Diagnosis with Topological Maps

Jürgen Rahmel
University of Kaiserslautern
Centre for Learning Systems and Applications
67653 Kaiserslautern, Germany
e-mail:*rahmel@informatik.uni-kl.de*

Abstract

This technical report is a compilation of several papers on the task of solving diagnostic problems with the help of topology preserving maps. It first reviews the application of Kohonen's Self-Organizing Feature Map (SOFM) for a technical diagnosis task, namely the fault detection in CNC-Machines with the KoDiag system [RW93], [RW94]. For emergent problems with coding attribute values, we then introduce fuzzy coding, similarity assignment and weight updating schemes for three crucial data types (continuous values, ordered and unordered symbols). These techniques result in a SOFM type network based on user defined local similarities, thus being able to incorporate a priori knowledge about the domain [Rah95].

1 Introduction

The Self-Organizing Feature Map SOFM [Koh89] is a widely known algorithm for data clustering and feature extraction. It realizes a mapping from a usually high-dimensional data space to a two-dimensional space of neurons located in the map layer of the network. The mapping is topology preserving, i.e. neurons, that are neighbors in the map represent data points, that are somewhat similar in input space. The SOFM has found many applications, some of which are in the domain of engineering. The KoDiag system [RW94] described in Sect. 3 has shown a way to build a diagnosis system for CNC-machines with the additional concept of semantic maps [RK89]. The possibility of assigning semantics to certain areas in the map is used to design an iterative, explained and user controlled diagnosis process. Thus, a diagnosis process can be performed, that simulates the way of human problem solving in such situations. The user is guided by a tree-structured graphical representation and has various possibilities of interaction to control the diagnosis process.

However, the data used in the KoDiag approach was containing binary values only. In order to use other data sets with different types like scalar values or ordered/unordered symbols for the attribute values, there are different coding techniques that can be applied to the data (see [Han88] for a short compilation of coding schemes). But different schemes

imply different representations and therefore impose a different bias on the clustering in the Kohonen algorithm, which uses euclidean distances between input data and weight vectors (cf. Section 2). Additionally, in some domains of application, certain correlations of attribute values might be known in advance. These correlations should be preserved by the resulting representation of the data set, which makes it difficult to find the appropriate coding scheme. The idea of this work is to construct a neural network that is based on the neighborhood preserving weight adaptation of the Kohonen algorithm, but uses local and global similarity relations instead of the minimal distance criterion to determine the winning node. Local similarity is determined seperately for the value of each attribute by means of a similarity matrix, that can easily be set up according to the facts known about the domain. Also, nonstandard concepts of similarity like nonsymmetry can be modeled that way. The result is propagated to the map neurons, which calculate their activation by the global similarity relation.

The report is organized as follows. In Section 2 we restate the Kohonen algorithm for the SOFM and explain the concept of semantic maps. Section 3 reviews the application of KoDiag, a SOFM-based system, for technical diagnosis and the results of this approach. In Section 4 we introduce fuzzy coding, similarity assignment and weight updating schemes for three crucial data types and describe the similarity calculations and weight dynamics. These techniques result in a SOFM type network based on user defined local similarities. The applicability of the techniques for diagnostic tasks and results are shown in Section 4.4. A summary will conclude the paper.

2 The Kohonen Self-Organizing Feature Map

For readers not familiar with the SOFM, this section shall present the original algorithm as proposed by Kohonen [Koh90] and the concept of semantic maps. Details of convergence analysis are omitted here and can be found (for the one-dimensional case) in [CF86] and [EOS92].

2.1 The Kohonen Algorithm

Let p Neurons $[N_1 \dots N_p]$ be arranged on a rectangular $p_1 \times p_2$ grid, $p_1 p_2 = p$ (other grid forms, e.g. hexagonal are possible, but not considered here). Those neurons of the so called Kohonen layer are fully connected to the neurons of the input layer $[X_1 \dots X_k]$, the activations of which will be set to the values of the current input vector $\mathbf{x} = (x_1, x_2, \dots, x_k)^T$ at each training step. Each connection from X_i to N_j has an associated weight value w_i^j, by $\mathbf{w}^j = (w_1^j, w_2^j \dots, w_k^j)^T$ we denote the weight vector of neuron N_j.

In order to obtain the topology preserving property of the SOFM, the weight adaptation at each training step not only affects one neuron but a whole group U_c of neurons around a center N_c of activation. The weight vector $\mathbf{w}^c$ of this neuron N_c is the one, that best fits to the current input vector $\mathbf{x}$ according to the applied distance or similarity measure. Kohonen proposes two different ways to determine N_c. The inner product $\mathbf{x}^T \mathbf{w}^j$ can be

used to measure the similarity of the input and weight vector of neuron N_j, if the vectors are normalized. An often more convenient criterion is the minimization of the euclidean distances between input and weight vectors. Thus, the center N_c is the one, for which the following holds:

$$\|\mathbf{x} - \mathbf{w}^c\| = \min_i\{\|\mathbf{x} - \mathbf{w}^i\|\}. \tag{1}$$

Different forms of the neighborhood are possible. U_c may be defined by a hard set with neurons nearer than some radius being in U_c, others being outside. In our case, we use the biologically more plausible continuous bell-shaped membership function

$$h_{rc} = h_0\, exp(-\frac{|r-c\|^2}{\sigma^2}), \tag{2}$$

where r and c are the lattice positions of neurons N_r and N_c, respectively, and $\sigma = \sigma(t)$ is a suitable decreasing function, like e.g.

$$\sigma(t) = \sigma_i \left(\frac{\sigma_f}{\sigma_i}\right)^{\frac{t}{t_{max}}}, \tag{3}$$

where t_{max} is the predefined duration of the training and σ_i and σ_f denote the initial and final radius of the neighborhood, respectively. After presenting an input vector and determining the winning neuron N_c by (1), weights are updated by the following rule

$$\mathbf{w}^r(t+1) \doteq \mathbf{w}^r(t) + \alpha\, h_{rc}\,(\mathbf{x}(t) - \mathbf{w}^r(t)), \tag{4}$$

where $\alpha = \alpha(t)$ is a decreasing function in time, called the learning rate. Summarizing, the algorithm consists of a finite alternating sequence of

1. presenting an input vector, randomly chosen from the data set,
2. determining the center neuron N_c by equation (1) and
3. adapting the weights of the neurons in the Kohonen layer by equation (4).

For a visualization of simple ordering processes and a demonstration of obtaining localized responses for particular input vectors, see [Koh90].

2.2 Semantic Maps

Using neural networks leads to the necessity of coding symbolic expressions into a distributed representation that can be processed by the network. But the similarity between objects as it is derived from their attributes is not necessarily reflected in the encoding of the symbols describing those objects. To choose a particular coding scheme always means to introduce a bias into the similarity assignment for the encoded items. In addition to

the model described in Sect. 2.1, Ritter and Kohonen propose in [RK89] to divide the input vector according to the semantics of the partial vectors. As the Kohonen model uses metric differences between vectors, encoding discrete symbols leads to a problem, because semantic similarities are commonly not reflected in the encoding vector. To reduce this influence, the input vector is assumed to be a concatenation of two or more fields, for example, one field specifying the symbol code and a second the attribute set, respectively. Denoting the fields by $\mathbf{x}_s$ and $\mathbf{x}_a$, the input vector $\mathbf{X}$ is written as:

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}_a \\ \mathbf{x}_s \end{pmatrix} = \begin{pmatrix} \mathbf{x}_a \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ \mathbf{x}_s \end{pmatrix}. \tag{5}$$

If the values in the attribute part $\mathbf{x}_a$ are weighted higher than those in the symbol part, its norm will predominate over the norm of the latter, leading to a clustering in the map, that mainly reflects the similarities of the attributes of the animals. But still, there are memory traces from the map to the symbol part, that can be used in the application phase to assign a symbol to a neuron in the map. Thus, the locations of the symbols of the input data are ordered on the map in a way, that reflects the contextual appearance of the items, controlled by the attributes. In the next section, this technique is extended for building a diagnosis system. In this system, however, we use training vectors that are composed of three different fields, so we define the *weighting* of the input pattern to be a triplet (k, l, m), where k,l and m denote the coding values for the vector components in the first, second and third field of the input, respectively ($0 \leq k, l, m \leq$).

3 The KoDiag Approach

This section describes the application of *semantic maps* [RK89] to the problem of diagnosis. The idea of such a diagnosis system is to make use of prior cases with known solutions, compare a new problem with the old cases and infer the solution of the most similar case as diagnosis in the current problem (a case-matching system in terms of the Case-based Reasoning community). The meaningfully localized responses of the SOFM allow for a more sophisticated and flexible diagnosis process than with common neural classifiers. After an introduction of notations, we review KoDiag, an application of semantic maps for diagnosis tasks.

3.1 Notations and general considerations

The data records for the KoDiag system stem from a compilation of diagnosis cases for a CNC-machine. First we define the necessary denotations to faciliate the discussion (adapted from [Wes91]).

Symptom. A *symptom* S_i is a technically measurable property and is denoted by $S_i = (A_i, V_i, v_{ij})$, where A_i is the name of the property (attribute), V_i is the range of possible

values for this attribute and v_{ij} is the actually measured value, $v_{ij} \in V_i \cup \{unknown\}$. The finite sets of all symptoms and attributes in the domain of interest are denoted by **S** and **A**, respectively.

In the sequel, as an attribute is unique for a symptom, we often identify a symptom with its attribute name. A *test* t_i is a measurement, that obtains the value v_i of attribute A_i. As tests are always coupled with the cost of their execution, the diagnosis process should only require a restricted number of tests. Unlike classification, a diagnosis process is a time dependent task. Therefore we define the current situation of the diagnosis process.

Situation. A *situation* $Sit(\tau)$ is defined by the values v_{ij} already known at time τ. Thus, $Sit(\tau) \subseteq \mathbf{S}$ with $S_i \in Sit \Leftrightarrow v_{ij} \in V_i$. We write short Sit, if the time is unambiguous.

Diagnosis. A *diagnosis* is a set of symptoms, that describes a faulty state of the machine. It will be associated with a name $D_j \in \mathbf{D} = \{D_1, D_2, \ldots, D_q\}$, the set of possible diagnoses. Thus, a diagnosis is a description of the source of error, but sometimes it will be combined with a solution for the problem.

Case. The necessary information, that characterizes a faulty machine state and provides a diagnosis for this problem is contained in a *case* $C_k = (Sit_k, D_k)$.

The experience of an expert concerning the diagnosis task can be expressed by cases, that associate a complete description of a situation with the diagnosis for that situation. The *case base* $\mathbf{C} = \{C_1, C_2, \ldots, C_m\}$ contains a history of problems with appropriate solutions. The goal of the diagnosis system will be to solve a current problem Sit_c by finding a case $C_j \in \mathbf{C}$, where Sit_j is the situation, that is most similar to the current problem. Then D_j will be proposed as the solution for Sit_c.

A striking point in diagnosis is the fact, that in general Sit_c is not known at the time of occurrence of the fault. A sequence $\mathbf{T} = (t_1, \ldots, t_{\tau_c})$ of tests has to be found, that determines the relevant $v_i \in Sit_c$ and minimizes the cost of testing. For this reason, a diagnosis system has to effectively realize at least two different tasks:

- the *classification component* compares $Sit_c(\tau)$ with the cases in **C** and proposes the most probable solution
- the *test selection component* determines a test $t_{\tau+1}$, derived from $Sit_c(\tau)$, to increase the amount of information about Sit_c.

In addition, explanatory functions should be available, that provide additional information for the user, e.g. concerning alternatives of proposed solutions, and allow flexible control of the diagnosis process. The graph in Figure 1 depicts the cyclic structure of this diagnosis process, which is an iteration of classifying the actually known amount of information $Sit_c(\tau)$ and trying to get more information, if necessary. In the beginning of the process, $Sit_c(\tau_0)$ is not necessarily empty. Like in our domain of application, there often are several symptoms, that are easily observed (for instance, the position of a moving part can be determined simply by looking at it). Additionally, the internal diagnosis system of a machine might output a machine error code, that will be the first clue to the problem. Thus, $Sit_c(\tau_0)$ often contains one or two of these obvious symptoms.

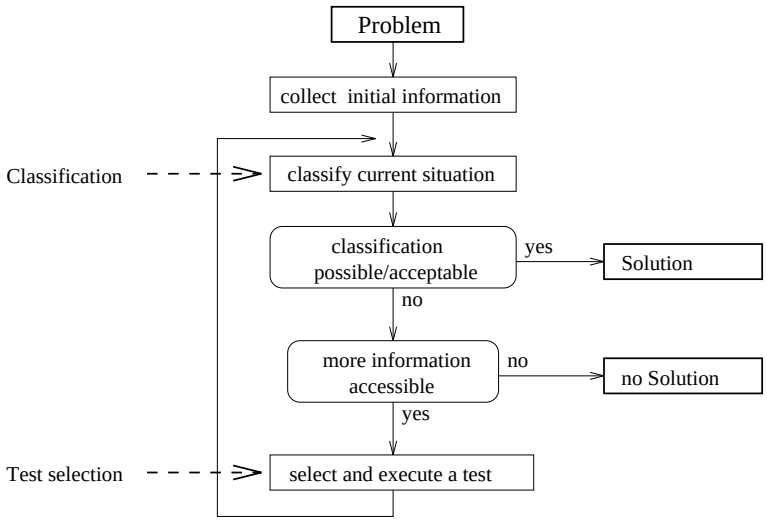


Figure 1: A common diagnosis process

3.2 Training of the KoDiag Network

This section describes the data we use in KoDiag, the encoding and the 2-phases training, a special training method for Kohonen semantic maps. The case examples, which we use in the KoDiag system, are part of a large database, also used for symbolic case-based reasoning systems like the PATDEX-System [Wes91]. They describe faulty states of CNC-machines. An example case is as follows:

```
Case:         Toolarm10

Description:  IoStateIN32     logical0
              Code            I41
              ToolarmPosition back
              IoStateOUT30    logical0
              IoStateOUT28    logical1
              Valve21Y2       switched
              IoStateIN37     logical1

Diagnosis:    IoCardFaultAtIN32i59
```

In the above case named *Toolarm10*, for example *IoStateIN32* is an attribute with value *logical0*. The string *IoCardFaultAtIN32i59* describes the diagnosis that is appropriate when attributes appear with the values stated in that case description. In the KoDiag system the records of the case database are coded into a bit string that is used as input vector (also called the training pattern) for the network. As stated above, the input vector consists of three parts: the attribute field, the value field and the diagnoses field. Figure 2 shows the way the appearance of attributes, values and diagnoses in a particular case is encoded.

For every attribute in the database, one bit in the attribute vector is reserved and the same applies for the values in the vector of values and for diagnoses in the diagnoses part, respectively.

