TEKNILLINEN KORKEAKOULU
Tietotekniikan osasto
Tietotekniikan koulutusohjelma

Process Modeling Using the Self-Organizing Map

Jaakko Hollmén

Diplomityö on tehty informaatiotekniikan syventymiskohteessa
Työn valvoja apulaisprofessori Olli Simula
Työn ohjaaja DI Pirkka Myllykoski

Otaniemi 15.2.1996

TEKNILLINEN KORKEAKOULU **DIPLOMITYÖN TIIVISTELMÄ**

Tekijä ja työn nimi:

Jaakko Hollmén
Process Modeling Using the Self-Organiznig Map
(Prosessien mallitus itseorganisoivan kartan avulla)

Päivämäärä: 15.2.1996 **Sivumäärä:** 50

Osasto: Tietotekniikan osasto
Professuuri: Tik-61 Tietojenkäsittelytekniikka (informaatiotekniikka)

Valvoja: Apulaisprofessori Olli Simula
Ohjaaja: DI Pirkka Myllykoski

Prosessin optimoinnin motivaationa ovat taloudelliset kannusteet. Tuotannollisessa ympäristössä, jossa kyseessä ovat suuret tuotannon volyymit, jopa pienet prosessiparannukset voivat tuoda suurta taloudellista hyötyä.
Tässä työssä esitetään tapa mallittaa teollista valmistusprosessia prosessimittauksiin perustuen. Mittauksina käytetään sisääntulevan raaka-aineen ominaisuuksia, itse prosessiparametrien asetuksia tuotannon aikana sekä lopputuotteen laatuominaisuuksia. Tätä dataa käytetään keinotekoisen hermoverkon opetuksessa. Hermoverkot ovat adaptiivisia malleja, jotka oppivat datasta ja joilla on kyky yleistää.
Prosessin malli muodostetaan itseorganisoivan kartan avulla. Itseorganisoiva kartta on opetuksessa käytetyn datan epälineaarinen regressiomalli. Tätä mallia voidaan esimerkiksi käyttää laatuparametrien ennustamisessa annetuilla prosessiasetuksilla sekä prosessiparametrien muutosten aiheuttamia vipuvaikutuksia.
Työssä esitetään myös malliin perustuva prosessisimulaattori, jonka avulla voidaan helpottaa prosessissa tehtyjen muutosten aiheuttamia vipuvaikutusten tutkimista.
Itseorganisoiva kartta jakaa mittausavaruuden osiin. Yksinkertaisia malleja voidaan sovittaa dataan, joka kuuluu tällaiseen paikalliseen alueeseen.
Työ on valmistunut osana Teknologian kehittämiskeskuksen TEKES:n "Oppivien ja älykkäiden järjestelmien sovellukset" -teknologiaohjelmaa. Työ on tehty Teknillisen korkeakoulun Informaatiotekniikan laboratoriossa, ja sen ovat rahoittaneet TEKES, Rautaruukin tutkimuskeskus sekä Rautaruukin ohutlevyryhmä Hämeenlinnassa.

Avainsanat: prosessi, neuroverkko, itseorganisoiva kartta, mallitus.

HELSINKI UNIVERSITY OF TECHNOLOGY **ABSTRACT OF THE MASTER'S THESIS**

Author and name of the thesis:
Jaakko Hollmén
Process Modeling Using the Self-Organizing Map

Date: 15.2.1996 **Number of pages:** 50

Department: Department of Computer Science
Professorship: Tik-61 Computer Sciences (Information Sciences)

Supervisor: Associate professor Olli Simula
Instructor: M.Sc. Pirkka Myllykoski

Process optimization is largely motivated by economic incentives. In a production environment, where large production volumes are involved, even small improvements can result in large economic gains.
In this work, way to model an industrial production process based on process measurements is presented. The measurements include the raw material characteristics, the process parameter settings during the production and the quality characteristics of the end product. This data is used for training an artficial neural network. The artificial neural networks are adpative models that learn from data and have the capability to generalize.
The process model is built with the Self-Organizing Map (SOM). The Self-Organizing Map is a non-linear regression model of the training data. This model can, for example, be used in predicting quality parameters with given process settings and in investigating the leverage effects of the process parameter changes.
In this work, a process simulator tool based on the model is also presented, with which the investigation of leverage effects is facilitated.
The Self-Organizing Map partitions the measurement space into local regions. Simple models can be fitted to the data belonging to a local region.
The work is part of the Technology Development Centre TEKES project "Applications of learning and intelligent systems". It has been carried out in the Laboratory of Computer and Information Science in Helsinki University of Technology, and it has been funded by TEKES, Rautaruukki Research Center and Rautaruukki thin sheet division in Hämeenlinna.

Keywords: process, neural network, Self-Organizing Map, modeling.

Contents

Glossary of terms and abbreviations

ANN	Artificial Neural Network
BP	Back-Propagation algorithm
BMU	Best-Matching Unit
MLP	Multi-Layer Perceptron
PCA	Principal Component Analysis
KLT	Karhunen-Loève Transform
RBF	Radial Basis Function
SOM	Self-Organizing Map
$\alpha(t)$	adaptation gain value
c	index of the best matching unit
$\mathbf{C_x}$	covariance matrix of the random variable $\mathbf{x}$
e_i	ith eigenvector
i	unit index
$h_{ci}(t)$	neighborhood kernel function
λ_i	ith eigenvalue
$\mathbf{m}_i(t), \mathbf{m}_i$	weight vector of the unit i
$m_{i_k}(t)$, m_{i_k}	component k of the weight vector $\mathbf{m}_i$
$\mu_\mathbf{x}$	mean of the random variable $\mathbf{x}$
N_c	neighborhood of the winner node c
r_k	location vector inside the array of neurons
$\sigma(t)$	neighborhood kernel width function
t	time variable
$\mathbf{x}(t), \mathbf{x}$	measurement vector
$x_k(t), x_k$	component k of the measurement vector $\mathbf{x}$

Preface

This study was carried out in the Laboratory of Computer and Information Science in Helsinki University of Technology. The work is a part of a national Technology Development Centre (TEKES) program "Applications of learning and intelligent systems".

I would like to thank my supervisor Olli Simula and instructor Pirkka Myllykoski for their support. I wish also to thank all the people in the Laboratory of Computer and Information Science and in the Neural Networks Research Centre for their help and support. Worth a special mention are Jari Kangas and Aapo Hyvärinen, who helped me during this work, although it was not one of their duties.

TEKES, Rautaruukki Research Center as well as Rautaruukki thin sheet division in Hämeenlinna financed this work and the process data was provided by Rautaruukki. I wish to thank them for financing this work.

I would also like to thank my family for all the support during my studies in Helsinki University of Technology as well as in Royal Institute of Technology in Stockholm.

In Otaniemi, February 15, 1996

Jaakko Hollmén

Chapter 1

Introduction

1.1 Background

Process optimization and control are largely motivated by economic incentives. This is especially the case when large volumes are involved in production. Small improvements in the process can result in large gains. Also, in the competitive market situation, continuous improvement is necessary to be able to maintain and improve the market position.

How can a complex process be optimized? It is necessary to understand the functioning of the process before one can proceed to optimize it. Modeling a given process aims at understanding the functioning the process and the relationships between the process variables. How should the system be optimized? How can one avoid causing negative side effects in one part of the process (and eventually global loss) while optimizing another, in other words, how to avoid suboptimizing? These are some of the questions one is faced with when trying to optimize a process.

In this study, methods are presented with which one can learn about the process characteristics with the aid of a model created by a neural network from process measurements. Data of the incoming raw material, process parameter settings and end product characteristics from individual products are used for building a non-linear regression model of the measurement data.

In particular, an artificial neural network called the Self-Organizing Map is used. Neural networks have been quite promising in complex application areas where traditional methods have failed. Due to their inherently non-linear nature, they can handle much more complex situations than the traditional methods.

The methods presented in this work give a good starting point to a process modeling and optimization effort.

1.2 Organization of this work

This chapter provides some background information and motivation for this study. The rest of the report is divided roughly into two parts. Chapters 2 to 4 include the theoretical foundation on which the case study in Chapter 5 is built on. In Chapter 6 conclusions are presented.

Chapters on theoretical aspects of this work are not necessarily self-contained, but should offer the reader somewhat thorough and clear presentation of the main principles. Especially, Chapter 2 on the Self-Organizing Map is intended to give a thorough discussion of the basic principles. For further details, several references are listed in the text.

Chapter 2

Self-Organizing Map

2.1 Artificial neural networks

Artificial neural networks are adaptive models that can learn from the data and generalize things learned. They extract the essential characteristics from the numerical data as opposed to memorizing all of it. This offers a convenient way to reduce the amount of data as well as to form a implicit model without having to form a traditional, physical model of the underlying phenomenon. In contrast to traditional models, which are *theory-rich and data-poor*, the neural networks are *data-rich and theory-poor* in a way that a little or no a priori knowledge of the problem is present [8]. Neural networks can be used for building mappings from inputs to ouputs of these kind of black boxes. The behavior of a black box system is not usually known. This is illustrated in Figure 2.1. These kind of systems occur often in practice.

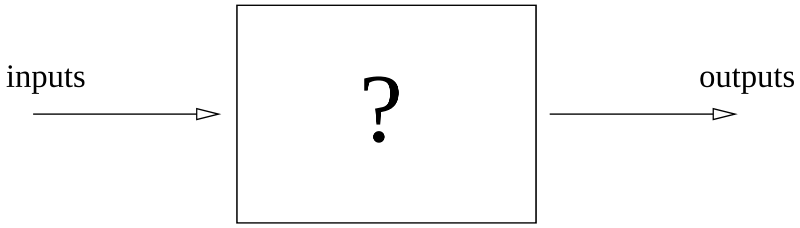


Figure 2.1: Black box

Neural networks are models as such. These models can be used to characterize the general case of the phenomenon at hand giving us the ideas how the phenomenon behaves in practice.

Artificial neural networks or shortly neural networks have been quite promising in offering solutions to problems, where traditional models have failed or are very complicated to build. Due to the non-linear nature of the neural networks, they are able to express much more complex phenomena than some linear modeling techniques.

Kohonen divides artificial neural networks into three categories [20]:

- Signal transfer networks
- State transition networks
- Competitive learning networks

In signal transfer networks, the input signal is transformed into an output signal. The signal traverses the network and undergoes a signal transformation of some kind. The network has usually a set of pre-defined basis functions, which are parametrized. The learning in these networks corresponds to changing parameters of these basis functions. Some examples are the multi-layer perceptron (MLP) networks that are taught with error back propagation algorithm (BP) and radial basis function (RBF) networks. More about these network models can be found in textbooks, for example in [3], [12].

In state transition networks the dynamic behavior of the network is essential. Given an input, the network converges to a stable state, which, hopefully, is a solution to a problem presented to it. Examples are Hopfield networks and Boltzmann machines. See [12] for reference.

In competitive learning networks, or self-organizing networks, all the neurons of the network receive the same input. The cells have lateral competition and the one with most activity "wins". Learning is based on the concept of winner neurons. A representative example of a network based on competitive learning is the Self-Organizing Map. The monograph by Kohonen [20] is the most complete book about this particular network model.

Learning in artificial neural networks is done in terms of adaptation of the network parameters. Network parameters are changed according to pre-defined equations called the learning rules. The learning rules may be derived from pre-defined error measures or may be inspired by biological systems. An example of an error measure in a network based on supervised learning could be the squared error between the output of the model and the desired output. This requires knowledge of the desired value for a given input. Learning rules are written so that the iterative learning process minimizes the error measure. Minimization might be performed by gradient descent optimization methods, for instance. In the course of learning, the residual between the model output and the desired output decreases and the model learns the relation between the input and the output.

