# Contents

1. Application of Kohonen and Counterpropagation Neural Networks  1

2. A Brief Introduction to Self-organizing Neural Networks  3
   2.1 Unsupervised Learning  3
   2.2 Supervised Learning: Counterpropagation Network  4

3. Implementation of Self-organizing Feature Maps in SONNIA  5
   3.1 Training of Kohonen Networks  5
   3.2 Counterpropagation Networks: Supervised Learning  7
   3.3 Network Topologies  7
   3.4 Generation of Kohonen Maps  8

4. A Typical Application of SONNIA  9

5. GUI - Graphical User Interface  17
   5.1 Starting the SONNIA GUI  17
   5.2 The Main Window of SONNIA  17
   5.3 File Menu  18
   5.4 Network Menu  20
   5.5 Maps Menu  23
   5.6 Map Context Menu  26
   5.7 Options Menu  30

6. Running SONNIA in Batch Mode (Reference Manual)  31
   6.1 Starting the CACTVS tclserver  31
   6.2 Loading CACTVS Packages  31
   6.3 SONNIA Commands  31
   6.3.1 Handling of Patterns  32
   6.3.2 Handling of Kohonen Networks  32
   6.3.3 Handling of Kohonen Feature Maps  35

7. Example of a SONNIA Tcl Script  40

8. File Formats  41
   8.1 The Data File  41
   8.2 The Prediction File  41
   8.3 The Network File  42
9 Program Installation 43
  9.1 IRIX6.5 ........................................................................................................ 43

Appendices 44
A List of Figures 44
B References on the Use of Kohonen Networks 45
1 Application of Kohonen and Counterpropagation Neural Networks

Teuvo Kohonen introduced an artificial neural network that he called self-organizing network. Learning in a Kohonen network is unsupervised, i.e., the property to be investigated is not used in the training process. In essence, a Kohonen network projects points in a multi-dimensional space into a space of lower dimensionality, usually into a two-dimensional plane. The results are then called maps of the high-dimensional space.

In this projection, the topology of the high-dimensional space is conserved in the lower-dimensional space, i.e., objects (points) that are close in the high-dimensional space are also close in the lower-dimensional space. Thus, in essence, Kohonen networks can be used for similarity perception; objects that are close in a Kohonen network are considered as similar. By the same token, Kohonen networks can be utilized for clustering objects; points adjacent in a Kohonen network can be considered as forming clusters.

The Kohonen learning algorithm can also be utilized for supervised learning. Objects then consist of pairs of information, the input vector for characterizing the objects, and an output vector for representing the property of the object to be studied. The output property can consist of a single value or of an entire vector of properties. Such supervised networks using Kohonen's training algorithm are called counterpropagation (CPG) networks.

A package for self-organized maps (SOM) is publicly available from the Kohonen group (http://www.cis.hut.fi/research/som_pak/ and http://www.cis.hut.fi/research/som-research/nncr-programs.shtml). We have, however, decided to develop a package of our own, the SONNIA software system, that is characterized by a variety of visualization tools, that allow easy and rapid analysis of the training process and the projection results. Of particular importance are those visualization techniques that comfortably support the study of datasets from chemistry and drug design. Methods for the visualization of two-dimensional chemical structures and of reaction equations are included in the program SONNIA.

A detailed introduction into Kohonen and counterpropagation neural networks can be found in:

J. Zupan, J. Gasteiger,
Neural Networks in Chemistry and Drug Design,

This book also reports on a variety of applications. In our group we have employed Kohonen and counterpropagation networks for the study of a series of problems. An overview of publications from our group can be found in the Appendix B (p. 45).

The objective of Chapter 2 is to give an brief overview to self-organizing neural networks. In Chapter 3 (p. 5) we describe the implementation of self-organizing neural networks in SONNIA with respect to the training process and the generation of Kohonen maps. As SONNIA can be run both in an interactive mode and in batch mode Chapter 4 (p. 9) introduces the user into the application of the SONNIA Graphical User Interface giving an example of a
typical application. A more detailed description of all features provided by the SONNIA GUI follows in Chapter 5 (p. 17).

The next two Chapters 6 (p. 31) and 7 (p. 40) refer to the use of SONNIA in the batch mode. Whereas Chapter 6 serves as a reference manual, Chapter 7 is provided as an example for the use of SONNIA in the batch mode.

The file formats of the data file, prediction file, and network file are explained in Chapter 8 (p. 41).

Finally the installation procedure and the files provided within the distribution CD-ROM of SONNIA are specified in Chapter 9 (p. 43).
2 A Brief Introduction to Self-organizing Neural Networks

2.1 Unsupervised Learning

Teuvo Kohonen introduced an algorithm for the construction of a non-linear projection of objects from a high-dimensional space into a lower-dimensional space (Ref.: Kohonen, T.; Self-organized formation of topologically correct feature maps, *Biol. Cybern. 1982, 43, 59-69*). The main feature of the target space covering the projection result is that it is only two- or three-dimensional. Despite of the reduction of dimensionality, internal relationships between points (the objects) building the high-dimensional space are largely preserved during this mapping procedure. Points that are neighbors in the high-dimensional space should also become neighbors in the resulting projection. The main advantage of this projection into low-dimensions is that it can easily be used for visualisation purposes. Relationships between objects in the high-dimensional space can be investigated by analyzing the projections. Figure 1 shows the architecture of a Kohonen neural network. Each column in this two-dimensional arrangement represents a neuron; each box of a column represents the weight of a neuron.

![Figure 1: Architecture of a Kohonen neural network](image)

The results in Kohonen networks are visualized as two-dimensional maps by looking at the block in Figure 1 from the top.

Throughout this manual the $x$- and $y$-direction are referred to as the *width* and *height* of the neural network as they are concerned with Kohonen maps. The *length* of the input vectors is oriented in the $z$-direction.

In summary, one iteration of the Kohonen algorithm contains the following steps:

1. An $m$-dimensional object, $X(t) = (x_1, x_2, x_3, \ldots, x_m)$, enters the network.
(2) The responses of all neurons are calculated.

(3) The "winning neuron", \( c \), which is most similar to the current object vector, \( X(t) \), is determined.

(4) The weights of neuron, \( c \), are corrected to improve its response to the same input in the next epoch.

(5) The weights of all neurons in the neighborhood of neuron, \( c \), are adapted to improve their response to \( X(t) \).
   For a single neuron the degree of correction decreases with increasing topological distance to \( c \).

(6) This iteration is repeated for the next \( m \)-variate object \( X(t + 1) \).

After termination of the training the response of the network is calculated for each object of the data set. The projection of the data set into the two-dimensional space is then performed by mapping each object, \( X(t) \), into the coordinates of its winning neuron.

### 2.2 Supervised Learning: Counterpropagation Network

The Kohonen learning algorithm can also be utilized for modeling problems, for relating objects with their properties. The objects are again represented by an \( m \)-dimensional vector that is used to construct the input block of a network. Each object is associated with a vector (can also be a single value) representing a property or various properties of this object. These properties are contained in the output block.

Such a network consisting of an input and output block is called a counterpropagation (CPG) network (Figure 2). For the training of such a CPG network, pairs of objects and their properties have to be presented to the network, essentially performing a supervised learning. Such a trained network can then be utilized for the prediction of properties of new objects.

![Figure 2: Architecture of a counterpropagation (CPG) neural network](image-url)
3 Implementation of Self-organizing Feature Maps in SONNIA

3.1 Training of Kohonen Networks

The Kohonen algorithm has already been briefly introduced in chapter 2 by steps (1) to (6). In the following, its implementation in SONNIA is explained in more detail.

Step (1) to (3): Determination of the "winning neuron"
Similarity is measured in SONNIA by the Euclidian distance. Therefore, the response of a neuron to an object $X(t)$ is defined as the Euclidian distance between the weight vector $W_j$ and the pattern $X(t)$:

$$\text{out}_c \leftarrow \min \|X(t) - W_j\| = \min \left\{ \sum_{i=1}^{m} [x_i(t) - w_{ji}]^2 \right\}$$

(eq. 1)

The Index $j$ refers to a particular neuron from among all $n$ neurons. $W_j$ and $X(t)$ have the same dimensionality $m$, and $t$ identifies a particular data pattern. The neuron having the smallest response is called the "winning neuron".

Step (4) to (5): Adaptation of the weights of the neurons
After a "winning neuron", $c$, is determined, the weights, $w_{ji}$, of the network are adapted. The index $t$ can be interpreted as a discrete time coordinate, that counts the number of presentations of patterns.

$$w_{ji}(t + 1) = w_{ji}(t) + h_{ji}(t, c) \cdot \left( x_i(t) - w_{ji}(t) \right)$$

(eq. 2)

The magnitude of the weight adaptation depends on two factors. The first factor is the neighborhood function $h_{ji}(t, c)$, a function of time $t$ (or the number of input patterns presented) and of the location of the "winning neuron", $c$. The second factor is the difference between the component of the pattern vector, $x_i(t)$, and its corresponding weight, $w_{ji}(t)$, of neuron $j$.

The neighborhood function, $h_{ji}(t, c)$, can again be separated into two parts, where $\eta(t)$ is the so-called learning-rate function and $\varphi_j(t, c)$ is a function that considers the influence of the topological distance between the neuron $j$ and the "winning neuron" $c$.

$$h_{ji}(t, c) = \eta(t) \cdot \varphi_j(t, c)$$

(eq. 3)

Usually, only the learning-rate function is time-dependent. The value of the learning rate in the next cycle is obtained by multiplying the current rate, $\eta(t)$, with a constant adaptation factor $\alpha$. The parameter $t_{\text{step}}$ controls how many training cycles have to pass until a change in the learning rate is made.

$$\eta(t + 1) = \eta(t), \quad \text{if} \quad t \neq n \cdot t_{\text{step}}$$

$$\eta(t + 1) = \alpha \cdot \eta(t), \quad \text{if} \quad t = n \cdot t_{\text{step}}, n \in \mathbb{N}$$

(eq. 4)
Figure 1 shows how the learning rate continuously decreases with \( t_{\text{step}} = 1 \). With \( t_{\text{step}} = 20 \) steps, the learning rate decreases in steps of 20 iterations.

![Figure 1: Dependency of the learning rate, \( \eta(t) \), on time, \( t \), shown for two different values of \( t_{\text{step}} \)](image)

As mentioned before, \( \varphi_j \) is the distance contribution to the neighborhood function. SONNIA offers the possibility to split \( \varphi_j \) into two alternative functions \( \varphi_{x(j)}(t,c) \) and \( \varphi_{y(j)}(t,c) \) representing the distance effect in \( x \)- and \( y \)-direction of the Kohonen network.

\[
\varphi_j(t,c) = \varphi_{x(j)}(t,c) = \max \left( 1 - \frac{d_{x(j)}(c)}{s_x(t)}, 0 \right), \quad \text{if} \quad \frac{d_{x(j)}(c)}{d_{y(j)}(c)} < \frac{s_x(t)}{s_y(t)} \tag{eq. 5}
\]
\[
\varphi_j(t,c) = \varphi_{y(j)}(t,c) = \max \left( 1 - \frac{d_{y(j)}(c)}{s_y(t)}, 0 \right), \quad \text{if} \quad \frac{d_{x(j)}(c)}{d_{y(j)}(c)} \geq \frac{s_x(t)}{s_y(t)}
\]

The spans \( s_x(t) \) and \( s_y(t) \) control which neurons undergo a correction of their weights. Starting from the "winning neuron", \( c \), only those neurons are adapted which have topological distances \( d_{x(j)}(c) \) and \( d_{y(j)}(c) \) that lead to a positive value of \( \varphi_j(t,c) \).

\[
s_x(t+1) = s_x(t) \quad \wedge \quad s_y(t+1) = s_y(t) \quad \text{if} \quad t \neq \lambda \cdot t_{\text{step}} \tag{eq. 6}
\]
\[
s_x(t+1) = s_x(t) - \Delta s_x \quad \wedge \quad s_y(t+1) = s_y(t) - \Delta s_y \quad \text{if} \quad t = \lambda \cdot t_{\text{step}}, \lambda \in \mathbb{N}
\]

Before training can be started, the values of the following parameters have to be specified: initial learning rate, \( \eta(t = 1) \), learning rate adaptation factor, \( \alpha \), initial learning spans, \( s_x(t = 1) \) and \( s_y(t = 1) \), learning span steps, \( \Delta s_x \) and \( \Delta s_y \), and the number of cycles with constant rate and span, \( t_{\text{step}} \). In the course of training, the area where neuron weights are adapted decreases as shown in Figure 2.
3.2 Counterpropagation Networks: Supervised Learning

Kohonen’s concept of self-organizing feature maps can also be applied to modeling problems where a quantitative relationship between independent and dependent variables is sought. The weight vector $W$ of the network which corresponds to $m$-dimensional patterns is expanded by an additional $m_{\text{out}}$-dimensional vector describing properties associated with the patterns (see Figure 2). In the training phase, the network is trained with the $m + m_{\text{out}}$-dimensional vectors. Only the first $m$ entries of the network and the patterns are used in the distance calculation to determine the “winning neurons”, whereas all weights $m + m_{\text{out}}$ are adapted. The trained network can be used to predict unknown property vectors. In the prediction phase, the “winning neurons” of the $m$-dimensional patterns are sought. The prediction of property vectors to these patterns is performed by returning the $m_{\text{out}}$-dimensional weight extension of the “winning neurons”.

3.3 Network Topologies

SONNIA distinguishes two different network topologies, a rectangular and a toroidal one. In the case of a rectangular network, the edge and corner neurons form the boundary of the network. Therefore, a corner has three neurons as neighbors, an edge neuron five other neurons. In toroidal networks, the neurons of opposite edges are considered to be neighbored. Then, each neuron is surrounded by eight other neurons (Figure 3).
3.4 Generation of Kohonen Maps

A Kohonen map of a given data set can be created from a trained network by finding the "winning neuron" for each pattern. As mentioned before, this mapping is a projection of data defined in a high-dimensional space into a two-dimensional rectangular plane. The map is colored by considering the occupation of neurons by data patterns using a color palette of up to ten intervals (classes).
SONNIA provides eight different methods for deriving Kohonen maps which are explained in chapter 5.5 (p. 23).
4 A Typical Application of SONNIA

The aim of this chapter is to give some examples how to work with the graphical user interface (GUI) of SONNIA. A dataset of 31 steroids binding to the corticosteroid binding globulin (CBG) receptor is investigated. This dataset was first compiled by Cramer et al. (Cramer, R.D., III; Patterson, D.E.; Bunce, J.D. J. Am. Chem. Soc. 1988, 110, 5959-5967.) for the presentation of the CoMFA method. Richards et al. (Good, A.C.; So, S.; Richards, W.G. J. Med. Chem. 1993, 36, 433-438.) used the same data for similarity calculations. Comparison of these printed and previously available computer-coded versions of this dataset showed several errors in structure coding. The dataset was therefore carefully recompiled by returning to the original literature [a) Dunn, J.F.; Nisula, B.C.; Rodbard, D. J. Crin. Endocrin. Metab. 1981, 53, 58-68. b) Mickelson, K.E.; Forsthoefel, J.; Westphal, U. Biochemistry 1981, 20, 6211-6218.].

This dataset was used in our group to model the corticosteroid binding globulin receptor activity with autocorrelation of molecular surface properties and neural networks (Wagner, M.; Sadowski, J.; Gasteiger, J. J. Am. Chem. Soc. 1995, 117, 7769-7775.). The 3D coordinates were calculated by the 3D structure generator CORINA (Sadowski, J.; Gasteiger, J.; Klebe, G. J. Chem. Inf. Comput. Sci. 1994, 34, 1000-1008.). The dataset has been extensively studied by different methods in several research groups. A stepwise description of an investigation of this CBG dataset follows.
Step 1: Open GUI

After the successful installation of SONNIA (see chapter 9, page 43) the graphical user interface of SONNIA is started with the command `cskm` and the main window of SONNIA’s Graphical User Interface appears on the screen (Figure 1).

Figure 1: Main window of SONNIA’s Graphical User Interface
Step 2: Read Datafile

To read in a datafile move the mouse cursor to the entry *File* in the menubar and press the left mouse button. Choose *Read...* from the drop down menu (Figure 2 top). After pressing the left mouse button on the button *Read...* highlighted in yellow the "SONNIA Read" dialog box is displayed (Figure 2 bottom).

Select "Data File" from the *Object: listbox, browse through the directory trees into the directory containing the file *steroids_s.dat* (RDF Code with 128 components for each compound) delivered with the SONNIA distribution, activate it by a single click with the left mouse button, and open it by pressing the *OK* button. A print-out of
an example data file is given in Chapter 8 (p. 41).

