HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Electrical and
Communications Engineering

Matti Pöllä

Modeling Anticipatory Behavior with Self-Organizing Neural Networks

Master's Thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science in Technology

Espoo, 1.5.2005

Supervisor: Prof. Timo Honkela
Instructor: Prof. Timo Honkela

TEKNILLINEN KORKEAKOULU DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Matti Pöllä
Työn nimi:	Ennakoivan käytöksen mallintaminen itseorganisoivilla neuroverkoilla
English title:	Modeling Anticipatory Behavior with Self-Organizing Neural Networks
Päivämäärä:	1.5.2005 **Sivumäärä:** 72
Osasto:	Tietotekniikan osasto
Professuuri:	T-61 Informaatiotekniikka
Työn valvoja:	Professori Timo Honkela
Työn ohjaaja:	Professori Timo Honkela

Yksi tärkeistä luonnollisten kognitiivisten järjestelmien ominaisuuksista on kyky tehdä päätöksiä ennakoitujen tapahtumien perusteella. Ennakoinnin avulla on usein mahdollista parantaa keinotekoisten kognitiivisten järjestelmien vikasietoisuutta, mutta joissain tapauksissa ennakointi on välttämätön toiminnan apuväline.

Tämä diplomityö käsittelee itseorganisoivien neuroverkkojen hyödyntämistä dynaamisen järjestelmän tila-avaruuden mallintamisessa ja ennakoivan järjestelmän toteuttamista tämän tila-avaruusmallin avulla. Työ käsittelee erityisesti itseorganisoiva kartta -algoritmia ja sen muunnelmia aikariippuvan tietoaineiston käsittelyssä. Growing neural gas -algoritmia käsitellään vaihtoehtona itseorganisoivalle kartalle.

Esitellyt koetulokset osoittavat, että itseorganisoivaa karttaa käyttäen on mahdollista toteuttaa ennakointiin kykenevä järjestelmä, joka pystyy myös sopeutumaan ympäristön muutoksiin. Growing neural gas -algoritmi osoittautuu kuitenkin hyödylliseksi tilanteissa, joissa tietyt tila-avaruutta koskevat oletukset eivät päde.

Avainsanat:	ennakoiva käytös, itseorganisoiva kartta, growing neural gas, tila-avaruus, SOM, GNG
Hyväksytty:	**Kirjasto:**

HELSINKI UNIVERSITY OF TECHNOLOGY ABSTRACT OF MASTER'S THESIS

Author:	Matti Pöllä
Title of thesis:	Modeling Anticipatory Behavior with Self-Organizing Neural Networks
Finnish title:	Ennakoivan käytöksen mallintaminen itseorganisoivilla neuroverkoilla
Date:	1st May, 2005 **Pages:** 72
Department:	Department of Computer Engineering
Chair:	T-61 Computer and Information Science
Supervisor:	Professor Timo Honkela
Instructor:	Professor Timo Honkela

A vital mechanism of high-level natural cognitive sysems is the anticipatory capability of making decisions based on predicted events in the future. While in some cases the performance of computational cognitive systems can be improved with anticipatory behavior, it has been shown that some cognitive tasks require anticipation.

In this thesis the use of self-organizing artificial neural networks in constructing a predictive state space model of a dynamic system is reviewed to implement a computational cognitive system employing anticipatory behavior. Specifically, the biologically inspired Self-Organizing Map (SOM) algorithm and its recurrent variants are discussed in the task of processing time-dependent information. The topologically dynamic Growing Neural Gas (GNG) algorithm is reviewed as an alternative to the SOM.

Simulation results show that the SOM can be used to implement an adaptive anticipatory system that not only anticipates future events but also adapts to changes in the environment. However, the GNG algorithm is useful in situations where certain prior assumptions on the operating environment do not apply.

Keywords:	anticipatory behavior, Self-Organizing Map, Growing Neural Gas, state space, SOM, GNG
Approved:	**Library code:**

Preface

This work has been carried out in the Laboratory of Computer and Information Science (CIS) at Helsinki University of Technology as a part of the cognitive systems research programme.

I would like to express my gratitude to the computational cognitive systems research group for the numerous discussions that supported the progress of this work. The whole personnel of the CIS laboratory deserves thanks for a friendly and an inspiring working environment.

I am also very thankful to my instructor Prof. Timo Honkela for the continuous encouraging attitude towards my work. I would also like to thank Prof. Jaakko Hollmén for providing valuable feedback about the manuscript and Tiina Lindh-Knuutila for proofreading the text.

Otaniemi, 1st May 2005

Matti Pöllä

Contents

List of Figures

List of Tables

Symbols and abbreviations

$S(t)$	A dynamic system at time t
$M(t)$	Model of a dynamic system at time t
A	Set of available actions
$\mathbf{M}$	Codebook vector set
$\mathbf{w}_i(t)$	Weight vector of network node i at time t
$\mathbf{x}(t)$	Input vector at time t
$\mathbf{y}(t)$	Output vector at time t
θ	Parameter vector
$L(\mathbf{x}\|\theta)$	Likelihood-function
$l(\mathbf{x}\|\theta)$	Log-likelihood-function
$E\{x\}$	Expected value of x

AI	Artificial Intelligence
AIC	Akaike Information Criterion
ANN	Artificial Neural Network
AR	Autoregressive
ARSOM	Activation-Based Recursive Self-Organizing Map
BMU	Best Matching Unit
GNG	Growing Neural Gas
MA	Moving Average
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
NG	Neural Gas
NN	Neural Network
PDF	Probability Density Function
RL	Reinforcement Learning
RSOM	Recurrent Self-Organizing Map
SARDNET	Sequential Activation Retention and Decay NETwork
SOM	Self-Organizing Map
STM	Short Term Memory
TKM	Temporal Kohonen Map
VQ	Vector Quantization
WSS	Wide Sense Stationary

Chapter 1

Introduction

The principle of causality states that in an event a cause must precede an effect. We have learned to think that what happens in the present time depends on a complex combination of events in the past and present but not on the future. Still, we do make decisions guided by our predictions of the future. We learn to anticipate events beforehand and modify our behavior in a way that seems suitable according to our anticipations.

This thesis in concerned with implementing anticipatory behavior in computational cognitive systems using similar mechanisms that operate in natural cognitive systems such as the human brain. Specifically, the contribution of this thesis is to introduce the use of prototype-based self-organizing neural networks as a predictive state space model of an autonomous agent employing anticipatory behavior.

1.1 Biological motivations

New developments in technology are often created to mimic certain desirable characteristics found in biological systems. This paradigm has a solid foundation considering the superior counterparts of technical inventions found in nature that have been developed by a long evolutionary process.

Perhaps one of the most significant examples of a technology inspired by biological systems is the development of artificial neural networks (ANN, or simply neural networks NN henceforth), which mimic the highly efficient and parallel information processing system found in the nervous system of humans and animals [30]. As opposed to the human-designed architecture of a modern computer, artificial neural networks were developed by studying the operation of actual biological neural networks.

Anticipatory behavior makes no exception in this regard. Traditionally the "intelligence" in an artificial environment has been implemented as a rule set that

maps certain perceptions to suitable predetermined actions. This kind of intelligence constitutes a reactive system that operates according to its perceptions about the current state of its environment.

In the context of human behavior, anticipation is a common tool of reasoning. For example, in a situation where one sees a thunderstorm approaching, people tend to avoid walking through open spaces in order to avoid get struck by lightning. In another example, a car driver would reduce his speed when noticing an icy part of the road. What is common to these examples is the act of foreseeing an unwanted state in the future – a lightning strike or a car crash – and changing current behavior to avoid the harmful consequences.

At the human level anticipatory behavior may seem trivial considering the previous examples. However, similar mechanism operate also in lower levels of natural systems. For example, Rosen considers [55] negatively phototropic plants, which move away from light, a manifestation of "wired-in" anticipatory behavior. In this case darkness in itself does not reward the plant in any way. However, darkness is correlated with characteristics, which reward the organism such as moisture and nutrition and by extending towards dark soil the plant is likely to be rewarded in the future.

1.2 Term definitions

The English language has a wide range of terms relating to anticipation[1]. To clarify the meanings of the commonly referred terms, a short discussion on the use of these terms in this thesis is presented in the following conforming to the definitions by Mihai Nadin in [47].

The word *predict* is often used to refer to an act of foreseeing the future state of some variable as in time series prediction. When we predict something the focus is on the predicted entity itself and further interpretations about its effects are not considered. As in time series prediction, the question of interest is to find a good approximation for a future value of a variable. What separates a prediction from a *guess* is the applicability of some perceived cause–effect relationship. For example, using the syntax rules of a natural language we can make reasonably good predictions for completing a sentence with one missing word. When flipping a coin we have to settle for guessing the outcome of the experiment.

A *forecast* is generally understood as a larger entity than prediction. Forecasting – as in the context of a weather forecast – involves a combination of several unknown factors for which some end result is predicted.

In the context of this thesis the word *anticipate* is used to express the action

[1]The word *anticipation* itself is derived from the Latin word *antecapere* i.e. *"to understand beforehand"*.

of making decisions by predicting the future state of a system. An *anticipatory system* is a system that employs anticipatory behavior.

1.3 Anticipatory systems

A foundation for the research on anticipatory systems was laid in "Anticipatory Systems" by Rosen [55] including the following definition of an anticipatory system:

> A system containing a predictive model of itself and/or its environment, which allows it to change state at an instant in accord with the model's predictions pertaining to a latter instant.

With the above definition, anticipatory behavior can be viewed as a union of two tasks:

- Constructing a model M which employs the same causal dependencies as the modeled system S to future states of the system S.
- Constructing a policy for relating specific states of the system S to specific actions according to expected reward gained from state–action pairs.

The first part of predicting the future state of the system is related to the large field of time series prediction. By looking at the history of a system one can build a model that can predict the future states of the system with some accuracy. In contrast to traditional time prediction tasks, the amount of variables to predict for anticipatory systems is typically large. Instead of predicting the exact value of a one-dimensional scalar variable the prediction task for an anticipatory system operating in a complex environment requires predicting the future value of a high-dimensional vector with only little interest on the prediction result of individual variables.

The latter component of forming a policy for selecting is related to the reinforcement learning problem which is discussed further in section 1.8.

Rosen's and Dubois' model of an anticipatory system

By Rosen's definition, an anticipatory system S defines its successive states as a function of the current state $S(t)$ and the future state of the model of itself $M(t + \Delta t)$. This kind of anticipatory system is defined by the following equations.

$$\begin{aligned} \frac{\Delta S}{\Delta t} &= \frac{S(t+\Delta t) - S(t)}{\Delta t} = f\left(S(t), M(t+\Delta t)\right) \\ \frac{\Delta M}{\Delta t} &= \frac{M(t+\Delta t) - M(t)}{\Delta t} = g\left(M(t)\right) \end{aligned} \tag{1.1}$$

This, however, leads to a contradiction. If M is the model of an anticipatory system S, then M should recursively contain a model of itself as the model of S. In Equation (1.1) the model M is a model of S without the predictive model guiding it.

Dubois [19] has refined the definition of an anticipatory system by defining the change of the model M similarly as the system S itself.

$$\begin{aligned} \frac{\Delta S}{\Delta t} &= \frac{S(t+\Delta t) - S(t)}{\Delta t} = f\left(S(t), M(t+\Delta t)\right) \\ \frac{\Delta M}{\Delta t} &= \frac{M(t+\Delta t) - M(t)}{\Delta t} = f\left(S(t), M(t+\Delta t)\right) \end{aligned} \tag{1.2}$$

In the special case of a complete model ($M = S$) Equation (1.2) could be written as

$$\frac{\Delta S}{\Delta t} = \frac{S(t+\Delta t) - S(t)}{\Delta t} = f(S(t), S(t) + \Delta t f(S(t), S(t+\Delta t))) \tag{1.3}$$

which is a self-referential system depending on the *future* state of the system itself.

The difference between the two definitions of an anticipatory system can be viewed by the way the model M of the system S is defined. If M is considered to be a model of an anticipatory system S, then the model should include the model contained by S. Further, each new level of modeling will always result in a new recursive requirement of a model-of-a-model[2] such as in Equation (1.3).

1.4 Reactive and anticipatory behavior

When an artificially intelligent system is driven solely by stimulus–response actions with no planning, the system operates reactively. While most high-level cognitive tasks require planning to some extent, it is interesting to consider the combination and interaction of reactive (Algorithm 1) and anticipatory (Algorithm 2) behavior.

Davidsson [16] uses children playing soccer as an example of reactive behavior in a situation where anticipatory behavior is considered preferable.

[2]Also known as the *homunculus problem*.

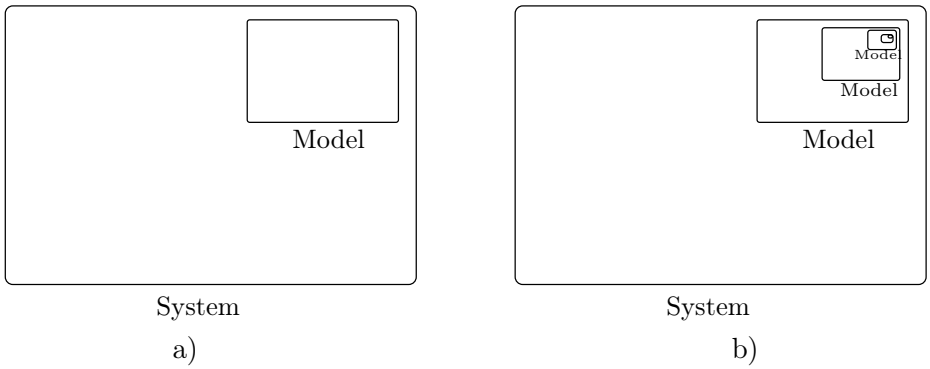


Figure 1.1: Rosen's (a) and Dubois' (b) model of an anticipatory system.

> Small children usually play a very primitive type of soccer which can be classified as a purely reactive behavior. If they are in control of the ball they kick it towards the opponent's goal, else if they see the ball they run towards it, if they do not, they look around for it.

In this case, the lack of a good predictive model of the future actions of the other players prevents the children from playing anticipatorily. As the players grow more experienced the playing tactics become extensively anticipatory by exploiting the predicted actions of other players including even "tricks" just to confuse the opponent's players' anticipations.

Another example on the interaction between reactive and anticipatory behavior is the class of adaptive mechanisms that are initially anticipatory and become reactive over time. Butz discusses [10] the initial hard practice of playing an instrument as requiring much planning and how it becomes more and more automatic and is eventually guided only by a correct feeling of its functioning.

Algorithm 1 Reactive behavior

```
1: state = sensors.getcurrentstate()
2: action = getsuitableaction(state)
3: actuators.executeaction(action)
```

Algorithm 2 Anticipatory behavior

```
1: state = sensors.getcurrentstate()
2: futurestate = predictfuturestate(state)
3: action = getsuitableaction(futurestate)
4: actuators.executeaction(action)
```

1.5 Recursion and incursion

The time evolution of a causal system can be formulated mathematically using recursion – defining an entity using itself. A first order causal recurrent system computes its states as

$$S(t) = f(S(t-1), \theta) \tag{1.4}$$

Where $S(t)$ is the state of the system at time t, θ is a set of domain-specific parameters and f is a function $f : \mathbb{R}^n \to \mathbb{R}^n$.

Mathematically it is also possible to reverse the temporal direction of recursion. For deterministic functions we can also define systems that compute their current state not only as a function of the past states but also the future states. A first order incursive system [19] computes its states as

$$S(t) = f(S(t+1), \theta) \tag{1.5}$$

A strong anticipatory system [19, 18, 9] can be defined as a system that computes its current state as a function of both past and future states.

$$S(t) = f(\ldots, S(t-2), S(t-1), S(t+1), S(t+2), \ldots; \theta) \tag{1.6}$$

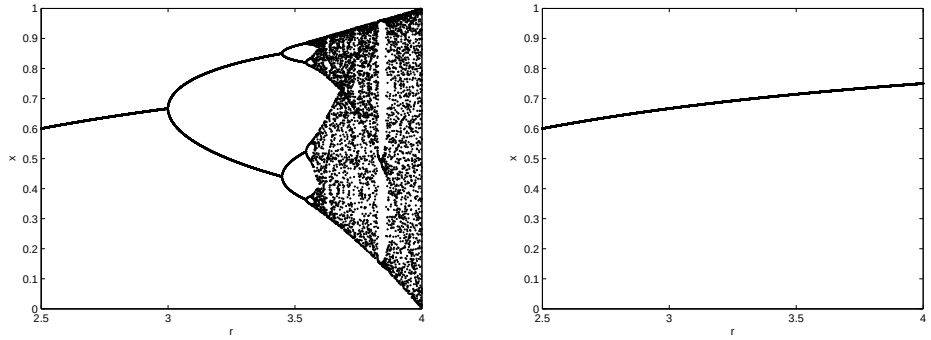
However, when considering complex real world systems, explicit information on the future state of a systems is typically uncomputable due to the massive amount of parameters and unsufficient information on the initial state. Thus, in practice, the information on the future will have to be replaced with information gained from a predictive model M of the system S. Now, the future states of the system in (1.6) can be replaced to define a weak anticipatory system [19]

$$S(t) = f(\ldots, S(t-2), S(t-1), M(t+1), M(t+2), \ldots; \theta) \tag{1.7}$$

where $M(t)$ is the state of the predictive model M at time t and θ is a set of domain-specific parameters.

Incursive control of a chaotic system

As an example, recursion and incursion are studied in a simple self-referential system which is commonly used to model population dynamics (presented originally in [19]). The logistic map – an equation frequently used to model the dynamics of a population size with a limitation posed by the sustainability of the environment is defined by



a) b)

Figure 1.2: Bifurcation diagram of (a) the logistic map $x(t) = r(1 - x(t-1))$ and (b) an incursive map $x(t) = r(1 - x(t+1))$. The bifurcation diagram shows the possible values of x (vertical axis) as a function of the parameter r (horizontal axis).

r	$x(t)$
$[0, 1]$	$x \to 0$ regardless of initial value $x(0)$
$[1, 2]$	$x \to \frac{r-1}{r}$
$[2, 3]$	oscillations followed by convergence to value $x \to \frac{r-1}{r}$
$[3, 1+\sqrt{6}]$	oscillation between two values
$[3.45, 3.54]$ (approx)	oscillation between four values
$r \geq 4$	chaotic

Table 1.1: Behavior of the logistic map for different values of the parameter r.

$$x(t+1) = rx(t)(1 - x(t)) \tag{1.8}$$

for which the values $\lim_{t\to\infty} x(t)$ are highly dependent on the parameter r (see Table 1.1) and with $r \geq 4$ the logistic map will become chaotic. This can be also seen in the bifurcation diagram in Figure 1.2 where the possible values of x are shown as a function of the parameter r.

If we replace $x(t)$ in the saturation factor $(1-x(t))$ of Equation (1.8) by $x(t+1)$ the logistic map becomes an incursive equation

$$x(t+1) = rx(t)(1 - x(t+1)) \tag{1.9}$$

which defines a first-order strong anticipatory system. Values $x(t+1)$ can now be computed self-referentially as

$$x(t+1) = rx(t)(1 - (rx(t)(1 - (rx(t)(1 - \dots))))) \tag{1.10}$$

in which the alternate series converges as

$$\begin{aligned} &1 - (rx(t)(1 - (rx(t)(1 - \dots)))) \\ &= 1 - rx(t) + (rx(t))^2 - (rx(t))^3 \cdots = \frac{1}{1 + rx(t)} \end{aligned} \tag{1.11}$$

Using (1.11) we can write (1.9) as

$$x(t+1) = rx(t)(1 - x(t+1)) = \frac{1}{1 + rx(t)} \tag{1.12}$$

which is stable for all values of r. Equation (1.12) is of the same form as Monod's model for population dynamics and it has been shown that bacterial growth obeys the Monod's model rather than the potentially chaotic logistic map of Equation (1.8) [7].

The conclusion of this section remains that while the above example does not prove anything about the implied anticipatory nature of natural population dynamics, it shows that a simple anticipatory modification to a recursive system can have a dramatic stabilizing effect.

1.6 State anticipation

When considering a dynamic system S, it is typical to view the time evolution of the system as a sequence of discrete-time states in a continuous state space $\mathbf{x} \in \mathbb{R}^n$. Each vector element x_i is a realization of a monitorable variable of the system. These states can also be divided into separate classes according to the effect a state has to the system.

When considering an anticipatory system S and M as the model of S, and the various ways of modifying the properties of S, Rosen suggests [55] the following method as the simplest:

> Let us imagine that the state space of S (and hence of M) to be partitioned into regions corresponding to "desirable" and "undesirable" states. As long as the trajectory in M remains in a "desirable" region, no action is taken by M through the effectors E. As soon as the M-trajectory moves into an "undesirable" region (and hence, by inference, we may expect the S-trajectory to move into the corresponding region at some later time, calculable from a knowledge of how the M- and S-trajectories are parameterized) the effector systems is activated to change the dynamics of S in such a way as to keep the S-trajectory out of the "undesirable" region.

The described scheme, also known as *state anticipation*, forms the basis for the study of implementing an anticipatory system in this thesis. In Chapter 4, a neural network -based framework is presented to build an adaptive state space representation of a dynamic system to implement anticipatory behavior.

1.7 Anticipatory agents

In the study of artificial intelligence an agent is considered an autonomous entity capable of perceiving its environment – sometimes including itself – and performing actions according to some decision making system. The agent perceives its environment by its *sensors* and uses its *actuators* to execute actions [57].

Further, an agent is considered to be *rational* if it chooses an appropriate action for each perception of the environment. A rational agent could, for example, monitor its environment for moving objects and avoid collisions by moving when objects approach the agent if we assume that a collision would have an unwanted effect on the agent.

Traditionally, the decision mechanism of an autonomous rational agent has been considered a reactive mapping from perceptions to actions. The agent might use its own history when constructing of modifying its perception–action policy and thus be aware of causal relations of events. However, this kind of agent is inherently *reactive* in the sense that each action is solely defined by its current state.

A reactive agent (Figure 1.3) receives information on itself and the environment with its sensors. A lookup-table-like decision mechanism is then used to associate each perceived state into a certain action. This policy table can be altered such that previous experiences are taken into account. However, in this case the causal inferences of the decision making system are mere reflections of the past as the agent lack a model of the world to simulate future events.

An anticipatory agent (Figure 1.4), on the other hand, selects its action according to an internal predictive model of the environment. An anticipatory agent is thus able to reason using causal dependencies of events that have occurred in the past. More importantly, an anticipatory agent is capable of simulating future events that have not occurred earlier in its history and select actions according to the simulated future state of the agent and its environment [14, 15, 21].

1.8 Anticipatory behavior and reinforcement learning

As discussed above, a subproblem in implementing anticipatory systems is the task of mapping the perceived states s_i of the system S into the set of available actions A.

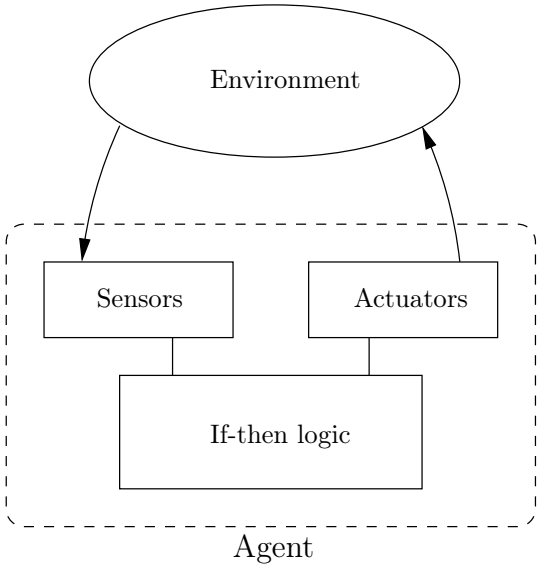


Figure 1.3: A reactive agent.

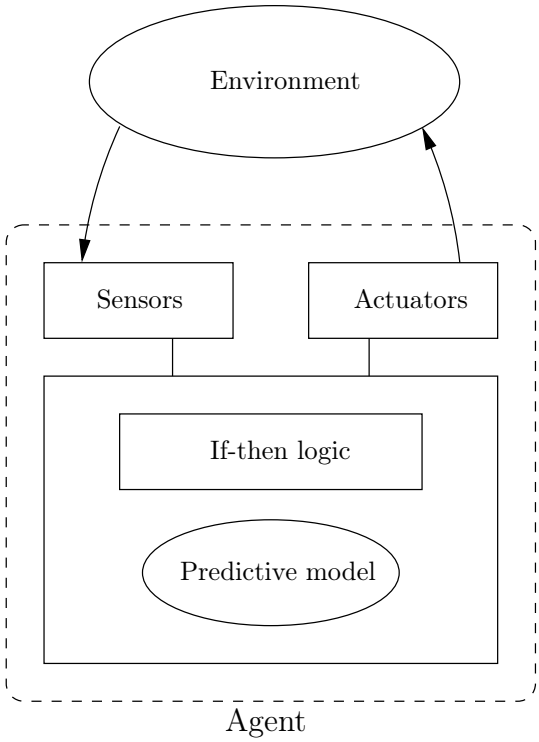


Figure 1.4: An anticipatory agent.

$$\pi \ : \ S \rightarrow A \tag{1.13}$$

An autonomous agent will typically form this policy without explicit knowledge on the effects of the actions using only the negative or positive reward provided by the environment.

The described problem of constructing a state–action policy using only the reward from the environment constitutes the reinforcement learning problem [64] where each action a_i moves the system into a new state s_{i+1} and rewards the agent with the associated reward r. In reinforcement learning the learning agent seeks to maximize the overall reward

$$R = r_0 + r_1 + ... + r_n \tag{1.14}$$

by selecting a policy π with a maximum action-value

$$\pi_*(s) = \arg\max_{a \in A} Q_{\pi_*}(s, a) \tag{1.15}$$

$$Q_\pi(s, a) = E_\pi\{R_t | s_t = s, a_t = a\} \tag{1.16}$$

Based on the above, the reinforcement learning problem can be understood as a subproblem of implementing an anticipatory system. In this thesis, however, the focus is on building a state space representation to determine *when* to act instead of focusing *how* to act. Further discussion on the application of reinforcement learning for anticipatory behavior can be found in Chapter 6.

Chapter 2

Multivariate time series prediction

2.1 Introduction

The task of predicting future values for a variable is one of the fundamental problems in statistical analysis. The problem of selecting a plausible model class and a suitable number of model parameters (model complexity) is an important part of the modeling process. Although statistical modeling of time series is generally focused on finding a good approximation for a single scalar variable, the same themes of model selection and model complexity apply for the case of modeling a multidimensional state space of a dynamic system.

This Chapter begins with an introduction of traditional linear models for time series analysis [8, 28, 12]. Further, a class of models implemented as neural networks are presented to extend the analysis to a more realistic case of multivariate predictions with nonlinear dependencies.

In the latter part of this Chapter, the selection of optimal complexity for a model based on information theoretical criteria is presented. Finally, the presented methods are reviewed based on their suitability to implement the predictive model of an anticipatory system of large dimensionality and a variable prediction interval.

2.2 Detecting regularities in a time series

One of the common problems of statistical analysis is the task of predicting future values of a time series using the previous observations of a phenomenon. Making reliable models for a time series is based on the assumption that the adjacent values $\{x(t), x(t+1), x(t+2), ...\}$ of a sequence x have a structure that

can be modeled.

An elementary method for finding a temporal structure in a sequence of numbers is to compute the autocorrelation i.e. the expected value of the product of two values of the sequence k time steps apart form each other.

$$r_x(k) = E[x(n-k)x(n)] \tag{2.1}$$

While autocorrelation indicates dependencies within a signal, cross-correlation can be used to investigate correlations between two signals x and y.

$$r_{xy}(k) = E[x(n-k)y(n)] \tag{2.2}$$

Correlations are also beneficial when determining the length of the temporal structures of a signal. Values $|r_x(n)| >> 0$ for small values of n are a sign of short periodic events in the time series while values $|r_x(n)| >> 0$ for large values n indicate long periodic structures.

In Figure 2.1 two seemingly random signals are presented together with their autocorrelation plots $r(k)$ for values $k = 1...300$. Looking at the signals in the time domain (Figure 2.1a and Figure 2.1b) does not reveal any clear temporal structures, but the autocorrelation plots (Figure 2.1c and Figure 2.1d) show that the first signal has a clear structure while the second signal appears uncorrelated.

What makes prediction of time serieses difficult is the presence of noise in nearly all observations that are used to construct the model of a time series. The fundamental problem is to find a model that captures the phenomenon which generates the observed data without adapting to the inherent noise in the training data. Thus, when modeling a time series using observations $y(t)$ the data is considered a sum of a deterministic component–which can be modeled–and a stochastic noise component

$$y(t) = x(t) + v(t) \tag{2.3}$$

where $x(t)$ is the model output and $v(t)$ is uncorrelated white noise.

The quality of a model is thus related to the error signal $e(n) = y(t) - x(t)$ such that for a successful model the remaining error shows no correlations $e(t) \sim N(0, \sigma)$. If the error signal is correlated, a deterministic part of the phenomenon has been left out of the model.

What makes selecting the complexity of a model especially complicated is the fact that with a limited set of observed data, one cannot specify exactly how much the data contains noise. A model could be constructed such that $y(t) = x(t)$ which would describe the specific data set precisely but as a result of adapting

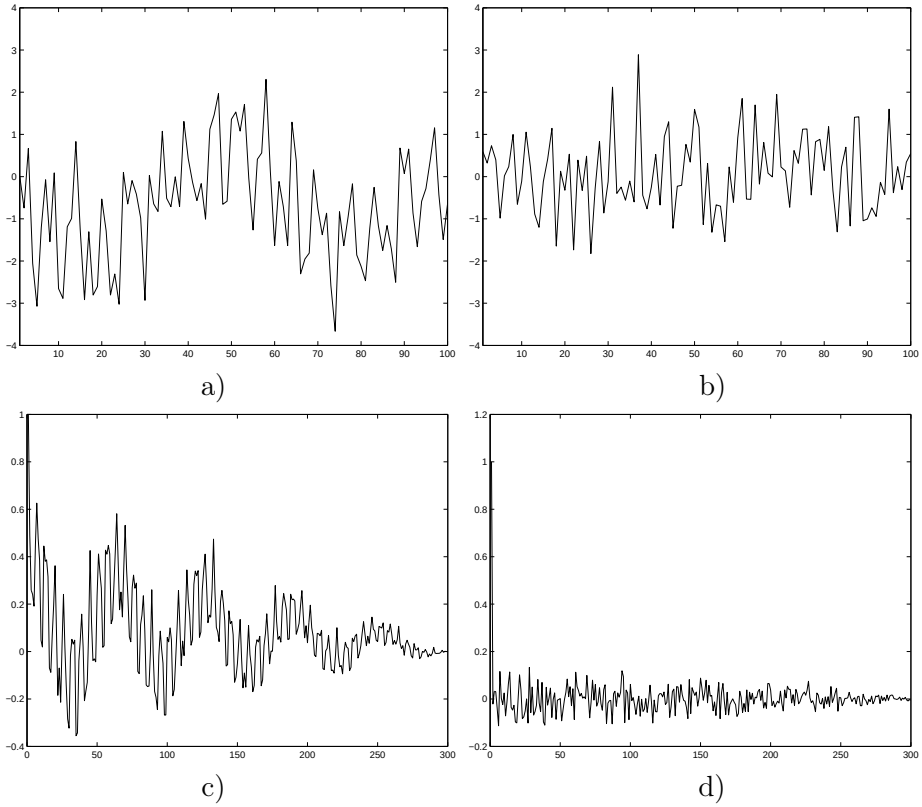


Figure 2.1: Autocorrelations of two seemingly random signals. The two signals in a) and b) seem random when viewed in the time domain. The corresponding autocorrelation plots c) and d) show that the first signal holds temporal structures while the second signal appears to be mere noise.

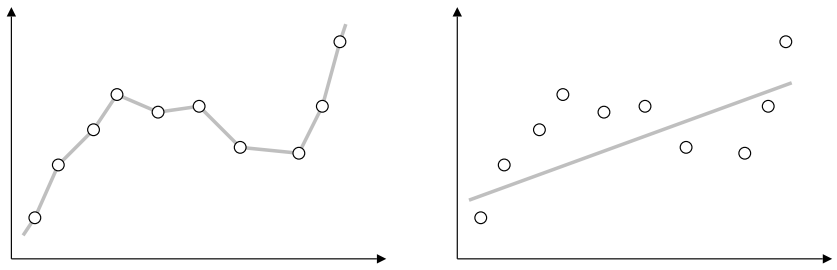


Figure 2.2: The bias-variance tradeoff can be illustrated using a simple curve-fitting problem. An over-fitted model (left) has little systematic error but the representation is specific to the used training data set. An under-fitted model (right) has little variance and thus a good generalization but suffers from large bias.

to noise the modeling result for a new data set observed from the same phenomenon would be poor (Figure 2.2).

In the following, a review of popular linear and nonlinear models for time series analysis is presented.

2.3 Linear models

Linear autoregressive moving average (ARMA) models of time-varying processes define adjacent values $\{x(1), x(2), x(3)...\}$ of a process as a linear combination of the previous values and the remaining error term [8, 28]. The small amount of parameters and the rigorous mathematical background has made ARMA a popular model class for time series prediction. Generalizations of the ARMA model for seasonal variation and external variables also exist [12].

The general ARMA model can be decomposed into an autoregressive (AR) part and a moving average (MA) part. A general autoregressive AR(p) model [8] of a process $x(n)$ can be defined as a weighted sum of the previous k values of x together with the coefficients a_i

$$x(n) = -\sum_{k=1}^{p} a(k)x(n-k) + v(n) \tag{2.4}$$

where $v(n)$ is the error term.

A moving average MA(q) model [8] defines successive values of a time series as the average of previous error terms as

$$x(n) = \sum_{k=0}^{q} b(k)v(n-k) \tag{2.5}$$

The two equations (2.4) and (2.5) can be combined to form the the general ARMA(p, q) model [8]

$$x(n) = -\sum_{k=1}^{p} a(k)x(n-k) + \sum_{l=0}^{q} b(l)v(n-l) \tag{2.6}$$

In equations (2.4), (2.5), and (2.6) the modeled process $x(t)$ is a one-dimensional scalar quantity. The model can be extended to multivariate processes $\mathbf{x}(t)$ by replacing the coefficients $\{a_1, a_2, a_3, ..., a_p\}$ and $\{b_0, b_1, b_2, ..., b_q\}$ with their matrix equivalents $\{\mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3, ..., \mathbf{A}_p\}$ and $\{\mathbf{B}_0, \mathbf{B}_1, \mathbf{B}_2, ..., \mathbf{B}_q\}$ to form a multivariate version of Equation (2.6)

$$\mathbf{x}(n) = -\mathbf{A}_k\mathbf{x}(n) + \mathbf{B}_k\mathbf{v}(n) \tag{2.7}$$

where $\mathbf{x}(n)$ and $\mathbf{v}(n)$ are $\mathbb{R}^d \times 1$ vectors.

Unlike in the scalar ARMA model, the estimation of the coefficients a_{ij} and b_{ij} is a complex operation and the amount of observed data to determine the coefficient matrices becomes much larger.

Drawbacks of the ARMA model class

The simplicity of the linear ARMA model has also its drawbacks which makes it a feasible model for only a very limited class of problems.

A precondition for applying the ARMA model is the stationarity of the data. A time series is stationary (wide sense stationary, WSS [29]) if the following conditions about the expected value and the autocorrelations of the process apply.

$$\begin{cases} E\{x(n)\} = m_x \\ r_x(n, n-k) = r_x(0, -k) \;\forall\; n \\ \mathrm{cov}(x(0), x(0)) < \infty \end{cases} \tag{2.8}$$

This is a severe limitation considering the realistic case of making predictions in a high-dimensional state space which is typically nonstationary.

The conventional ARMA model is typically used to predict the next value $x(n+1)$ of a sequence given the previous values. A one-step prediction task such as this can be useful in situations where longer predictions are not needed. Also, the prediction horizon of the ARMA model need not be the next value but some other future value of the sequence. In addition, the ARMA model can be applied recursively [67] as

$$\begin{aligned}\hat{x}(n+4) &= \hat{x}(\hat{x}(n+3)) \\ &= \hat{x}(\hat{x}(\hat{x}(n+2))) \\ &= \hat{x}(\hat{x}(\hat{x}(\hat{x}(n+1))))\end{aligned} \tag{2.9}$$

to use the predictions of the ARMA model to predict further values of the sequence. However, the prediction accuracy decreases dramatically when applying this to longer prediction lengths.

Another severe limitation of the linear model class is the problem of selecting the proper length of the fixed time window defined by the coefficients p and q. This can be seen as a manifestation of the general model complexity selection problem which is discussed further in the end of this Chapter. Although sophisticated tools exist for selecting a proper complexity for a linear AR model, the solution is typically problem dependent and thus not suitable to model the dynamics of a system in the state of continuous change.

The most fundamental limitation the linear models for modeling a dynamic system is the fact that time dynamics typically are not linear and thus cannot be captured by linear models. This calls for applying nonlinear models often implemented as artificial neural networks.

2.4 Nonlinear models

In the previous section, a general linear model for modeling a random process was presented. In this section, the modeling problem is extended by allowing the temporal dependencies of the signal to be nonlinear. In practice, these nonlinear models for time series are implemented as artificial neural network structures. In the following, a group of network topologies is reviewed as model of time-varying processes.

2.4.1 Multilayer perceptron

The multi-layer perceptron (MLP) is a general class of neural networks constructed of an input and an output layer of neurons and a variable number of hidden layers between them [6] (Figure 2.3). The characteristic feature of MLP networks is that the connections between the neurons are all in the forward direction of the signal flow and no recurrent connections exist. One of the strengths of MLP networks is that for differentiable transfer functions the derivatives of the error function of the network can be expressed as a function of the network weights and biases known as the error back-propagation [56, 71].

MLP networks are commonly used to learn a static mapping $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ and applied in pattern recognition tasks [30, 6, 60, 66].

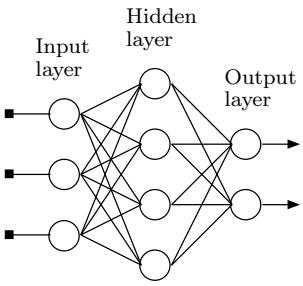


Figure 2.3: A fully connected multilayer perceptron network with two layers of adaptive weights, four hidden units and two output units.

The MLP network processes input patterns without discriminating the order in which the inputs are presented meaning that the MLP topology does not natively store information about time. External memory mechanics are thus needed to include the time dimension in the processing. For example, a delay line of length N can be connected to the inputs of a MLP network such that the input pattern $\mathbf{x}$ of the network is constructed of the current input and N previous values as

$$\mathbf{x}(n) = \begin{bmatrix} x(n) \\ x(n-1) \\ x(n-2) \\ \vdots \\ x(n-N) \end{bmatrix}$$

and the network can learn a complex nonlinear mapping using the N previous inputs.

The described delay line method does has a severe limitation considering the general case of processing time-dependent data. Typically, there is no prior information on the period length of the temporal pattern or at least an upper limit for the period length cannot be defined. Also, the period length can be varying. For an MLP network the delay line length determines a constant window length N with the discouraging effect that patterns longer than the time window will be ignored. On the other hand, for events with a short period length a wide time window can disturb the learning process.

2.4.2 Recurrent topologies

The generic MLP network type can be extended with internal memory mechanics by adding recurrent connections between the network nodes. The idea of recurrent connections in neural networks is to store the activation state of the network and use the stored activations as context information when processing new inputs.

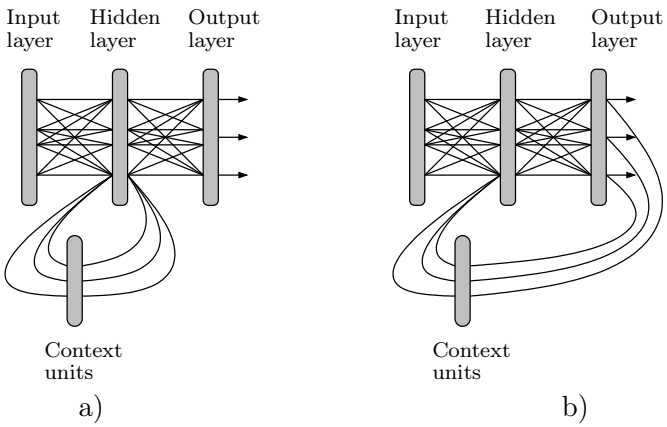


Figure 2.4: In the Elman topology a) the activations of the neurons in the hidden layer have a recurrent connection through a set of context neurons that can be used to store information on the temporal context of activations. In the Jordan topology b) the recurrent connection is taken from the network outputs.

The Elman topology network [22] adds a hidden layer of neurons to the feed-forward structure of a MLP network (Figure 2.4a). These context neurons are used to record the activation state of the hidden layer of the network with the consequence that previous activation states in the hidden layer effects the processing of each new input pattern.

A variation of the Elman network by Jordan [36] stores information on the activation state of the network outputs (Figure 2.4b).

One of the beneficial properties of the recurrent connections is that the network is able to learn sequences of variable length instead of defining a fixed time window as in the case of a MLP network used with a delay line.

2.5 Model validation methods

As discussed in the introduction, a common problem in constructing a model for a time series is to select a proper number of parameters [13]. A model with too few parameters will typically result in a too rigid model whereas a model with too many parameters is likely to be overfitted to the training data at hand.

Maximum likelihood is one of the popular methods for validating the quality of a model [1, 58]. In the maximum likelihood method a random variable X is assumed to distributed according to $p(X, \theta)$, where θ is a parameter vector $\{\theta_1, \theta_2, ..., \theta_k\}$. The idea now is to select the parameter vector θ such that the observed realizations of X gain maximum likelihood.

In the case of discrete variables with a probability density function $P(x_i, \theta)$ the

likelihood function can be written as

$$L(\theta) = \prod_{i=1}^{n} p(x_i, \theta_1, ..., \theta_k) \tag{2.10}$$

For a continuous probability density function f (2.10) can be written as

$$L(\theta) = \prod_{i=1}^{n} f(x_i, \theta_1, ..., \theta_k) \tag{2.11}$$

Often to simplify the differentations $\frac{\partial L}{\partial \theta_i}$ Equation (2.11) is transformed into a logarithmic version

$$l(\theta) = \log L(\theta) = \sum_{i=1}^{n} \log f(x_i, \theta_1, ..., \theta_k) \tag{2.12}$$

When the model class has been selected the remaining task is to find an appropriate complexity for the model after which the parameters can be selected optimally.

2.5.1 Methods for determining optimal model complexity

When modeling an observed process y the available data is typically limited to a set of N samples. A common measure for the modeling error is the mean squared error

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (y_i - x_i)^2 \tag{2.13}$$

in which y is the observed data and x is the model output. For a good model the error (2.13) should obviously be small. Still, it should be noted that since the observations typically contain noise and, on the other hand, the observed data set cannot be regarded as a complete representation of the process. A model should be able to distinguish between the noise term and the process itself so that only the process itself is modeled.

The modeling error can now be written as a sum of the two components

$$E\{(x-y)^2\} = \underbrace{\{E\{x\} - y\}^2}_{(\text{bias})^2} + \underbrace{E\{x - E\{x\}\}^2}_{\text{variance}} \tag{2.14}$$

where the first term is the systematic error of the model and the second term is the variance [6]. An overly rigid model will result in a large bias term and little variance. On the other hand an overfitted model will have very little bias but high variance which can be interpreted as learning the data set instead of the underlying phenomenon.

A common scheme for finding an appropriate complexity of a model is to first select a region of evaluated parameter amounts and then study the quality of the complexity level by comparing the models with some criterion. In this case the parameters of the models are all optimal since the parameter selection is done separately for each complexity.

When considering the optimal amount of parameters for a model class M the process is as follows. First, the maximum complexity of a model is decided as $k_{\max}$. After this, the k parameters of each model M_k are selected – for example – by maximizing the likelihood function i.e. selecting the parameters $\theta = \theta_1 ... \theta_k$ so that the probability of observing the observed data is at its maximum

$$\max_{\theta} L(\theta) = \max_{\theta} \prod p(x_i; \theta) \tag{2.15}$$

Usually the computation of the optimal parameters θ can be significantly simplified by maximizing the logarithm of the likelihood function.

$$\max_{\theta} \log L(\theta) = \max_{\theta} l(\theta) \tag{2.16}$$

The parameters can now be selected such that

$$\frac{\partial l(\theta)}{\partial \theta} = 0 \tag{2.17}$$

After this, the suitable model $M_1 ... M_k$ can be selected by a score known as information criterion [13].

2.5.2 Numerical information criteria

A popular way of balancing the bias-variance dilemma known as cross validation divides the observed data set into separate parts using the other part to construct an optimal model and the other part to validate the quality of the model. In the beginning of the cross validation process the observed data is divided into N subsets. For each data set the model parameters can be optimized using all other $N - 1$ data sets. After this the modeling error is computed and averaged over the N data sets. According to cross validation the model which results in the smallest mean modeling error has the optimal complexity [13].

Although cross validation is regarded as a data oriented method it should be noted that the selection of the amount of data subsets has an effect on the result. Also, when considering time dependent processes the division has to be made such that the time dependent information is remained.

In cross validation [61] each data value x_i belongs to exactly one data set. While this guarantees the representation of the whole data set in the modeling it restricts the amount of training data sets. Another approach to the selection of the data subsets is to ignore the restriction of each sample x_i belonging to just one subset and allow a sample to be selected into several subsets. This method is known as bootstrapping [20]. While bootstrapping does not even guarantee that all data samples are used at all, it enables the use of a large amount of training data sets and thus overcomes this restriction of cross validation. The score itself is computed similarly to cross validation as the mean modeling error with the smallest score indicating the best model.

2.5.3 Two-part information criterion

Akaike information criterion

A commonly used information criterion for the selection of an optimal model complexity considers the problem in two regards. The Akaike information criterion (AIC) [58] considers both the complexity of the model and the goodness of the fit to the observed data set based on the log likelihood function which estimates the expected log likelihood.

According to Akaike [58] the mean expected log likelihood is smaller by term $k/2$ than the real model's expected log likelihood

$$l_n(k) = E\{l'(\hat{\theta})\} = l'(\hat{\theta}') - \frac{k}{2} \tag{2.18}$$

where $l'(\theta')$ is the log likelihood of the real distribution. An optimal model is thus a one that minimizes the AIC criterion

$$AIC(k) = -2l(\hat{\theta}) + 2k \tag{2.19}$$

Minimum description length and normalized maximum likelihood

Another viewpoint for modeling is offered by the idea that every data set can be coded into a string of symbols of a finite alphabet and use any regularities inside the code to compress information. Minimum description length (MDL) by Rissanen [51, 5] is inspired by the principle of selecting the simplest hypothesis

of all candidates to explain a phenomenon.[1]

According to Rissanen, any set x of discrete data can be coded using a binary prefix code of length $L(x)$. A perfect binary prefix code also defines a distribution

$$p(x) = 2^{-L(x)} \tag{2.20}$$

For a model $p(x|\theta)$ the optimal code length is thus

$$L(x|\theta) = -\log_2 p(x|\theta) \tag{2.21}$$

The minimum description length becomes

$$\min_{\theta}\{L(x|\theta) + L(\theta)\} \tag{2.22}$$

in which the first term $L(x|\theta)$ describes the length of coding the data and the second term $L(\theta)$ the code length of the parameters. The model $p(x|\theta)$ that has the minimum description length according to (2.22) is considered a good generalization of the observed data.

The use of MDL still requires prior information on the parameter distribution to compute the code length for the parameters. A strictly data driven method called normalized maximum likelihood (NML) [62] can be derived by normalizing over the parameter space

$$L(\theta) = \frac{\pi(\theta)'}{\sum_y \pi(\theta)} \tag{2.23}$$

which makes the NML a strictly data driven information criterion for selecting the complexity of a model.

[1] Known popularly as *Ockam's razor* [4], which is usually phrased *"entities are not to be multiplied beyond necessity"*.

Chapter 3

Self-organizing neural networks for temporal modeling

3.1 Introduction

The Self-Organizing Map (SOM) algorithm by Kohonen [40] can be used to visualize, cluster and analyze large amounts of multidimensional data in an unsupervised manner. However, in its original form the SOM algorithm lacks the ability to represent temporal information. Several extensions have been developed to add this functionality to the basic SOM algorithm based on delay mechanics, recurrent connections and leaky activation potential.

In the beginning of this Chapter, the SOM algorithm is introduced in its original form. After this, a number of temporal SOM algorithms are presented and reviewed first in the context of classifying finite-length sequences and then discussed in the context of a continuous learning task such as the one in the anticipatory AI system of an autonomous agent.

As an alternative to the SOM, the class of Neural Gas algorithms by Martinetz and Fritzke [43, 46, 25] is presented as a different approach to the task of estimating the state space of a dynamic system.

3.2 The Self-Organizing Map

The Self-Organizing Map algorithm by Kohonen [40] is an neural network algorithm for creating a topologically correct nonlinear projection of high dimensional data into a neuron lattice of lower dimensionality. The SOM is typically used to visualize a large multidimensional data set to discover groups of similar patterns and thus transform a large amount of data into a simple visual presentation.

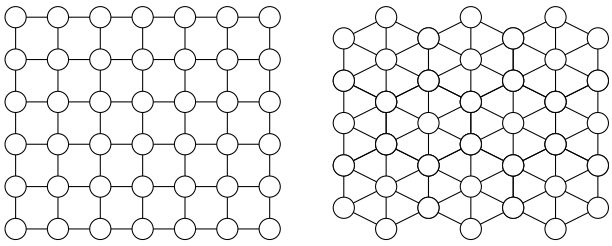


Figure 3.1: A rectangular and a hexagonal SOM network topology.

A SOM network is typically constructed as a two-dimensional rectangular grid (Figure 3.1) of N neurons (known also as nodes, prototype vectors or codebook vectors).

$$\mathbf{M} = \{\mathbf{m}_1, \mathbf{m}_2, \mathbf{m}_3, ..., \mathbf{m}_N\} \tag{3.1}$$

Each of the SOM nodes $\mathbf{m}_i$ is associated with a n-dimensional weight vector $\mathbf{w}_i$ which defines the node as a point in the input space $\mathbb{R}^n$.

$$\mathbf{w}_i = \{w_{i1}, w_{i2}, w_{i3}, ..., w_{nN}\} \tag{3.2}$$

At each step of the training process, the input vectors $\mathbf{x} \in \mathbb{R}^n$ are compared with the SOM codebook vectors to organize the codebook vector set $\mathbf{M}$ such that similar vectors (according to Euclidian distance metric) reside close to each other on the SOM grid to form groups of similar vectors. As a result of the training process, the input data vectors become coded in the SOM such that for each training vector $\mathbf{x}$ used in the training process there exists a close equivalent in the SOM codebook. Each training vector $\mathbf{x}$ can then be viewed in relation to the rest of the training data by looking at the location of its best matching equivalent in the codebook vector set. Detecting similarities (and dissimilarities) between two multidimensional vectors of the training set thus reduces to looking at the distance between the two vectors on the two dimensional SOM grid.

3.2.1 Training algorithm

The SOM combines Hebbian learning[1] and competitive learning in an unsupervised manner such that the organizing process advances – as the name implies – without any guidance or feedback. The unsupervized nature of the learning process distincts the SOM from many NN types such as the MLP which is trained by presenting a desired output value for each input pattern.

The training algorithm involves processing each input data vector $\mathbf{x}$ through the following two steps

[1] see Hebb's postulate [31].

(i) The input vector $\mathbf{x}$ is compared with each SOM codebook vector $\mathbf{m}_i$ and the closest equivalent according to the Euclidian distance between the vectors is selected as the best matching unit $\mathbf{m}_c$.

$$||\mathbf{x} - \mathbf{m}_c|| = \min_i \{||\mathbf{x} - \mathbf{m}_i||\} \tag{3.3}$$

(ii) The BMU neuron and its neighboring units are shifted towards the input vector $\mathbf{x}$ such that the BMU neuron is changed the most and the shift amount for the neighboring for the neighboring units is defined by a neighborhood function, which is a monotonically decreasing function of $||\mathbf{r}_i - \mathbf{r}_c||$ where $\mathbf{r}_i$ is the two-dimensional coordinate vector indicating the location of unit i in the SOM plane.

$$\mathbf{m}_i(t+1) = \mathbf{m}_i(t) + h_{ci}(t)[\mathbf{x}(t) - \mathbf{m}_i(t)] \tag{3.4}$$

A typical choice for a neighborhood function is a Gaussian kernel function (Figure 3.2)

$$h_{ci}(t) = \alpha(t) \cdot \exp\left(-\frac{||\mathbf{r}_i - \mathbf{r}_c||^2}{2\sigma(t)^2}\right) \tag{3.5}$$

where $0 < \alpha(t) < 1$ is a time-dependent decreasing parameter defining the learning rate and $\sigma(t)$ is a time-dependent parameter defining the decreasing neighborhood width. For convergence it is necessary that $h_{ci}(t) \to 0$ when $t \to \infty$. The Gaussian neighborhood function of Equation (3.5) suffers from heavy computational load which can be avoided by using a computationally lighter function such as the bubble neighborhood function (Figure 3.2)

$$h_{ci}(t) = \begin{cases} 1, & ||\mathbf{r}_i - \mathbf{r}_c|| \leq d \\ 0, & ||\mathbf{r}_i - \mathbf{r}_c|| > d \end{cases} \tag{3.6}$$

where d is a parameter defining the width of the neighborhood.

3.2.2 Visualization methods

The SOM algorithm can be an extremely useful tool in visualizing large data sets since the distance on the map lattice is a simplified measure of the similarity of two data vectors. In addition to labeling the codebook vectors according to their input vectors there exists other methods to depict the SOM projection.

A commonly used visualization tool for the SOM is the U-matrix [69] (Figure 3.3). In the U-matrix plot each SOM codebook vector is shown as a color shade defined by the similarity between the neuron and its neighboring neurons. Thus,

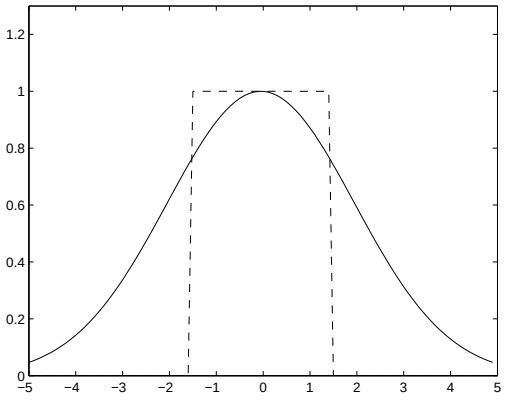


Figure 3.2: SOM neighborhood functions–a Gaussian (solid line) and a bubble neighborhood (dashed line).

areas where similar SOM nodes form a cluster, will show as a homogenous color area and these clusters are separated by a ridge.

When working with temporal data – which is the case in the following chapters – the trajectory of the BMU neurons can be used to visualize temporal structures in the data set. A BMU trajectory (Figure 3.4) is constructed simply by connecting the BMU nodes $\mathbf{m}_c$ of adjacent input data vectors with a line. In the case of slow transitions in the input space $\mathbb{R}^n$ the BMU trajectory can be used to predict the possible future values of the pattern sequence. This property of the SOM will be used extensively in Chapter 4.

Sammon's mapping [59] is another method of projecting multi-dimensional information into a two dimensional representation. The mapping is based on organizing the two-dimensional representations of the multidimensional vectors such that an energy function is minimized. For large data sets the computation of the Sammon's mapping becomes very heavy and thus it typically can not be applied to the whole data set. Instead, the SOM codebook vectors can be analyzed using the Sammon's mapping such as in Figure 3.5.

3.2.3 Processing temporal data with the SOM

The SOM algorithm, as presented above, processes the input vectors in a way that is independent of the order of the data. Sequences $\{\mathbf{x}(1), \mathbf{x}(2), \mathbf{x}(3)\}$ and $\{\mathbf{x}(3), \mathbf{x}(1), \mathbf{x}(2)\}$ will thus find their best matching units $\{\mathbf{m}_{c1}, \mathbf{m}_{c2}, \mathbf{m}_{c3}\}$ regardless of the order in which the inputs are presented.

Although the algorithm itself is unaware of the time dimension of the data, in some situations we can use our prior knowledge about the data to gain implicit temporal information from the SOM projection of the input data. For example,

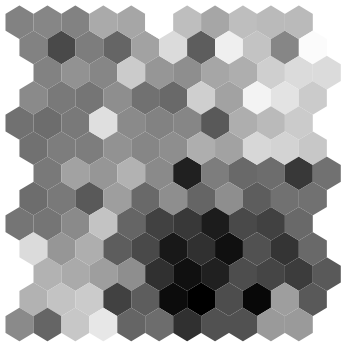


Figure 3.3: In the U-matrix visualization clusters of similar codebook vectors show as dark areas. In this case, a cluster of similar nodes is located in the middle of the lower half of the U-matrix.

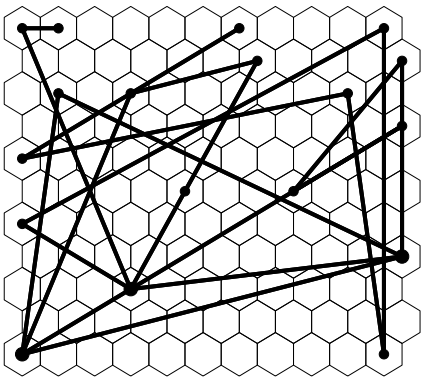


Figure 3.4: Trajectory of the BMU units for successive input vectors.

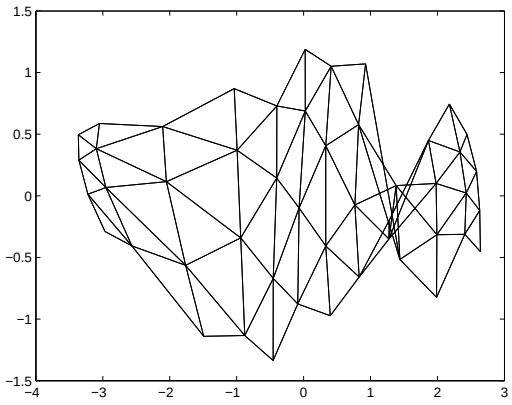


Figure 3.5: Sammon's mapping of the codebook vectors of a SOM network.

if the differences between adjacent patterns $\mathbf{x}(t)$ and $\mathbf{x}(t+1)$ are known to be small, the corresponding SOM neurons $\mathbf{m}_t$ and $\mathbf{m}_{t+1}$ of the adjacent patterns are likely to be placed near each other in the SOM space.

Another view to the relation of the network topology and the temporal structure of the input sequence is provided in [63], where the topology was constructed such that only the adjacent BMU neurons of an input sequence were connected.

This time topology property of the SOM algorithm will be discussed further in Chapter 4.

In many situations though the implicit time topology preservation of the SOM is not a sufficient way to represent temporal information. For this reason, many extensions to the SOM algorithm have been developed to encode the time dimension either with external or internal memory mechanics.

3.3 Variants of the SOM algorithm

As discussed in the previous section, the SOM algorithm does not store information about past inputs. The ability to recognize temporal structures requires a mechanism to store information about the adjacency of individual inputs. This mechanism – known as the short term memory (STM) – can be implemented in various ways ranging from a simple delay line to various schemes implementing a leaky integrator potential of the activation of the neurons. A rough division between the various extensions of the SOM algorithm can be made between the placement of the STM mechanics in regard to the SOM itself [17].

External STM models implement the memory as a separate element from the SOM. Typically this is done with an independent memory circuit connected to

the input or output of the SOM as in the case of a tapped delay line. Internal STM models implement the STM as a part of the modified SOM algorithm using, for example, a leaky integrator potential in the inputs or the outputs of the SOM neurons.

3.3.1 External STM mechanics

The simplest way of implementing a short-term memory follows the idea of autoregression introduced in Chapter 2. The temporal sequence is led through a chain of unit delays which store the previous N input values. The SOM can now be fed with a data vector which is constructed of N adjacent input values together with the current input.

$$\mathbf{x}(n) = \begin{bmatrix} x(n) \\ x(n-1) \\ x(n-2) \\ \vdots \\ x(t-N) \end{bmatrix} \tag{3.7}$$

Thus the tapped delay line STM effectively transforms the time dimension to space dimensions. Kangas [38, 37] has applied such scheme for isolated phoneme recognition and segmentation.

A serious obstacle of the delay line STM is – as in the case of an AR model – the difficulty of defining the suitable length N for the time window. Because the number of used delay units dictates the memory length, all information on the process history beyond $\mathbf{x}(n-N)$ will be ignored.

3.3.2 Internal STM mechanics

Various modifications of the SOM algorithm include the STM into the SOM algorithm itself using the biologically inspired idea of a leaky activation potential [17, 11, 41, 35]. The idea behind the leaky activation potential is to represent each neuron with a potential P_i decaying over time.

$$P_i(t) = \lambda P_i(t-1) + I_i(t) \tag{3.8}$$

where $0 < \lambda < 1$ is a decay parameter defining the memory depth and $I_i(t)$ is the activity of neuron i at time t. This approach overcomes the need to define a fixed-length time window of the delay-line by introducing a more flexible concept of memory depth that can be adjusted with the parameter λ. Additionally, the idea of a leaky potential is a biologically more plausible way of representing

temporal information. In the following, a number of modified SOM algorithms based on the leaky potential are presented.

Temporal Kohonen Map (TKM)

The concept of a leaky integrator potential associated to neuron activity was introduced in the Temporal Kohonen Map algorithm (TKM) by Chappel and Taylor [11]. The TKM extends the SOM by defining a time-varying activity state for each neuron. When the SOM searches for a best matching unit neuron for an input pattern, only a single prototype of the SOM is activated and for the following input vector the BMU is searched regardless of the result of the previous result. In the TKM, the sharp instantaneous activation of the SOM codebook vectors is replaced with a continuous activation potential which once activated, gradually decays and loses its potential.

In the TKM the decaying activation potential for neuron i is defined as

$$V_i(t) = dV_i(t-1) - \frac{1}{2}||\mathbf{x}(t) - \mathbf{w}_i(t)||^2 \tag{3.9}$$

where $0 \leq d \leq 1$ is a constant defining the memory depth, $\mathbf{x}(t)$ is the input vector, and $\mathbf{w}_i(t)$ is the weight vector of neuron i. The activity for neuron i at time t becomes

$$V_i(t) = -\frac{1}{2}\sum_{k=0}^{t-1} d^k||\mathbf{x}(t-k) - \mathbf{w}_i(t-k)||^2 + d^t V_i(0) \tag{3.10}$$

The winning neuron $\mathbf{c}$ is then selected as

$$V_{\mathbf{c}}(n) = \max_i\{V_i(n)\} \tag{3.11}$$

The TKM can be used to classify n-length sequences by representing each known sequence with an associated TKM reference node $\mathbf{m}_i \in \mathbb{R}^n$ defined by the last BMU neuron of the sequence. Thus the sequences “aabcba” and “aabccb” would be associated to separate reference nodes. Nevertheless, the resolution of the TKM is limited to the number of the map codebook nodes. Also, as noted by James et al. [35] the TKM suffers form loss of context as a result of the decaying activation potential. This means that the classification of longer sequences will rely heavily on the last values of the sequence and thus the sequences “aabcba” and “babcba” would easily be classified as the same sequence. According to James this property limits the use of TKM to only short sequences.

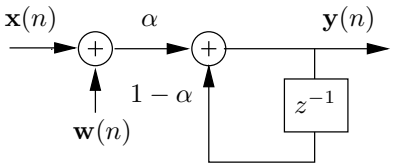


Figure 3.6: Schematic picture of a RSOM neuron [41].

Recurrent SOM (RSOM)

A modified version of the TKM algorithm called the recurrent SOM [41] simply moves the leaky integrator potential from the output of the neuron into the input. In this case the feedback quantity is a vector, which makes it possible to capture also the direction of the error.

The input activity of an RSOM neuron i is defined recursively as

$$\mathbf{y}_i(t) = (1-\alpha)\mathbf{y}_i(t-1) + \alpha(\mathbf{x}(t) - \mathbf{w}_i(t)) \tag{3.12}$$

where α is the leaking coefficient analogous to the factor d in the TKM.

The best matching unit is selected such that

$$\mathbf{y}_c = \min_i\{||\mathbf{y}_i||\} \tag{3.13}$$

measuring the smallest error between the current input and the sustained previous inputs of the RSOM node.

SARDNET

The SARDNET (Sequential Activation Retention and Decay NETwork) model by James [35] uses activation retention and decay to form a very dense representation of temporal sequences with only few training iterations.

What makes the SARDNET model very different from the previous algorithms is that the order in which the map nodes are activated during an input sequence is used to classify sequences. This makes the SARDNET capable of storing a large amount of information about input sequences without the resolution limitation of the TKM and RSOM. For example, consider the case of processing vectors with maximum length l and each vector component having p possible values. In this case for p^n possible input vectors there needs to exist lp^n map nodes to represent them all. This is a significant improvement to methods such as the TKM that would require p^{nl} nodes for the same task.

The SARDNET algorithm is defined by the following steps:

1. Initialize a SOM network consisting of N codebook vectors.

2. For an input pattern $\mathbf{x}$ determine the BMU node by the smallest Euclidean distance.
3. Set the activation of the BMU neuron to 1.0.
4. Adjust the weights of the BMU neuron and its neighbors by the standard SOM training rule (3.4).
5. Exclude the current BMU neuron from subsequent competition for the current sequence.
6. Decrement the activation values for all other active nodes.

ARSOM

A variant of the SOM called Activation-based Recursive Self-Organizing map (ARSOM) [33] extends the idea of inter-neuron recurrent connections of the Elman [22] and Jordan [36] networks to the domain of Self-Organizing Maps. In the ARSOM model, the activation of the whole map is taken into account when processing successive inputs.

In an ARSOM network each neuron is associated with two weight vectors which define the response of the neuron for each new input (known as content) and for each activation state of the network (known as context). The content weight vector is similar to the one of the standard SOM algorithm

$$\mathbf{w}_i^x = \{w_{i1}^x, w_{i2}^x, w_{i3}^x, ..., w_{iN}^x\} \tag{3.14}$$

where N is the input dimension of the map. Connections between the neurons are defined by the context weight vector

$$\mathbf{w}_i^y = \{w_{i1}^y, w_{i2}^y, w_{i3}^y, ..., w_{iM}^y\} \tag{3.15}$$

where M is the number of neurons in the network.

Using the weight vectors of (3.14) and (3.15) we can write the activation of the ARSOM node i as

$$y_i(t) = TF\left(||\mathbf{x}(t) - \mathbf{w}_i^x||, \alpha\right) \cdot TF\left(||\mathbf{y}(t-1) - \mathbf{w}_i^y||, \beta\right) \tag{3.16}$$

which is a product of the *content response* and *context response* of the codebook unit. In (3.16) $TF(\cdot)$ is a transfer function which maps the distances $||\mathbf{x}(t) - \mathbf{w}_i||$ to the range $[0, 1]$. Note that the transfer function $TF(\cdot)$ is the same for the both responses so no distinction is made between the feedforward and recurrent connections.

The BMU unit is then selected as

$$y_c = \max_i \{y_i(t)\} \tag{3.17}$$

and the weights are updated according to the update rules

$$\begin{aligned} \Delta \mathbf{w}_i^x &= \phi h_{ci}(\mathbf{x}(t) - \mathbf{w}_i^x) \\ \Delta \mathbf{w}_i^y &= \psi h_{ci}(\mathbf{y}(t-1) - \mathbf{w}_i^x) \end{aligned} \tag{3.18}$$

where ϕ and ψ are the learning rates.

3.3.3 Limitations of the SOM algorithm

Despite the usefulness of the SOM algorithm it has some inherent limitations when considering the task of processing temporal patterns. These limitations include the following:

- The static topology of the SOM is a limitation when considering the resolution of the projected probability density estimate. One has to decide beforehand the size of the SOM lattice and the dimensionality of the projection. This could be a serious limitation in a situation where no *a priori* information is available about the underlying data distribution.
- In an online learning situation the time dependent parameters of the SOM lead to a compromise between the plasticity of the network and the memory length for old data. This problem called *catastrophic forgetting* [24] is common in neural networks with a static topology.

In the following, two neural network algorithms, namely the Neural Gas and the Growing Neural Gas, based on unsupervised learning, are presented as an option for using the SOM when constructing a temporal state space model.

3.4 Other types of self-organizing networks

3.4.1 Neural Gas

An alternative to the SOM approach to quantizing multidimensional datasets offers the family of Neural Gas (NG) algorithms. Instead of reducing the dimension of the data set as in the SOM algorithm, the Neural Gas algorithm defines a set of N nodes in the original input space $\mathbb{R}^n$. Connections between the nodes

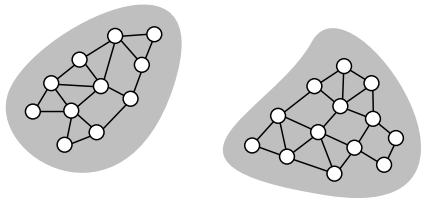


Figure 3.7: The Neural Gas is able to represent a probability density function consisting of two separate regions (gray areas) due to the dynamic topological connections.

are defined without a regular static SOM-like grid. Instead, the topology of the network is dynamic and able to adapt to changes in the PDF generating the input samples (Figure 3.7).

Martinetz [43, 46] justifies the use of Neural Gas networks over the SOM as follows:

> To obtain optimal results concerning the conservation of the topology of the mapping as well as the optimal utilization of all neural units, the topology of the employed network has to match the topology of the manifold of the data which is represented. This requires prior knowledge about the topological structure of the manifold, which is not always available of might be difficult to obtain if the topological structure of the manifold is very heterogenous, e.g. composed of subsets of different effective dimensions or disjunct and highly fractured.
>
> For this reason, it is desirable to employ a more flexible network capable of (i) quantizing topologically heterogenously structured manifolds and (ii) learning the similarity relationships among the input signals without the necessity of prespecifying a network topology.

The Neural Gas network consists of a set of N nodes with each one associated with a weight vector $\mathbf{w}_i \in \mathbb{R}^n$. The network topology is defined by a connectivity matrix $\mathbf{C}$, $C_{ij} = \{0, 1\}$ where the value 1 denotes a connection between nodes i and j. The connectivity matrix is then used to define the neighborhood of a network node using the graph distance. Each connection C_{ij} is also associated with an age parameter such that old connections can be removed during the training in favor of new connections.

The Neural Gas training algorithm is defined by the following steps:

1. Assign initial values to the weight vectors $\mathbf{w}_i \in \mathbb{R}^n$ and set all C_{ij} to zero.
2. Select an input vector $\mathbf{v}$ of the input manifold M.
3. For each unit i determine the number k_i of neurons j with

$$||\mathbf{v} - \mathbf{w}_j|| < ||\mathbf{v} - \mathbf{w}_i|| \tag{3.19}$$

by, e.g., determining the sequence $(i_0, i_1, ..., i_{N-1})$ of neurons with

$$||\mathbf{v} - \mathbf{w}_{i_0}|| < ||\mathbf{v} - \mathbf{w}_{i_1}|| < ... < ||\mathbf{v} - \mathbf{w}_{i_{N-1}}|| \tag{3.20}$$

4. Perform an adaptation step for the weights according to

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \epsilon \cdot e^{-k_i/\lambda}(\mathbf{v} - \mathbf{w}_i(t)),\ i = 1, ..., N. \tag{3.21}$$

5. If $C_{i_0 i_1} = 0$, set $C_{i_0 i_1} = 1$ and $t_{i_0 i_1} = 0$. If $C_{i_0 i_1} = 1$, set $t_{i_0 i_1} = 0$.

6. Increase the age of all connections of i_0 by setting $t_{i_0 j} = t_{i_0 j} + 1$ for all j with $C_{i_0 j} = 1$.

7. Remove all connections of i_0 which exceed their lifetime by setting $C_{i_0 j} = 0$ for all j with $C_{i_0 j} = 1$ and $t_{i_0 j} > a_{\max}$. Continue with 2.

The Neural Gas algorithm has also been applied to time series prediction as in [44] where the attractor of the Mackey-Glass time series [42] was represented with a NG network.

Also, the recursive and recurrent modifications of the SOM have been applied to the NG algorithm. In [70] the recursive use of the network activation is used for the SOM and NG networks. In [65] the proposed extension to the NG algorithm is similar to the RSOM algorithm [41].

3.4.2 Growing Neural Gas

A modification of the Neural Gas algorithm called Growing Neural Gas (GNG) by Fritzke [25] extends the NG algorithm with a growing network size. The capability of adding neurons while the training advances makes the GNG algorithm suitable for online learning situations where the complexity of the input data is unknown. Another virtue of the GNG algorithm compared to the SOM is that it does not include any time-dependent parameters.

Consider a set of A neurons each associated with a weight vector $\mathbf{w}_i \in \mathbb{R}^n$. A set of N un-weighted edges connecting the neurons defines the topology of the network. Each neuron i represents a point in the unknown n-dimensional input distribution $P(\xi)$ that the GNG algorithm is used to approximate.

The algorithm starts with only two neurons that have initially random weight vectors. As the training advances new neurons are added to regions where the quantization error for input signals is large.

The complete GNG algorithm is given by the following:

0. Create two neurons a and b with random weight vectors $\mathbf{w}_a$ and $\mathbf{w}_b$ in $\mathbb{R}^n$.
1. Generate an input signal ξ according to $P(\xi)$.
2. Find the nearest unit (in Euclidian metric) s_1 and the second nearest unit s_2 for the input signal.
3. Increment the age of all edges connecting to s_1.
4. Increment the local error counter variable of s_1 with the squared quantization error.

$$\Delta(\text{error})(s_1) = ||\mathbf{w}_{s_1} - \xi|| \tag{3.22}$$

5. Move all topological neighbors of s_1 towards ξ by fractions ϵ_b and ϵ_n, respectively, of the total distance:

$$\begin{aligned} \Delta\mathbf{w}_{s_1} &= \epsilon_b(\xi - \mathbf{w}_{s_1}) \\ \Delta\mathbf{w}_n &= \epsilon_b(\xi - \mathbf{w}_n) \text{ for all neighbors } n \text{ of } s_1 \end{aligned} \tag{3.23}$$

6. If s_1 and s_2 are connected by an edge, set the age of this edge to zero. If an edge does not exist, create it.
7. Remove edges with an age larger than $a_{\max}$. If this results in points having no emanating edges, remove them as well.
8. If the number of input signals generated so far is an integer multiple of a parameter λ, insert a new unit as follows:
   - Determine the unit q with the maximum accumulated error.
   - Insert a new unit r halfway between q and its neighbor f with the largest error variable:

$$\mathbf{w}_r = 0.5(\mathbf{w}_q + \mathbf{w}_f) \tag{3.24}$$

   - Insert edges connecting the new unit r with units q and f, and remove the original edge between q and f.
9. Decrease the error variables of q and f by multiplying them with a constant d.
10. If a stopping criterion (e.g. net size or some performance measure) is not yet fulfilled go to step 1.

The GNG algorithm results in a network that approximates the unknown probability density with a resolution that can be pre-defined without the need to select the size of the network beforehand.

3.5 SOM-based methods for process control

The Self-Organizing Map algorithm [40] has proven useful in situations where large amounts of multidimensional data is available but dependencies between individual variables are difficult to determine. This is a typical situation in process monitoring where the SOM algorithm has been successfully applied as an additional tool besides traditional statistical methods. This section introduces the use of the SOM in process monitoring, visualization and control. In the end of the section some real life applications are also presented.

3.5.1 Self-organized map of process data

Controlling dynamic processes typically involves a large amount of numerical information. Usually, the process can be controlled by changing certain process parameters in order to direct the process into a suitable state. Sensory information of various kinds can also be used to examine the current state of a process.

What makes the problem of process control difficult is that very often the large amount of process data is of little use, unless we have some conception of how changes in one variable effect some other variable. Traditional statistical methods can be used to analyze these dependencies but as the number of variables grows, the problem becomes overwhelming with traditional methods.

The SOM algorithm can be applied in process monitoring, visualization, and control to overcome the problem of identifying dependencies in a large amount of multi-dimensional data [34]. In this case, the SOM is trained with the available process data to form a vector-quantized mapping of the process state space. After training, the current state of a process can be associated with the best matching unit on the map to visualize the process (see Figure 3.8).

3.5.2 Detecting error states

One of the most typical tasks in process control is to avoid drifting to an error state [2] which is indicated by some abnormal interval in a monitored process variable. While this kind of monitoring can be done for each variable individually it is often useful to monitor the whole set of variables on the SOM plane.

Two different approaches have been presented [39] to detect abnormal (error) states. In the first one the process state map has been trained using data which has been collected during normal process behavior. Drifting to error states can be detected by looking at the quantization error $e(t)$ of the current state of the process $\mathbf{x}(t)$at each time instant t.

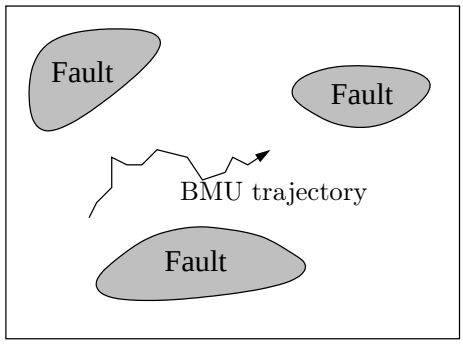


Figure 3.8: SOM visualization of a process state space with undesirable error state regions.

$$e(t) = ||\mathbf{x}(t) - \mathbf{m}_c(t)|| \tag{3.25}$$

When the process advances normally the quantization error remains small because the training data has been collected to give a good representation of normal behavior. Correspondingly, abnormal states will result an increase in the quantization error.

The second approach for detecting error states uses training data received during process faults. In this case the error states are mapped as error regions (Figure 3.8) on the map and drifting to one of these sates can be monitored as the BMU trajectory entering one of these regions.

3.5.3 Substitution of missing data

In some cases data may be missing from the vector describing the state of the process [32]. This could be due to a faulty sensor or some other error in the monitoring. A virtue of the SOM monitoring method is the redundant capability to track the process state even though some input were missing. This can be done by excluding the missing components of the state vector from the BMU search routine.

The process state map can also be used to predict the missing data values. In [26] this was done by evaluating the best matching neuron of the input vector using the available vector components. The missing data value(s) in the input vector could be approximated by the value found in the BMU neuron. It has also been suggested [40] that the human brain could process missing information in a similar manner.

3.5.4 Applications

The method of monitoring a dynamic system using the SOM as a representation of the system state space is a very general solution which could be applied to a wide range of problems. However, the list of real life examples of this method are involved with industrial processes.

In integrated circuit design the Self-Organizing Map can be used to aid the design process [26]. In this case the map is constructed using vectors formed of design parameters. The map can be thus used to indicate the state of the device and give an estimate of its life time.

In [68] the SOM-based monitoring method is applied to separating three materials in a chemical distillation process. The article points that the SOM-based control is especially useful in situations where mathematical equations give only a simplified model or are unsatisfying for other reasons.

In [48] the SOM monitoring method has been discussed in the context of power plants. A method for finding deviations in the process by following the BMU trajectory of the process state on the map is presented. A discussion on finding the cause of the deviation is also presented by comparing the weight distributions of the best matching map unit and the process state vector. In [27] the method is extended to build a neuro-fuzzy system with fuzzy state membership functions corresponding to the clusters found on the SOM.

Other examples include learning coordination of a robot arm [45] [52] and process monitoring in the manufacture of pharmaceutical products [50].

Chapter 4

Self-organizing neural networks for adaptive state space representation

4.1 Introduction

As discussed in Chapter 1 the operation of an anticipatory system can be decomposed into two separate tasks: constructing a meaningful internal model of the dynamic state space of the agent and its environment and, secondly, determining a policy for selecting appropriate actions for each perceived and predicted state of the environment in the present time and future.

This Chapter introduces a prototype-based neural network framework for modeling the state space of a dynamic system. Topological connections of the networks are used as an implicit representation of the time dependent state transitions which can be used to anticipate future states of the system.

Models based on the Self-Organizing Map are compared with Neural Gas algorithms to illustrate the different virtues of these two algorithm families.

4.2 State space as a pattern sequence

As discussed in Chapter 1 an agent is considered an entity which perceives its environment with sensors and performs actions with its actuators. The perceived information about the environment can be understood as a time-dependent signal which can be received from various sensors including

- visual perceptions

- auditory signals sensed as air pressure variations
- tactile signals received from haptic sensors
- temperature sensors
- kinesthetic senses

Each of these signals can be expressed as a discrete-time process $x(n)$. For example, in a system with visual, auditory, and tactile sensors the sensed state could constitute of three signals $x_v(n)$, $x_a(n)$, and $x_t(n)$. These individual sensory signals can be combined into a single time-varying vector quantity $\mathbf{x}(n)$ summarizing the sensed state of the system.

$$\mathbf{x}(n) = \begin{bmatrix} x_v(n) \\ x_a(n) \\ x_t(n) \end{bmatrix}^T \tag{4.1}$$

Though often instead of using the raw sensory signals, preprocessing tools can be applied to extract features and reduce the possibly large amount of sensory input variables.

The perceived state of the agent can thus be described as a point in the state space $\mathbb{R}^d$ were d is the sum of the dimensions of the individual inputs. The pattern sequence of successive perceived states can be written as

$$\begin{bmatrix} \mathbf{x}(0) \\ \mathbf{x}(1) \\ \mathbf{x}(2) \\ \vdots \\ \mathbf{x}(n) \\ \vdots \end{bmatrix} = \begin{bmatrix} x_1(0) & x_2(0) & x_3(0) & ... & x_n(0) \\ x_1(1) & x_2(1) & x_3(1) & ... & x_n(1) \\ x_1(2) & x_2(2) & x_3(2) & ... & x_n(2) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_1(n) & x_2(n) & x_3(n) & ... & x_n(n) \\ \vdots & \vdots & \vdots & ... & \vdots \end{bmatrix} \tag{4.2}$$

which can be viewed as a trajectory of the system's state in the state space.

Based on the feedback that the agent receives the state space can be partitioned into regions of similar states. A rational agent focuses its actions to preserve itself in a region of states where the feedback is positive. Also, a rational agent will avoid state space regions where negative reward is to be expected.

Using the above state space notation we can implement an anticipatory system by selecting appropriate actions in the present time to gain reward in the future by predicting the future state of the system based on an internal representation of the dynamics of the state space.

4.3 Prototype-based state space models

Constructing a model of a multi dimensional state space of an anticipatory agent aims to build an internal representation of the agent's environment. The problem is essentially a probability density estimation task, for which many approaches exist [6].

When creating a domain-specific AI system for a particular task, the available prior information about the data could be used to select a specific model class known to suit the phenomenon and use parametric methods to estimate the perceived state space distribution. However, this kind of estimate would only suit the specific problem. The general case of a multi dimensional state space with no prior information about the distribution encourages the use of non-parametric methods which make no assumptions on the data.

In the following sections, the problem of modeling a state space PDF is approached using prototype-based neural networks which can be seen as a multi-dimensional vector quantization of the space with added connections between the prototype nodes. Time is represented implicitly by the connections of the networks such that states occurring adjacently in time are assumed to reside close to each other in the network topology. This assumption of "slow state transformations" is – although a strong one – the only assumption that is made about the state space. This can be achieved by selecting a high frequency for the sampling of the continuous sensory input signals and a reasonably large network size such that the transition from state $\mathbf{x}(n)$ to state $\mathbf{x}(n+1)$ will be represented as a transition between two network nodes that reside close to each other in the network topology.

4.4 Implicit time topology

In the presented method of modeling the state space using a prototype-based neural network, the time dimension has only an implicit representation coded by the network connections.

Consider a sequence of vectors $\mathbf{x}(n)$ consisting of discretized values of continuous signals sampled such that the length of the difference vector $||\mathbf{x}(n) - \mathbf{x}(n+1)||$ has an upper limit. If a vector quantization network such as a SOM of NG is trained using this data set, the prototypes organize such that the continuous transitions of the system in the input space can be tracked by the BMU neurons in the network as in Figure 4.1. The connections of the network thus imply the temporal order in which the patterns are perceived.

Alternatively, the state space estimation task could be done using a topology-free vector quantization method as in Figure 4.2 where the neighborhood of the nodes is defined only in the input space. This, however, could lead to problems when estimating a non-convex distribution where a topology-free estimate would

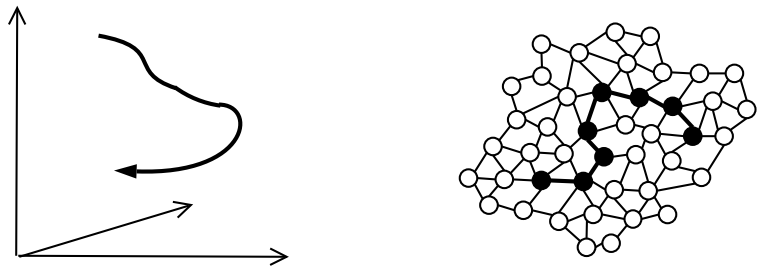


Figure 4.1: Continuous transitions in the high dimensional state space (left) can be tracked on the quantized prototype-based network (right).

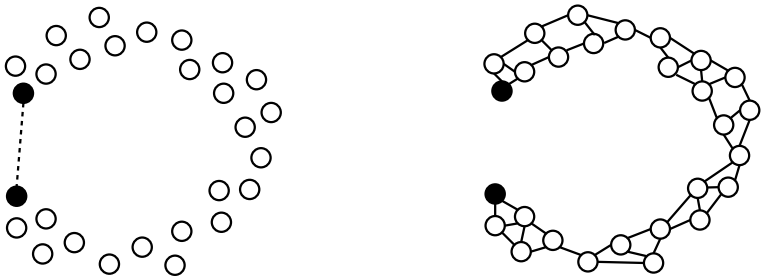


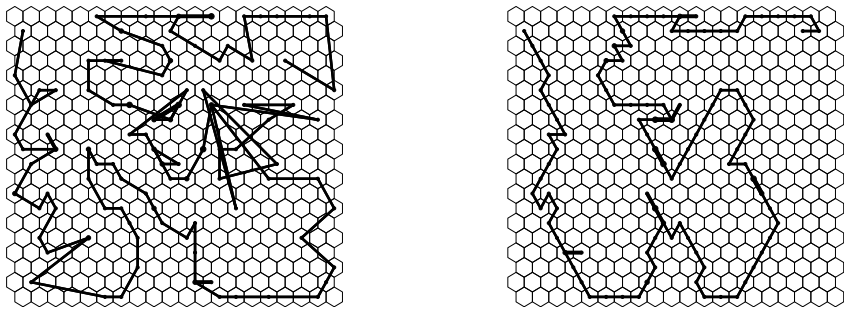
Figure 4.2: A topology-free vector quantization of a state space (left) implies a connection between two closest nodes. A Neural Gas -type network (right) avoids this as the distance between the nodes is computed as the number of connections between the nodes.

suggest a connection between two nodes that do not occur close to each other in time. A Neural Gas -type network is able to avoid such misinterpretations as the distance between two nodes is defined using the amount of connections between the nodes.

4.5 SOM-based state space representations

As discussed in Chapter 3, several methods for analyzing time dependent information using the SOM have been developed based on external delay mechanics and leaky integrator potentials. Although modified SOM algorithms are typically proposed for processing finite-length one-dimensional sequences, the principle of recurrent connections can also be applied when using the SOM as state space estimate.

The discussion on using the topological connections of the SOM network to represent time implicitly relies on assumption that two states occurring close to each other in time will be mapped close to each other on the SOM grid. While this assumption applies in the case of slow transformations in a convex data set, in a more general case the BMU trajectory on the SOM plane can be unpredictable.



a) b)

Figure 4.3: BMU trajectories of a standard SOM (a) and a one with contextual restriction in the BMU search algorithm (b).

Using recurrent connections in the inputs of the SOM nodes, similar to RSOM, the BMU search of the SOM can be altered such that the previous input patterns are considered when selecting the BMU node for a new input vector. In Figure 4.3 the trajectory for a standard SOM and a one with a neighborhood-guided BMU search is presented. As seen in the figure, the modified BMU search results in a more predictable trajectory.

4.6 Neural Gas -based state space representations

Modeling a state space using Neural Gas -type networks is much similar to the case of a SOM network. The crucial difference between the two network types is the way the distance between two nodes in the network is defined. The SOM has the advantage of a simple and constant grid topology, where the computation of the distance reduces to computing the Euclidian distance of two neurons. In the case of a rectangular SOM network the distance can alternatively be measured in Manhattan distance.

In NG networks, however, the only metric to define distances between two nodes is the amount of connections between the nodes. While the NG topology is a more flexible one, it requires more computation as the connections change over time.

The performance of the two types of networks estimating state spaces of different forms is tested in Chapter 5 including non-convex and disjoint distributions.

Algorithm	Dimensionality reduction	Dynamic topology	Dynamic size	Time-dependent parameters
SOM	yes	no	no	yes (α and σ)
NG	no	yes	no	none
GNG	no	yes	yes	none

Table 4.1: Comparison of different state space representation methods.

4.7 Discussion

In this Chapter, the idea of representing the state space of a dynamic system using a prototype-based neural network was introduced. The topological connections of the network was then used as an implicit representation of the temporal dynamics of the state space. This way, state anticipation (as described in Chapter 1) could be implemented by examining the network neighborhood of node representing the current state of the system and executing appropriate control actions to avoid unwanted states.

The most significant property of the proposed method is the capability to extend the topology-preserving nature of the SOM projection to the time dimension in the case of continuous-valued variables.

One significant advantage of the presented framework is its robustness against missing data. In a situation where the state space is of high dimensionality the effect of an individual data values decreases and the remaining dimensions (for which data exists) can be used to compute the BMU unit.

Chapter 5

Simulations

5.1 Introduction

In Chapters 3 and 4 a framework for implementing an anticipatory model of a dynamic system based on self-organizing neural networks was presented. In this Chapter, the SOM-based state space model is simulated and compared with the Neural Gas -based implementation.

In the first simulation the previously introduced scheme of monitoring the state of a process using a SOM projection of the process states is used to anticipate error states. The latter part of the simulations demonstrate the operation of the SOM and GNG networks for different kinds of distributions. The case of a single convex data set is extended to a case of nonstationary and nonconvex state space.

5.2 A SOM-based adaptive anticipatory system

In Chapter 3 the implicit time-representation of the SOM projection was used in visualizing the progress of an industrial process (see [2, 26, 34, 39]) by examining the trajectory of the BMU node corresponding to the current process state. In the simulations the same method is extended to apply anticipatory control operations to avoid error states.

In this simulation a dynamic system is described by six-dimensional vector $\mathbf{x} \in \mathbb{R}^6$ which represents the sensory inputs used to monitor the state of the system (Figure 5.1). A SOM projection $\mathbb{R}^6 \rightarrow \mathbb{R}^2$ was created to visualize the process on a two-dimensional surface. The process state variables were bounded to the interval $[-1,\ 1]$ to avoid re-normalizing the SOM. The normal behavior of the process was defined such that each of the process state variables x_i were required to remain in the interval $[-0.8,\ 0.8]$ and values $[-1,\ -0.8]$ or $[0.8,\ 1]$ in any of

the components were interpreted as undesired states.

At each time instance a random shift was added to the system's state such that successive states were computed as

$$\mathbf{x}(t) = \mathbf{x}(t-1) + \mathbf{v}(t) \tag{5.1}$$

where $\mathbf{v}(t) \sim N^6(0, 0.1)$ is Gaussian noise. For each state of the system the BMU node $\mathbf{m}_c$ was selected to classify the state one of the three categories

1. Normal operation, where each state variable of the BMU node $\mathbf{m}_c$ satisfy $-0.8 \leq m_i \leq 0.8$.
2. Undesired error states where $|m_i| > 0.8$.
3. States in the near vicinity of error states for which $||\mathbf{r}_i - \mathbf{r}_e|| < b$, where b is a parameter defining the width of the buffer region.

For the first class of states no actions were taken. If the current state was found represented as an error state the process was initialized to its original state. This rule is equivalent to the reactive paradigm where only the current state of the process is evaluated. However, if the current state was found to belong into the third category, the process was also initialized in anticipation of an undesired state in the near future.

In Figure 5.2 the BMU trajectories of a state of a reactive and an anticipatory system are shown. The reactive system goes through several faults (shown as SOM cells with a large black dot) while the anticipatory system avoids them by anticipating nearby errors and initializing its state to the allowed interval.

Adapting to changes in the environment can be taken into account by updating the SOM during the process (Figure 5.3). For example, the system might end up in a state which has no good representation on the SOM projection. This can be detected as increase in the quantization error between the system state and the corresponding best matching unit.

5.3 Dynamic SOM and Neural Gas -based state space models

In the following examples the SOM-based modeling of the state space of a dynamic system is compared to the topologically dynamic Growing Neural Gas algorithm.

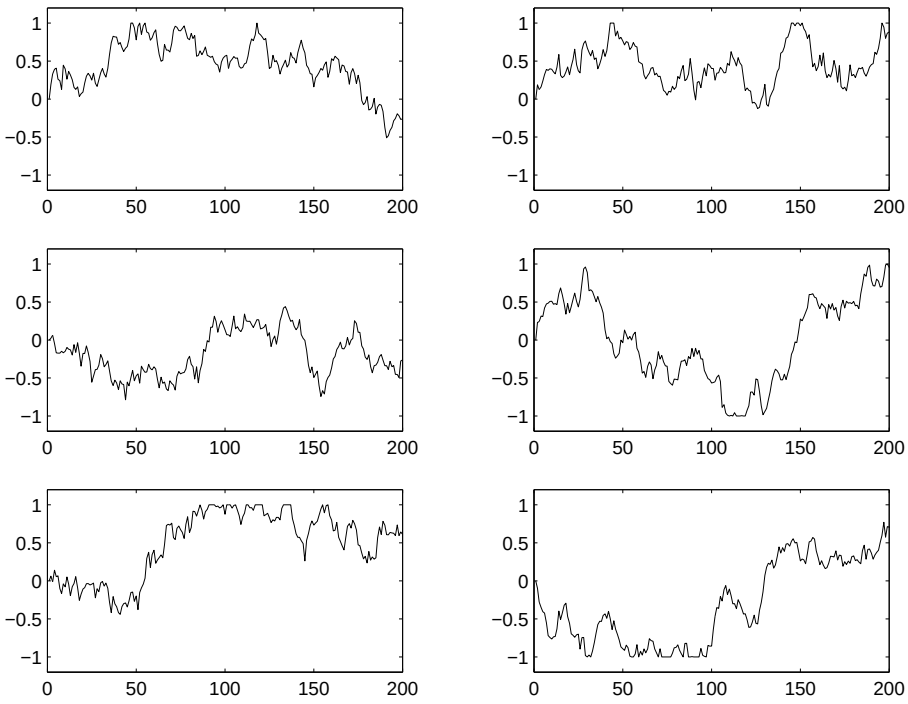
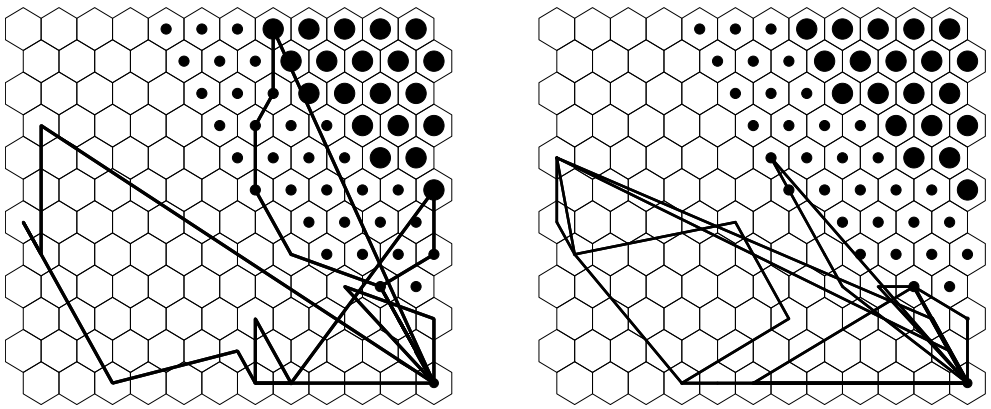


Figure 5.1: Component values of the six-dimensional random process.



a) b)

Figure 5.2: SOM trajectories of a a) reactive and an b) anticipatory system. Error states are marked with large black dots and the buffer region around the error states are marked with small dots.

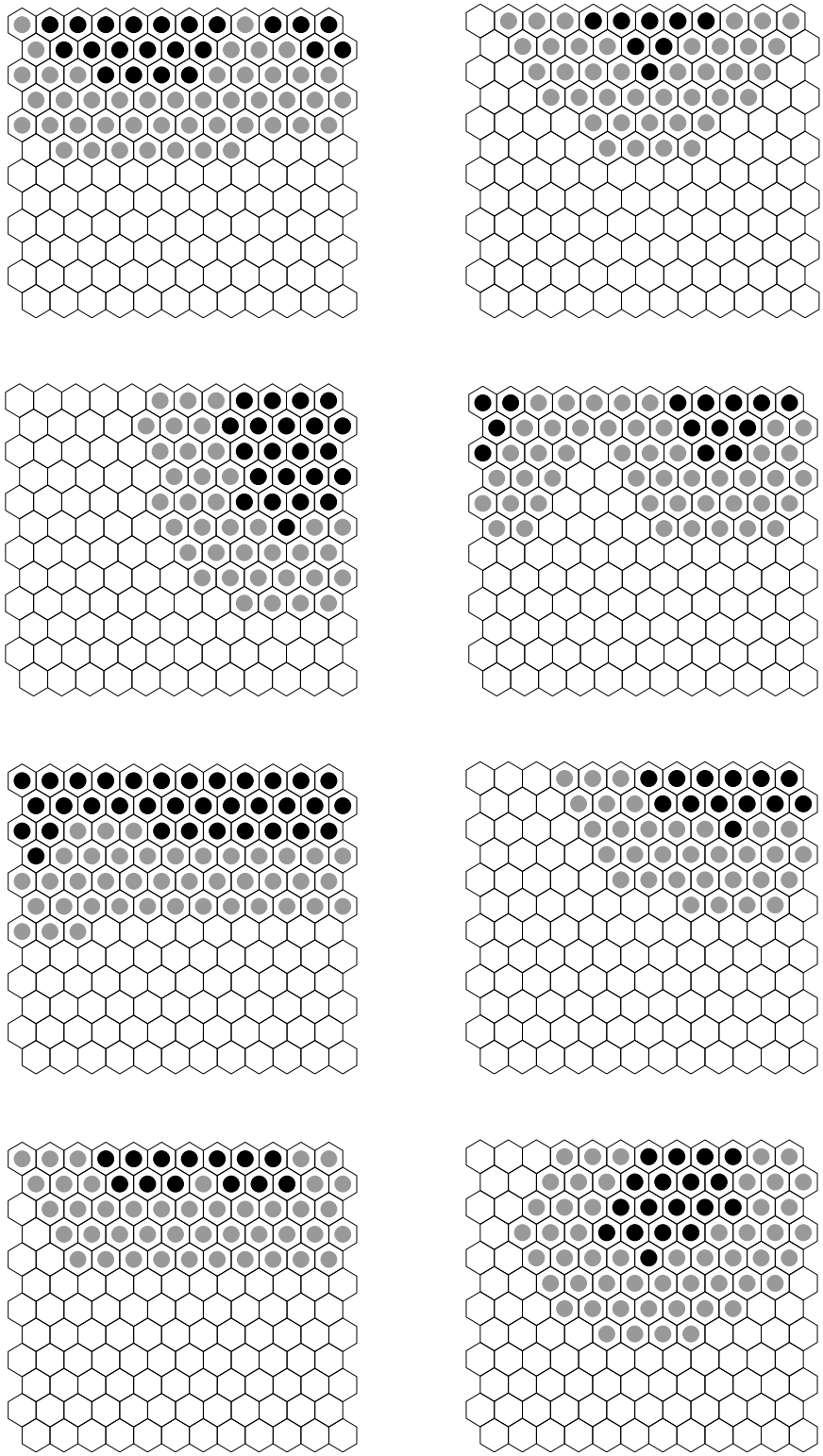


Figure 5.3: The SOM mapping of the state space is able to learn new undesired regions during the simulation. The buffer region around the error states marked as gray and the error states using black dots.

Modeling a non-convex distribution

One of the virtues of the SOM is the ability to describe a multidimensional dataset with a simple static network topology which is typically two-dimensional hexagonal or rectangular grid. For a simple convex data set[1] the topology of the SOM network can be used to approximate the mutual distances of the nodes in the input space. However, if the input data distribution consists of two or more separate regions or if the region is not convex the SOM topology does not describe well the input data distribution.

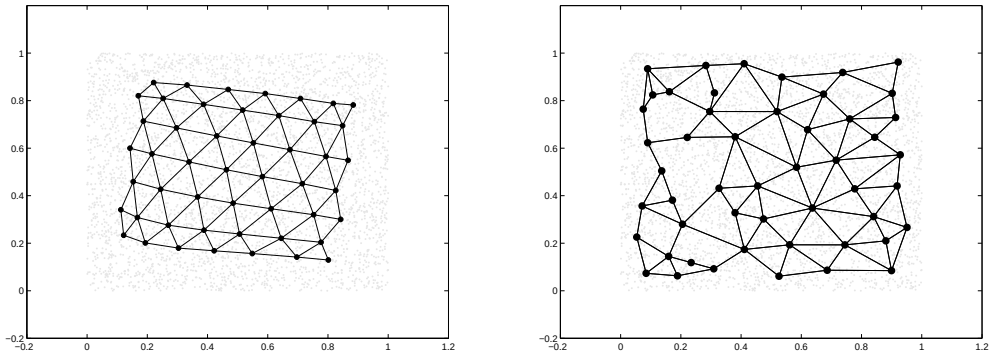
In this simulation, three data distributions were selected to compare the performance of the SOM and GNG in approximating the distribution and especially how the connections between the nodes describe the structure of the underlying PDF.

In Figure 5.4 a SOM and a GNG network are trained with a data set that favors the static topology of the SOM. Both of the networks learn an evidently good representation of the data set and the connections between the nodes have a strong connection to the distance between the nodes in the input space such that only nodes that reside close to each other in the input space have a connection in the network. A significant difference in the two vector quantized representations is how the outer boundary regions are represented. As seen in Figure 5.4a the SOM representation does not extend to the outer limits the network of Figure 5.4 is distributed evenly across the whole data set.

As a comparison to the SOM-friendly setting of the previous experiment, a non-convex dataset was modeled with the two VQ algorithms. In Figure 5.5 the result for a SOM (a) and a GNG (b) representation of a "U-shaped" data distribution is presented. As seen in Figure 5.5a the SOM representation of the data does not capture the shape of the distribution and many of the nodes fall in an area where no input exist. Also, the topological connections between the nodes in the extreme ends of the data distribution has the consequence that the connections can not be used to gain reliable information about the distance between individual nodes. In Figure 5.5b the corresponding result for the GNG algorithm is presented. This result shows how the GNG is able to capture the shape of the distribution such that the distance between two neurons measured as the network distance correlates strongly with the Euclidian distance between the nodes in the input space.

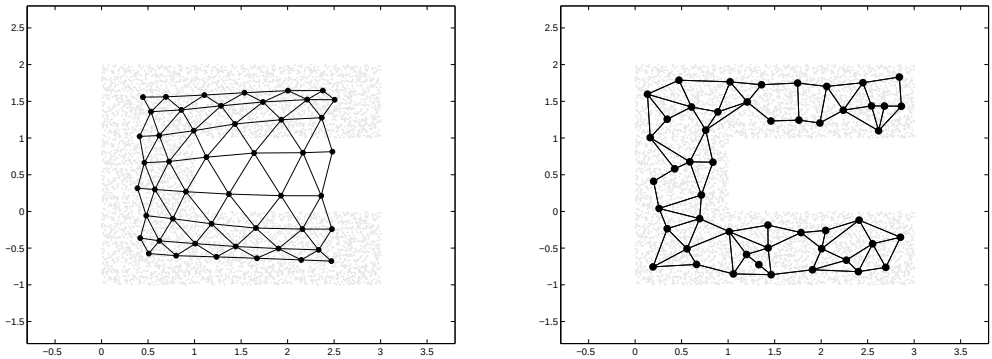
Another example of the two algorithms in the approximation task of a non-convex distribution is shown in Figure 5.6. As in the previous experiment the GNG is able to recognize the separate components of the PDF of the input data while the SOM stretches between the three parts. This result shown in Figure 5.6a is an extreme example of the possible hazards of relating the distance of two nodes in the SOM space to the corresponding distance in the input space.

[1]Here the term *convex data set* is used to describe a data set which has been generated from a convex probability distribution.



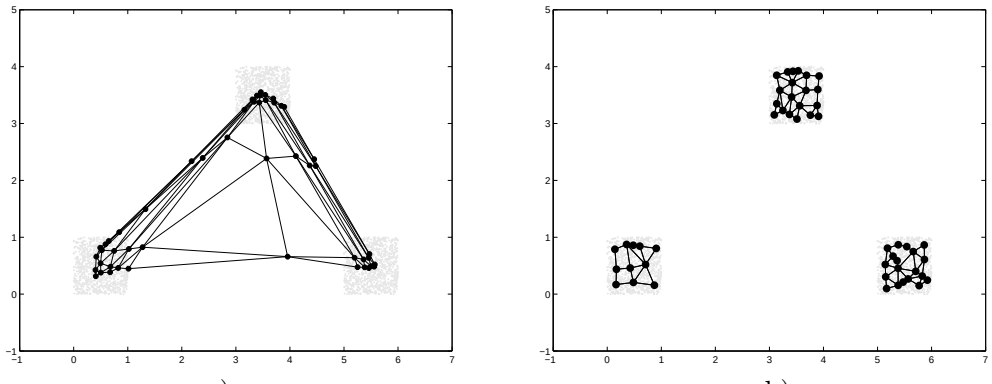
a) b)

Figure 5.4: A SOM (a) and a GNG (b) approximation of a uniformly distributed rectangular two-dimensional data set.



a) b)

Figure 5.5: A SOM (a) and a GNG (b) network approximation of a non-convex two-dimensional data set.



a) b)

Figure 5.6: SOM (a) and GNG (b) approximations of a PDF with separate regions.

	connection length mean	connection length variance
SOM (data set #1)	0.0787	0.0023
GNG (data set #1)	0.0973	0.0035
SOM (data set #2)	0.5801	0.0063
GNG (data set #2)	0.1244	0.4065

Table 5.1: Results of the node connection length experiment.

Adaptation to a nonstationary environment

A severe restriction of many classical statistical analysis tools, such as the ARMA model, is the requirement for the stationarity of the data (see Equation 2.8). When considering an autonomous agent with several sensory inputs operating in a complex environment the perceived data is know to be highly nonstationary since the environment itself changes over time.

In this experiment the adaptation to a changing state space distribution is tested for a GNG network.

In Figure 5.7 the initial state of the GNG representation of a data set generated from a uniformly $[-0.5,\ 0.5]$ distributed two-dimensional PDF is depicted together with the data points. After this, new inputs are fed to the network from a two-dimensional normal distribution $N^2(1, 1)$. The network adaptation to the new distribution is shown in Figure 5.7. The significance of this experiment is in the fact that the adaptation process is totally independent of time. This is a notable advantage of the GNG network type considering the situation of a continuous learning in a nonstationary environment.

Topological connection lengths

When using a prototype-based network as a model of the state space distribution as described in Chapter 4 the topological connections of the network can be related to time. If the precondition of slow transitions in the state space holds, the transitions in the BMU trajectory of the NN model will also occur between nodes that reside close to each other in the network.

As the anticipation on future states is done using the metrics of the NN model instead of the input space, it is important to consider how the connection lengths of the NN nodes relate to the corresponding distance in the input space.

In Figures 5.8, 5.9 and 5.10 the SOM and GNG quantizations of a data set is shown with the histogram of the connection length distribution for both networks. The important result of this experiment is that as a result of the dynamic topology of the GNG network, the variance of the connection lengths in the input space is small compared to the connection lengths of the SOM.

Looking at the connection length statistics of Table 5.1 it is apparent how the variance of the connection lengths is very large (0.4065) for the two-part PDF by comparison to the other variances. In the easy case of the single rectangular

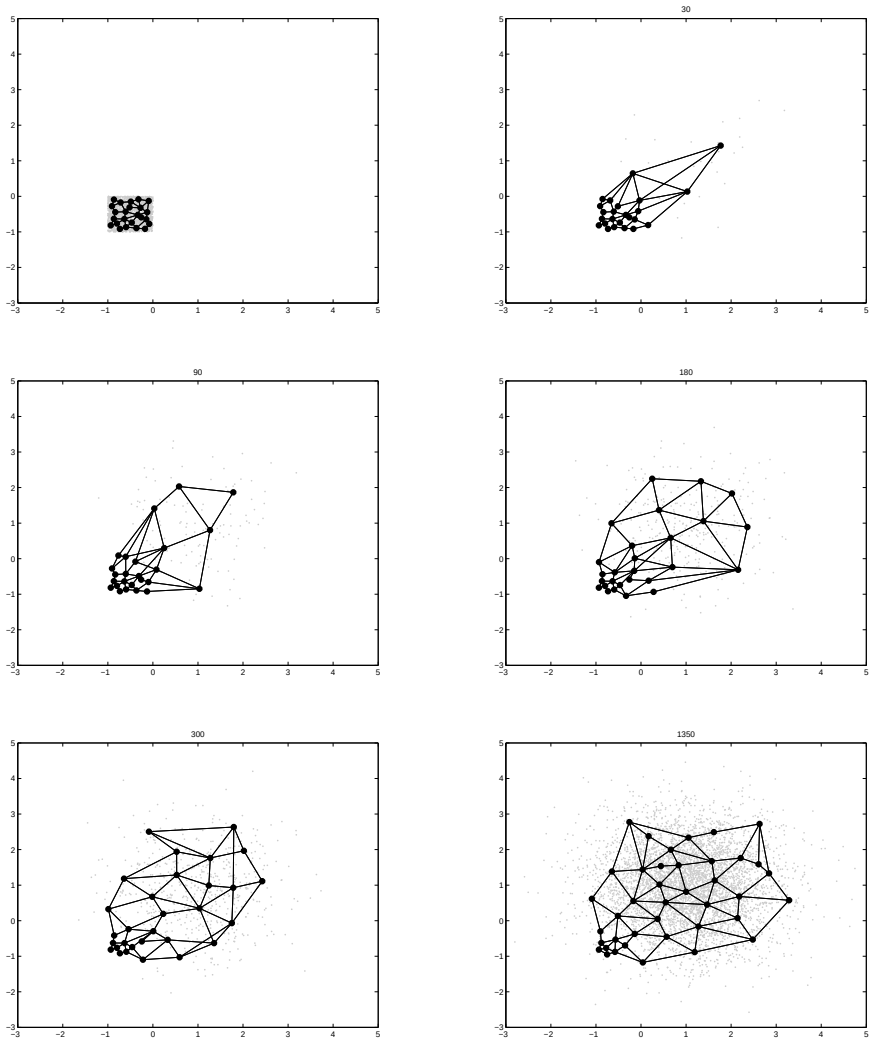


Figure 5.7: A GNG network adapting to a nonstationary data set.

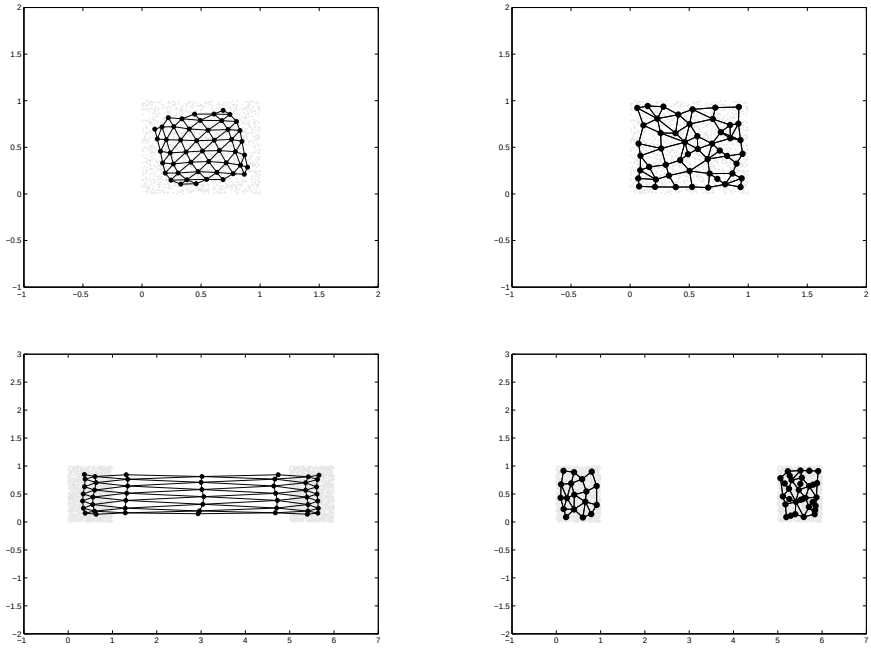


Figure 5.8: SOM (left) and GNG (right) approximations of a simple rectangular uniform distribution (upper) and a two-part distribution (lower).

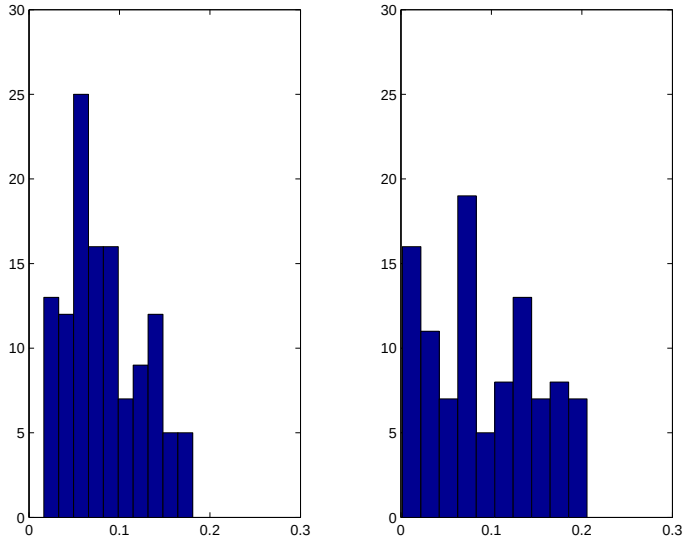


Figure 5.9: Connection length histograms for a SOM (left) and a GNG (right) network.

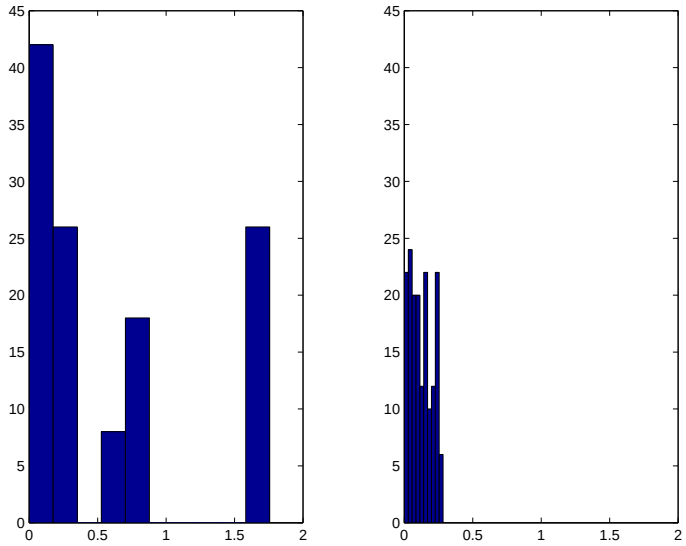


Figure 5.10: Connection length histograms for a SOM (left) and a GNG (right) network approximating a two-part distribution.

distribution the difference between the SOM are GNG are very small.

The conclusion of this experiment remains that the topological connections of the SOM network should not be directly associated to the corresponding distances in the input space. Specifically, when using the network topology as an implicit representation of time (as in the previous chapter), the shape of the state space distribution has a crucial influence to the applicability of this scheme.

Chapter 6

Discussion

6.1 Conclusions

In this thesis, the use of self-organizing neural networks was presented as a means for implementing a predictive model of an anticipatory system. This approach has a solid foundation considering the mechanics of natural cognitive systems employing anticipatory behavior. In many current implementations of anticipatory systems the design process follows the top-down paradigm of selecting and developing a model using the prior knowledge about the problem. In this regard the scientific contribution of this work is the alternative approach of implementing anticipation as an emergent feature of a dynamic state space model.

Two classes of neural network algorithms based on unsupervised learning were reviewed by their applicability to quantize the state space of a dynamic system. The SOM has been previously used to monitor the state of an industrial process and the idea of anticipating unwanted states using the topology of the SOM network was derived from the previous work by Tryba et al. [68].

The described scheme was used to implement a simple anticipatory system using the SOM topology as an implicit representation of the time dimension. However, for more complex state space distributions the SOM was found to have undesirable characteristics caused by the static rectangular topology.

The GNG algorithm was found preferable to the SOM when modeling a more complex state space. In the case of a non-convex state space the GNG was able to form a topology in which the implicit representation of time was plausible.

6.2 Future work

6.2.1 From reactive to anticipatory behavior

As discussed in the previous chapters anticipatory behavior is a vital component of a biologically plausible AI system. The experiments in Chapter 5 show how self-organizing neural networks can be used to implement a preventive state anticipatory system.

In the experiments performed in Chapter 5 the AI system made decisions combining reactive and anticipatory. However, the decision making system was simplified to a binary "OR" operation.

In reality, the balancing between reactive and anticipatory behavior is often much more complex including dependencies on the context. In an unpredictable environment where reliable predictions about the future are not available it would be suitable to "turn off" anticipatory mechanisms. On the other hand in a situation where predictions are very accurate the anticipatory component could be used with good results.

6.2.2 Learning and forgetting

Although neural networks are capable of learning large amounts of data, they usually suffer from forgetting old information. Especially in networks that have a static topology – such as the SOM – suffer from new data overwriting previously learned information also known as catastrophic forgetting [3, 53, 54, 23, 24].

If an anticipatory system employs a SOM model, there is an inevitable tradeoff between the plasticity of the learning and the capability of recall previously learned information. Specifically the two parameters α and σ play a crucial role in finding a balance between a system with large plasticity and a system with good capability to store previous information.

Contrary to the standard SOM algorithm where the two parameters decay over time, in a continuous learning the parameters have to be either selected as constants or alternatively address the problem with some other mechanism.

In feed-forward type networks catastrophic forgetting has been addressed by generating artificial input data which is then fed to the network together with new inputs. This method known as pseudorehearsal successfully retains a good representation of previous inputs in the network without the need for a infinite-length short term memory.

For unsupervised networks such as the SOM the idea of internally generated pseudodata is not applicable because it would severely disrupt the training algorithm itself. In [49] a scheme based on using the codebook vectors of the SOM as reminders of previous data was introduced to alleviate catastrophic

forgetting in a SOM network in a continuous learning task. Using this method the forgetting can be converted from catastrophic to gradual which is the case in biological networks.

The GNG algorithm, however, has only constant parameters and is thus more suitable in an online learning system.

6.2.3 State space estimates with explicit time representation

The discussed state space estimates represent time implicitly as a result of the gradual changes in the state of the system. While this approach is applicable for continuous state space transitions, the state space estimate could be constructed such that information on the temporal order of the inputs would affect the result.

Recurrent extensions to the SOM and NG type networks could be applied to overcome the precondition of slow transition in the state space as in [41] and [65]. Explicit information about the order of the inputs could extend the presented framework to support discontinuous transitions making the model significantly more plausible to real-world applications.

6.2.4 Anticipating when to act and how to act

As discussed in Chapter 1 the research on anticipatory behavior is strongly related to the reinforcement learning [64] paradigm – the problem of learning a policy of *how* to act in each situation to gain the highest possible overall reward without any explicit training information.

In this thesis, the subject of anticipatory behavior was approached more from the viewpoint of constructing an internal predictive model to predict *when* actions should be taken. Naturally, a sophisticated anticipatory agent would benefit from the combination of these two – anticipating *when and how* to act instead of selecting a generic action to resume the system to a normal state.

Bibliography

[1] Hirotugu Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, AC-19(6):716–723, 1974.

[2] 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.

[3] Bernard Ans, Stéphane Rousset, Robert M. French, and Serban Musca. Self-refreshing memory in artificial neural networks: learning temporal sequences without catastrophic forgetting. *Connection Science*, 16(2):71–99, June 2004.

[4] Robert Ariew. *Ockham's Razor: A Historical and Philosophical Analysis of Ockham's Principle of Parsimony.* PhD thesis, Champaign-Urbana, University of Illinois, 1976.

[5] Andrew Barron, Jorma Rissanen, and Bin Yu. The minimum description length principle in coding and modeling. *IEEE Trans. Information Theory*, 44(6):2743–2760, 1998.

[6] Cristopher Bishop. *Neural Networks for Pattern Recognition.* Oxford Universiy Press, 1995.

[7] Antonio Bonomi and A. G. Fredrickson. Protozoan feeding and bacterial wall growth. *Biotechnology and Bioengineering*, 18(2):239–252, 1976.

[8] George Box, Gwilym Jenkins, and Gregory Reinsel. *Time Series Analysis: Forecasting and Control.* Prentice Hall, Englewood Cliffs, New Jersey, 1994.

[9] Mark E. Burke. Further properties of derived scalar strong anticipatory systems. In *CASYS'03, 6th International Conference on Computing Anticipatory Systems*, AIP Conference Proceedings 718, pages 219–227. American Institute of Physics, 2003.

[10] Martin V. Butz, Olivier Sigaud, and Pierre Gérard. *Anticipatory Behavior in Adaptive Learning Systems*, chapter Internal Models of Anticipations

in Adaptive Learning Systems. Lecture Notes in Artificial Intelligence. Springer, 2003.

[11] Geoffery J. Chappel and John G. Taylor. The temporal kohonen map. *Neural Networks*, 6:441–445, 1993.

[12] Chris Chatfield. *Time-Series Forecasting*. Chapman & Hall/CRC, 2001.

[13] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. John Wiley, 1991.

[14] Paul Davidsson. A linearly quasi-anticipatory autonomous agent architecture: Some preliminary experiments. In C. Zhang and D. Lukose, editors, *Distributed Artificial Intelligence Architecture and Modelling (Lecture Notes in Artificial Intelligence 1087)*, pages 189–203. Springer-Verlag: Heidelberg, Germany, 1996.

[15] Paul Davidsson. Linearly anticipatory autonomous agents. In W. Lewis Johnson and Barbara Hayes-Roth, editors, *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, pages 490–491, New York, 5–8, 1997. ACM Press.

[16] Paul Davidsson. *Anticipatory Behavior in Adaptive Learning Systems*, chapter A Framework for Preventive State Anticipation. Lecture Notes in Artificial Intelligence. Springer, 2003.

[17] Guilherme de A. Barreto and Aluizio F. R. Araújo. Time in self-organizing maps: An overview of models. *International Journal of Computer Research*, 10(2):139–179, 2001.

[18] Daniel M. Dubois. Computing anticipatory systems with incursion and hyperincursion. In *Computing Anticipatory Systems: CASYS - First International Conference*, AIP Conference Proceedings 437, pages 3–29. American Institute of Physics, 1998.

[19] Daniel M. Dubois. *Anticipatory Behavior in Adaptive Learning Systems*, chapter Mathematical Foundations of Discrete and Functional Systems with Strong and Weak Anticipations. Number 2684 in Lecture Notes in Artificial Intelligence. Springer, 2003.

[20] Bradley Efron and Robert J. Tibshirani. *An Introduction to the Bootstrap*. Chapman & Hall/CRC, 1993.

[21] Bertil Ekdahl, Eric Astor, and Paul Davidsson. Toward anticipatory agents. In M. Wooldridge and N. R. Jennings, editors, *Intelligent Agents - Theories, Architectures, and Languages, Lecture Notes in Artificial Intelligence 890*, pages 191–202. Springer-Verlag: Heidelberg, Germany, 1995.

[22] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[23] Robert M. French. Catastrophic interference in connectionist networks: Can it be predicted, can it be prevented? In Jack D. Cowan, Gerald Tesauro, and Joshua Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 1176–1177. Morgan Kaufmann Publishers, Inc., 1994.

[24] Robert M. French. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Science*, 3(4):128–135, 1999.

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

[26] Karl Goser. Self-organizing map for intelligent process control. In *Proceedings of WSOM'97, Workshop on Self-Organizing Maps, Espoo, Finland, June 4–6*, pages 75–79. Helsinki University of Technology, Neural Networks Research Centre, Espoo, Finland, 1997.

[27] Karl Goser, K. Schuhmacher, M. Hartung, K. Heesche, B. Hesse, and A. Kanstein. Neuro-fuzzy systems for engineering applications. In R. V. Mayorga, editor, *AFRICON '96. Incorporating AP-MTT-96 and COMSIG-96. 1996 IEEE AFRICON. 4th AFRICON Conference in Africa. Electrical Energy Technology, Communication Systems, Human Resources*, volume 2, pages 759–64. IASTED-Acta Press, Anaheim, CA, USA, 1996.

[28] James D. Hamilton. *Time Series Analysis*. Princeton University Press, 1994.

[29] Monson H. Hayes. *Statistical Digital Signal Processing and Modeling*. John Wiley & Sons, New York, NY, 1996.

[30] Simon Haykin. *Neural Networks*. Prentice Hall, 2nd edition, 1999.

[31] Donald O. Hebb. *The Organization of Behaviour*. John Wiley and Sons, 1949.

[32] Jaakko Hollmén. Process modeling using the self-organizing map. Master's thesis, Helsinki University of Technology, 1996.

[33] Kevin Hynnä and Mauri Kaipainen. Activation-based recursive self-organizing maps: A general formulation and empirical results. *Neural Processing Letters*, 2004. accepted for publication.

[34] Heikki Hyötyniemi. Optimal control of dynamic systems using self-organizing maps. In Stan Gielen and Bert Kappen, editors, *Proc. ICANN'93, International Conference on Artificial Neural Networks*, pages 850–853, London, UK, 1993. Springer.

[35] Daniel L. James and Risto Miikkulainen. SARDNET: a self-organizing feature map for sequences. In G. Tesauro, D. Touretzky, and T. Leen,

editors, *Advances in Neural Information Processing Systems 7*, pages 577–84, Cambridge, MA, USA, 1995. MIT Press.

[36] Michael I. Jordan. Attractor dynamics and parallelism in a connectionist sequential machine. In *Proceedings of the 8th annual Conference on Cognitive Science Society*, pages 531–546, 1986.

[37] Jari Kangas. Time-delayed self-organizing maps. In *Proc. IJCNN-90, International Joint Conference on Neural Networks, San Diego*, volume II, pages 331–336, Los Alamitos, CA, 1990. IEEE Computer Society Press.

[38] Jari Kangas. *On the Analysis of Pattern Sequences by Self-Organizing Maps*. PhD thesis, Helsinki University of Technology, 1994.

[39] 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.

[40] Teuvo Kohonen. *Self-Organizing Maps*, volume 30 of *Springer Series in Information Sciences*. Springer, