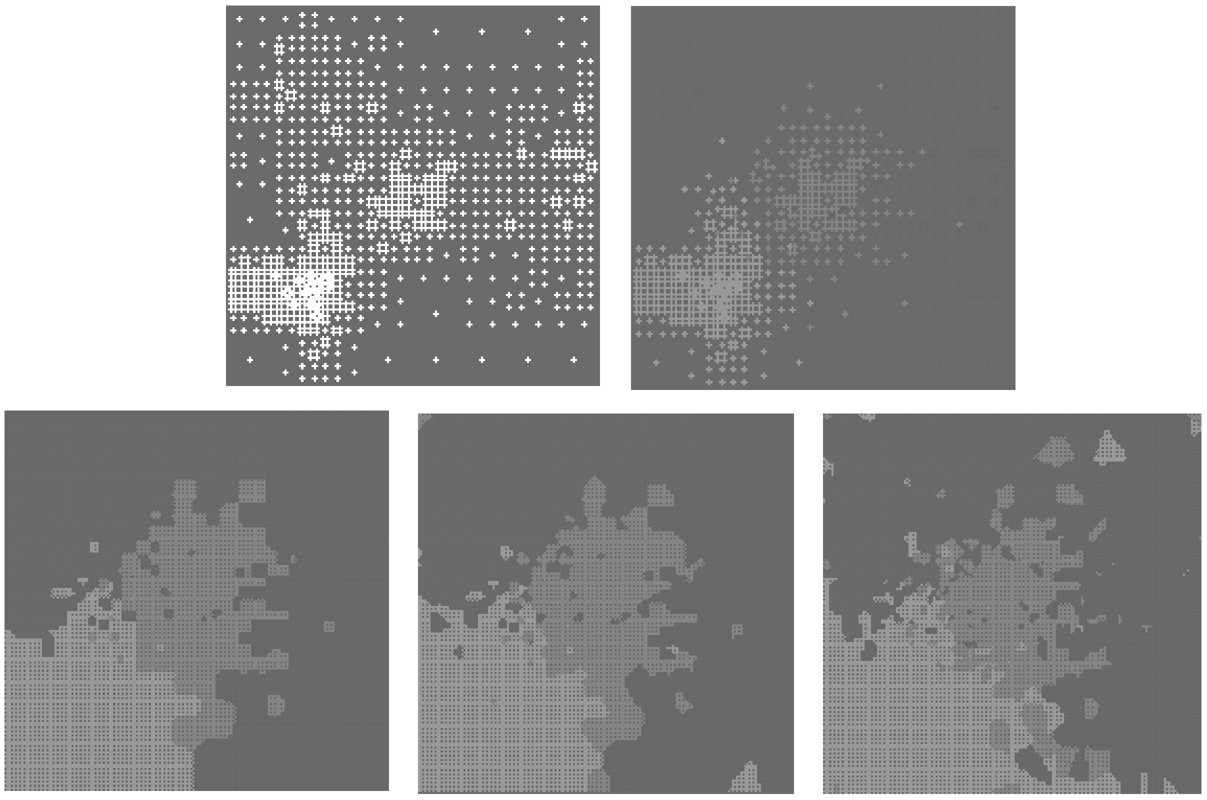


Figure 20: Neuron positions and class assignment of the entire feature space after the several phases of the process ($\delta_{max} = 5$).