**Step 3: Create Network**

Open the "SONNIA Network" window (Figure 4) by pressing Network in the menubar and selecting Create... from the drop down menu to set up the architecture, learning strategy and initialization (Figure 3).

Figure 3: Network menu

In this example the Kohonen algorithm (field Algorithm) and a toroidal topology (field Topology) will be used. These are also the default settings in the "SONNIA Network" window (Figure 4). In the Neuron Dimensions field the length of the input vector is determined by the value given in the dark grey field behind Input:. Change the content of this field to 128. Proceed with editing the values given for Width: and Height: in the field Network Size and set them to 5 resp. 4. Finally modify the initialization method to gauss in the field Initialization by clicking on the button and selecting gauss from the list appearing below that button. Leave this dialog box by pressing the Create button.

Figure 4: Define parameters for the network architecture, learning strategy and initialization
Step 4: Train Network

The dialog box “SONNIA Training” can be accessed from the Network menu by the item Train.... This dialog box allows the user to control the parameters for the training of the neural network. In the upper part Control Flags activate the entries Dyn. Learning Rate, Dyn. Learning Span, and Error Check (in this order). The activated items are marked with a small yellow square.

Continue with the input of the values for Cycles:, Span(x):, Span(y):, Rate:, Intervall:, Step(x):, Step(y):, and Rate Factor: in the Parameters part as given in Figure 5.

Start the training of the neural network by pressing the Train button. The ”SONNIA Monitor” window appears showing the progress of the dynamic error as a function of the number of training cycles (Figure 6).

Figure 5: Set-up parameters for the training of the neural network

Figure 6: The ”SONNIA Monitor” window showing the change of the dynamic error during the training phase
Step 5: Create a Feature Map

The visualization of a trained network is supported by the Maps menu. Select Create in the Maps menu to create a map showing the occupancy of the neurons. Choose occupancy from the pop-up window shown in the upper part of Figure 7. Then, the map will appear as illustrated in the bottom part of Figure 7. In this example the red colored neurons are occupied by a single compound, the orange ones by two and so on.

Figure 7: Creating a map showing the occupancy of the neurons
Step 6: Visualize Structures

In order to visualize structures a structure file has to be read in first. The procedure is the same as described for the datafile in Step 2 on page 11 except for the following differences: Select "Structure File" from the Object: listbox and read in the file `steroids_s.ctx`. Clicking on a neuron with the left mouse button and pressing the control key simultaneously causes this neuron to be selected. It is possible to select more than one neuron. This can be done by the subsequent selection of single neurons or by moving the mouse and spanning a rectangle while keeping both the left mouse button and the control key pressed. In Figure 8 a) the neuron in the bottom left corner has been selected.

![Figure 8: Visualization of the contents of a marked neuron in the CACTVS Structure Browser](image)
Pressing the right mouse button on the map causes the map context menu shown in Figure 8 b) to appear. Move the mouse cursor to the entry *Export Structures...* in the map context menu and release the right mouse button. Then the contents of the marked neurons are displayed in the CACTVS Structure Browser [Figure 8 c)]. A detailed description of the features provided by the SONNIA GUI follows in Chapter 5.
5 GUI - Graphical User Interface

SONNIA’s Graphical User Interface has been designed for interactively generating Kohonen neural networks. It allows the visualisation of Kohonen maps. Additionally, various kinds of manipulations can be performed on the displayed maps.

5.1 Starting the SONNIA GUI

The SONNIA GUI is started from a UNIX shell by calling the shell script named `cskm`. The location of `cskm` depends on the specific installation of SONNIA. Usually, `cskm` is located in the directory `/usr/local/bin`.

5.2 The Main Window of SONNIA

After executing `cskm` the main window shown in Figure 1 appears. It is divided into four different parts:

1. Main menu: This menu bar controls the operations performed by SONNIA.
2. Palette window: It contains a palette which allows the interpretation of the colors in the displayed maps. The palette is automatically updated for the map where the mouse cursor is positioned.
3. Area of maps: The generated Kohonen maps are displayed here. The number of maps shown can be adjusted by the user.
4. Message window: Some operations, e.g., the loading of a data file, return a message displayed in this window. Error messages are also displayed here and are colored in red.

A schematic overview of the menu follows:

<table>
<thead>
<tr>
<th>File</th>
<th>Network</th>
<th>Maps</th>
<th>Options</th>
<th>Help</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read...</td>
<td>Create..</td>
<td>Create</td>
<td>Window Layout..</td>
<td>About...</td>
</tr>
<tr>
<td>Write...</td>
<td>Train..</td>
<td>Delete</td>
<td>Reference by Records</td>
<td>Reference by Ident</td>
</tr>
<tr>
<td>Quit</td>
<td></td>
<td>Palette Editor..</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The item Create in the Maps menu allows to choose one of the color modes average output, average output (conflicts), minimum output, maximum output, centroids output, most frequent output, occupancy, projection distortion, average neuron-input distance, neuron-input variance, and “smoothness”.
Selecting Delete in the Maps menu opens a pop up menu with the two alternatives of deleting either the marked or all maps currently displayed.
5.3 File Menu

The input and output of SONNIA files is controlled via the File menu shown in Figure 2. First, the user has to choose whether a file should be read or written. The File menu also offers the possibility to quit SONNIA.

Depending on the choice made in the File menu a file dialog box, either "SONNIA Read" or "SONNIA Write", appears. These windows differ only in the data object that can be transferred. "SONNIA Read" is shown in Figure 3.
With the "SONNIA Read" option, files of chemical structures or of chemical reactions can be specified. Furthermore networks, maps, and data files can be loaded. By activating "SONNIA Write", objects can be saved. These objects may be networks, maps, predictions, or images. If the user activates the marked button next to the "Object:" listbox only images or maps of selected Kohonen maps are written. Images are saved as GIF files. Two image files are generated for each map, one for the map and one for the color palette. They are automatically numbered by inserting ".map1", ".map2", ".pal1", ".pal2", etc. in the specified file name.

Both dialog boxes "SONNIA Read" and "SONNIA Write" allow the user to browse through his/her directory trees. A file is specified either by clicking on a name appearing in the "File:" listbox on the left hand side or by entering its name in the "Files:" field. Pressing the OK button starts the file operation.
5.4 Network Menu

All parameters influencing network generation or training can be set by using the Network menu (Figure 4).

Figure 4: Network menu

If the user selects Create... the "SONNIA Network" window shown in Figure 5 appears. The architecture of the Kohonen network can be selected with the options contained in this window.

Figure 5: Determination of network architecture with the "SONNIA Network" window

The "SONNIA network" window consists of five different sections:

1. "Algorithm"
   With Kohonen a network for mapping purposes (unsupervised learning) is created. If the aim is to predict a certain property vector (supervised learning) Counterprop. has to be selected to start a Counterpropagation network.

2. "Topology"
   The user has to choose one of the two possible network topologies: toroidal or rectangular.

3. "Neuron Dimensions"
   The length of the vectors that are used for the training of the network has to be specified in the Input box.
When Counterprop. is selected from the Algorithm section the Output box gets active. The length of the predicted property vector has then to be entered here.

4. "Network Size"

Width and Height determine the size of the Kohonen network (see Figure 1). By clicking the PCA button, a principle component analysis (PCA) is started to determine the maximum variance of the data by rotating the coordinate system. The distances of the coordinates that span the largest range in this rotated coordinate system are used to determine the ratio between Width and Height of the network. It is also recommendable to set a value for the ratio of neurons to patterns (NP-ratio).

5. "Initialization"

One of the three initialization modes random, gauss, or torus has to be chosen.
random: The weights of the Kohonen network are initialized with equally distributed pseudo random numbers from [-1.0, 1.0].
gauss: For each input dimension the mean and the standard deviation of the data set currently loaded are determined. Using these parameters, the Kohonen network is initialized layer by layer with normally distributed pseudo random numbers.
torus: The weights of the network are initialized in a way that they would form a torus in 3D space, whose size depends on the data set loaded. This initialization should not be mixed up with the network topology described in point 2. It requires a data set having an input dimension of three.

The internal random number generator is initialized using the seed value.

The Kohonen network is generated by clicking the Apply Button.

After selecting Train in the Network menu, the "SONNIA Training" window pops up, which allows one to set the training parameters. As shown in Figure 6, the "SONNIA Training" window consists of two sections, one for the "Control Flags" and the other one for the "Parameters".

![SONNIA Training Window](image)

Figure 6: The "SONNIA Training" window allows the specification of training parameters
1. "Control Flags"
   By setting or unsetting the buttons Dyn. Learning Rate and Dyn. Learning Span the user can choose which learning parameters are adapted during training. If at least one of these buttons is set, it is additionally possible to set the Error Check and/or Stop Thresholds buttons (see Parameters).

2. "Parameters"
   The parameters Cycles \([t_{\text{max}}]\), \(\text{Span}(x)\) \([s_x(t = 0)]\), \(\text{Span}(y)\) \([s_y(t = 0)]\), and Rate \([\eta(t = 0)]\) always have to be specified. Depending on the selected control flags some additional entry boxes are activated: Interval \([t_{\text{step}}]\), Step\(x\) \([\Delta s_x]\), Step\(y\) \([\Delta s_y]\), Rate Factor \([\alpha]\), Stop Span \([s_{\text{stop}}]\), and Stop Rate \([\eta_{\text{stop}}]\). Within the interval of \(t_{\text{step}}\) cycles a dynamic error value is calculated by summing the squared distances between the winning neuron’s weight vector and the pattern and then normalizing the value by \(t_{\text{step}}\) thus obtained. If the Error Check button is set, the learning parameter is adapted only if the error becomes larger than in the previous \(t_{\text{step}}\) cycles. Using the Stop Thresholds option, the training can be aborted without performing all \(t_{\text{max}}\) cycles. This happens, if the span values \(s_x(t)\) and \(s_y(t)\) are smaller than \(s_{\text{stop}}\) and/or the learning rate \(\eta(t)\) is smaller than \(\eta_{\text{stop}}\).

Training is started by pressing the Apply button. By default, a "SONNIA Monitor" (Figure 7) window pops up showing the development of the dynamic error during training defined above.

![SONNIA Monitor](image)

Figure 7: The "SONNIA Monitor" window shows the course of the dynamic error during the training phase

After training has been started the Apply button of the "SONNIA Training" window turns into a red Stop button which allows the user to interrupt the training at any time (Figure 8).

![SONNIA Stop](image)

Figure 8: Training can be aborted by pressing the red Stop button
5.5 Maps Menu

The Maps menu shown in Figure 9 controls the visualization of Kohonen maps. A map can be generated by pressing the desired color mode from the Create pop up menu.

Figure 9: The Maps menu

Eight different color modes are available:

- **average output**: Neurons are indexed according to the average feature value of the patterns contained in them.
- **average output (conflicts)**: In contrast to the previous mode, neurons containing patterns with feature values of different intervals (or classes) are marked to be conflict neurons.
- **minimum/maximum output**: Neurons are indexed according to the extreme property values of patterns.
- **centroids output**: The pattern most similar to the weight vector determines the index of the neuron.
- **most frequent output**: Neurons are indexed with the most frequent interval occurring among the patterns.
- **occupancy**: Neurons are indexed according to the number of patterns contained in them.
- **projection distortion**: For each pattern occupying the neuron under consideration the $N$-th most similar patterns of the data set are determined and the “winning neurons” to those patterns are found. By summing the topological distances between the considered neuron and all other neurons a value $f$ is calculated by eq. 1.

$$f = \frac{1}{N \cdot n_p} \sum_{j}^{N \cdot n_p} \max \left( d_x(j)(c), d_y(j)(c) \right)$$  \hspace{1cm} (eq. 1)

The value of $f$ is a measure of the distortion caused by the projection from a high-dimensional space into a two-dimensional space. The index $j$ covers all $n_p$ next patterns in the data set.
- **average neuron-input distance**: The neurons are indexed via the average distance between the pattern vectors and weight vector.
- **neuron-input variance**: The neuron index indicates the variance of the distances between the pattern vectors and weight vector.
- **smoothness**: The neuron indices are determined by the average distance between the weight vectors of the considered neuron and its direct neighbors by

\[
g = \frac{1}{n} \sum_{j}^{N} |W_c - W_j|
\]  

(eq. 2)

where \( n \) is the number of neurons in the first neighborhood. It should be noted that \( n \) depends on the chosen network topology.

To use the *projection distortion* mode, the user has to set the number of considered neighbors in an additional window.

Each displayed Kohonen map is centered in one of the frames that build the area of maps (Figures 1 and 10). A single map can be marked by double-clicking its frame. A green frame around a map indicates the selection of maps.

Maps can be removed by pressing the *Delete* button. The user can either delete all maps or only marked maps.

If the mouse cursor is moved inside a frame which contains a Kohonen map, a small window pops up describing how the map has been created (Figure 10).

![Image](image.png)

Figure 10: Pop up window with description of the map
The "SONNIA Palette Editor" shown in Figure 11 is started by choosing Palette Editor... from the Maps menu. The palette parameters of the Kohonen map to be generated, can be set in the "Parameters" part of the "SONNIA Palette Editor". A map may consist of up to ten Colors. The property which is used for coloring the map is selected in the Output box. In the case of an ordinary Kohonen mapping the first property is chosen by entering "1". In the Counterprop. mode a "1" is the first component of the patterns the network is trained with. Hence maps can also be colored by elements of the input vectors. The output field begins with the entry m+1 (see Figure 2, page 4).

![Figure 11: The "SONNIA Palette Editor"](image1)

Usually, the neurons are colored according to their occupancy. This method is used if the option normal is selected in the Projection field. A second way of coloring is performed if Projection is set to reverse. In this case the "winning pattern" is determined for each neuron of the network. Its property values are then used for coloring the map. The setting of Intervals determines how the interval borders of the color palette are calculated. With normal, the resulting intervals are equidistant. Selecting adapted, the borders are calculated in a way, that the same number of patterns currently loaded is contained in the different intervals.

Clicking on one of the palette colors displayed in the "Palette" part of the window causes the "SONNIA Color Editor" to pop up, which allows one to change colors (Figure 12). RGB (Red/Green/Blue) values can be defined by entering values in the fields Red, Green, and Blue or by moving the corresponding sliders. In addition, the user can select a new color from the RGB palette displayed in the upper left corner.

![Figure 12: The "SONNIA Color Editor"](image2)
5.6 Map Context Menu

Several operations can be performed on a displayed Kohonen map. If the map has been generated from a network with toroidal topology the shifting of the map is allowed. When a neuron of a map is clicked with the left mouse button, a crosshair appears. This neuron is the starting point. The crosshair can be moved to all other neurons by keeping the left button pressed. After releasing the button, the map is shifted by moving the starting point of the map to the last position of the crosshair.

The upper six entries of the context menu depicted in Figure 13 allow one to zoom, rotate, or mirror a map. Clicking on neurons and pressing the control key simultaneously causes neurons to be selected or unselected. Several neurons can be selected or unselected by keeping the buttons pressed and spanning a rectangle with the mouse. The radio buttons Select Neurons and Unselect Neurons control whether neurons are selected or unselected.

![Image of context menu]

Figure 13: The map context window allows several operations on displayed maps
The "SONNIA Tiling" window is started by clicking *Tile...*. If the network of the current map is based on a toroidal topology, the map is displayed in a tiled manner by putting several identical maps together like tiles on a wall (Figure 14). The number of tiles is automatically adapted to a change of the window size. Only one Kohonen map is shown for rectangular networks. Palette and tiled maps can be saved as GIF images.

Figure 14: The "SONNIA Tiling" window
Chemical structures and reactions can be viewed in the CACTVS browser. Two possibilities for displaying these objects have been implemented into the graphical user interface of SONNIA. Using the entry Export Structures... of the Context Menu, a CACTVS browser is started and the contents of all marked neurons is displayed (Figure 15), if a reference structure or reaction file has already been specified via the File menu (Read... structure file).

Figure 15: The contents of the marked neurons of the left map is displayed in the CACTVS Structure Browser
Centroids - structures or reactions whose pattern vector is most similar to a weight vector - can be visualised by Export Centroids... The objects are displayed depending on the orientation of the current map as shown in Figure 16.

Figure 16: Centroids can be displayed in the CACTVS Structure Browser

Delete Map removes the current Kohonen map from the maps area.
5.7 Options Menu

Some general SONNIA settings can be made within the Options menu of the Main Window of SONNIA shown in Figure 17.

![Options menu]

Figure 17: The Options menu

The "SONNIA Layout" window, obtained by pressing Window Layout..., controls the appearance of the area of maps (Figure 18). The Size box contains the width of the squared map frames. Additionally, the number of Visible Rows and Visible Columns as well as the number of Background Rows and Background Columns can be modified.

![SONNIA Layout window]

Figure 18: The "SONNIA Layout" window

The appearance of the "SONNIA Monitor" window (Figure 7) can be suppressed by deactivating the Control Window button.
6 Running SONNIA in Batch Mode (Reference Manual)

SONNIA can also be used in batch mode. The batch mode is extremely useful when large datasets should be analyzed or a sequence of many networks with, for example, different network or training parameters for the same dataset should be trained.

SONNIA has been designed as a Tcl-library loadable to the CACTVS system (http://www2.ccc.uni-erlangen.de/software/cactvs), and, therefore, all SONNIA commands can easily be integrated into a Tcl-script. The full Tcl functionality like loops, conditions etc. is provided.

The Gd package which is part of the CACTVS system allows the user to generate GIF images of Kohonen maps.

6.1 Starting the CACTVS tclserver

The tclserver is started via the script named `csts`; this opens a tclserver shell. A Tcl script containing SONNIA commands can be called with the Tcl command `source scriptname`. Alternatively, the SONNIA script can be started directly by `csts -f scriptname`.

It should be mentioned that the CACTVS tclserver also allows an interactive mode for using SONNIA. In the tclserver shell, one can carry out single commands and wait for the answer of the server.

6.2 Loading CACTVS Packages

The entire CACTVS system comprises a collection of different dynamic libraries. SONNIA has been designed as an additionally loadable library. Using the standard installation, no specification has to be made, where the SONNIA library is located. If the library has been moved to a different directory the CACTVS variable `cactvs(cmdxpath)` has to be appended. This can be achieved with the Tcl command `lappend cactvs(cmdxpath) directory`.

The SONNIA library, as any other CACTVS library, is loaded to the CACTVS system by `cmdx load library` here: `cmdx load kmap`.

6.3 SONNIA Commands

The commands implemented in the SONNIA library satisfy the general Tcl syntax requirements.

SONNIA commands can be divided into three different groups according to the objects they handle. The first group of commands deals with the patterns. The second group controls the network. With the commands belonging to that group, network parameters like size or topology of the network and training parameters are set. With the third group of commands, Kohonen maps are generated.
The following conventions are used throughout this reference manual: Commands and options are written in **bold face** whereas variables of argument parameters are in *italics*. Furthermore, the following special characters are used:

?? Arguments enclosed in question marks are optional

... Ellipsis. Several values may be provided for the previous arguments, or the previous argument can be specified multiple times, for example "map..."

| Separator. Only one of the arguments separated by this character can be specified. Braces enclose the alternatives.

### 6.3.1 Handling of Patterns

**kmap data destroy**
The current data set is deleted.

**kmap data read** *filename*
A new data set is read from the file *filename* which must have the pattern format explained in section 8.1. The number of correctly loaded patterns is returned. It is recommended to check this number after reading a data set in batch mode.

**kmap data write** *filename*
The data set read by **kmap data read** is written in tabular form to the file *filename*.

### 6.3.2 Handling of Kohonen Networks

**kmap net autocreate** *neuron_dim* *np_ratio* ?*xy_ratio*?
The size of a Kohonen network is calculated, *width* and *height* are returned. The values are determined, depending on the data currently loaded. The variable *neuron_dim* specifies the number of input dimensions. *np_ratio* gives the ratio between the number of patterns in the data set and the number of neurons in the Kohonen neural network. With the third optional parameter *xy_ratio*, the ratio between returned *width* and *height* is controlled. If a real value is given to this parameter *width* and *height* are calculated according to the number of neurons of the network resulting from *np_ratio*. If no value is specified for *xy_ratio* a principal component analysis (PCA) of the loaded patterns is performed. During the PCA, the coordinate system is rotated which results in maximum variance of the data. The distances of the coordinates that span the largest range in this rotated coordinate system are used to determine the ratio between *width* and *height* of the network.

**kmap net create** *neuron_dim* *width* *height*
A Kohonen network of *height* neurons vertically, and *width* neurons horizontally is created. Each neuron of this network is represented by a *neuron_dim*-dimensional vector.

**kmap net destroy**
The current network is deleted.

**kmap net initialize** *mode* ?*seed*?
The Kohonen network has to be initialized before training. This is done by the **kmap net initialize** command. Three different initialization methods are implemented which can be chosen by *mode*:
**random**: The weights of the Kohonen network are initialized with equally distributed pseudo random numbers from \([-1.0, 1.0]\).

**gauss**: For each input dimension the mean and the standard deviation of the data set currently loaded is determined. Using these parameters, the Kohonen network is initialized layer by layer with pseudo random numbers having normal distribution.

**torus**: The weights of the network are initialized in a way that they form a torus in 3D space. The size of this torus depends on the data set loaded. This method requires a data set having an input dimension of three.

If the `kmap net initialize` command is omitted the network is initialized in `random` mode by default. With the second parameter `seed` the random number generator itself can be initialized. If `seed` has the value zero, the current sequence of random numbers is used. Otherwise the seed value is used for starting the generation of a new sequence of random numbers.

**kmap net train cycles**
The network created with `kmap net create` and possibly initialized with `kmap net initialize` is trained with the data loaded by `kmap data read`. During training the patterns of the data set are presented to the Kohonen network in a random sequence `cycles` times, but only exactly once in one epoch of training. The parameters controlling the adaptation of the network weights can be specified using the `kmap net set` command (see below).

**kmap net smooth ?cycles?**
The weights of the Kohonen network can be smoothed. This is done by replacing the current weight of the neuron under consideration by an average weight value calculated from the weight itself and its four orthogonal neighbors. This procedure is performed `cycles` times, but at least once.

**kmap net read filename**
Size and weights of a Kohonen network are read from file `filename`. If the Kohonen network was designed as a pseudo "counterpropagation" network applied to modeling problems the string "counterpropagation" followed by the number of input dimensions of the neurons is returned. Otherwise one obtains the string "kohonen" and the total number of weights of the neurons (**kmap net usex** and **kmap net predict**).

**kmap net write filename**
The definition and all weights of the Kohonen network are written to the file `filename`. If a data set is loaded, the mapping of the patterns onto the neurons is calculated and is also written into `filename`.

**kmap net set parameter ?value ...?**
With `kmap net set` all parameters concerning the configuration of the Kohonen network and the training course can be specified. For example the command `kmap net set topology t` determines the topology of the network.

<table>
<thead>
<tr>
<th>parameter ?value ...?</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>indim in_dim</code></td>
<td><code>in_dim</code> gives the dimension of the data patterns. Normally, this parameter needs not to be specified since it is automatically set to the dimension of a neuron.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Value ...?</td>
</tr>
<tr>
<td>-----------</td>
<td>------------</td>
</tr>
<tr>
<td>outdim out_dim</td>
<td></td>
</tr>
<tr>
<td>initrate rate0</td>
<td></td>
</tr>
<tr>
<td>initspan xspan0 yspan0 (rel</td>
<td>abs)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>ratefact rate_factor</td>
<td></td>
</tr>
<tr>
<td>spanstep dspanx dspany (rel</td>
<td>abs)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>stoprate stop_rate</td>
<td></td>
</tr>
<tr>
<td>stopspan stop_span</td>
<td></td>
</tr>
<tr>
<td>control cycles flags</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
continuation of: **control cycles flags**

<table>
<thead>
<tr>
<th>parameter</th>
<th>value</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td></td>
<td>d: During cycles training iterations, the distances between patterns and their winning neurons is summed. If the obtained value is smaller than the value of the previous cycles training steps the parameters chosen by s and r are decreased. The learning rate is multiplied by rate_factor and the learning span is decreased by dspanx and dspany. If flag d is not set, the chosen learning parameters are changed every cycles steps.</td>
</tr>
<tr>
<td>topology</td>
<td>top_type</td>
<td>The topology of the Kohonen network is determined by top_type. If top_type is r, a rectangular topology is chosen. With a top_type of t, the topology is toroidal. The default is t.</td>
</tr>
</tbody>
</table>

**kmap net get parameter**
The current values of the network parameters **indim**, outdim, xdim, ydim, zdim, topology, control, initrate, ratefact, initspan, spanstep, stoprate, and stopspan are returned.

**kmap net usex indices**
As introduced in section 3.2, Kohonen networks using the counterpropagation algorithm can also be applied to modeling problems, where certain properties represented by vectors with real numbers should be predicted. In this case, the neurons of the Kohonen network consist of the pattern describing the object and the associated property vector. Correspondingly the total neuron dimension neuron_dim (**kmap net create**) has to be given by summing the dimensions of pattern and property vectors. During training, only the first indices layers of the Kohonen network are used to calculate the distance between a pattern and the weights of a neuron. The remaining weights which are not considered in the determination of the winning neuron, can be interpreted as predictions for the property vectors (**kmap net predict**).

**kmap net predict filename**
Normally, all weights of a neuron are used to find the neuron that is most similar to the pattern under consideration, also called the "winning neuron". If only the first part of the weight vectors is used in this distance calculation (**kmap net usex**), the remaining second part can be interpreted as a prediction for the corresponding part of the data pattern.

With the command **kmap net predict**, a prediction file **filename** is produced. Its format has already been described in section 8.2.

### 6.3.3 Handling of Kohonen Feature Maps

Within the SONNIA Tcl library, access to Kohonen maps is realised via handles. Whenever a new map is created it is associated with a handle, a string, e.g., "map0", which allows the correct identification of a specific map. This handle concept allows to deal with several Kohonen maps of the present network simultaneously. Commands creating new map handles are **kmap map create** and **kmap map read**.

If the CACTVS Gd library has been loaded to the tclserver before generating Kohonen maps, each map is additionally supplied with a Gd handle. This handle refers to a data structure representing the Kohonen map as a GIF image with one pixel per neuron. Each operation, like rotating or mirroring, is only performed on the Gd data structure. With the commands explained here and the Gd commands it is possible to write Tcl/Tk scripts having a graphical output. In that case, it is required that the Tk image of a Kohonen map has the same name as the
corresponding map handle. The best example of an application using this feasibility is the graphical user interface of SONNIA.

\texttt{kmap map create}\ colormode

For Kohonen network and the data set present the mapping of patterns onto neurons is calculated. Using this projection an output value is determined for each neuron according to the setting of \texttt{colormode} and \texttt{outindex} (\texttt{kmap net set}). Based on this output a color palette is created to set up the Gd data structure. If the creation of a map has been successful its map handle is returned.

The following modes for \texttt{colormode} are allowed for coloring Kohonen networks:

<table>
<thead>
<tr>
<th>colormode</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>average output value of patterns assigned to the neuron</td>
</tr>
<tr>
<td>m</td>
<td>minimum output value of patterns assigned to the neuron</td>
</tr>
<tr>
<td>M</td>
<td>maximum output value of patterns assigned to the neuron</td>
</tr>
<tr>
<td>x</td>
<td>average output value, but conflicts between different classes are highlighted</td>
</tr>
<tr>
<td>s</td>
<td>output value of the pattern most similar to weight vector of the neuron</td>
</tr>
<tr>
<td>f</td>
<td>most frequent color index calculated from output values of patterns assigned to the neuron</td>
</tr>
<tr>
<td>n</td>
<td>number of patterns assigned to the neuron</td>
</tr>
<tr>
<td>d</td>
<td>projection distortion at the neuron (section 5.5)</td>
</tr>
<tr>
<td>q</td>
<td>average distance between of pattern vectors and weight vector of the neuron</td>
</tr>
<tr>
<td>e</td>
<td>variance of the distances between pattern vectors and weight vector of the neuron</td>
</tr>
<tr>
<td>g</td>
<td>smoothness (section 5.5)</td>
</tr>
</tbody>
</table>

\texttt{kmap map list}\ (maphandle | selected | all)

A list of map handles is returned satisfying the condition set up by the last argument. If the last argument is \texttt{all}, the list contains map handles of each Kohonen map present. If the \texttt{selected} mode is chosen, only map handles of maps selected by \texttt{kmap map select} are returned. Furthermore, only a certain map handle can be specified via the argument \texttt{maphandle}. If a map belonging to this handle is currently present \texttt{maphandle} is returned, otherwise the output is an empty string.

\texttt{kmap map destroy}\ maphandle ...

All Kohonen maps listed in this command are deleted.

\texttt{kmap map select}\ (maphandle | all)

The Kohonen map specified by \texttt{maphandle} is selected. If \texttt{all} is used instead of a certain map handle, each map currently present is selected. Selected maps can be listed with \texttt{kmap map list}.

\texttt{kmap map unselect}\ (maphandle | all)

The Kohonen map specified by \texttt{maphandle} gets unselected. If \texttt{all} is used, each map currently selected is removed (\texttt{kmap map list}).

\texttt{kmap map write}\ maphandle ... \ filename

All Kohonen maps characterised by the listed map handles (\texttt{maphandle} ...) are written to an ASCII file
filename. Since operations performed on maps like scrolling or shifting manipulate the Gd data structure only, maps are saved in their original orientation matching the Kohonen network.

**kmap map read filename**
Maps are read from the file filename created with **kmap map write**. Each map read is labelled with an explicit map handle, e.g. "map0". A list of handles of all maps included in filename is returned.

**kmap map set parameter value**
The parameter defined here control how the next Kohonen maps will be created. The following parameters can be applied:

<table>
<thead>
<tr>
<th>parameter value</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>color palettemode ncolors</td>
<td>The palette of Kohonen maps will consist of ncolors colors indicating different classes in the loaded data set. Depending on the selected output property (<strong>kmap map set outindex</strong>) the assignment of patterns into classes can be achieved in two ways. If palettemode is set to 0 the classes are defined by dividing the interval between minimum and maximum value of the output property into ncolors intervals of same size. If palettemode is 1 the borders of classes are calculated in a way that all ncolors intervals approximately cover the same number of patterns.</td>
</tr>
<tr>
<td>projection (n</td>
<td>r)</td>
</tr>
<tr>
<td>outindex colorindex</td>
<td>A pattern in SONNIA consists of an input part which is used for training and an output part containing properties which can be used for coloring the Kohonen map. The integer value of colorindex species which property is used. The smallest allowed number is 1.</td>
</tr>
</tbody>
</table>

**kmap map show parameter**
Unlike **kmap map get** this command shows global parameters which do not refer to a certain Kohonen map. The supported parameters are: **color**, **outindex**, and **projection**.

**kmap map zoom maphandle deltaxzoom**
The size of the Kohonen map maphandle displayed by the CACTVS tkserver is changed depending on deltaxzoom. Each map has its own zoom factor. A zoom factor of one means that a neuron of a map is visualised by one pixel. The new zoom factor is calculated by adding deltaxzoom. Depending on the sign of deltaxzoom the map is increased or decreased.

**kmap map scroll maphandle hshift vshift**
Kohonen maps of networks having toroidal topology can be shifted horizontally and vertically. Besides the map handle maphandle the number of neurons the map should be shifted in horizontal hshift and vertical direction vshift have to be specified. If a Tk image maphandle exists it is updated automatically.
**kmap map mirror** *maphandle* (*v* | *h*)
With *v* the Tk image *maphandle* is flipped vertically, with *h* horizontally. This is only possible when the Gd library is loaded.

**kmap map rotate** *maphandle* (*l* | *r*)
With *r* the Tk image *maphandle* is turned clockwise, with *l* counter-clockwise. This is only possible when the Gd library is loaded.

**kmap map get** *maphandle* parameter
Parameters belonging to a certain Kohonen map *maphandle* are returned. The following parameters can be displayed:

<table>
<thead>
<tr>
<th>parameter</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>colors</em></td>
<td>number of colors of the palette used for creating the map</td>
</tr>
<tr>
<td><em>width</em></td>
<td>width of map in its current orientation</td>
</tr>
<tr>
<td><em>height</em></td>
<td>height of map in its current orientation</td>
</tr>
<tr>
<td><em>topology</em></td>
<td>topology of the Kohonen network the map is based on</td>
</tr>
<tr>
<td><em>property</em></td>
<td>description (string) how the map has been created by <strong>kmap map create</strong></td>
</tr>
<tr>
<td><em>conflicts</em></td>
<td>number of conflicts. A conflict is defined as the situation where two patterns belonging to different classes are assigned to one neuron.</td>
</tr>
<tr>
<td><em>occupancy</em></td>
<td>number of occupied neurons</td>
</tr>
<tr>
<td><em>data</em></td>
<td>filename (full pathname)</td>
</tr>
<tr>
<td><em>zoom</em></td>
<td>current zoom factor of the map</td>
</tr>
<tr>
<td><em>rotate</em></td>
<td>index that indicates the rotation state of the map</td>
</tr>
<tr>
<td><em>gdhandle</em></td>
<td>Gd handle of the Kohonen map image</td>
</tr>
<tr>
<td><em>gdpalette</em></td>
<td>Gd handle of the palette image of the map</td>
</tr>
<tr>
<td><em>palettetype</em></td>
<td>type of palette, equidistant (0) or adapted to data set (1)</td>
</tr>
<tr>
<td><em>palette</em></td>
<td>list of float numbers defining the palette of the map</td>
</tr>
</tbody>
</table>

**kmap map findneuron** *maphandle* *x* *y*
Different operations like rotating, shifting, and mirroring can be performed on Kohonen maps. This command converts neuron coordinates *x* and *y* of map *maphandle* into the neuron coordinates of the corresponding immovable network.

**kmap map createpalette** *maphandle*
For a given Kohonen map *maphandle* this command creates a Gd image of the palette belonging to this map. Its Gd handle is available with **kmap map get** *maphandle* *gdhandle*.

**kmap map getcontents** *maphandle* (*record* | *label*) (*selected* | (*x1* *y1* *x2* *y2*))
After a Kohonen network has been trained, each pattern of a data set is assigned to a certain neuron. The so-called “winning neuron” is that neuron of the network which is most similar to the input pattern. In this way, each neuron can be occupied by several patterns. The contents of neurons can be returned using this command. The 5\textsuperscript{th} argument specifies the output mode: Using *record* a list of record numbers of the patterns...
in the data file is created. With **label** this list contains the labels of patterns instead of record numbers. The 6th argument determines the neurons, the patterns are returned for: One can either choose the selected neurons (**kmap map selectneuron**) with **selected** or specify a rectangular region by the coordinates \(x_1, y_1, x_2, y_2\), and \(y_2\). If a Kohonen map has already been shifted, mirrored, etc. \(x_1, y_1, x_2,\) and \(y_2\) denote the neurons in the manipulated map.

**kmap map getcentroid** \(\text{maphandle} (\text{record} | \text{label}) x_1 y_1 x_2 y_2\)

During the creation of a Kohonen map the patterns of the currently loaded data set are assigned to the neurons of the network. This command returns a list of centroids. From all patterns falling in a single neuron, that pattern which is most similar to the neuron weights, is called to be its centroid. The 5th argument controls the output mode. The output is either a list of record numbers, if **record** is used or a list of pattern labels, if **label** is used. \(x_1, y_1, x_2,\) and \(y_2\) define a rectangular region of neurons in the Kohonen map for which the centroids are determined.

**kmap map changecolor** \(\text{ncolors} \text{colorindex} r g b\)

In SONNIA a palette of Kohonen maps can have up to ten colors. In the palette having \(\text{ncolors}\) colors the color with the number \(\text{colorindex}\) is changed. The new color is defined via the RGB model, where \(r\) stands for the red, \(g\) for the green, and \(b\) for the blue part.

A change in the colors of a palette effects only the map images created after this change has happend.

**kmap map getcolor** \(\text{ncolors} \text{colorindex}\)

The color with the index \(\text{colorindex}\) of the palette consisting of \(\text{ncolors}\) colors is returned as list of the red, green, and blue part of that color defined in the RGB model.

**kmap map getrgb** \(\text{maphandle} x y\)

The RGB values of a certain neurons specified by \(x\) and \(y\) in the Kohonen map \(\text{maphandle}\) are returned.

**kmap map selectneuron** \(\text{maphandle} x_1 y_1 x_2 y_2\)

The neurons of the map \(\text{maphandle}\) in the region \(x_1, y_1, x_2,\) and \(y_2\) are added to the selection (**kmap map getcontents**).

**kmap map unselectneuron** \(\text{maphandle} x_1 y_1 x_2 y_2\)

Neurons of the region defined by \(x_1, y_1, x_2,\) and \(y_2\) get unselected.
7 Example of a SONNIA Tcl Script

The following example shows a Tcl script that can be loaded into the tkserver using the Tcl command `source`. The script contains a loop over the variable $f$, which varies the size of the networks. The loop is divided into the parts `internals`, `data`, `network`, `map`, and `output`.

```tcl
# internals
lappend cactvs(cmdxpath) /home/andreast/CMDX/Linux2.0
cmdx load kmap

# data
kmap data read steroids_s.dat

set count 0
foreach f {{3 2} {6 5} {8 6}} {

# network
kmap net create 128 [lindex $f 0] [lindex $f 1]
kmap net set initrate 0.9
kmap net set initspan 2 1.5 abs
kmap net set ratefact 0.9
kmap net set spanstep 0.2 0.15 abs
kmap net train 930

# map
kmap map set color 0 3
kmap map set outindex 2
set maphandle [kmap map create f]

# output
kmap net write steroids_s$count.knet
kmap map write $maphandle steroids_s$count.map
kmap map destroy $maphandle
incr count
}
```

In the section `internals` of the Tcl script the location of the dynamic SONNIA library is specified and the library is loaded. After that, a data set of patterns of steroidal compounds is loaded. The total dimension of the pattern is 130, a 128-dimensional molecular descriptor and a two-dimensional property vector.

First, the parameters of the Kohonen net are set in the network part, then the training is performed. In the next block of commands, a Kohonen map of the second property having three colors is generated. The Tcl handle of the map is stored in the variable maphandle.

The Kohonen network and the Kohonen map are written into files and the map is finally deleted.
8 File Formats

8.1 The Data File

The data file contains the entire information about patterns in ASCII format. Usually, it has the file extension ".dat". An example of a data file is given in the next lines:

```plaintext
!!Name: steroids_s.ctx
! Input vector created by rcode
! Parameters:
! lower distance border: 0.00000000, Upper distance border 12.80000000
! input dimension: 128, non weighted
!!Property: radial distribution function A1
.... 8.830693e-02 8.825346e-02 8.786038e-02 .... -6.279 2 !aldosterone
.... 1.721332e+00 1.721456e+00 1.722374e+00 .... -5.000 3 !androstanediol
.... 1.721336e+00 1.721485e+00 1.722592e+00 .... -5.000 3 !5-androstenediol
.... -5.563186e-01 -5.564730e-01 -5.576087e-01 .... -5.763 3 !4-androstenedione
.... .... ....
```

One leading exclamation mark makes a line a comment line. Special keywords have two leading exclamation marks. The keyword !!Name: is used to name the ensemble of patterns. The second keyword !!Property: can be used to describe the patterns. In the example given, the entries one to three of the pattern vectors in a line are the cartesian coordinates followed by the electrostatic potential (ESP).

The patterns are listed after the leading comments and definitions. One line contains one pattern. The first entries form the vectors the network is trained with. They are followed by properties which are used to generate Kohonen maps or which should be predicted. Finally, each pattern can be associated with a label describing the record. The entries of pattern vectors, network input and properties, represent the only required data. All further information is optional.

SONNIA data files for modeling problems by a counterpropagation network also have the file format described here.

8.2 The Prediction File

The results of a prediction performed by SONNIA are stored in prediction files. Usually, they have the file extension ".prd". A line in these ASCII files contains information in the following order:

- coordinates of the winning neuron
- distance to the winning neuron considering only those parts of the vectors that are used in the distance calculation
- distance to the winning neuron of the remaining parts of the vectors
- weights of the "winning neuron"
- pattern vector
- label associated with the data pattern
8.3 The Network File

Information about neuron weights and training parameters is stored in a network file, which is characterized by the file extension ".knet".

The first part of this file, enclosed by the statements BEGIN PAR and END PAR, contains network and training parameters.

All network weights are stored in the second part of this file, layer by layer, from BEGIN CS to END CS. If a network is built by $x$ times $y$ neurons – each neuron having $z$ weights – this part contains $z$ blocks representing the different network layers. Each block contains $x$ lines and each line, in turn, has $y$ weight elements.

In the following example the network has the dimensions $x=5$, $y=4$, and $z=128$. The weights for two network layers are given, the remaining 126 blocks are omitted.

BEGIN PAR

dim: 128 5 4

top_type: t

counter: 20000

cooling: 2000

dnc_flags: 7

span0_x: 1.0

span_x: 0.6

span0_y: 1.0

span_y: 0.6

span_step: 0.2

rate0: 0.800

rate: 0.652

rate_fact: 0.95

min_err: 0.00

END PAR

BEGIN CS

-2.112540 -1.565817 0.150467 -0.161108

-0.207792 -0.047749 -0.065630 0.087860

-0.106258 -0.115443 -0.055201 1.719950

-0.554965 1.722430 1.722374 -0.554965

-0.557609 -1.683210 -0.200422 -0.559072

5.388159 -0.179605 -0.179605 -0.179605

-0.179605 -0.179605 -0.179605 -0.179605

-0.179605 -0.179605 -0.179605 -0.179605

-0.179605 -0.179605 -0.179605 -0.179605

-0.179605 -0.179605 -0.179605 -0.179605

...  

...  

...  

END CS
9 Program Installation

9.1 IRIX6.5

The distribution CD-ROM contains the compressed file

    cactvskmapdist-IRIX6.5-3.99.tar.gz

and the installation script

    installme.

1. Create a subdirectory, e.g. tmpdir.

2. Copy cactvskmapdist-IRIX6.5-3.99.tar.gz and installme from the mounted CDROM to the subdirectory created in the first step.

3. Start the installation script in the subdirectory tmpdir with the command ./installme.

4. Add the subdirectory with the start scripts – as determined during the execution of the installation script – to the environment variable PATH in your .login or .cshrc files.

Caution: Use SONNIA only with the CACTVS tools version which is distributed with SONNIA
A List of Figures

1 Architecture of a Kohonen neural network ........................................... 3
2 Architecture of a counterpropagation (CPG) neural network ....................... 4
1 Dependency of the learning rate, $\eta(t)$, on time, $t$, shown for two different values of $t_{\text{step}}$ .... 6
2 Dependency of $\varphi_j(t, c)$ on $d_j(t)$ in the course of training shown for $t_{\text{step}} = 20$ and $\Delta s = 1$ .... 7
3 Number of neighbors depending on the network topology .......................... 8
4 Transformation of a Torus into a plane .................................................. 8
1 Main window of SONNIA’s Graphical User Interface ................................. 10
2 Read a datafile ......................................................................................... 11
3 Network menu .......................................................................................... 12
4 Define parameters for the network architecture, learning strategy and initialization ................................................. 12
5 Set-up parameters for the training of the neural network ......................... 13
6 The ”SONNIA Monitor” window showing the change of the dynamic error during the training phase 13
7 Creating a map showing the occupancy of the neurons ............................. 14
8 Visualization of the contents of a marked neuron in the CACTVS Structure Browser ......................................................... 15
1 Main window of SONNIA’s Graphical User Interface ................................. 18
2 The SONNIA File menu ............................................................................ 18
3 The ”SONNIA Read” file dialog box .......................................................... 19
4 Network menu .......................................................................................... 20
5 Determination of network architecture with the ”SONNIA Network” window .................................................... 20
6 The ”SONNIA Training” window allows the specification of training parameters ................................................................. 21
7 The ”SONNIA Monitor” window shows the course of the dynamic error during the training phase ........................................... 22
8 Training can be aborted by pressing the red Stop button ............................ 22
9 The Maps menu ......................................................................................... 23
10 Pop up window with description of the map .............................................. 24
11 The ”SONNIA Palette Editor” ................................................................. 25
12 The ”SONNIA Color Editor” ..................................................................... 25
13 The map context window allows several operations on displayed maps .............................................................. 26
14 The ”SONNIA Tiling” window ................................................................. 27
15 The contents of the marked neurons of the left map is displayed in the CACTVS Structure Browser ................................................................. 28
16 Centroids can be displayed in the CACTVS Structure Browser ................. 29
17 The Options menu ..................................................................................... 30
18 The ”SONNIA Layout” window ............................................................... 30
B References on the Use of Kohonen Networks

Unsupervised learning

- separation of compounds of different biological activity
  The Coding of the Three-dimensional Structure of Molecules by Molecular Transforms and Its Application to Structure – Spectra Correlations and Studies of Biological Activity
  J. H. Schuur, P. Selzer, J. Gasteiger

  Locating Biologically Active Compounds in Medium-Sized Heterogeneous Datasets by Topological Autocorrelation Vectors: Dopamine and Benzodiazepine Agonists
  H. Bauknecht, A. Zell, H. Bayer, P. Levi, M. Wagener, J. Sadowski, J. Gasteiger

- comparison of chemical libraries
  Bewertung der Ähnlichkeit und Vielfalt von Verbindungsbibliotheken mit räumlichen Autokorrelationsvectoren und neuronalen Netzen
  J. Sadowski, M. Wagener, J. Gasteiger
  Assessing Similarity and Diversity of Combinatorial Libraries by Spatial Autocorrelation Functions and Neural Networks
  J. Sadowski, M. Wagener, J. Gasteiger

- analysis of the similarity and diversity of combinatorial libraries
  same reference as preceding item

- comparison of different molecular descriptors for modeling biological activity
  Autocorrelation of Molecular Surface Properties for Modeling Corticosteroid Binding Globulin and Cytosolic Ah Receptor Activity by Neural Networks
  M. Wagener, J. Sadowski, J. Gasteiger

- analysis of the results of high-throughput-screening
  Analysis of High-Throughput Screening Results by Neural Networks and Different Structure Representations
  A. Teckentrup, J. Gasteiger, H. Briem
  in preparation
- **evaluation of molecular surface properties**
  
  *Representation of Molecular Electrostatic Potentials by Topological Feature Maps*
  J. Gasteiger, C. Rudolph, J. Sadowski, J. Zupan

  *Abbildung elektrostatischer Potentiale muscarinischer und nicotinischer Agonisten mit künstlichen neuronalen Netzen*
  J. Gasteiger, X. Li

  *Mapping the Electrostatic Potential of Muscarinic and Nicotinic Agonists with Artificial Neural Networks*
  J. Gasteiger, X. Li

  *The beauty of molecular surfaces as revealed by self-organizing neural networks*
  J. Gasteiger, X. Li, A. Uschold

  *Evaluation of Molecular Surface Properties Using a Kohonen Neural Network: Studies on Structure-Activity Relationships*
  S. Anzali, G. Barnickel, M. Krug, J. Sadowski, M. Wagener, J. Gasteiger
  in: *Neural Networks in QSAR and Drug Design*  

  *The Comparison of Molecular Surfaces by Neural Networks and Its Application to Quantitative Structure Activity Studies*
  J. Polanski, J. Gasteiger, M. Wagener, J. Sadowski

- **knowledge extraction from reaction databases**
  
  *Organische Reaktionen mit Hilfe neuronaler Netze klassifiziert: Michael-Additionen, Friedel-Crafts-Alkylierungen durch Alkene und verwandte Reaktionen*
  L. Chen, J. Gasteiger

  *Organic Reactions Classified by Neural Networks: Michael Additions, Friedel- Crafts Alkylations by Alkenes, and Related Reactions*
  L. Chen, J. Gasteiger

  *Knowledge Discovery in Reaction Databases: Landscaping Organic Reactions by a Self-Organizing Neural Network*
  L. Chen, J. Gasteiger
Classification of Organic Reactions: Similarity of Reactions Based on Changes in the Electronic Features of Oxygen Atoms at the Reaction Sites
H. Satoh, O. Sacher, T. Nakata, L. Chen, J. Gasteiger, K. Funatsu

- classification of samples based on chemical analysis
  Classification of Multicomponent Analytical Data of Olive Oils using Different Neural Networks
  J. Zupan, M. Novic, X. Li, J. Gasteiger

- overview of the use of Kohonen networks in drug design
  The Use of Self-Organizing Neural Networks in Drug Design
  in "3D QSAR im Drug Design - Volume 2", p. 273-299

Supervised learning

- simulation of infrared spectra
  The Coding of the Three-dimensional Structure of Molecules by Molecular Transforms and Its Application to Structure – Spectra Correlations and Studies of Biological Activity
  J. H. Schuur, P. Selzer, J. Gasteiger

  Infrared Spectra Simulation of Substituted Benzene Derivatives on the Basis of a Novel 3D Structure Representation
  J. Schuur, J. Gasteiger

  The Prediction of the 3D Structure of Organic Molecules from Their Infrared Spectra
  M. C. Hemmer, V. Steinhauer, J. Gasteiger