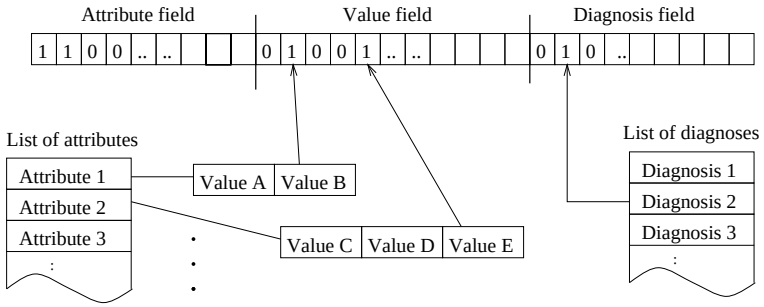


Figure 2: The coding of case in KoDiag

Despite the fact, that no modelling of the real CNC-machine is available in addition to the case database for use in the background of the diagnostic system, some semantic information can be extracted from the contextual appearance of features in the case records. To strengthen the formation of clusters in the Kohonen layer due to the feature part of the input vector, we designed the 2-phases training process. Two pattern P1 and P2 are called a *2-phases pattern*, if the information contained in P1 is a proper subset of the information in P2.

Thus, the coding process described above constructs training pattern P2. Pattern P1 is obtained by setting the bits of the feature values to zero. The training of the network is first carried out with P1 for a number of epochs. This leads to a coarse clustering of the Kohonen map that is driven by the feature context in the case records. For about the last quarter of the training time, the full pattern is presented as input vector and structures in the layer are refined according to the feature values of the particular cases. In the context of 2-phases training, the weighting (k,l,m) for a training pattern defines a weighting of (k,0,m) in the first phase and (k,l,m) in the second. Evaluation of the training results of KoDiag's Kohonen network with and without the 2-phases training is described in Sect. 3.5

3.3 Diagnosis with the KoDiag system

The concept of semantic maps was used to realize KoDiag, a neural system based on the SOFM, that imitates the diagnosis process described above. The training of the network results in storing the trained cases in a spatially ordered fashion that is determined by the presence (or absence) of the most frequent S_i. This ordering process gains additional information by the contextual appearances of the attributes and values in the encoded cases. As the output of the SOFM is a structured map and not a category information, we need a method to extract the classification knowledge from the map. Among the methods discussed in literature are the use of linguistic variables and fuzzy decision theory [PC92] or the U-matrix method [Ult92]. We will describe a method based on iterative operations

on sets of responding neurons. This process is carried out in a stepwise manner, directed by the test selection component of KoDiag. In this point our approach to CBR differs from other connectionist models or systems in that field, e.g. [Thr89], [BJ89]. From a general point of view, as stated above, diagnosis splits up into two tasks - classification and test selection - and this iterative and natural way of problem solving can be preserved for the computer aided approach. In addition, the system should be able to provide multiple solutions and name alternatives for rejected outputs.

3.3.1 Classification

The diagnosis process of KoDiag is interactively controlled by the user of the system. If a CNC-machine failure occurs, the machine gives some error messages, e.g. lighting a particular bulb or stating a message on a display. This error message can be the first input into the KoDiag system. In technical domains it is difficult even for an expert to decide whether a given attribute value is of pathological nature or not. Error messages of the machine are certainly pathological values, but it is not that clear for other parts of the machine. Therefore, KoDiag has to tolerate a wrong user input in the beginning of the diagnosis process.

KoDiag keeps track of the user inputs by a set $P_u(\tau)$ of excited neurons. In the beginning, the set $P_u(\tau)$ is empty. If a new attribute/value pair (A_i, v_{ij}) is entered by the user at time $\tau + 1$, the corresponding input vector $\mathbf{x}_{\tau+1}$ for that symptom is presented to the network, activating a set $P_i(\tau + 1)$ of neurons in the map. To be independent of the order of user inputs and robust against some wrong (i.e. not pathological) data, the union of the sets is taken as a description of the input up to this point:

$$P_u(\tau + 1) \doteq P_u(\tau) \cup P_i(\tau + 1). \tag{6}$$

The classification of the current situation $Sit_c(\tau + 1)$ is finished by determining the diagnosis D_j that is activated most, if we trace the connections from the neurons in $P_u(\tau+1)$ to the part $\mathbf{x}_d$ of the input layer. This diagnosis $D_j(\tau + 1)$ is proposed as solution for $Sit_c(\tau + 1)$. These actions are repeated as often as the user wants to make an input without being asked by the system. Each step adds a level to the tree in the graphical output window of KoDiag. Figure 3 shows an example of the solution tree after the first input. The user decided to select attribute *IOStatusIN32* from the menu and entered the value *logical0* which could be read from a display and which is supposed to be of pathological nature in the current situation. KoDiag then claculates as described above a number of suggested diagnoses. If the user wants to enter another attribute/value pair, the second attribute box will be drawn as another son of the first one and the new list of diagnoses is then determined.

But, as the number of neurons in $P_u(\tau)$ is growing, no convergence of the process can be expected. In fact, it is not necessary to make more than two or three inputs to KoDiag without being asked by the system. To reduce the size of $P_u(\tau)$ and to increase the quality of suggested diagnoses, the test selection facility of KoDiag is to be called.

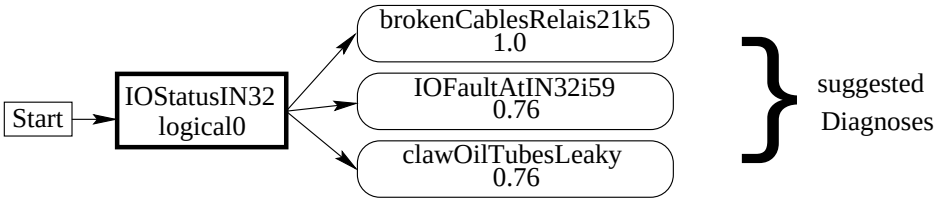


Figure 3: The solution tree after the first user input

3.3.2 Test Selection

As shown above, there might be a need to execute one or more additional tests to gain more information about the current situation. If the user decides to stop the deliberate entering of attributes and corresponding values, the KoDiag system can take control over the diagnosis process. The test selection strategy chooses, depending on $Sit_c(\tau)$, the tests t_i that are most likely to be useful in the actual context and with respect to the case base $\mathbf{C}$, see Fig. 4. This is done by using the set $P_u(\tau)$ of neurons responding to $Sit_c(\tau)$ and tracing back their connections to the attributes part $\mathbf{x}_a$ of the input layer. The strongest (not yet specified) attribute A_i is suggested to be tested next by t_i.

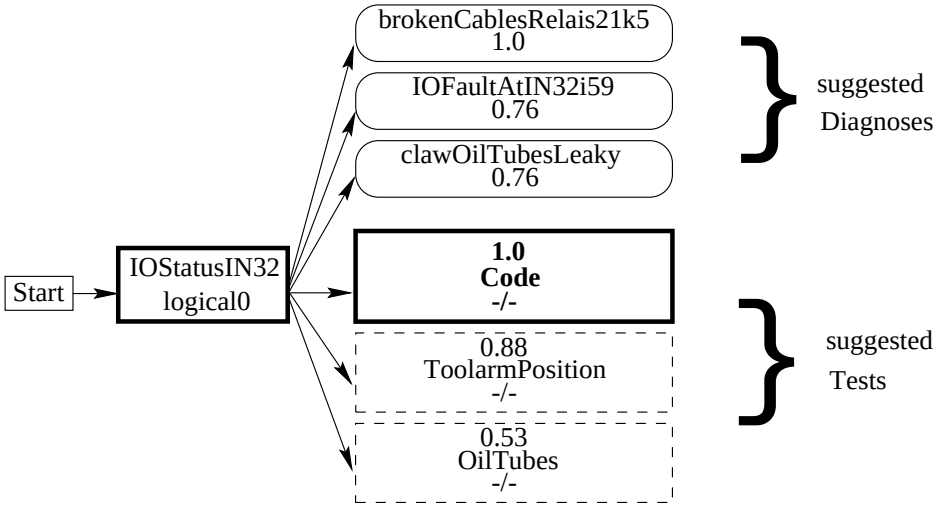


Figure 4: Test selection called after the first user input

By executing the test and entering the value v_i, the user starts another cycle of the diagnosis process, but this time, the intersection of the two neuron sets involved is calculated:

$$P_u(\tau+1) \doteq P_u(\tau) \cap P_i(\tau+1). \tag{7}$$

This is an allowed operation now, because the attribute/value pair was entered on request of the system, so there is no deliberate ordering in the input of the user. As

described for the classification, KoDiag is calculating a diagnosis proposal for the new situation $Sit_c(\tau+1)$. While repeating these classification and test selection steps, the size of $P_u(\tau)$ now is decreasing and the diagnosis process is guaranteed to converge. The end of the diagnosis process is reached, if

1. KoDiag proposes a unique diagnosis,
2. the user accepts the output with respect to the relative importances of the diagnoses (see Fig. 5) or
3. the set $P_u(\tau)$ is empty, due to poor conformity of the present situation with the learned cases.

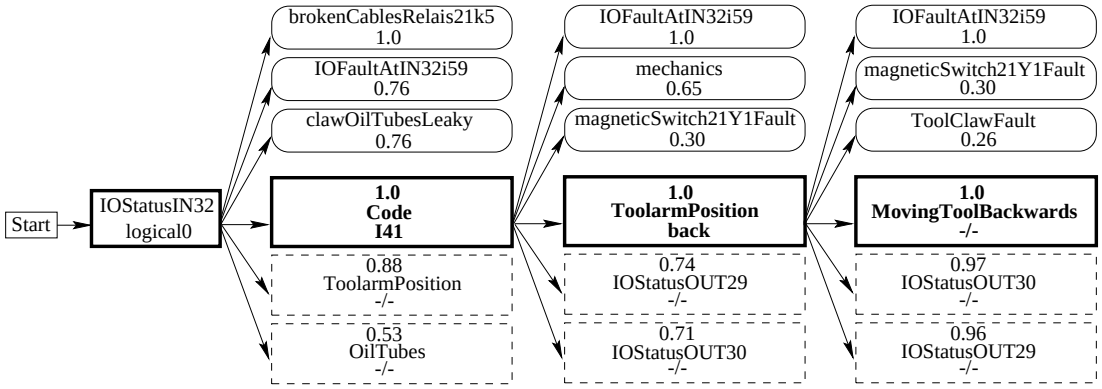


Figure 5: A complete solution tree with acceptable diagnosis *IOFaultAtIN32i59*

As shown in Fig. 5, the relative importances that are given together with the list of attributes allow the user to choose a test from the list. The test actually executed need not be the one with the hightest score, if other considerations are in favor of a test with a slightly lower score. The relative weightings of the diagnoses help in deciding how well the top diagnosis is established in comparison to other alternatives. Figure 6 resumes the diagnosis process of KoDiag.

3.4 Explanation

As indicated in Figure 6, KoDiag not only proposes one solution or test per step but a bundle of alternatives with relative ratings. The user is free in choosing alternatives according to his own considerations, and is supported by the graphical display of the solution tree, containing all chosen and neglected alternatives and showing the currently followed path. This allows for a recapitulation of the whole process at any time instead of having to accept a unique, uncommented solution as with common neural classifiers. Intermediate results are stored in the nodes of the tree, thus a rejection of the current path and resuming at a deliberate level of the tree is possible. Figure 7 shows an example for this. After entering value *I55* for the attribute *Code*, the test selection component

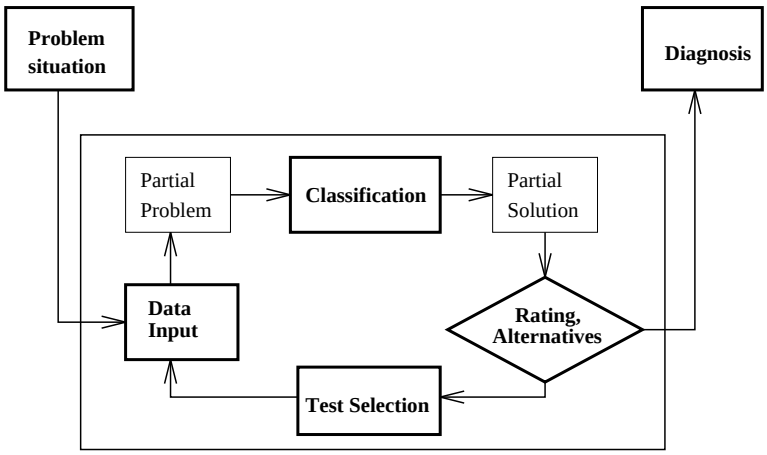


Figure 6: The diagnosis process in KoDiag, schematically

suggested checking the attribute *OilTubes*. But as execution of this test is very expensive, the user may decide to follow the slightly less weighted path below and enter the value for attribute *ToolarmPosition*. In this example, a unique diagnosis was then proposed by the system.

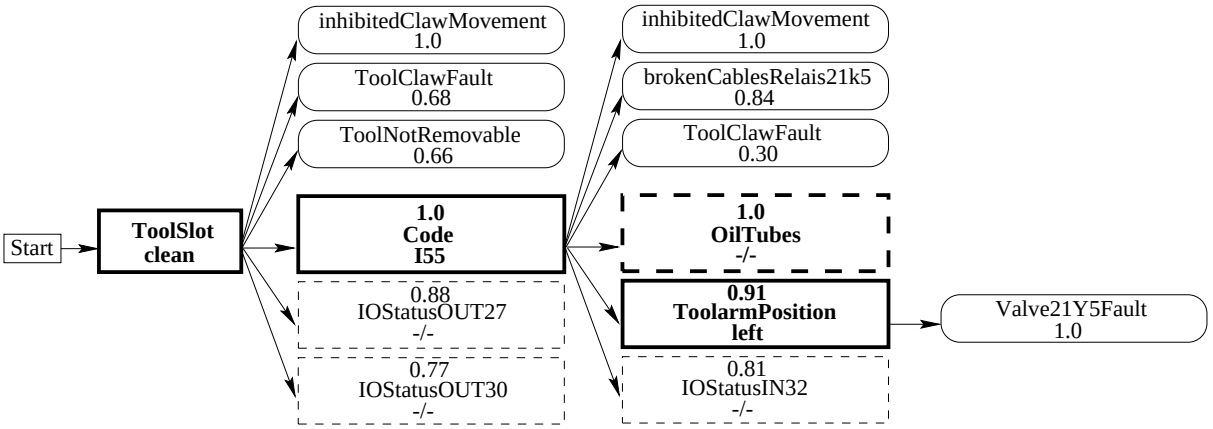


Figure 7: An example for rejection of an attribute

Another idea allows separation of similar weighted diagnoses. As the representation of the cases in the network is ruled by relative frequencies and not by discriminant functions, it it sometimes convenient to shorten the iterative process by comissioning KoDiag to find out the test, which will differentiate between two selected diagnoses. This can be achieved by collecting the most important attributes of either diagnosis, determining the discriminating attribute or attribute value.