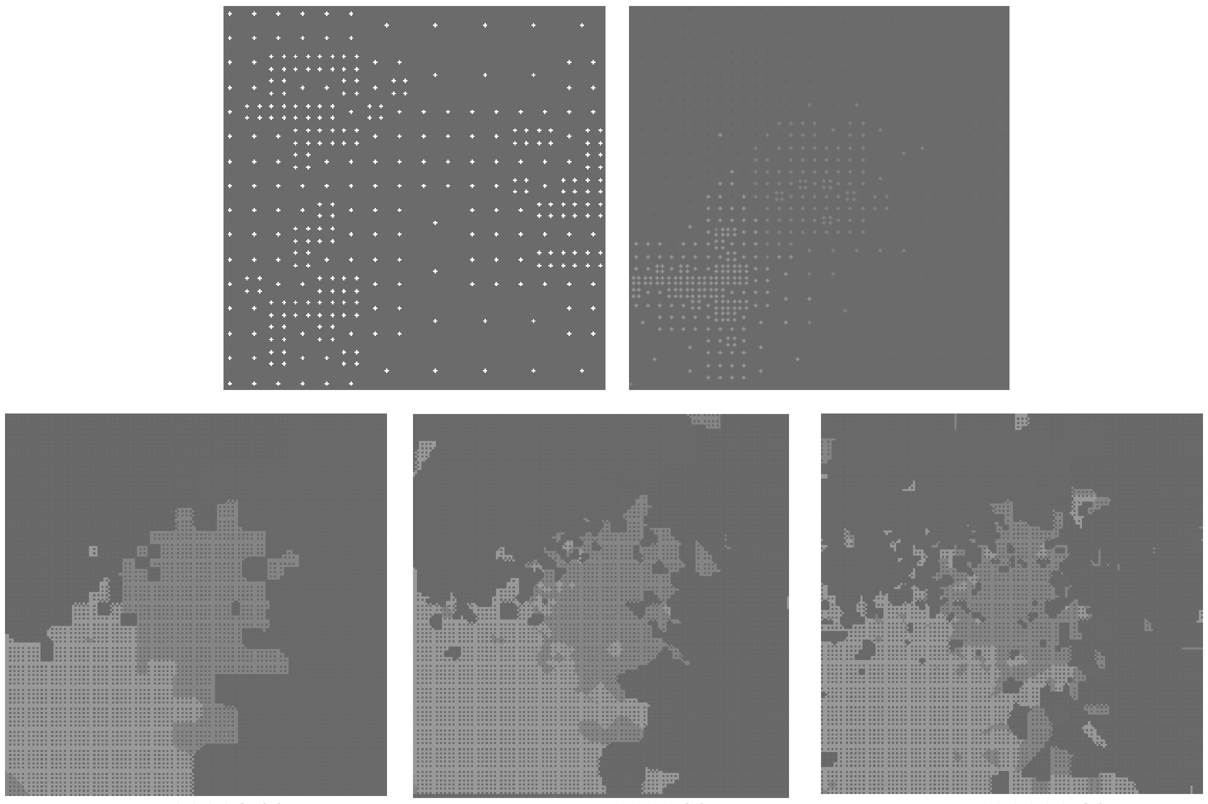


Figure 21: Neuron positions and class assignment of the entire feature space after the several phases of the process ($\delta_{max} = 10$).

Figure 22: Development of the classification errors in all "Melting Octree" tests.

6.3 Comparison with Backpropagation and LVQ

The following table shows the reclassification results obtained by out method and compares them with those of 2 backpropagation networks and 2 Kohonen-maps. In both cases, the "Melting Octree" algorithm provides better results even though we used more neurons with the Kohonen-Map II (2500). Another remarkable result is, that while the backpropagation networks and Kohonen-Maps already convergate, the results of the "Melting Octree" in some cases can still be ameliorated by "bearing" more child neurons. Figure 22 shows that in some of the tests the reclassification errors don’t yet convergate at all.

Table 3: Reclassification errors in % for 2-D tests

δ_{max} / type of net	3-class test	4-class test
5	0.466667	5.399996
10	0.933333	4.639999
30	2.333333	7.639999
Kohonen-Map I	3.202135	6.482593
Kohonen-Map II	4.336224	7.322929
Backpropagation I	5.403603	9.003601
Backpropagation II	5.003335	9.083633

(Learning cycles: 100000 for Backpropagation,
250000 for Kohonen-Map I, 900 neurons
1000000 for Kohonen-Map II, 2500 neurons)

Figures 23 and 24 show again the reclassification of the entire feature space for the 3-class and 4-class problem.