The training must be stopped at the right time. If training continues for too long, it results in overlearning. Overlearning means that the neural network extracts too much information from the individual cases forgetting the relevant information of the general case.

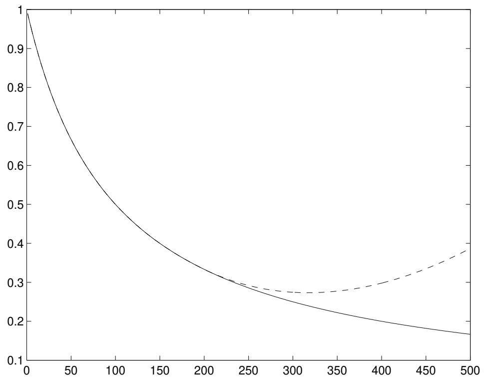


Figure 2.2: The model residual versus time for the training and the testing set

In the Figure 2.2 we can see two different curves. The difference between the network output and desired output, or the model residual is plotted as a function of training time. We can see that the model residual decreases for the training set marked with a solid line but starts to increase for the testing set marked with the dashed line. When the network starts to learn the characteristics on individual samples rather than the characteristics of the general phenomenon, the model residual for the testing set starts to increase. The model is departing from the general structure of the problem to learning about the individual cases instead.

Usually, the neural network performance is tested with a testing set which is not part of the training set. The testing set can be seen as the representative cases of the general phenomenon. If the network performs well on the testing set, it can be expected to perform well on the general case, as well.

Cross-validation methods can also be used to avoid overlearning. In cross-validation, we switch the places of the training set and the testing set and compare the performance of the resulting networks.

It is essential to understand the characteristics of a particular neural network model before using it. In this way, one can avoid many pitfalls of neural networks.

In the next section, the attention is on a particular neural network model called the Self-Organizing Map.

2.2 Self-Organizing Map (SOM)

The Self-Organizing Map is one of the most popular neural network models. It belongs to the category of competitive learning networks. The Self-Organizing Map is based on unsupervised learning, which means that no human intervention is needed during the learning and that little needs to be known about the characteristics of the input data. We could, for example, use the SOM for clustering data without knowing the class memberships of the input data. The SOM can be used to detect features inherent to the problem and thus has also been called SOFM, the Self-Organizing Feature Map.

The Self-Organizing Map was developed by professor Kohonen [20]. The SOM has been proven useful in many applications [22]. For closer review of the applications published in the open literature, see section 2.3.

The SOM algorithm is based on unsupervised, competitive learning. It provides a topology preserving mapping from the high dimensional space to map units. Map units, or neurons, usually form a two-dimensional lattice and thus the mapping is a mapping from high dimensional space onto a plane. The property of topology preserving means that the mapping preserves the relative distance between the points. Points that are near each other in the input space are mapped to nearby map units in the SOM. The SOM can thus serve as a cluster analyzing tool of high-dimensional data. Also, the SOM has the capability to generalize. Generalization capability means that the network can recognize or characterize inputs it has never encountered before. A new input is assimilated with the map unit it is mapped to.

The Self-Organizing Map is a two-dimensional array of neurons:

$$\mathbf{M} = \{\mathbf{m}_1, ..., \mathbf{m}_{pxq}\}$$

This is illustrated in Figure 2.3. One neuron is a vector called the codebook vector

$$\mathbf{m}_i = [m_{i1}, ..., m_{in}]$$

This has the same dimension as the input vectors (n -dimensional). The neurons are connected to adjacent neurons by a neighborhood relation. This dictates the topology, or the structure, of the map. Usually, the neurons are connected to each other via rectangular or hexagonal topology. In the Figure 2.3 the topological relations are shown by lines between the neurons.

One can also define a distance between the map units according to their topology relations. Immediate neighbors (the neurons that are adjacent) belong to the neighborhood N_c of the neuron $\mathbf{m}_c$. The neighborhood function should be a decreasing function of time: $N_c = N_c(t)$. Neighborhoods of different sizes in a hexagonal lattice are illustrated in Figure 2.4. In the smallest hexagon, there

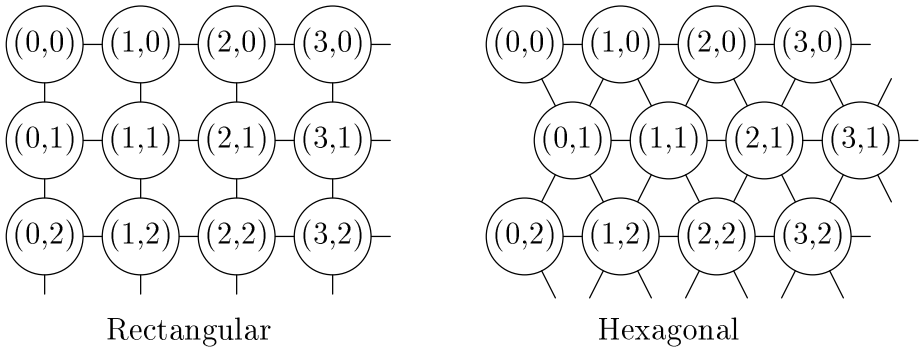


Figure 2.3: Different topologies

are all the neighbors belonging to the smallest neighborhood of the neuron in the middle belonging to a hexagonal lattice. The topological relations between the neurons are left out for clarity.

In the basic SOM algorithm, the topological relations and the number of neurons are fixed from the beginning. This number of neurons determines the scale or the granularity of the resulting model. Scale selection affects the accuracy and the generalization capability of the model. It must be taken into account that the generalization and accuracy are contradictory goals. By improving the first, we lose on the second, and vice versa.

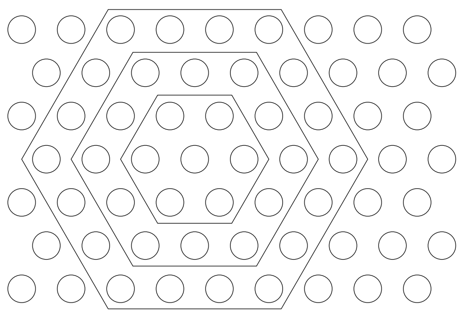


Figure 2.4: Neighborhood of a given winner unit

2.2.1 Data preprocessing

The data fed to a SOM includes all the information that a network gets. If erroneous data is fed to the SOM, the result is also erroneous or of bad quality. Thus, Self-Organizing Map, as well as the other neural network models, follow the "garbage in - garbage out" principle. This is the motivation for data preprocessing. Especially, when analyzing real-life data, preprocessing is of paramount importance.

Focusing on a subset of data

If we are interested in a certain aspect or the subset of the input data, we should naturally use only that portion of the data. Almost needless to say, certain analyses focus on a totally different subset than the others.

Removing erroneous data

Errors in the data must be removed. If the data is downloaded from a database as a query, the result is likely to include erroneous data because of the lack of database integrity. Erroneous data must be filtered using a priori knowledge of the problem domain and common sense. For example, in databases, missing values are usually presented as zeros. Zeros are typical errors due to the lack of database integrity. These kind of errors show up in the probability density function presentation as peaks at zero possibly outside the normal range of the variable. In the case of uncertainty, these kind of values can be replaced with "don't care" values. In training of a SOM, input vectors with missing values can be used [36]. Another approach is to remove the vectors from the training set if they have missing values. This has the negative side effect of reducing the training set size.

Data encoding

If the data is coded in a non-metric scale, i.e. the metric distance can not be used as a measure of similarity, the coding must be transformed. Groupings and class memberships are examples of this kind of coding. Having groups 1 to 10, we can not say that the group number 9 is more similar to the group number 10 than the group number 1. N groups can be divided into one-of-n coding using N components. For group N, the Nth component is 1, others are 0.

Measurement must be made quantifiable, because the Euclidean distance is commonly used as a measure of similarity. Coding must be in harmony with the similarity measure used. Symbolic data cannot be processed with the SOM as such, but can be transformed to a suitable form. See [34] for reference.

Scaling

It is common that the components of the input data are scaled to have unit variance.

$$Var(X) = 1$$

This can be achieved by dividing the components by the square roots of their corresponding variances. This assures that for each component, the difference between two samples contribute approximately an equal amount to the summed distance measure between an input sample and codebook vector.

Because the similarity measure usually loses identity of component differences via a summation, or treats all components equally, the components must contribute approximately as much to the similarity measure. Otherwise, a component with large variance would shadow components with small variance and thus only the components with large variance would contribute to the distance measure used as a similarity measure.

2.2.2 Initialization

Kohonen presents three different types of network initializations [20]: random initialization, initialization using initial samples and linear initialization.

Random initialization

Random initialization means simply that random values are assigned to codebook vectors. This is the case if nothing or little is known about the input data at the time of the initialization.

Initialization using initial samples

Initial samples of the input data set can be used for codebook vector initialization. This has the advantage that the points automatically lie in the same part of the input space with the data.

Linear initialization

One initialization method takes advantage of the principal component (PCA) analysis of the input data. Principal component analysis is discussed more closely in Chapter 3. The codebook vectors are initialized to lie in the same input space that is spanned by two eigenvectors corresponding to the largest eigenvalues of the input data. This has the effect of stretching the Self-Organizing Map to the same orientation as the data having the most significant amounts of energy.

2.2.3 Training

Training is an iterative process through time. It requires a lot of computational effort and thus is time-consuming. The training consists of drawing sample vectors from the input data set and "teaching" them to the SOM. The teaching consists of choosing a winner unit by the means of a similarity measure and updating the values of codebook vectors in the neighborhood of the winner unit. This process is repeated a number of times.

In one training step, one sample vector is drawn randomly from the input data set. This vector is fed to all units in the network and a similarity measure is calculated between the input data sample and all the codebook vectors. The best-matching unit (BMU) is chosen to be the codebook vector with greatest similarity with the input sample. The similarity is usually defined by means of a distance measure. For example in the case of Euclidean distance the best-matching unit is the closest neuron to the sample in the input space. The Euclidean norm of the vector $\mathbf{x}$ is defined as

$$||\mathbf{x}|| = \sqrt{\sum_{i=1}^{n} {\mathbf{x}_i}^2}$$

Then, we can define the Euclidean distance in terms of the Euclidean norm of the difference between two vectors:

$$d_E(\mathbf{x}, \mathbf{y}) = ||\mathbf{x} - \mathbf{y}||$$

The best-matching unit, usually noted as $\mathbf{m}_c$, is the codebook vector that matches a given input vector $\mathbf{x}$ best. It is defined formally as the neuron for which

$$||\mathbf{x} - \mathbf{m}_c|| = \min_i \{||\mathbf{x} - \mathbf{m}_i||\}$$

After finding the best-matching unit, units in the SOM are updated. During the update procedure, the best-matching unit is updated to be a little closer to the sample vector in the input space. The topological neighbors of the best-matching unit are also similarly updated. This update procedure streches the BMU and its topological neighbors towards the sample vector.

In the Figure 2.5 we see an illustration of the update procedure. The codebook vectors are situated in the crossings of the solid lines. The topological relationships of the SOM are drawn with lines. The input fed to the network is marked by an `x` in the input space. The best-matching unit, or the winner neuron is the codebook vector closest to the sample, in this example the codebook vector in the middle above `x`. The winner neuron and its topological neighbors are updated by moving them a little towards the input sample. The neighborhood in this case consists of the eight neighboring units in the figure. The updated network is shown in the same figure with dashed lines.

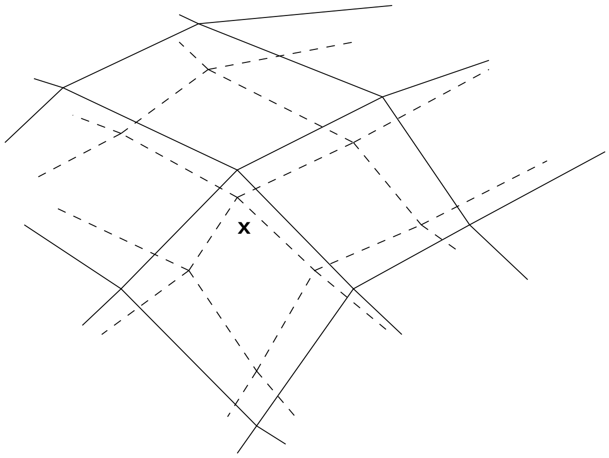


Figure 2.5: updating the best matching unit and its neighbors

The computational effort consists of finding a best-matching unit among all the neurons and updating the codebook vectors in the neighborhood of the winner unit. If the neighborhood is large, there are a lot of codebook vectors to be updated. This is the case in the beginning of the training process, where it is recommended to use large neighborhoods. In the case of large networks, relatively larger portion of the time is spent looking for a winner neuron. All these considerations depend on the time spent on each of these phases depending on particular software and hardware used.

Update rule

By this update procedure described above, the net forms an elastic net that during learning folds onto the "cloud" formed by the input data. The codebook vectors tend to drift there where the data is dense, while there tends to be only a few codebook vectors where data is sparsely located. In this manner, the net tends to approximate the probability density function of the input data [20].

The Self-Organizing Map update rule for a unit $\mathbf{m}_i$ is the following:

$$\mathbf{m}_i(t+1) = \mathbf{m}_i(t) + h_{ci}(t)[\mathbf{x}(t) - \mathbf{m}_i(t)]$$

where t denotes time. This is, as mentioned above, a training process through time. The $\mathbf{x}(t)$ is the input vector drawn from the input data set at time t. h_{ci} is a non-increasing neighborhood function around the winner unit $\mathbf{m}_c$. More on the subject of the neighborhood function in the next section.

Neighborhood function

The neighborhood function includes the learning rate function $\alpha(t)$ which is a decreasing function of time and the function that dictates the form of the neighborhood function. The form of the latter function also determines the rate of change around the winner unit. The neighborhood function can be written as

$$h_{ci}(t) = \alpha(t) \cdot \exp(-\frac{\|\mathbf{r}_i - \mathbf{r}_c\|^2}{2\sigma(t)^2})$$

in the case of the Gaussian neighborhood function around the winner neuron $\mathbf{m}_c$.

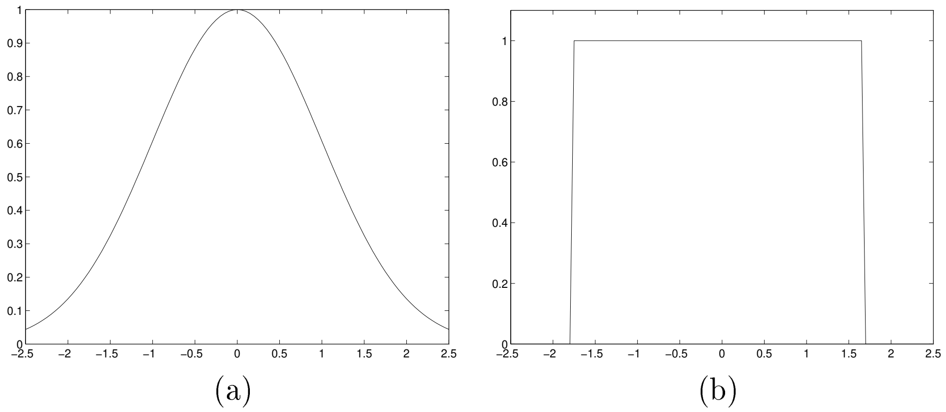
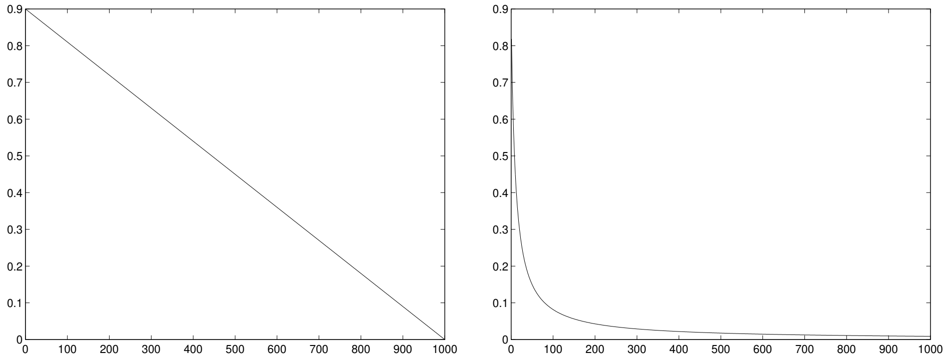


Figure 2.6: Neighborhood function values

A variety of neighborhood functions can be used. We can constrain the neigbourhood function to be non-increasing around the winner unit $\mathbf{m}_c$. Thus, the neighborhood function can also be constant around the winner unit. One choice for a neighborhood function is to use a Gaussian kernel around the winner neuron as described above. This is computationally demanding as the exponential function has to be calculated, but can well be approximated by the "bubble" neighborhood function. The bubble neighborhood function is a constant function in the defined neighborhood of the winner neuron, that is, every neuron in the neighborhood is updated the same proportion of the difference between the neuron and the presented sample vector. The bubble neighborhood function is a good compromise between the computational cost and the approximation of the Gaussian.

In the Figure 2.6 the two used forms of the neighborhood function are illustrated. In the left, we see the form of a Gaussian (a), on the right the bubble function (b).



(a) (b)

Figure 2.7: Learning rates as functions of time

Learning rate

Learning rate is a decreasing function of time. Two forms that are commonly used are a linear function of time and a function that is inversely proportional to the time t. These are illustrated in the Figure 2.7. Linear alpha function (a) decreases to zero linearly during the learning from its initial value whereas the inverse alpha function (b) decreases rapidly from the initial value Both the functions in the Figure have the initial value of 0.9. The initial values for α must be determined. Usually, when using a rapidly decreasing inverse alpha function, the initial values can be larger than in the linear case. The learning is usually performed in two phases. On the first round relatively large initial alpha values are used ($\alpha = 0.3, ..., 0.99$) whereas small initial alpha values ($\alpha = 0.01, ..., 0.1$) are used during the other round. This corresponds to first tuning the SOM approximately to the same space than the inputs and then fine-tuning the map. There are several rules-of-thumb for picking suitable values. These have been found through experiments and can be found in the monograph by Kohonen [20].

Alpha values are defined to be

$$\alpha(t) = \alpha(0)(1.0 - t/rlen)$$

for the linear case and

$$\alpha(t) = C\alpha(0)/(C + t)$$

for the inverse function. C can be, for example $C = rlen/100$. $rlen$ is the running length of the training, or number of samples fed to the network. These are the values used in the programming package SOM_PAK [21].

By choosing a suitable initial learning rate and a suitable form for the learning rate function, we can considerably affect the result.

2.2.4 Visualization

The Self-Organizing Map is an approximation to the probability density function of the input data [20]. It can be used in visualization [13]. In the next sections, we present common ways to visualize the Self-Organizing Map.

U-matrix

U-matrix (unified distance matrix) representation of the Self-Organizing Map [39] visualizes the distances between the neurons. The distance between the adjacent neuons is calculated and presented with different colorings between the adjacent nodes. A dark coloring between the neurons corresponds to a large distance and thus a gap between the codebook values in the input space. A light coloring between the neurons signifies that the codebook vectors are close to each other in the input space. Light areas can be thought as clusters and dark areas as cluster separators. This can be a helpful presentation when one tries to find clusters in the input data without having any a priori information about the clusters.

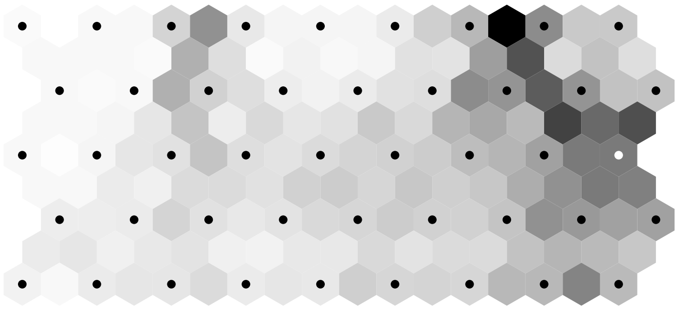


Figure 2.8: U-matrix representation of the Self-Organizing Map

In the Figure 2.8 we can see the neurons of the network marked as black dots. The representation reveals that these are a separate cluster in the upper right corner of this representation. The clusters are separated by a dark gap. This result was achieved by unsupervised learning, that is, without human intervention. Teaching a SOM and representing it with the U-matrix offers a fast way to get insight of the data distribution.

Sammon's mapping

Sammon's mapping [27] is a non-linear mapping that maps a set of input points on a plane trying to preserve the relative distance between the input points approximately. It can be used to visualize a SOM by mapping the values of codebook vectors on a plane. Furthermore, the topological relations can be drawn using lines between neighboring neurons to enhance the net-like look. Sammon's mapping can be applied directly to data sets, but is computationally very intensive. The SOM quantizes the input data to a small number of codebook vectors, so the burden of computation is not so heavy.

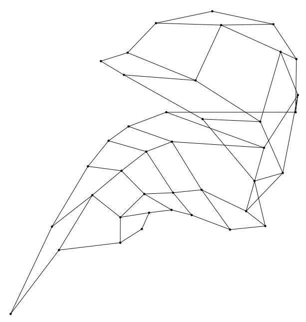


Figure 2.9: Sammon's mapping of the codebook vectors of SOM

The Figure 2.9 is an illustration of the Sammon's mapping. The form of the Sammon's mapping can be considered as a hint of the form of the set of codebook vectors and thus the input data.

Component plane representation

By component plane representation we can visualize the relative component distributions of the input data. Component plane representation can be thought as a sliced version of the Self-Organizing Map. Each component plane has the relative distribution of one data vector component. In this representation, dark values represent relatively small values while white values represent relatively large values. By comparing component planes we can see if two components correlate. If the outlook is similar, the components strongly correlate.

This is a clear visualization of correlation between the vector components. For example, there is correlation between the components (j), (k) and (l), for example. By picking a same neuron in each plane (in the same location), we could assemble the relative values of a codebook vector of the network.

2.2.5 Validation

We can create models as we like, but before a model can be reliably used, it must be validated. Validation means that the model is tested so that we can be sure that the model gives us reasonable and accurate values. What we mean by this depends largely on the application and our requirements.