3.5 Results for the KoDiag Approach

This description of the results refers to a subset of 101 cases of a larger database. Those cases contain 62 attributes, 125 attribute values and 41 diagnoses, therefore the input vector is made up of 228 bits. Throughout the evaluation process we use testvectors with weighting (1,1,0), so no diagnoses are given as input. We had main interest in the behaviour of KoDiag when information of the cases was lost or not available, because when used interactively as a diagnostic tool the system should be robust against missing data. On one hand, the user may forget some input, one the other hand, inquiries about some attribute values may be too expensive to be made. For that reason, test vectors are generated, which hold only a fraction of the original information in the fields of attributes and attribute values.

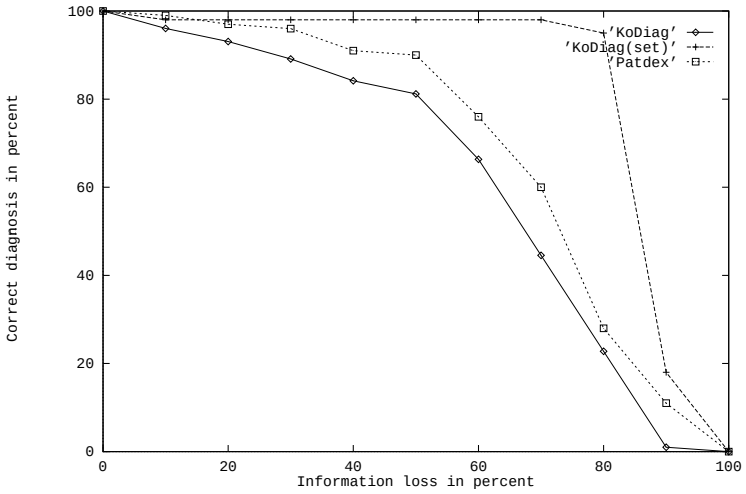


Figure 8: Comparison of KoDiag and Patdex

Figure 8 shows the dependence of obtaining the correct diagnosis and loss of information. Two curves are drawn for KoDiag, as we used two different evaluation methods. The one denoted *KoDiag* refers to the 'unambigous' approach, where a result was considered correct, only if the system was able to suggest a diagnosis clearly separated from its alternatives. The curve named *KoDiag(set)* is obtained, when a set of nearly equally-weighted diagnoses are suggested and that set included the correct diagnosis. In this example the network size was 15×15, the weighting was (1,1,1) and training duration was 60 epochs (no 2-phases training). For comparison the third curve named *Patdex* is drawn. It refers to the performance of the state-of-the-art CBR-expert system Patdex [Wes91], based on the same database and evaluated according to the 'unambigous' approach.

The performance of KoDiag for larger information losses can be increased by the 2-phases training method in combination with a different weighting of the input vector. If we set the weighting to (1,0,0.5), the vector of attributes has most influence on the structuring of the Kohonen map. Thus, in the first phase the arising clustering mainly reflects the appearance of diagnoses in the context of merely the attributes. No attribute

values are considered in the training yet. In the second phase, the full vector is supplied to the network with weighting (1,1,0.5). Figure 9 shows a comparison of the results of a normal training and the 2-phases training where patterns were changed after 30 epochs, with a total training duration of 40 epochs. Evaluation again is based on the 'unambigous' approach.

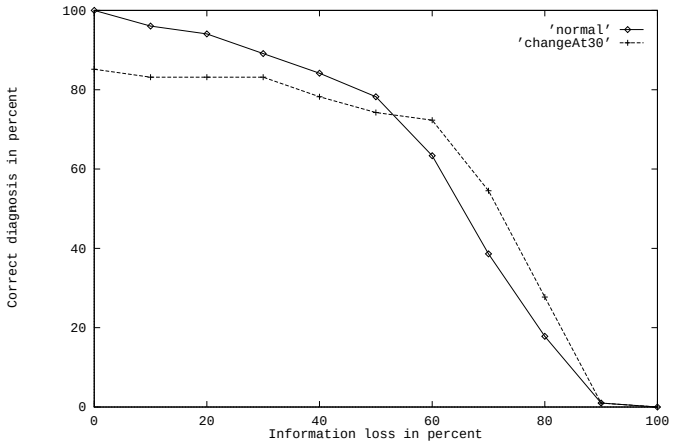


Figure 9: Comparison of normal and 2-phases training

By changing patterns after about 70-80% of training time, the results for the range of 55-85% information loss can be increased by up to 15 percentage points, then beating even the Hi-level expert system Patdex in that range. This is paid for by reduced performance when (nearly) full information is available, but because a diagnostic tool is rarely used as a stand-alone system, a suggestion of a few diagnoses at an early point can be seen as a very useful output and in fact this was the main goal in the construction of KoDiag.

The stepwise diagnosis process and the explanation and control functions distinguish KoDiag from the traditional approaches like [Thr89] or [BJ89] using neural nets as one-shot black-box classifiers. However, the power of the system is limited in two ways. First, what differs most from the way we humans solve diagnosis problems is the test selection strategy. The quality of the test selection in KoDiag strongly depends on the cases in $\mathbf{C}$. In the domain mentioned above, the test selection component was very effective, leading to good solutions after only a few cycles. But is has to be noted that one can construct case bases, for which the strategy is very ineffective. Second, the use of semantic maps is not an appropriate method to circumvent the anomalies introduced by certain coding strategies. The similarity of attribute values is implicitly determined by the encoding, e.g. the similarity of values v_i and v_j will be judged differently, depending on the use of interpolation coding, continuous coding or any other coding scheme ([Han88]). The first drawback can be approached by hybrid systems that support different methods, e.g. model-based, for the selection of tests. The rest of the paper will propose a way to strengthen the above diagnostic system by making similarities and dissimilarities in the data explicit. Thus, additional domain knowledge is used to improve the performance of the diagnosis process.

4 Similarity-based clustering

In the previous section, it was shown how the Kohonen network and its extension to semantic maps can be applied to construct a diagnostic system. However, the data used for the CNC-machine application was restricted to binary values. In more general domains at least three different data types are needed for an adequate representation of the underlying information of attribute A_i:

- continuous values ($V_i = [a; b]$),
- unordered symbols ($V_i = \{v_{i1}, v_{i2}, \ldots, v_{in}\}$)
- ordered symbols ($V_i = \{v_{i1}, v_{i2}, \ldots, v_{in}\}$ and $v_{i1} < v_{i2} < \cdots < v_{in}$).

In KoDiag, each $\mathbf{x}_v|_{A_i}$ was realized as a binary vector with $|V_i|$ dimensions, distance calculation and weight adaptation were uniform over all $\mathbf{x}_v$. Now, each of the three types will use a different processing policy. The ordered symbols need a weight updating scheme, that produces valid weight vectors only (see below), and for the continuous values, there might be a need for a range dependent distance/similarity. For that reason, the neurons for the former part $\mathbf{x}_v$ of the values in the input vector $\mathbf{x}$ are replaced by a collection of cells $[M_1 \ldots M_l]$, $l = |\mathbf{A}|$, each locally processing the values of one attribute A_i. The M_i calculate a similarity relation and propagate a similarity value Sim to the neurons of the map layer, where the global similarity of this input vector is determined. We leave the vectors $\mathbf{x}_a$ of attributes and $\mathbf{x}_d$ as they were in KoDiag. That way, we preserve the properties of KoDiag concerning the flexible diagnosis process and handling of unknown values (see Section 4.3).

4.1 Local similarity

This section describes the local similarity assignment for the three data types mentioned above. We need a way to effectively assign a similarity $Sim_{ij} = Sim(v_{ij}, w^j|_{A_i})$ between the current input value for attribute A_i and the local weights of map neuron j. The notion of fuzzy sets and numbers has shown to be a natural way for defining and understanding the similarity of different values of attributes. An introduction to these concepts can be found in [Bez81] or [KG88].

4.1.1 Continuous values

Continuous values are difficult to handle with neural networks. Their range V_i can be discretized, but finding the best boundaries for discretization involves a priori knowledge, which often is not available for the data in question. Also different ranges for different attributes A_i require normalization of the V_i onto the interval $[0; 1]$ to equalize their influence on the global distance measure. But normalization has some unwanted side effects. Outliers will compress the data more than necessary while normalization fixes the predefined

range that is currently seen in the data. Future extension of the range, a crucial need for incremental versions of the approach, is therefore possible only with a complete retraining. Additionally, in practice it has shown to be necessary to have a range dependent similarity function for continuous values, i.e. equal distances of two points v_i and v_j to a third one not necessarily mean equal similarity of those pairs. As an example, consider an attribute *temperature* in continuous coding. We might wish to declare distinct temperature zones (normal, critical) and within the critical zone, a deviation of temperatures shall influence similarity more rapid than in the normal zone. This kind of additional domain knowledge often is available, but not used in diagnostic applications of neural networks.

Similarity assignment. For the above reasons, we use a similarity measure on unnormalized data, that is defined with the help of *Triangular Fuzzy Numbers* and *Trapezoidal Fuzzy Numbers* (TFN and TrFN, respectively). A range dependent similarity can now be achieved by dividing the range into several regions sim_k of equal similarity assignment. Figure 10 shows the definition of $sim_{1,2,3}$ with three TrFN, each determining the membership $\mu_k(x_i)$ of value x_i to sim_k, where $x_i = v_{ij}$ in the case of continuous values.

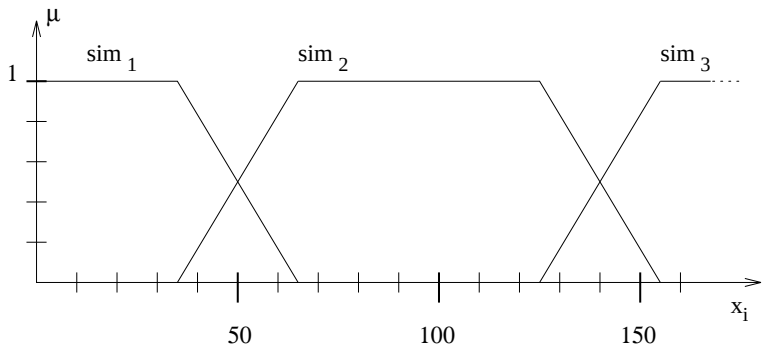


Figure 10: Definition of similarity ranges

Now, the sim_k can seperately be defined by TFNs in a way that takes into account the relative position of x_i and w^j on the range scale. The TFN is based on the position of x_i and similarity can decrease differently whether w^j is above or below x_i. For the definition as in Figure 11, the x_i and w^j are considered more similar, if $x_i < w^j$ and similarity decreases faster, if $x_i > w^j$. Of course, there is no restriction of the general method to TFNs. We chose them for convenience of their definition.

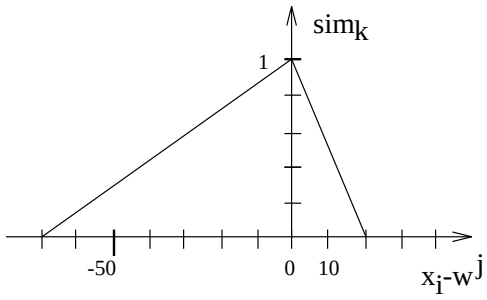


Figure 11: Range dependent similarity

The local similarity Sim_{ij} for the continuous attribute A_i compared to the corresponding weight of Neuron N_j is then calculated by summing up the contributions of the different ranges:

$$Sim_{ij}(x_i, w^j) \doteq \sum_k \mu_k(x_i)\, sim_k(x_i, w^j). \tag{8}$$

Note that at most two of the μ_k are not equal to zero, if the sim_k are defined as shown above.

Weight adaptation. The weight adaptation for continuous values is performed in a straightforward way. Depending on its distance from the winning node N_c in the map, the neuron N_j will be assigned a learning factor $\lambda = \alpha\, h_{jc}$. The adaptation rule then reads:

$$w^j(t+1) \doteq w^j(t) + \lambda\,(x_i - w^j(t)). \tag{9}$$

4.1.2 Unordered symbolic values

We subsume under the term *unordered symbolic values* all sets of symbolic values V_i, that do not have a unique linear ordering. Unordered values appear in many real applications of diagnosis and case-based systems, an example is a set of colors like V_i ={*red, black, blue*}. Even for such simple sets with no welldefined ordering, it is often necessary to assign lateral similarities between certain members of the set. In a particular application it might be known in advance, that *black* is more similar to *blue* than to *red*. The usual orthogonal coding with one bit/color is not able to cope with such needs. Additionally, we want to allow a nonbinary coding of the values to adopt a way of expressing uncertainty of measurements.

Similarity assignment. An approach to employ background knowledge is the use of a similarity matrix $\mathbf{F} = (f_{kl})$, $1 \le k, l \le |V_i| = n$, that contains the lateral similarities between the components v_{ik} and v_{il} of entities $v_i \in V_i$. The degree of presence of a value v_{ik} will be encoded by $x_{ik} \in [0;1]$, thus providing a means to express imprecision of observations. The matrix $\mathbf{F}$ will transform the vector $\mathbf{x}_i$ into a vector $\mathbf{sim}_i$ of entity similarities, that is used for further processing. The values f_{kl} are set by the user or calculated from existing taxonomies within the set V_i. The row vector $\mathbf{f}_k = (f_{k1}, f_{k2}, \ldots, f_{kn})$ contains the similarities between all entities and v_{ik}. Thus, the f_{kl} express the degree of membership of v_{il} in the set $\{v_{ik}\}$, denoted by μ_{kl} in the theory of fuzzy sets. Note that for reasons emerging from cognitive psychology, it is not allowed to use those similarities in a transitive way, i.e. inferencing f_{km} from f_{kl} and f_{lm}. The f_{kl} encode the imprecision and exchangeability of the elements of V_i. Thus, we calculate the vector $\mathbf{sim}_i$ as

$$\mathbf{sim}_i = \begin{pmatrix} sim(v_{i1}, x_i) \\ \vdots \\ sim(v_{in}, x_i) \end{pmatrix} \doteq \mathbf{F}\,\mathbf{x}_i = \begin{pmatrix} \sum_l f_{1l}\, x_{il} \\ \vdots \\ \sum_l f_{nl}\, x_{il} \end{pmatrix}. \tag{10}$$

This vector $\mathbf{sim}_i$ represents the activation of the different entities in V_i due to the current input $\mathbf{v}_i$ for the attribute A_i. The weight vector $\mathbf{w}_i^j \doteq \mathbf{w}^j|_{A_i}$ of neuron N_j then is compared with $\mathbf{sim}_i$ by

$$Sim_{ij}(\mathbf{x}_i, \mathbf{w}^j) \doteq \mathbf{sim}_i^T \mathbf{w}_i^j = \sum_n sim_{in}\, w_{in}^j \tag{11}$$

resulting in the local similarity Sim_{ij} for neuron N_j with respect to attribute A_i. If we normalize $\mathbf{sim}_i$ before applying equation (11), then Sim_{ij} is reflexive.