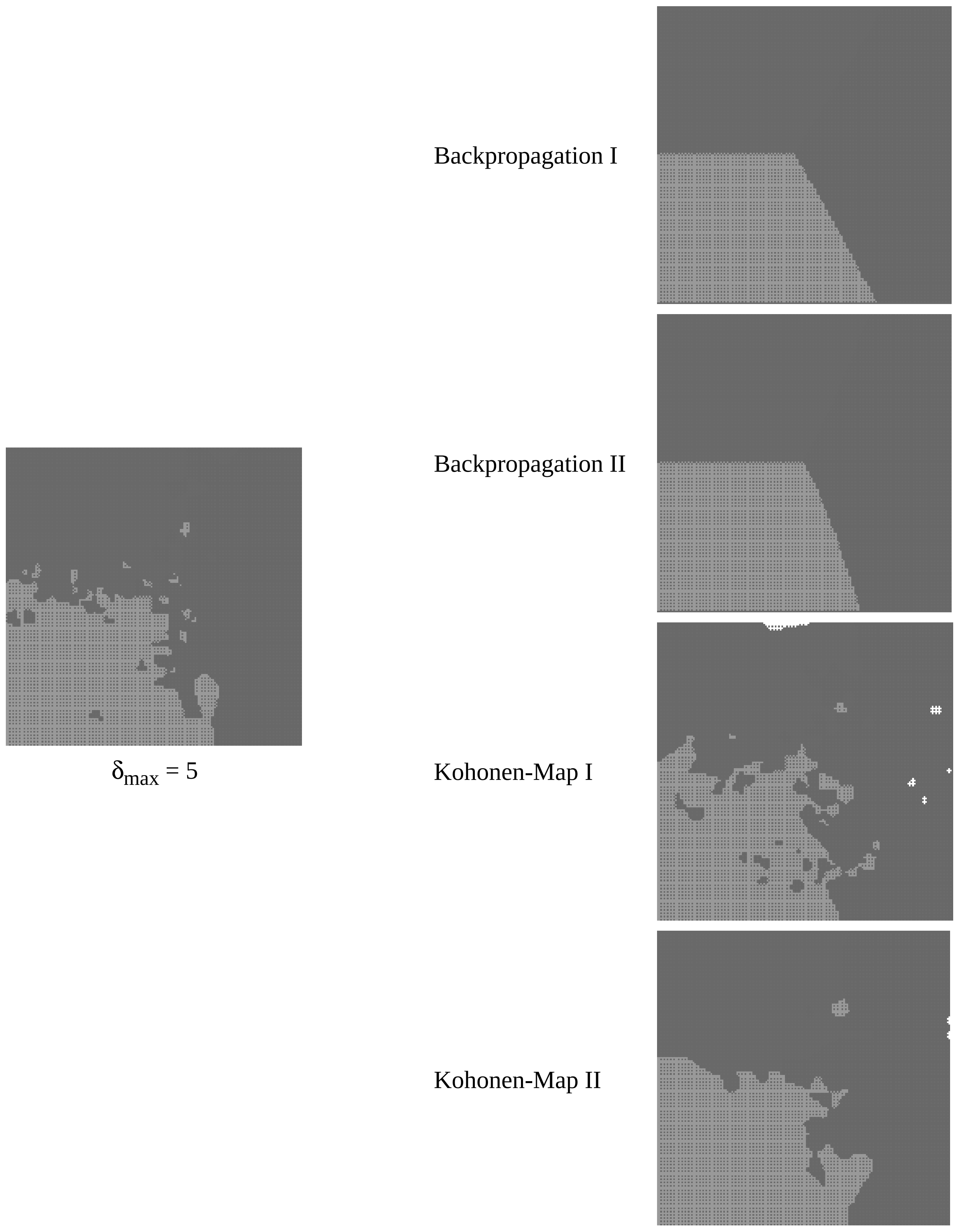


Figure 23: Comparison of final reclassification results for the 3-class tests.

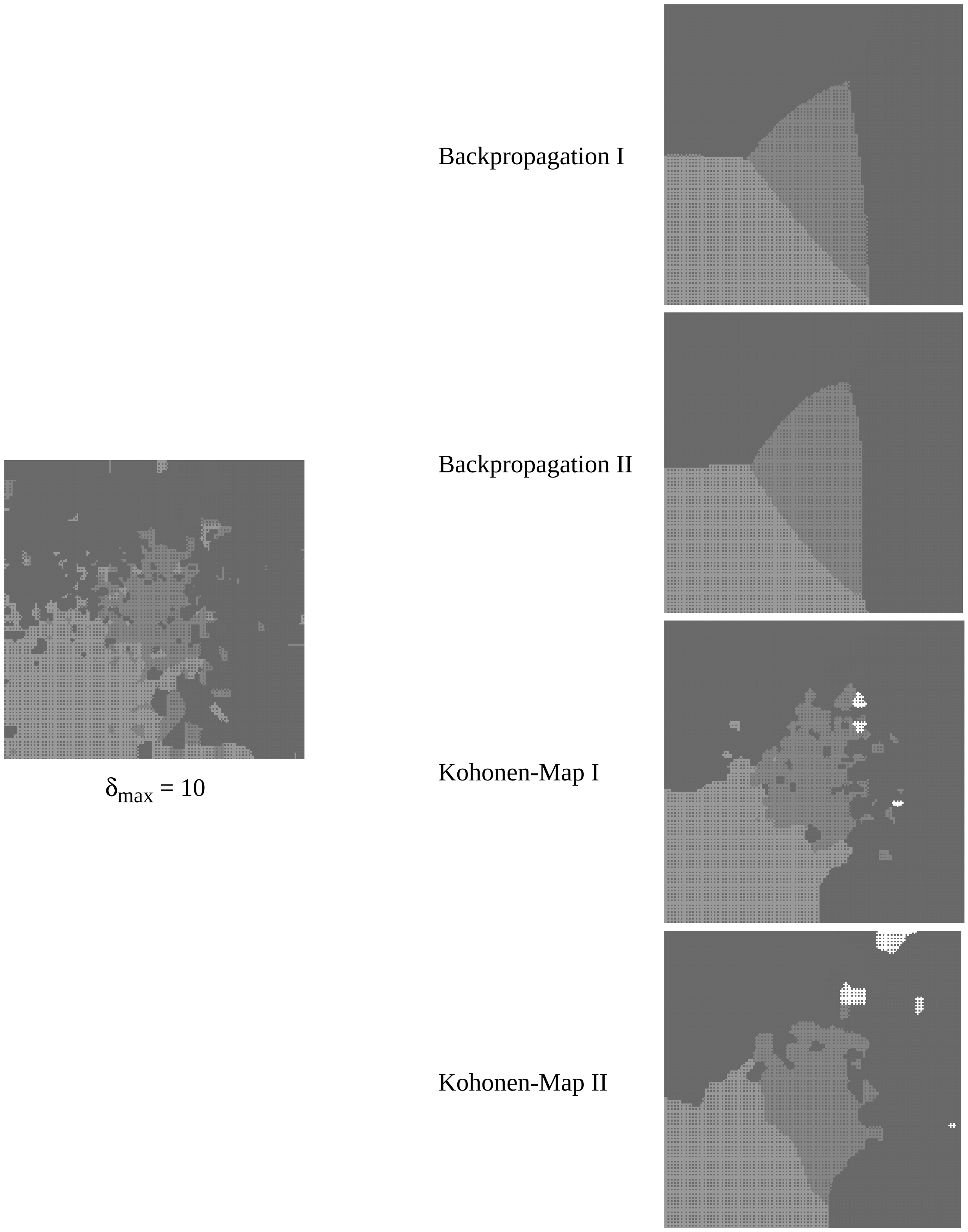


Figure 24: Final reclassification results for the 4-class tests.

7. Application on medical image data sets

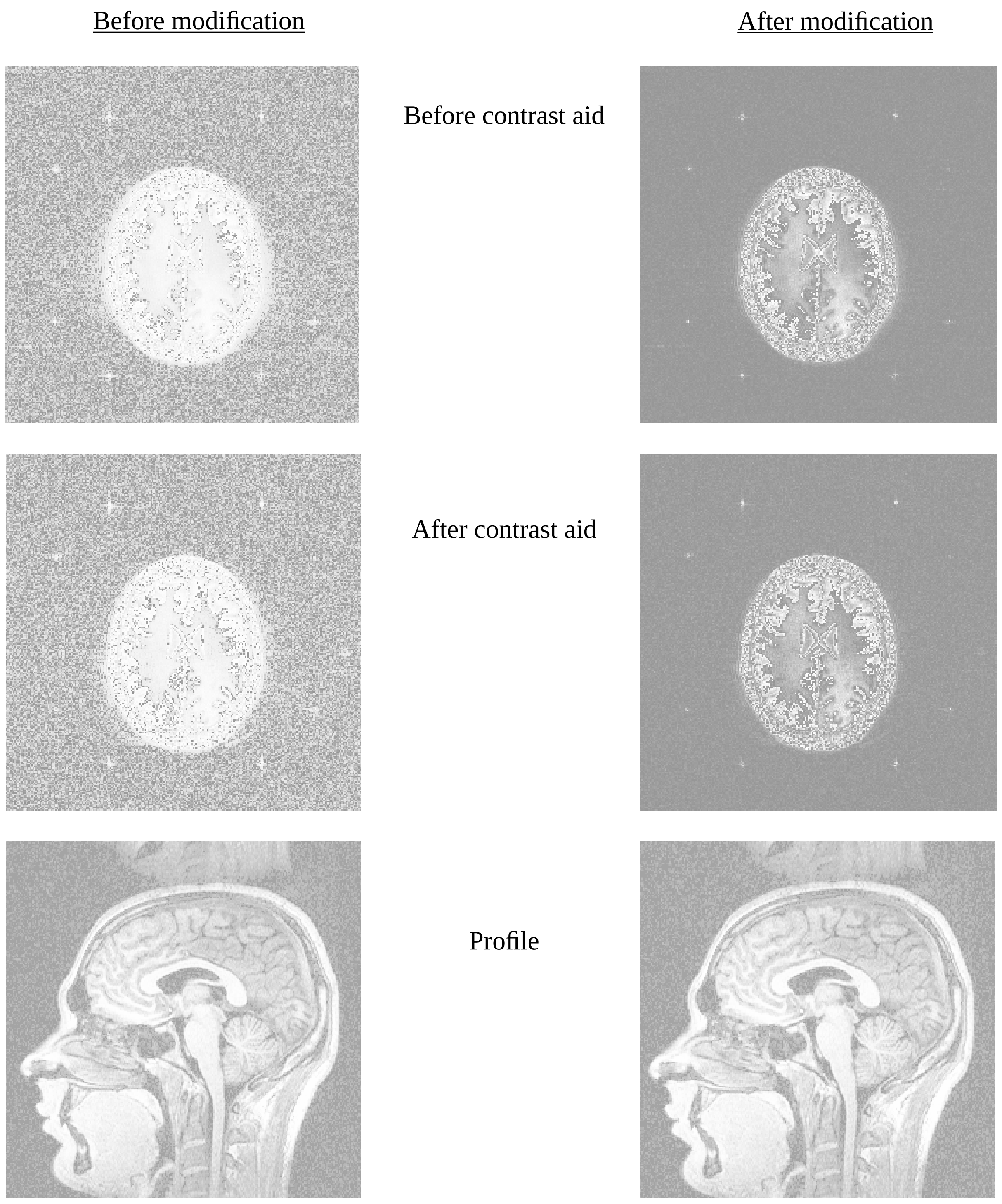


Figure 25: Future applications on medical data images.

Figure 25 shows pictures of very interesting and challenging classification problems: the tissue extraction from MRT data sets in medical imaging. The upper pictures show 2-D slices taken from a brain tumor patient (meningiom) by spin-echo technique before adding a contrast aid. The middle pictures show the same slice after contrasting the tissue. The tumor in the middle of the brain can now be recognized more easily. The right pictures always correspond to the pictures on the left sides. They were produced by applying several image processing techniques to the original pictures in order to try to ameliorate the image quality of the originals (left sides). These results show that even after the image processing it is very hard to locate the tumor so that more sophisticated methods are requested.

In order to develop a method being able to extract these damaged areas automatically, we have to introduce intensive preprocessing to find appropriate texture descriptors to be classified later on. This is very interesting in regard to the support of physician’s and surgeon’s work and in order to avoid unnecessary surgeries for diagnosis purposes.

8. Acknowledgements

The authors would like to thank the DBP Telekom/Germany that made this research possible being involved in medical image data analysis. Furthermore, we would like to thank the International Computer Science Institute, Berkeley, CA for giving us accomodation and computing support and the Deutsches Krebsforschungszentrum in Heidelberg for providing us with CRT brain data. Frank Seibert deserves further thank as he implemented and adapted the Kohonen and backpropagation procedures for our purpose. Finally, many thanks to Prof. Dr.-Ing. J.L. Encarnação, who gave us the opportunity to come to Berkeley and to develop the “Melting Octree”.

9. References

/1/ Amari, S.: “Mathematical Foundations of Neurocomputing.” Proceedings of the IEEE, Vol. 78, No. 9, September 1990, pp. 1443 - 1463

/2/ Amari, S.: “Neural Theory of Association and Concept-Formation.” Biological Cybernetics, Vol. 26, 1977, pp. 175 - 185

/3/ Oja, E.: “ A Simplified Neuron Model as a Principal Component Analyzer.” J. Mathematical Biology, Vol. 15, 1982, 267 - 273

/4/ Kohonen, T.: “Self-Organization and Associative Memory”, 3rd ed., Berlin, Heidelberg, Germany: Springer-Verlag, 1989

/5/ Kohonen, T.: “The Self-Organizing Map”. Proceedings of the IEEE, Vol. 78, No. 9, September 1990, pp. 1464 - 1480

/6/ Kirby, M.; Sirovich, L.: “Application of the Karhunen-Loeve Procedure for the Characterization of Human Faces”, IEEE Pattern Analysis and Machine Intelligence, Vol. 12, No. 1, January 1991, pp. 103 - 108

/7/ Rumelhart, D.; Hinton, G.; Williams, R.: “Learning Internal Representations by Error Propagation” in Parallel Distributed Processing: Explorations of the Microstructure of Cognition. Vol. 1: Foundations, Cambridge, Mass. : MIT Press, 1986, pp. 318 - 362

/8/ Duda, R.; Hart, P.: “Pattern Classification and Scene Analysis”. NewYork: John Wiley & Sons, 1973

/9/ Groß, M.; Seibert, F.: “Neural Network Image Analysis for Environmental Protection” to appear in the Proceedings of the GI-Workshop on Visualization for Environmental Protection, Springer Verlag, Germany

/10/ Groß, M.: “The Analysis of Visibility - Environmental Inferactions between Computer Graphics, Physics and Physiology”. Computers & Graphics, Vol. 15, No. 3, 1991, pp. 407 - 415

/11/ Groß, M.: “Image Analysis for Advertisement Purposes - A Computational Model of Visual Perception”. Computers & Graphics, Vol. 2, 1992..

/12/ Groß, M.; Seibert, F.: “Introducing Neural Networks for Visualization - The Analysis of Multidimensional Data Sets with Feature Mapping”. submitted to the Visual Computer References

/13/ Oja, E.: “Subspace Methods of Pattern Recognition”, Research Studies Press, England, 1983

/14/ Fukunaga, K.: Introduction to Statistical Pattern Recognition, 2nd Ed.. London, New York: Academic Press, 1990

Appendix A: Notations

$\mathbf{x} = (x_1, \dots , x_N)$	data vector
N;	dimension
$j = 1 \dots N$:	index for the dimension
$\{X\}_T = \{\{X\}_{T1}, \dots, \{X\}_{TM}\}$:	training data set
M:	number of classes
$k = 1 \dots M$:	index for the classes
$\{X\}_{Tk}$:	training data set of elements of class k
C_k:	class assignment, symbol for a class
$\mathbf{m} = (m_1, \dots, m_N)$:	weight vector of a neuron, also cluster centroid in feature space
$\{\mathbf{m}\}$:	set of neurons
$i = 1 \dots C$:	index for the neurons
C:	number of neurons/cluster centroids
ε:	width of the receptive field
ε_i:	width of the receptive field of neuron i
$P(\mathbf{x})$:	probability of $\mathbf{x}$
$P_e(\mathbf{x})$:	error probability of $\mathbf{x}$
b:	decision boundary for a one-dimensional example
d:	euclidian distance
d_{il}:	distance between neuron i and datum $\mathbf{x}_l$
$l = 1 \dots L$:	index for the training data
L:	number of elements in $\{X\}_T$
$t = 1 \dots T$:	simulation time
T:	maximum number of training cycles
$\{X\}_{T\varepsilon i}$:	Training data in the receptive field of neuron i
a, c, e, f:	inizes, that specify a selected element of a set {}
J:	number of neurons with the largest classification errors
$\| E \|$:	confusion matrix of order O(MxC)
$\mathbf{m}_m$:	mother neuron
$\mathbf{m}_{ch}$:	child neuron
rand:	uniform distributed random value {0...1}
$\{X\}$:	set of data vectors $\mathbf{x}$
σ:	small positive constant for the LVQ
$\mathbf{x}_{celli}$:	center coordinate for the octree cell i
δ_{max}:	maximum data density of one octree cell
γ:	instance of a neuron from the Bayes border
β:	angle for the collision handling

Appendix B: Parameter tables of documented tests

Table 4: Parameter for the 2-D tests on 3 classes

$\delta_{\mathbf{max}}$ **= 5**	$\delta_{\mathbf{max}}$ **= 10**	$\delta_{\mathbf{max}}$ **= 30**
ALPHA0 = 1.000000	ALPHA0 = 1.000000	ALPHA0 = 1.000000
T0 = 7500	T0 = 5000	T0 = 2500
T1 = 10	T1 = 10	T1 = 10
NOISE = 0.010000	NOISE = 0.010000	NOISE = 0.010000
EPSILON_MAX = 5	**EPSILON_MAX = 10**	**EPSILON_MAX = 30**
J_VALUE = 10	J_VALUE = 10	J_VALUE = 10
ERROR_MAX = 0.000000	ERROR_MAX = 0.000000	ERROR_MAX = 0.000000
T2 = 10	T2 = 10	T2 = 30
T3 = 135	T3 = 167	T3 = 190
ADJUST_EPSILON = 1	ADJUST_EPSILON = 1	ADJUST_EPSILON = 1
AVOID_COLLISION = 1	AVOID_COLLISION = 1	AVOID_COLLISION = 1

Table 5: Parameter for the 2-D tests on 4 classes

$\delta_{\mathbf{max}}$ **= 5**	$\delta_{\mathbf{max}}$ **= 10**	$\delta_{\mathbf{max}}$ **= 30**
ALPHA0 = 1.000000	ALPHA0 = 1.000000	ALPHA0 = 1.000000
T0 = 30000	T0 = 30000	T0 = 30000
T1 = 10	T1 = 10	T1 = 10
NOISE = 0.010000	NOISE = 0.010000	NOISE = 0.010000
EPSILON_MAX = 5	**EPSILON_MAX = 10**	**EPSILON_MAX = 30**
J_VALUE = 10	J_VALUE = 10	J_VALUE = 10
ERROR_MAX = 0.000000	ERROR_MAX = 0.000000	ERROR_MAX = 0.000000
T2 = 15	T2 = 15	T2 = 15
T3 = 95	T3 = 142	T3 = 181
ADJUST_EPSILON = 1	ADJUST_EPSILON = 1	ADJUST_EPSILON = 1
AVOID_COLLISION = 1	AVOID_COLLISION = 1	AVOID_COLLISION = 1

Table 6: Parameter for the 3-D tests on 2 classes

δ_{max} = 10	**δ_{max} = 20**
ALPHA0 = 1.000000	ALPHA0 = 1.000000
T0 = 5500	T0 = 5500
T1 = 10	T1 = 10
NOISE = 0.010000	NOISE = 0.010000
EPSILON_MAX = 10	**EPSILON_MAX = 20**
J_VALUE = 10	J_VALUE = 10
ERROR_MAX = 0.000000	ERROR_MAX = 0.000000
T2 = 10	T2 = 10
T3 = 150	T3 = 200
ADJUST_EPSILON = 1	ADJUST_EPSILON = 1
AVOID_COLLISION = 1	AVOID_COLLISION = 1

Table 7: Net configurations for the 2-D tests on 4 classes

Backpropagation net I	**Backpropagation net II**
TYPE = BACKPROPAGATION	TYPE = BACKPROPAGATION
INPUT = 0	INPUT = 0
TARGET = 1_OF_N	TARGET = 1_OF_N
ACTIVATION = SYMMETRIC	ACTIVATION = SYMMETRIC
LEARNING = NORMAL	LEARNING = NORMAL
LAYERS = 3	LAYERS = 4
LAYER#1 = 2 * 1 * 1, 0 , 0 , 0	LAYER#1 = 2 * 1 * 1, 0 , 0 , 0
LAYER#2 = 50 * 1 * 1, 0 , 0 , 0	LAYER#2 = 50 * 1 * 1, 0 , 0 , 0
LAYER#3 = 4 * 1 * 1, 0 , 0 , 0	LAYER#3 = 40 * 1 * 1, 0 , 0 , 0
	LAYER#4 = 4 * 1 * 1, 0 , 0 , 0

Table 8: Net configurations of the Kohonen-Maps for all 2-D tests

Kohonen-map net I	**Kohonen-map net II**
TYPE = KOHONEN_MAP	TYPE = KOHONEN_MAP
LAYERS = 2	LAYERS = 2
LAYER#1 = 2 * 1 * 1	LAYER#1 = 2 * 1 * 1
LAYER#2 = 30 * 30 * 1	LAYER#2 = 50 *50 * 1

Table of Contents