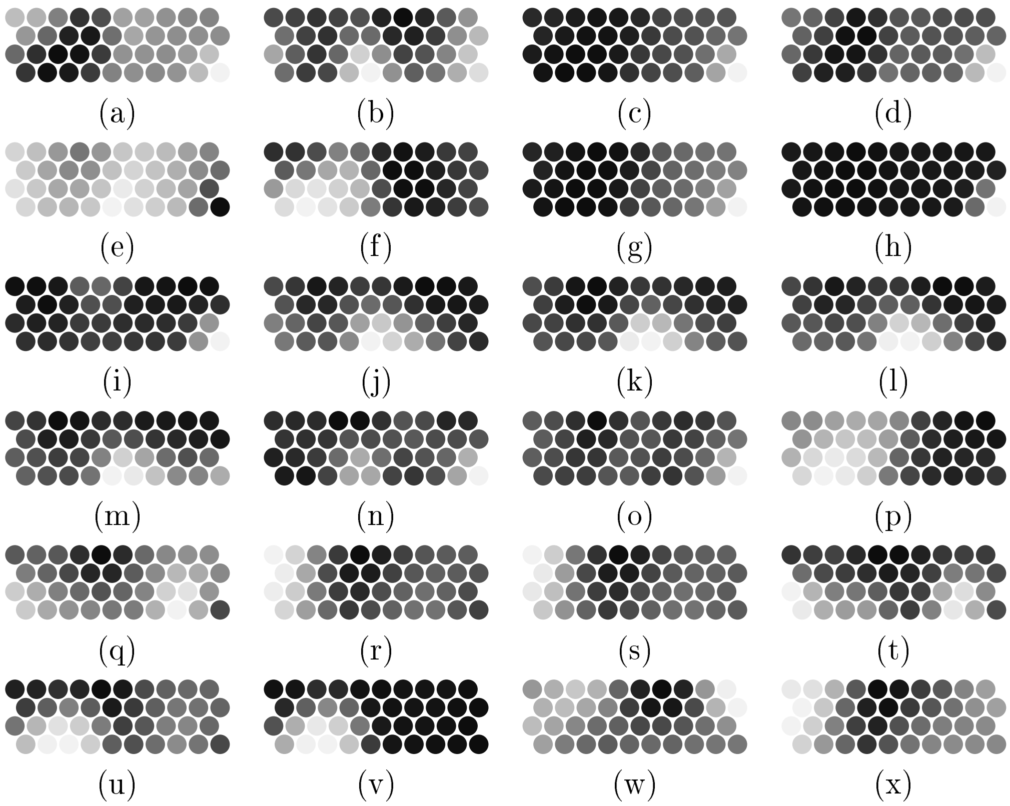


Figure 2.10: The component plane representation of the SOM

The validation must be done by using an independent test set. The independent test set is a set similar to the input set but not a part of the training set. The testing set can be seen as a representative of the general case.

Quantization error of an input vector is defined as the difference between the input vector and the closest codebook vector. For a set of input vectors, we can reflect on the similarity of the input data set and the SOM by investigating the distribution of the quantization errors. The range of quantization error tells the smallest and the largest amount of error.

2.3 Applications

Self-Organizing Map has proven useful in many technical applications. Many applications have been published in the open literature. Representative examples can be found in an article by Kohonen et al. [22].

In an industrial setting, the SOM has been applied, for example, in process and machine state monitoring [1] [2] [4] [7] [11] [18], fault identification [41] and in robot control [35].

In process and systems analysis, the use of SOM is well motivated [22]. The number of state variables may exceed the number of measurements by an order of magnitude. Also, the state variables may be non-linearly related. In this case, the analystical model of the plant in question would not be identifiable from the measurements.

In fault diagnosis, the SOM has two functions. Firstly, the SOM can be used in detecting the fault and in identifying it. One can detect faults even if there are no measurements of the faulty states by investigating the quantization error between the SOM and the measurements. If the quantization error exceeds a pre-defined limit, a faulty state has occurred. If one needs also to identify faults, representative examples of the faulty situations must have been recorded.

In an application, measurements were made from a computer system in a network environment. The system was measured in terms of utilization rates of the central processing unit and traffic volumes in the network. It was not known a priori what the characteristic states in the operation of the system would be. The SOM was used in clustering data measured from the system, or to form a representation of the characteristic states. Such a representation would be useful in monitoring the current state of the system.

In the Figure 2.11 the U-matrix representation and the Sammon's mapping of the SOM is presented. These representations reveal the characteristic states of the system. The SOM can in this way be used in monitoring the current state of the system. This kind of monitoring application is based on the ability of SOM to represents the density function of input data on a low-dimensional display [22].

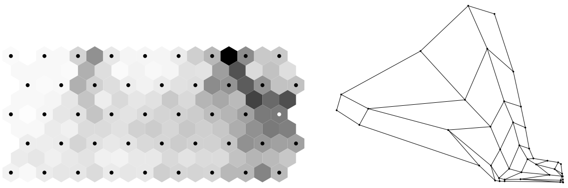
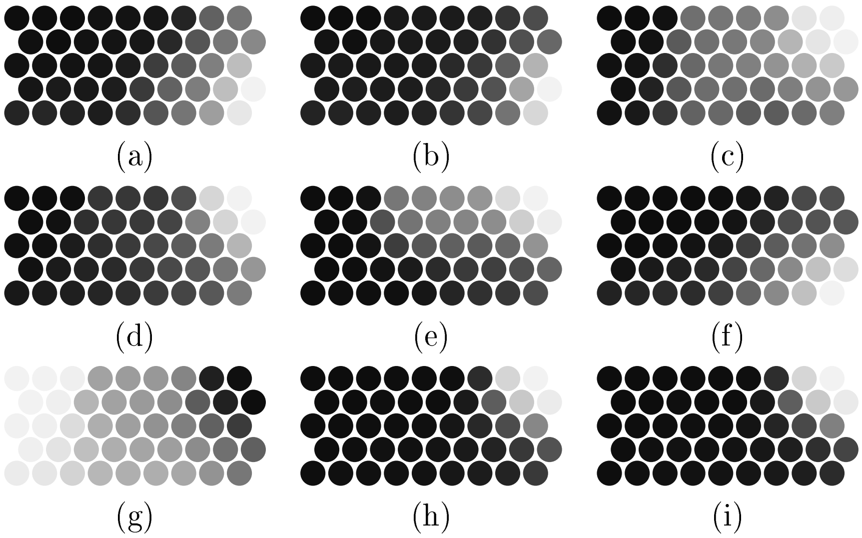


Figure 2.11: U-matrix and Sammon's mapping of the SOM



(a) (b) (c) (d) (e) (f) (g) (h) (i)

Figure 2.12: Relative distributions of the measurements in planes representation

The component plane representation in 2.12 shows the relative component distribution of the measurements. If something is known about the characteristic states of the system, the component planes can help in giving states descriptions, or labels. Then, a mapping from the measurement space to map units is a mapping to a name describing the state.

Chapter 3

Principal component analysis

3.1 Principal component analysis

Principal component analysis (PCA) is a classical statistical method. This linear transform has been widely used in data analysis and compression. The following presentation is adapted from [9]. Some of the texts on the subject also include [30], [31]. Principal component analysis is based on the statistical representation of a random variable. Suppose we have a random vector population $\mathbf{x}$, where

$$\mathbf{x} = (x_1, ..., x_n)^T$$

and the mean of that population is denoted by

$$\mu_{\mathbf{x}} = E\{\mathbf{x}\}$$

and the covariance matrix of the same data set is

$$\mathbf{C}_x = E\{(\mathbf{x} - \mu_{\mathbf{x}})(\mathbf{x} - \mu_{\mathbf{x}})^T\}$$

The components of $\mathbf{C}_x$, denoted by c_{ij}, represent the covariances between the random variable components x_i and x_j. The component c_{ii} is the variance of the component x_i. The variance of a component indicates the spread of the component values around its mean value. If two components x_i and x_j of the data are uncorrelated, their covariance is zero ($c_{ij} = c_{ji} = 0$). The covariance matrix is, by definition, always symmetric.

From a sample of vectors $\mathbf{x}_1, \ldots, \mathbf{x}_M$, we can calculate the sample mean and the sample covariance matrix as the estimates of the mean and the covariance matrix.

From a symmetric matrix such as the covariance matrix, we can calculate an orthogonal basis by finding its eigenvalues and eigenvectors. The eigenvectors $\mathbf{e}_i$ and the corresponding eigenvalues λ_i are the solutions of the equation

$$\mathbf{C}_x \mathbf{e}_i = \lambda_i \mathbf{e}_i, i = 1, ..., n$$

For simplicity we assume that the λ_i are distinct. These values can be found, for example, by finding the solutions of the characteristic equation

$$|\mathbf{C}_x - \lambda \mathbf{I}| = 0$$

where the $\mathbf{I}$ is the identity matrix having the same order than $\mathbf{C}_x$ and the $|.|$ denotes the determinant of the matrix. If the data vector has n components, the characteristic equation becomes of order n. This is easy to solve only if n is small. Solving eigenvalues and corresponding eigenvectors is a non-trivial task, and many methods exist. One way to solve the eigenvalue problem is to use a neural solution to the problem [30]. The data is fed as the input, and the network converges to the wanted solution.

By ordering the eigenvectors in the order of descending eigenvalues (largest first), one can create an ordered orthogonal basis with the first eigenvector having the direction of largest variance of the data. In this way, we can find directions in which the data set has the most significant amounts of energy.

Suppose one has a data set of which the sample mean and the covariance matrix have been calculated. Let $\mathbf{A}$ be a matrix consisting of eigenvectors of the covariance matrix as the row vectors.

By transforming a data vector $\mathbf{x}$, we get

$$\mathbf{y} = \mathbf{A}(\mathbf{x} - \mu_{\mathbf{x}})$$

which is a point in the orthogonal coordinate system defined by the eigenvectors. Components of $\mathbf{y}$ can be seen as the coordinates in the orthogonal base. We can reconstruct the original data vector $\mathbf{x}$ from $\mathbf{y}$ by

$$\mathbf{x} = \mathbf{A}^T \mathbf{y} + \mu_{\mathbf{x}}$$

using the property of an orthogonal matrix $\mathbf{A}^{-1} = \mathbf{A}^T$. The $\mathbf{A}^T$ is the transpose of a matrix $\mathbf{A}$. The original vector $\mathbf{x}$ was projected on the coordinate axes defined by the orthogonal basis. The original vector was then reconstructed by a linear combination of the orthogonal basis vectors.

Instead of using all the eigenvectors of the covariance matrix, we may represent the data in terms of only a few basis vectors of the orthogonal basis. If we denote the matrix having the K first eigenvectors as rows by $\mathbf{A}_K$, we can create a similar transformation as seen above

$$\mathbf{y} = \mathbf{A}_K(\mathbf{x} - \mu_{\mathbf{x}})$$

and

$$\mathbf{x} = {\mathbf{A}_K}^T \mathbf{y} + \mu_{\mathbf{x}}$$

This means that we project the original data vector on the coordinate axes having the dimension K and transforming the vector back by a linear combination of the basis vectors. This minimizes the mean-square error between the data and this representation with given number of eigenvectors.

If the data is concentrated in a linear subspace, this provides a way to compress data without losing much information and simplifying the representation. By picking the eigenvectors having the largest eigenvalues we lose as little information as possible in the mean-square sense. One can e.g. choose a fixed number of eigenvectors and their respective eigenvalues and get a consistent representation, or abstraction of the data. This preserves a varying amount of energy of the original data. Alternatively, we can choose approximately the same amount of energy and a varying amount of eigenvectors and their respective eigenvalues. This would in turn give approximately consistent amount of information in the expense of varying representations with regard to the dimension of the subspace.

We are here faced with contradictory goals: On one hand, we should simplify the problem by reducing the dimension of the representation. On the other hand we want to preserve as much as possible of the original information content. PCA offers a convenient way to control the trade-off between loosing information and simplifying the problem at hand.

As it will be noted later, it may be possible to create piecewise linear models by dividing the input data to smaller regions and fitting linear models locally to the data.

Now, consider a small example showing the characteristics of the eigenvectors. Some artificial data has been generated, which is illustrated in the Figure 3.1. The small dots are the points in the data set.

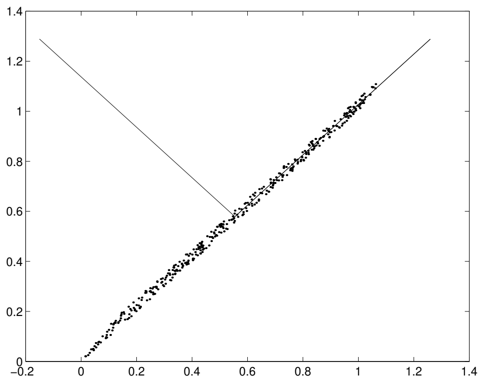


Figure 3.1: Eigenvectors of the artificially created data

Sample mean and sample covariance matrix can easily be calculated from the data. Eigenvectors and eigenvalues can be calculated from the covariance matrix. The directions of eigenvectors are drawn in the Figure as lines. The first eigenvector having the largest eigenvalue points to the direction of largest variance (right and upwards) whereas the second eigenvector is orthogonal to the first one (pointing to left and upwards). In this example the first eigenvalue corresponding to the first eigenvector is $\lambda_1 = 0.1737$ while the other eigenvalue is $\lambda_2 = 0.0001$. By comparing the values of eigenvalues to the total sum of eigenvalues, we can get an idea how much of the energy is concentrated along the particular eigenvector. In this case, the first eigenvector contains almost all the energy. The data could be well approximated with a one-dimensional representation.