Weight adaptation. The weights between neurons N_j of the map and input cells M_i performing the above similarity processing are now vectors of dimension $|V_i|$. Nevertheless, the weight updating rule for the unordered symbolic values has the same form as (9), now with vectors instead of scalar values:

$$\mathbf{w}^j(t+1) \doteq \mathbf{w}^j(t) + \lambda\,(\mathbf{x}_i - \mathbf{w}^j(t)). \tag{9'}$$

Please note, that the similarities contained in matrix $\mathbf{F}$ are used for determining the winning node only. Weight adaptation is directed to the unchanged input vector $\mathbf{x}_i$. Thus, the weights still represent original data, in contrast to a network with preprocessing, that codes the similarity into the components of the input vector. This fact is essential, if the application of the network, e.g. in a diagnosis task (cf. Section 3), is based on weight inspection rather than black box usage of the trained network.

4.1.3 Ordered symbolic values

This section deals with the sets $V_i = \{v_{i1}, v_{i2}, \ldots, v_{in}\}$ for which exists a strict ordering $v_{i1} < v_{i2} < \cdots < v_{in}$. An example is the set $\{min,\ low,\ high,\ max\}$, where the ordering is obvious from the semantics of the v_{ij}. To a certain degree, the ordered symbols are treated in a manner comparable to the unordered symbols.

Similarity assignment. As in the case of unordered symbols, we make use of a matrix $\mathbf{F} = (f_{kl}) = (\mu_{kl})$ to handle the lateral similarity of elements v_{ik} and v_{il}, but for the ordered symbols we force $f_{kl} = 0$ for $|k - l| > 1$. This constraint is grounded in the strict ordering of the elements and the resulting scale-like properties of V_i. Now, since at most two of the f_{kl} are not zero, the row vector $\mathbf{f}_k$ is convex and thus a fuzzy number on the referential V_i. The same restriction applies to the input vector $\mathbf{x}_i$.

We calculate the vector of entity similarities $\mathbf{sim}_i$ and the global similarity Sim_{ij} for neuron N_j and attribute A_i as in equations (10) and (11), respectively.

Weight adaptation. Again, the weights between neurons N_j of the map and the input cells are vectors of dimension $|V_i|$. But for ordered symbols, the orthogonal coding with one bit per entity v_{ik} has more degrees of freedom than are allowed with respect to the strict ordering of the elements. Figure 12 shows this problem for the weights in the three dimensions of an example set {*low, medium, high*}. The weight adaptation scheme has to ensure, that the resulting weight vector $\mathbf{w}$ only moves on the path between *low* and *medium* or between *medium* and *high* (bold line). The direct crossing from *low* to *high* (dotted line) is forbidden, because it will lead to nonconvex and inconsistent representations of $\mathbf{w}$.

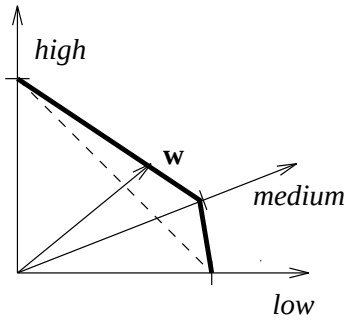


Figure 12: Allowed weight configurations

This is accomplished by a projection of V_i onto an interval $Q_i \subset \mathbb{R}$. We associate each position of a value v_{ik} with a number q_{ik}, such that $\forall_{k,l} : v_{ik} < v_{il} \Rightarrow q_{ik} < q_{il}$ and compute a number $q_{x_i} = \sum_n x_{in}\, q_{in}$, representing the projection of $\mathbf{x}_i$. Analoguously, we compute q_{w^j} for weight vectors of neuron N_j. The weight adaptation rule then operates on the projection values:

$$q_{w^j}(t+1) \doteq q_{w^j}(t) + \lambda\,(q_{x_i} - q_{w^j}(t)). \tag{12}$$

The resulting new value $q_{w^j}(t+1)$ has to be transformed back into a convex representation. The additional constraint $\sum_l w_l^j = 1$ makes this operation unique and yields the vector $\mathbf{w}^j(t+1)$.

4.2 Global similarity

The global similarity controls the clustering process of the map. The original equation (1) is no longer applicable for determining the winning node, when the local similarities are set as described above. The winning node N_c is determined by $c = \text{argmax}\{SIM_i\}$, $i = 1, \ldots, p$, where the global similarity SIM_j of a neuron N_j and the input vector $\mathbf{x}$ obtained from case C_k is calculated by

$$SIM_j \doteq \frac{1}{2|C_k|}\left(\sum_i x_{ai} w_{ai}^j + \sum_i Sim_{ij} + \sum_i x_{di} w_{di}^j\right). \tag{13}$$

The normalization factor contains the size $|C_k|$ of the current training case, what is meant to be the number of symptoms in Sit_k. Various simulations have indicated, that the choice for similarity assignment as it is described here leads to results that are of the same or slightly better quality as those of the normal SOFM (cf. Section 4.4). Thus, the gain of the application of these schemes is the ability to explicitly define lateral and nonstandard similarity concepts whenever available, in order to control the resulting topology of the map according to the knowledge available on local structures.

4.3 Modeling unknown values

In case matching systems like KoDiag, there exist two types of unknown values, that have to be dealt with. The first type are values of attributes, that are irrelevant for the case in question, e.g. all symptoms S_i that are not part of the situation described by a case C_k are not relevant for the diagnosis in C_k and thus are unknown. The second type consists of missing values for attributes appearing to be part of the case that seems to be appropriate for the current situation.

The first type of unknown values (or better: unknown symptoms) is easily modeled by the components of the vector $\mathbf{x}_a$ of attributes. As the connection weights of this part of the input and the value part are updated simultanuously, each component w_{ai} of a neurons weight vector $\mathbf{w}_a$ represents the importance of attribute A_i for that neuron. In a diagnostic application, the test selection component will make use of exactly those vectors $\mathbf{w}_a$ to find out about the relevance of attributes.

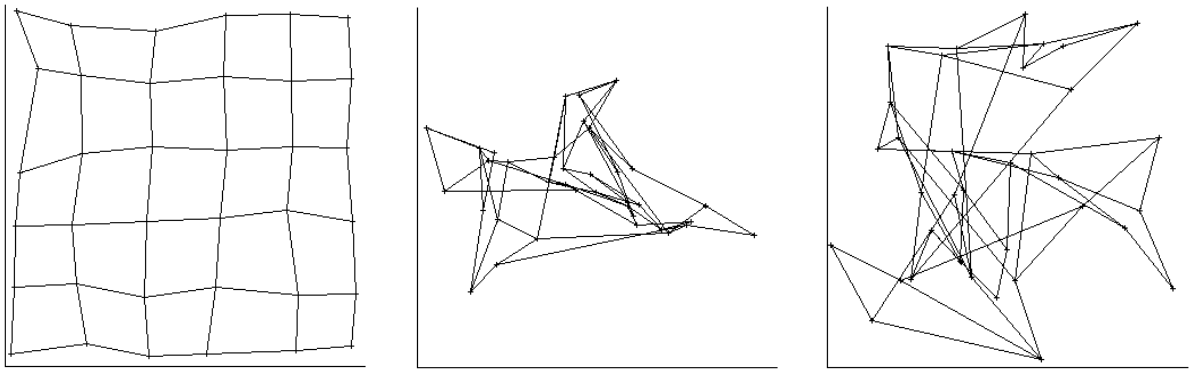
Unknowns of the second type occur, if the test suggested by the test selection component could not be executed for any reason. It can be seen from Section 3 that the strategy employed in KoDiag is optimistic in the sense that missing values are not considered to be restricting the scope of the current situation $Sit_c(\tau)$ in the diagnosis process.

4.4 Applicability and Results

First we demonstrate the necessity of the special weight update scheme for the ordered symbolic values. The referential set is selected as $V = \{v_1, v_2, \ldots, v_6\}$ with $v_1 < v_2 < \cdots < v_6$. We chose only two attributes with values $v_{ik} \in V$ and generated 300 training vectors by selecting the values for A_1 and A_2 randomly from V. The training result of the value vector can be visualized in a two-dimensional display. In order to localize the symbolic values, they were associated with numbers 0, 0.2, ... , 1.0 in an order preserving manner. Therefore, we have $(\mathbf{v}_1 \times \mathbf{v}_2) \in [0;1]^2$ and thus represent each neuron position in V^2 as a point in the unit square, connecting adjacent neurons with lines. Figure 13 shows the results for a 6×6-network after training.

The network in part (a) was trained with the weight updating scheme in equation (12) and spans the space as it is to be expected (cf. [Koh90], Fig. 3 for a continuous-valued equivalent). The similarity matrix $\mathbf{F}$ was initialized with small values for directly adjacent elements of V. Part (b) shows the effect of setting $\mathbf{F} = \mathbf{I}$, the identity matrix of the same rank. The network in (c) had the matrix initialized as in (a), but was trained with the normal weight update rule (9′). Since both of the nets (b) and (c) failed to produce a topological ordering of the input space, we conclude that the information contained in the lateral similarity and the update scheme are essential mechanisms in modeling ordered symbols through orthogonal coding.

In order to verify the diagnostic capabilities, we tested the approach presented in this paper against the model described in Section 3 for the CNC-machine domain that provided no additional information on lateral similarities. Several simulations were executed for both models and the trained networks were evaluated according to missing information.



(a) (b) (c)

Figure 13: Effects of weight update scheme

As the diagnosis task is to be solved iteratively and interactively, we had main interest in the behaviour of the models, when information of situations was lost or not available. Therefore, test vectors were generated, that contained only a fraction of the original amount of information in the fields of attributes and diagnoses. Figure 14 shows a comparison of the mean scores of five test runs for each model, depending on the information loss in the test vectors. The results are quite comparable, with a slight advantage for the proposed model, demonstrating the applicability even if no additional domain knowledge is coded into the similarity matrix.

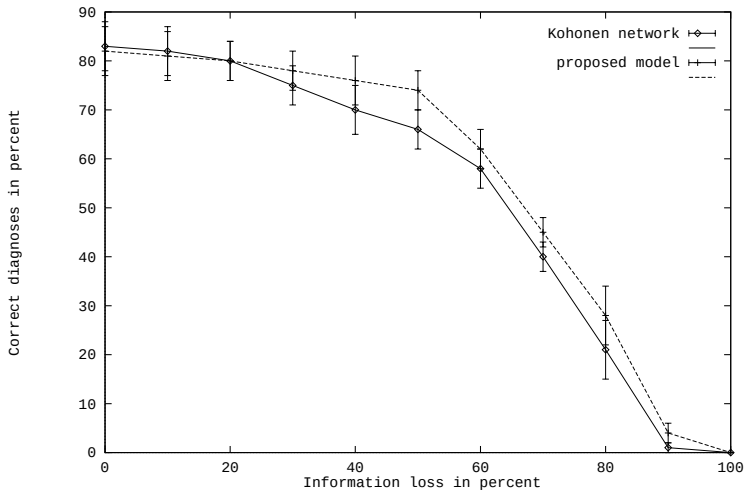


Figure 14: Comparison: Kohonen network against proposed model

Simulations with a case base of car sales demonstrated the difference between our model and the normal Kohonen network. Customer preferences, e.g. for price (continuous), engine

power (ordered) or color (unordered), can now be modeled from the experiences of the sales persons and given to the network. This provides valuable information that might not be contained in the sequence of training patterns. The diagnosis process described above is now a process that searches a car according to the customers specifications and that is influenced by the relations modeled by the lateral similarities in different attributes and values.

The advantage of the proposed model appears for such cases, where no full matches to the specification could be found. Then, alternative solutions are suggested that fit better to the customers needs because the different local similarities favoured trained cases, the normal Kohonen model wouldn't have selected.

5 Summary

The subject of this work is the application of topological maps for diagnostic tasks. After some general remarks on diagnosis processes and their requirements, we reviewed an application of the SOFM in the area of CNC-machine diagnosis. This application raised questions on how to encode continuous and symbolic data properly, in order to incorporate additional domain knowledge that often remains unused. We proposed similarity assignment and weight updating for fuzzy coded inputs of three different data types. The model is able to explicitly deal with lateral local similarity between entities and supports range dependent similarity as well as the nonstandard concept of nonsymmetry. The resulting network is a SOFM, that is based on local and global similarity and still provides semantically meaningful weight access to realize a diagnosis process as the one reviewed. The results demonstrated the applicability of the proposed schemes.

Currently, models with increased topology preserving properties and dynamic learning abilities are being developed in order to create diagnosis systems with more flexibility and adaptability. Future effort will be concerned about introducing situation dependent similarity, extending the explanatory facilities of the classification and test selection components as well as integrating the model in hybrid planning and decision making systems.

References

[Bez81] J.C. Bezdek. *Pattern recognition with fuzzy objective function algorithms.* Plenum Press, 1981.

[BJ89] L. Becker and K. Jazayeri. A connectionist approach to case-based reasoning. In *Proccedings DARPA Workshop on Case-based Reasoning.* Morgan Kaufmann, 1989.

[CF86] M. Cottrell and J.-C. Fort. A stochastic model of retinotopy. *Biological Cybernetics*, 53:405–411, 1986.

[EOS92] E. Erwin, K. Obermayer, and K. Schulten. Self-organizing maps: ordering, convergence properties and energy functions. *Biological Cybernetics*, 67:47–55, 1992.

[Han88] P.J.B. Hancock. Data representation in neural nets: an empirical study. *Connectionist Models Summer School*, 1988.

[KG88] A. Kaufmann and M.M. Gupta. *Fuzzy Mathematical Models in Engineering and Management Science*. North-Holland, 1988.

[Koh89] T. Kohonen. *Self-Organization and Associative Memory*. Springer, 3rd edition, 1989.

[Koh90] T. Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990.

[PC92] W. Pedrycz and H.C. Card. Linguistic interpretation of self-organizing maps. In *IEEE Int. Conference on Fuzzy Systems*, pages 371–378, 1992.

[Rah95] J. Rahmel. Similarity-based self-organized clustering. In *EPIA 95: Workshop Fuzzy Logic and Neural Networks in Engineering*, 1995.

[RK89] H. Ritter and T. Kohonen. Self-organizing semantic maps. *Biological Cybernetics*, 61:241–254, 1989.

[RW93] J. Rahmel and A.v. Wangenheim. The KoDiag System: Case-based Diagnosis with Kohonen Networks. In P.J.G. Lisboa and M.J. Taylor, editors, *Proc. of the Workshop on Neural Network Applications and Tools*. Computer Society Press, 1993.

[RW94] J. Rahmel and A.v. Wangenheim. KoDiag: A Connectionist Expert System. In *International Symposium on Integrating Knowledge and Neural Heuristics*, Pensacola, FL, 1994.

[Thr89] P. Thrift. A neural network model for case-based reasoning. In *Proceedings DARPA Workshop on Case-based Reasoning*. Morgan Kaufmann, 1989.

[Ult92] A. Ultsch. Self-organizing neural networks for knowledge acquisition. In Neumann B., editor, *Proc. ECAI*, pages 208–210, 1992.

[Wes91] S. Wess. PATDEX/2 - ein System zum adaptiven, fallfokussierenden Lernen in technischen Diagnosesituationen. Technical report, SWP-91-01, University of Kaiserslautern, 1991.