Sometimes it is desirable to investigate the behavior of the system under small changes. Assume that this system, or phenomenon is constrained to a n-dimensional manifold and can be approximated with a linear manifold. Suppose one has a small change along one of the coordinate axes in the original coordinate system. If the data from the phenomenon is concentrated in a subspace, we can project this small change δ_x to the approximative subspace built with PCA by projecting δ_x on all the basis vectors in the linear subspace by

$$\delta_y = \mathbf{A}_K \delta_x$$

where the matrix $\mathbf{A}_K$ has the K first eigenvectors as rows. Subspace has then a dimension of K. δ_y represents the change caused by the original small change. This can be transformed back with a change of basis by taking a linear combination of the basis vectors by

$$\delta_x = \mathbf{A}_K{}^T \delta_y$$

Then, we get the typical change in the real-world coordinate system caused by a small change δ_x by assuming that the phenomenon constrains the system to have values in the limited subspace only.

Chapter 4

Models

4.1 General

The instinct of survival of man has driven him to investigate his surroundings. Intelligence and the ability to reason and to adapt have been crucial to the survival of man. Models can be thought of as explicit expressions of the surrounding world. Models can help us to understand our surroundings and to answer questions about it. A general definition of a model could be presented: “Model is an object (or an abstraction) that facilitates the processing of another object”.

Kohonen states in his book [20]: Model, especially an analytical one, usually consists of a finite set variables and their quantitative interactions that are supposed to describe, e.g., states and signals in a real system, often assumed to behave according to known, simplified laws of nature.

Many kinds of models exist. The model that is best suited for a specific purpose must be chosen carefully taking into account the following considerations:

- View-point on the problem
- Scale (or resolution or granularity)
- Domain of applicability

First, view-point must be decided. It is impossible to create a model that could answer all the questions. An accurate model has a restricted view on the problem. A more general model can exist on higher abstraction levels. Generality and the level of detail are contradictory: a detailed model cannot be general and a general model can not contain small details.

Secondly, scale of the model must be chosen. The problem itself does not contain any information on the scale in which the observer considers relevant, so the scale selection remains a problem of the observer. Some heuristics can be developed to pick a suitable scale, but these always contain assumptions of the observer.

Thirdly, the domain of applicability must be understood. Every model has its limitations and the model is not valid outside its scope. This restricts the use of a model.

Numerous modeling techniques have been used. These tend to be specific to a certain discipline and constrained by tradition. All model-making seem to suffer from the previously named problems.

Neural networks have been quite promising in modeling complex real-life phenomena. As mentioned earlier, they are *data-rich and theory-poor* [8] models in a sense that a few assumptions must be made to form such a model.

In the next two sections, we study ways of modeling with the SOM. First, a way to create regression models is presented. Secondly, a method is described to expand this kind of model to fit local models directly to the data.

4.2 SOM as a regression model

The SOM is a nonparametric regression model. This provides a data-driven abstraction method of the phenomenon described by input data. It is possible to study the general case by building this kind of model from many individual cases.

The codebook vectors of the SOM represent the general form of the data and quantize the input space. Along with the training, the elastic net is streched to cover the cloud of data in the input space. We can use this representation as a model or build models on this abstraction. We can visualize the SOM by drawing a U-matrix or a component plane representation described earlier in section 2.2.4. These provide us information about the correlations between components, division of data in the input space and relative distributions of the components.

The problem of scale selection must be tackled. The scale selection is the problem of determining the smallest level of detail, or the granularity. This corresponds, in a plain SOM representation, to picking a suitable number of codebook vectors. Heuristics can be developed to do this, but this always requires that assumptions are made by the observer. Training a SOM with a large number of neurons requires a lot of computational effort and one may end up modeling small details. A SOM with a small number of neurons might not in turn grasp the essential.

Suppose we train a SOM with input vectors of dimension n. SOM is then a representation of the general case with no regard to which components of the input vector are independent variables and which are dependent variables. We have not committed us to a certain relationship between the vector components or named any components as the "inputs" or the "outputs" of this relationships.

We can constrain any component of the input vector to be constant and to fetch the rest of the vector values with the aid of known values. The forecasted values are then the values of the BMU with regard to the known values of the input vector.

The credibility measure of the predicted values can be approximated by the difference between the codebook vector and the input vector.

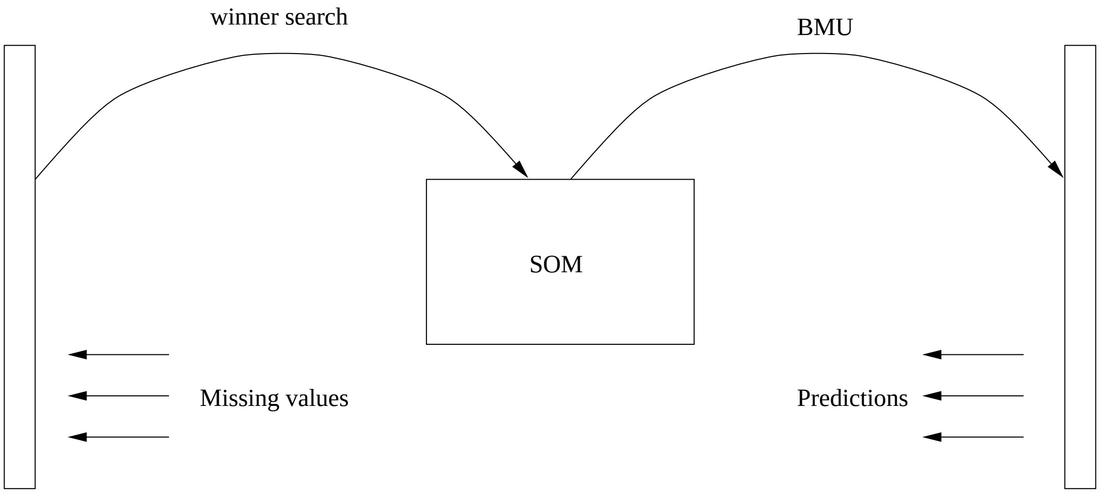


Figure 4.1: Fetch the unknown variables

A prediction can be created by seeking the best-matching unit for the a vector with unknown components with regard to the known components. The winner unit is searched with regard to the known components only. The predicted values can be fetched from the best-matching unit.

A same kind of approach was used in robot control by Ritter et al. [35]. The SOM was used as an adaptive look-up table for fetching suitable output variable values for given input variable values. Output values correspond to control actions with given input values. Output values were taught with a different learning rule. This corresponds to "picking the most suitable question for our question, and finding the corresponding answer and considering that for a final answer". Kohonen suggests that this might be the way brain operates, namely by fusing different kinds of data together [20]. In the second application in time series prediction by Walter et al. [42] the SOM was used to partition the state-space of a time-dependent system. Each state of the system was mapped to a codebook vector, and the next state was predicted by an autoregressive model of the previous values of the system. The autoregressive models were specific to a certain state-space.

This method is indeed very simple. It is not committed to any "inputs" or "outputs", but can be used to predict any wanted "input-output" relationship. It would be desirable that the number of known components would be larger than the number of unknown components.

4.3 SOM and local model fitting

The SOM representation is a generalization of the underlying data [19]. It can be used as a basis for further processing. The SOM representation is a lattice of discrete points in the n-dimensional input space. The SOM can be used to partition the input data to smaller regions by associating input data with their best-matching units. Each data point in input space has, by definition, one best matching unit. The area in the input space for which the codebook vector is the BMU is called Voronoi tesselation. Voronoi tesselations partition the input space into disjoint sets.

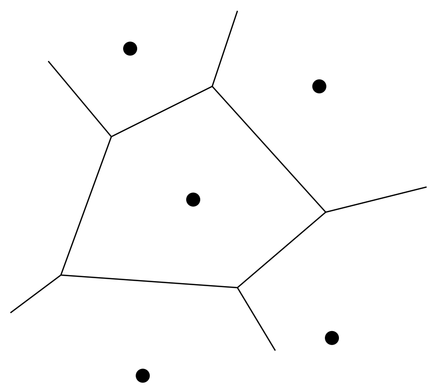


Figure 4.2: Voronoi tesselations in the input space

A model can be created by fitting a model to the data in the Voronoi tesselation. By way of doing this, one can create models that are local to the specific Voronoi tesselation. These models desrcibe the behavior of the system in this local space only. One could also combine data coming from a neuron and its neighboring units to form a larger amount of data covering a larger amount of input space thus enlarging the area of interesting operation points.

Whereas the SOM codebook vectors are local averages of the training data, PCA represents also the first-order terms of the data. By restricting the input space of PCA to one Voronoi tesselation only, one can take advantage of the non-linear elasticity of SOM and its capability to partition the input space and building linear regression models with PCA.

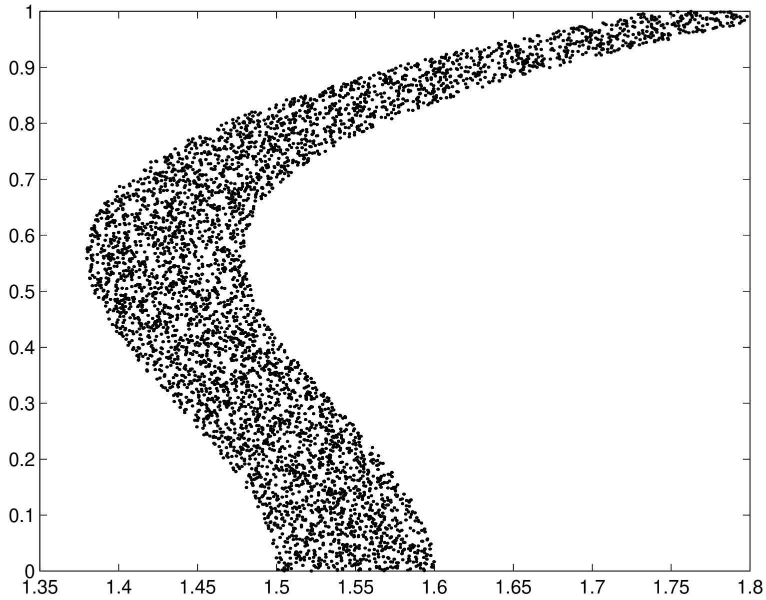


Figure 4.3: The training data used to train a SOM

In the Figure 4.3 5000 thousand artificially created codebook vectors are illustrated. These are used in training a SOM. In addition to replacing the 5000 data vectors with 10 codebook vectors that describe the original data, it partitions the input space.

The one-dimensional SOM in the Figure 4.4 quantizes the input space. Data is associated to neurons by the similarity criterion. One could build a model based on data belonging to one of the Voronoi tesselations. This kind of model would be local in nature. Neurons of the SOM are marked with small circles in the figure. The topological relationships are drawn as lines.

If the data clearly resided in a linear subspace, these local linear models could be interpreted as first derivative rules and thus be used for sensitivity analysis. Often one would like to study the behavior of a system under small changes.

The goal here is to develop methods with which one could understand the structure of the multidimensional data manifold by applying SOM to the training data and PCA to each of the Voronoi tesselations in the input space. Similar work has been reported in [5], [14], [15], [16], [17].

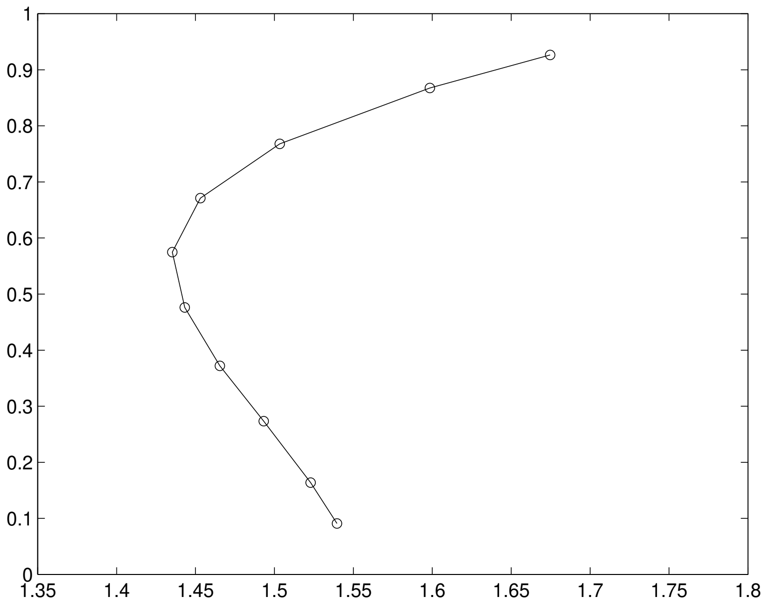


Figure 4.4: The SOM with 10 codebook vectors

Chapter 5

Case study: Rautaruukki

5.1 Problem domain

Process optimization is largely motivated by the economic incentive. Because of the large volumes involved in production, even small improvements can result in large productivity gains. Under competition, continuous improvement is also necessary to maintain and to improve a market position.

Artificial neural networks have been applied in monitoring and control of industrial processes in [1], [2], [4], [7], [10], [11], [18], [23], [24], [25], [26], [28], [29], [32], [33], [37], [38], [40], [41], [43].

Rautaruukki produces steel coils to be used, for instance, by the construction industry. The process consists of rolling operations and an annealing process. The steel rolls may also be coated with plastic or zinc.

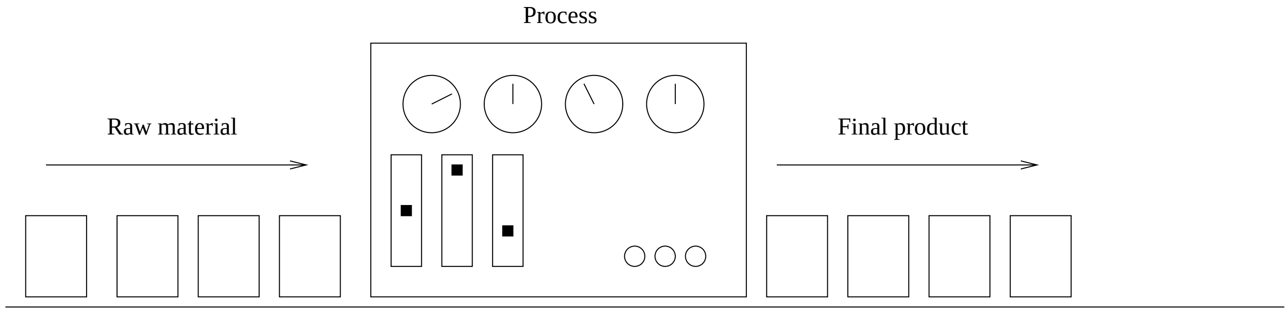


Figure 5.1: An oversimplified model of the process (black box approach)

The products leave the production line one by one. Each of these products has attributes associated with it. These attributes are the raw material charcteristics in the form of element concentrations, the process parameter settings during the processing and the end product characteristics in the form of the quality parameters. The quality parameters reflect the mechanical properties of the steel. The quality parameters are determined through standard procedures.

After steel strips have been hot rolled the strips are pickled. The strip goes through a pickling process. A pickling process is an acid bath, which removes an oxide layer from the surface of the strip. After pickling, the strip are rolled in four consecutive roll stands to make the steel strip thinner.

After rolling, the strip is coiled and warmed up to a hot temperature in a bell-shaped oven. The roll is warmed and then let to cool up during a long period of time. This part of the process is called annealing. Annealing process improves the microstructure of the steel: on a microlevel, the steel particles become ordered. After annealing, the coils are rolled with a temper mill to improve formability.

The material chain in the process is illustrated in the Figure 5.2. The incoming raw material consists of the hot rolled coils from suppliers.

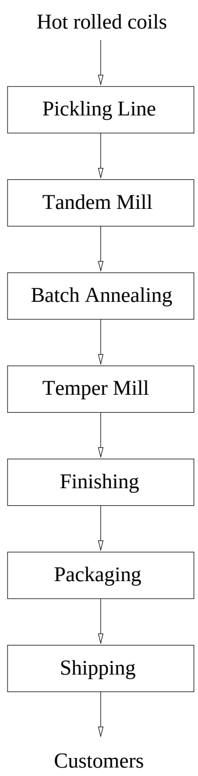


Figure 5.2: The material flow in the process

After the treatments described above, the coils are ready to be packed and shipped to the customer. The coils may also be coated with zinc or plastic. The coating process was not a part of this analysis.

5.2 Process data

The production system stores all the necessary information to identify the operations done on a per product basis. Data of the products can be retrieved from the factory database with a database query. This data is used in modeling the process.

As in the Figure 5.3 the process data describes attributes of individual products. These attributes are the attributes of the incoming raw material in the form of element concentrations, the process parameter settings during the producing of a coil and the quality parameters of the final product shipped to the customer. The model variables from the process measurements can by summed in following:

- The raw material characteristics in terms of element concentrations
- The process parameter settings during the production of a product
- The quality characteristics of a particular product

A process model is constructed using these measurements. Model can be utilized to predict quality parameter values with given raw material and process parameter settings. In this way, we can study how different process settings contribute to the end product quality.

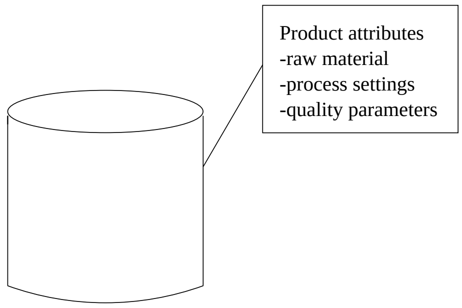


Figure 5.3: attributed product

The above mentioned parameters are gathered in a 28-dimensional vector

$$\mathbf{x} = [x_1, ..., x_{28}]^T$$

describing the product and its attributes. This vector is used to train a Self-Organizing Map. The approach is similar to that used in [2] in a process monitoring application. The difference is that in this case the measurement vector is a measurement of one particular product, whereas in [2] the measurement vector consists of the corresponding attributes of a sampled process state at a given time.

In this particular case study, passive data is used. Passive data is data collected during the normal operation of the plant. This kind of data does not cover all the possible operation points, but only those points already encountered in practice during the normal operation.

Data preprocessing

This case study focused on a particular product grade. The data reflecting these products was filtered from the database. The data with missing values were removed. The data was scaled so that each component had variance one. The training set had approximately 3000 training vectors.

Training a SOM

Several Self-Organizing Maps were trained with a different number of codebook vectors and varying teaching parameters. Of all these SOMs, the one with the smallest quantization error was picked. This was the SOM with 25 times 15 codebok vectors. Bubble neighborhood was used as the neighborhood function. The learning rate was chosen to decrease linearly as a function of time. Training was done in two phases. During the first cycle, the initial alpha value $\alpha = 0.4$, the map was taught with 35000 samples and the initial neighborhood had the value 15. The respective parameter values for the second cycle were 0.04, 150000 and 6.

The U-matrix representation

In this case, the U-matrix representation can be used to see if the SOM quantizes the input space evenly.

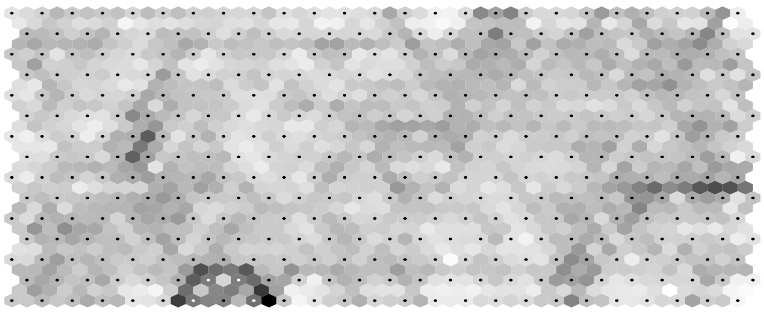


Figure 5.4: The U-matrix representation of the SOM

If there are no visible clusters, as is the case in Figure 5.4, the SOM quantizes the input space evenly, that is, the distances between the codebook vectors are approximately the same.

Sammon mapping of the SOM

Figure 5.5: Sammon mapping of the SOM

Sammon's mapping is used to visualize the relative distances between the codebook vectors of the SOM. As the SOM approximates the probability density function to some extent, there are a lot a codebook vectors, where the data is dense. This can be seen in the Figure 5.5.

5.3 Using SOM as a regression model

5.3.1 Predicting quality parameters

We predict the quality parameters by fetching direct copies of the missing components of the best-matching unit in the SOM. The best-matching unit is searched with regard to the known components only. In this case, we are interested in predicting the quality parameter values with given incoming raw material and process parameter settings. We thus map vectors with known components to the Self-Organizing Map and fetch the values of quality parameters from the best-matching unit of the input vector.

The choice of the missing components are up to the user. No committment to which components are inputs and which are outputs is made.

To validate the model, we removed a random sample of 500 vectors from the input data to form an independent testing set. This set was not used during training. For these vectors, the quality parameters were known.The quality parameters were omitted from the testing set vectors and the SOM was used to predict the values of the quality parameters by fetching direct copies of the quality parameters of the best-matching unit with regard to the known components. The predicted values for the quality parameter values can be compared with the corresponding known values.

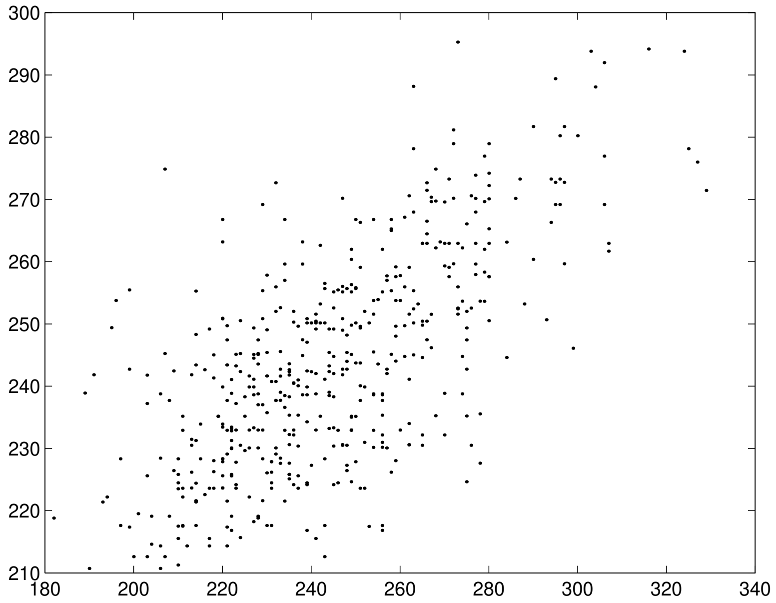


Figure 5.6: The real values versus the predicted values of the quality parameter

The points in Figure 5.6 are the pairs of real values and predicted values. Ideally, these should be on a line where both are equal. It can be seen that the predicted values lie around this line and are most accurate in the middle of the quality parameter range.

The Figure 5.7 shows the sorted real values of the quality parameter as points, and the predicted values of the respective quality parameters as a connected line.

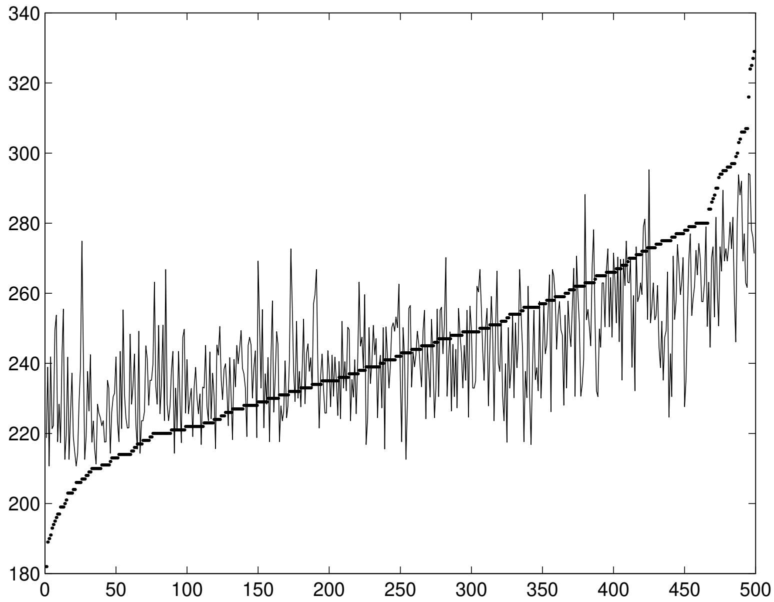


Figure 5.7: The predicted values and the sorted, real values

This shows that the predicted values for small values of the quality parameter are consistently above the real value. Predictions for the large values of the quality parameter are consistently below the right value.

The Figures above show that this method, despite its relative simplicity, can produce predictions with good accuracy. This method predicts well on average. The expected value of the prediction error is 0.6 and the standard deviation for the prediction error is 19 MPa.

5.3.2 Sensitivity analysis

By doing instantaneous or consecutive predictions under small changes made in the process parameter space we can investigate the leverage effects of the parameter changes. This is crucial for two reasons. Firstly, random variation causes small perturbations in the process parameters. This kind of noise is always present in a production process. Secondly, one can make improvements in the process by changing standard operating procedures in a directions that result in products of better quality.

In the Figure 5.8 the mechanism behind the tool is shown. A small change along one of the coordinates in the measurement space is made. The new vector of all components is mapped to the SOM. If the best-matching unit changes, the values of that unit are shown to the user. The interpretation of this is that the

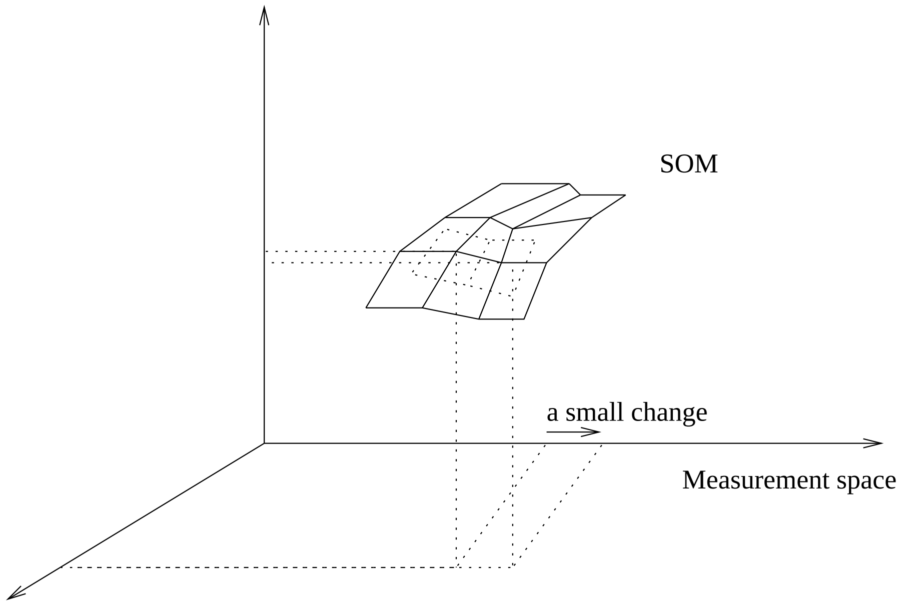


Figure 5.8: The leverage effect

the small change in one of the measurement space axes caused the other values to change. In the Figure 5.8 the component value on the axis pointing upward decreased a little caused by the small increase in the measurement on the axis pointing to the right. The third component remained the same.

A software tool was developed to help in the mentioned goals. In the Figure 5.9 the different functions of the tool are described.

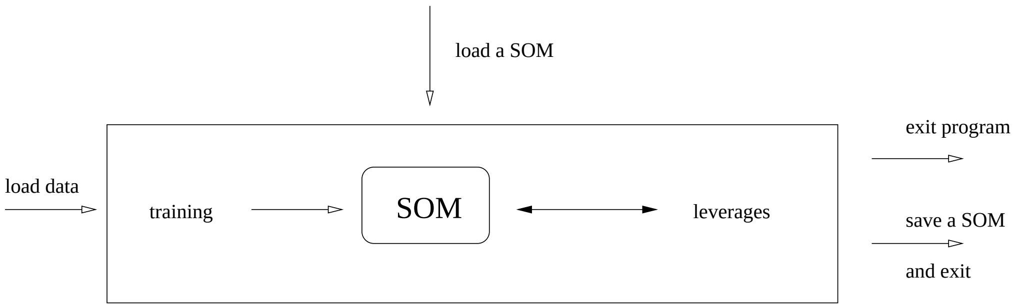


Figure 5.9: Functions of the tool program

One can perform all the parts needed to train a SOM from pre-processed data. First, data is read to the program. With the aid of the tool, the training process of the Self-Organizing Map can be handled. Instead of training a SOM, one can also load in a SOM and also save a SOM for later inspection and use.

This representation is used in giving instantaneous predictions under small changes made by the user. This reveals the leverage effects in a given operation point under small changes. It must be remembered that same changes can have different effects in different operations points.

The parameters settings are controlled by the user. The user has the choice of locking one or several parameters, thus limiting the operating point to a certain location in the measurement space. The tool updates only those components which are not locked. If no parameters are locked, the shown values are the direct copies of the values in the codebook vector.

As the user changes one of the parameters, the best-matching unit is constantly searched for. If the best-matching unit changes, all the other parameter settings are changed according to the values in that particular codebook vector. This corresponds to a kind of a projection from the parameter space to the SOM codebook vectors.

The user can then investigate the leverage effects caused by the small parameter changes. In this way, the operator can "play-along" with the process and learn about the dynamic behavior of the process.

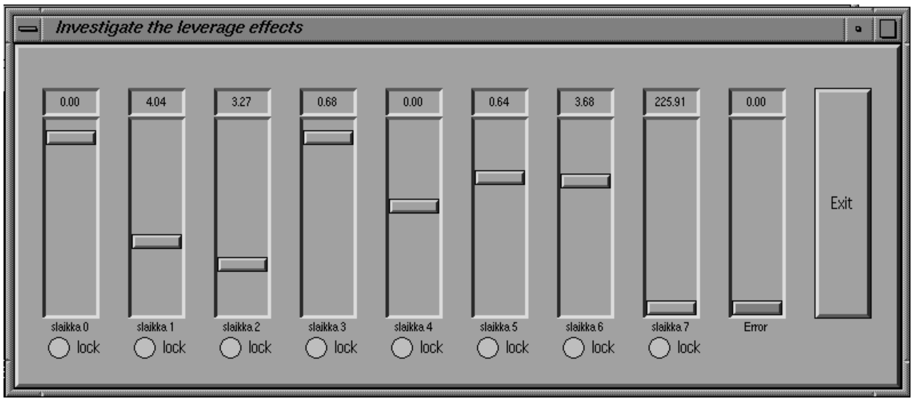


Figure 5.10: The interface of the tool for analyzing the leverage effects

The graphical interface of the leverage analyzer tool is illustrated in Figure 5.10. A static picture of the tool does not reveal much of its function or the purpose, but hands-on experience has proven it useful.

Besides the prediction, a measure of reliability is shown. The measure of reliability used is the quantization error between the codebook vector and the input vector set by the user. Locking parameters can have negative side-effects: we can drift away from the surface defined by the SOM. The quantization error provides a way to detect this.

No commitment is made to which parameters are independent and which are independent parameters. Naturally, one is interested in finding a combination of raw material characteristics and process parameter setting that produce the best possible quality, but one could as well predict best possible incoming raw material given the quality characteristics of the end product and the process parameter settings.

5.4 Using SOM and local model fitting

As explained in section 4.3, the SOM partitions the input space into Voronoi tesselations, which are disjoint areas in the input space. One may try to fit local models to the data belonging to one of these Voronoi tesselations. At the simplest form, these models would be linear. We could also fit a model to adjacent, neighboring Voronoi tesselations thus enlarging the area of interest. The locality of the data set is dependent on the division made by the SOM. Division is determined by the number of codebook vectors and the teaching process.

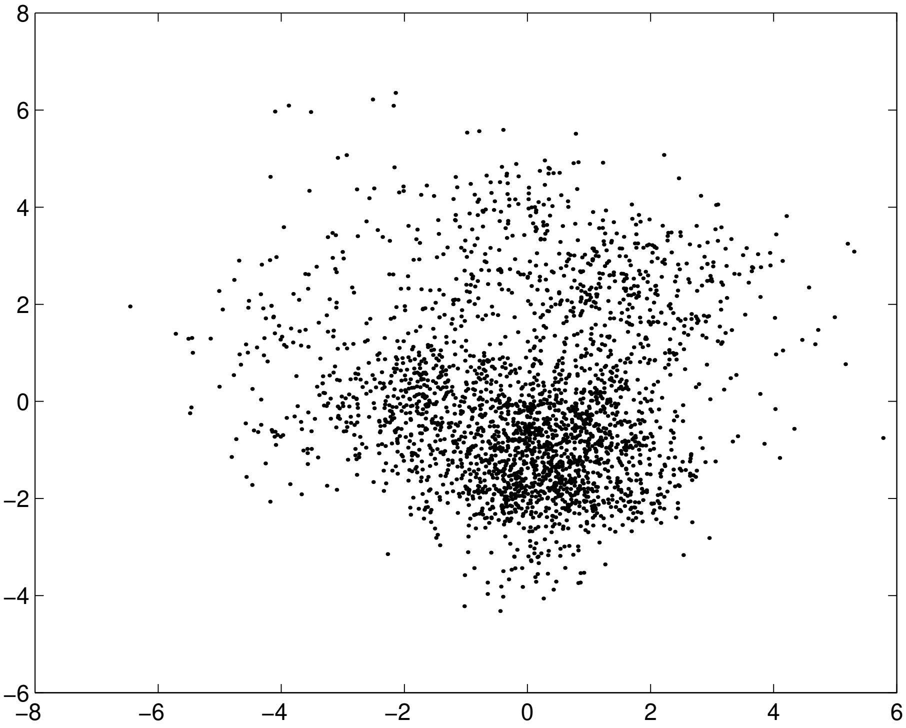


Figure 5.11: The training data projected on the two first eigenvectors

In the Figure 5.11 the training data used in prediction in the previous section was projected on the first two eigenvectors having the largest eigenvalues. Only about 22.6 % of the energy was concentrated around this linear subspace. The eigenvalues decreased rather slowly which indicated that there was a lot of energy in all the directions. This seems quite natural taking into account that there was no partitioning of the input space and that the phenomenon at hand is relatively complex. Also, moderate measurement noise may have part in this.

An effort was made to build local linear models of the data. The training data was divided into Voronoi tesselations based on the best-matching unit of input samples. The covariance matrix and the sample mean were calculated based on the data belonging to the Voronoi tesselations under interest. By investigating the eigenvalues in descending order, it was noted that there was no rapid decrease indicating that the data would reside in a linear subspace. Despite partitioning, the eigenvalues decreased slowly indicating that there was energy in essentially all the directions of the original coordinate system.

Scale selection seems to be the key problem in finding proper partitionings for the input space.

It may be argued that the failure to describe the data set in terms of local subspaces was due to small data set size after partitioning, moderate noise in the measurements and improper partitioning of the input space. Further work has to be done in order to develop better solutions for this problem.

Chapter 6

Conclusions

In this work, way to model a production process from the process measurements is presented. The artificial neural network, with which the process model is built, is the Self-Organizing Map (SOM). The model is based on the capability of the SOM to build non-linear regression models of the data.

The attributes of individual products were used as process measurements. These attributes were used in training the Self-Organizing Map.

The model's ability to describe the process depends also on the attributes' capability to describe the behavior of the process. In case there are missing model variables, that is, some essential process variables not a part of the model, the model cannot be expected to give good results.

These attributes used in modeling include the element concentrations of the incoming raw material, the process parameter settings during the production of a particular product and quality characteristics of the end product.

With the aid of the model one can predict quality parameters and study the leverage effects of the process parameter changes. A software tool is presented to facilitate this. The method is applicable if there is a lot of data from the process. These results serve best if only a little is known about the behavior of the process. In this way, the model serves the process specialist in the learning of the essential from large amounts of measurement data.

The quality of the model itself will decrease as the the amount of noise in the measurements increases. Before any modeling technique can produce meaningful results, the inputs, that is, the measurements from the process must be consistent. Making the inputs consistent should be the first phase in any process improvement scheme [6]. It must be remembered that no modeling technique can compensate the lack of good data.

Bibliography

[1] Jarmo T. Alander, Matti Frisk, Lasse Holmström, Ari Hämäläinen, and Juha Tuominen. Process error detection using self-organizing feature maps. In T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas, editors, *Artificial Neural Networks*, volume II, pages 1229–1232, Amsterdam, Netherlands, 1991. North-Holland.

[2] Esa Alhoniemi. Monitoring of complex processes using the self-organizing map. Master's thesis, Helsinki University of Technology, 1995.

[3] Christopher M. Bishop. *Neural Networks for Pattern recognition.* Oxford University Press, 1995.

[4] Simon Cumming. Neural networks for monitoring of engine condition data. *Neural Computing & Applications*, 1(1):96–102, 1993.

[5] Pierre Demartines and Jeanny Hérault. CCA: "curvilinear component analysis. In *Proc. of 15th workshop GRETSI. sep 1995, Juan-Les-Pins France*, 1995.

[6] Norman M. Edelson, Melinda L. Ellis, Anil N. Kharkar, Brian E. Stutts, and Suzanne deTreville. A sequential 3-phase process improvement strategy. In *ASQC Quality Congress Transactions - Nashville*, 1992.

[7] F. Firenze, L. Ricciardiello, and S. Pagliano. Self-organizing networks: A challenging approach to fault diagnosis of industrial processes. In Maria Marinaro and Pietro G. Morasso, editors, *Proc. ICANN'94, Int. Conf. on Artificial Neural Networks*, volume II, pages 1239–1242, London, UK, 1994. Springer.

[8] Neil A. Gershenfeld and Andreas S. Weigend. The future of time series: Learning and understanding. In *Time Series Prediction: Forecasting the Future and Undestanding the Past*, 1993.

[9] Rafael C. Gonzalez and Richard E. Woods. *Digital image processing.* Addison Wessley Publishing Company, 1992.

[10] K. Goser, S. Metzen, and V. Tryba. Designing of basic integrated circuits by self-organizing feature maps. *Neuro-Nimes*, 1989.

[11] Tom Harris. A Kohonen S.O.M. based, machine health monitoring system which enables diagnosis of faults not seen in the training set. In *Proc. IJCNN-93-Nagoya, Int. Joint Conf. on Neural Networks*, volume I, pages 947–950, Piscataway, NJ, 1993. IEEE Service Center.

[12] Robert Hecht-Nielsen. *Neurocomputing.* Addison-Wesley, 1989.

[13] Jukka Iivarinen, Teuvo Kohonen, Jari Kangas, and Sami Kaski. Visualizing the clusters on the self-organizing map. In Christer Carlsson, Timo Järvi, and Tapio Reponen, editors, *Proc. Conf. on Artificial Intelligence Res. in Finland*, number 12 in Conf. Proc. of Finnish Artificial Intelligence Society, pages 122–126, Helsinki, Finland, 1994. Finnish Artificial Intelligence Society.

[14] Jyrki Joutsensalo. Nonlinear data compression and representation by combining self-organizing map and subspace rule. In *Proc. ICNN'94, Int. Conf. on Neural Networks*, pages 637–640, Piscataway, NJ, 1994. IEEE Service Center.

[15] Jyrki Joutsensalo and Antti Miettinen. Self-organizning operator map for nonlinear dimension reduction. In *1995 IEEE Int. conf. on Neural Networks proc.*, 1995.

[16] Jyrki Joutsensalo, Antti Miettinen, and Martin Zeindl. Nonlinear dimension reduction by combining competitive and distributed learning. In F. Fogelman-Soulié and P. Gallinari, editors, *Proc. ICANN'95, Int. Conf. on Artificial Neural Networks*, volume II, pages 395–400, Nanterre, France, 1995. EC2.

[17] Nandakishore Kambhatla and Todd K. Leen. Fast non-linear dimension reduction. In *1993 IEEE Int. conf. on Neural Networks proc.*, volume III, 1993.

[18] Mika Kasslin, Jari Kangas, and Olli Simula. Process state monitoring using self-organizing maps. In I. Aleksander and J. Taylor, editors, *Artificial Neural Networks, 2*, volume II, pages 1531–1534, Amsterdam, Netherlands, 1992. North-Holland.

[19] Teuvo Kohonen. What generalizations of the Self-Organizing Map make sense. In Maria Marinaro and Pietro G. Morasso, editors, *Proc. ICANN'94, Int. Conf. on Artificial Neural Networks*, volume I, pages 292–297, London, UK, 1994. Springer.

[20] Teuvo Kohonen. *Self-Organizing Maps.* Springer, Berlin, Heidelberg, 1995.

[21] Teuvo Kohonen, Jussi Hynninen, Jari Kangas, and Jorma Laaksonen. *SOM_PAK: The Self-Organizing Map Program Package.* Helsinki University of Technology, Laboratory of Computer and Information Science, 1995. Available via anonymous ftp at internet address cochlea.hut.fi (130.233.168.48).

[22] Teuvo Kohonen, Erkki Oja, Olli Simula, Ari Visa, and Jari Kangas. Engineering applications of the self-organizing map. Manuscript submitted to a journal.

[23] Jouko Lampinen and Ossi Taipale. Optimization and simulation of quality properties in paper machine with neural networks. In *Proc. ICNN'94, Int. Conf. on Neural Networks*, pages 3812–3815, Piscataway, NJ, 1994. IEEE Service Center.

[24] Thomas Martinetz, Peter Protzel, Otto Gramckow, and Gunter Sorgel. Neural network control for steel rolling mills. In Bert Kappen and Stan Kielen, editors, *Proceedings of the Third Annual SNN Symposium on Neural Networks Nijmegen*, pages 281–286, September 1995.

[25] C. P. Matthews and K. Warwick. Practical application of self organising feature maps to process modeling. In *Proc. of the Int. Conf. on Engineering Applications on Neural Networks*, 1995.

[26] Gary S. May. Manufacturing ics the neural way. *IEEE Spectrum*, 1994.

[27] John W. Sammon, Jr. A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers*, C-18(5):401–409, May 1969.

[28] Pietro Morasso, Alberto Pareto, Stefano Pagliano, and Vittorio Sanguineti. Self-organizing neural network for diagnosis. In Stan Gielen and Bert Kappen, editors, *Proc. ICANN'93, Int. Conf. on Artificial Neural Networks*, pages 806–809, London, UK, 1993. Springer.

[29] D. Niebur and A. J. Germond. Unsupervised neural net classification of power system static security states. *Int. J. Electrical Power & Energy Systems*, 14(2-3):233–242, April-June 1992.

[30] Erkki Oja. *Subspace methods of pattern recognition*, volume 6 of *Pattern recognition and image processing series.* John Wiley & Sons, 1983.

[31] Erkki Oja. Neural networks, principal components, and subspaces. *International Journal of Neural Systems*, 1(1):61–68, 1989.

[32] Thomas Poppe, Dragan Obradovic, and Martin Schlang. Neural networks: Reducing energy and raw materials requirements. *Siemens Review*, 1995.

[33] Jose C. Principe and Ludong Wang. Non-linear time series modeling with Self-Organization Feature Maps. In *Proc. NNSP'95, IEEE Workshop on Neural Networks for Signal Processing*, pages 11–20, Piscataway, NJ, 1995. IEEE Service Center.

[34] Helge Ritter and Teuvo Kohonen. Self-organizing semantic maps. Technical report, Helsinki University of Technology, Faculty of Information Technology, Laboratory of Computer and Information Science, 1989.

[35] Helge Ritter, Thomas Martinetz, and Klaus Schulten. *Neural Computation and Self-Organizing Maps.* Addison-Wesley Publishing Company, 1992.

[36] T. Samad and S. A. Harp. Self-organization with partial data. *Network: Computation in Neural Systems*, 3(2):205–212, May 1992.

[37] Olli Simula and Jari Kangas. *Neural Networks for Chemical Engineers*, volume 6 of *Computer-Aided Chemical Engineering*, chapter 14, Process monitoring and visualization using self-organizing maps. Elsevier, Amsterdam, 1995.

[38] Viktor Tryba and Karl Goser. Self-Organizing Feature Maps for process control in chemistry. In T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas, editors, *Artificial Neural Networks*, pages 847–852, Amsterdam, Netherlands, 1991. North-Holland.

[39] A. Ultsch and H.P. Siemon. Kohonen's self organizing feature maps for exploratory data analysis. In *Proc. INNC'90, Int. Neural Network Conf.*, pages 305–308, Dordrecht, Netherlands, 1990. Kluwer.

[40] Alfred Ultsch. Self organized feature maps for monitoring and knowledge acquisition of a chemical process. In Stan Gielen and Bert Kappen, editors, *Proc. ICANN'93, Int. Conf. on Artificial Neural Networks*, pages 864–867, London, UK, 1993. Springer.

[41] Mauri Vapola, Olli Simula, Teuvo Kohonen, and Pekka Meriläinen. Representation and identification of fault conditions of an anaesthesia system by means of the Self-Organizing Map. In Maria Marinaro and Pietro G. Morasso, editors, *Proc. ICANN'94, Int. Conf. on Artificial Neural Networks*, volume I, pages 350–353, London, UK, 1994. Springer.

[42] Jörg Walter, Helge Ritter, and Klaus Schulten. Non-linear prediction with self-organizing maps. In *Proc. IJCNN-90-San Diego, Int. Joint Conf. on*

Neural Networks, volume 1, pages 589–594. IEEE Service Center, Piscataway NJ, 1990.

[43] X.Yao, A.K.Tieu, X.D.Fang, and D.Frances. Neural network application to head & tail width control in a hot strip mill. In *Proc. ICNN'95, Int. Conf. on Neural Networks*, 1995.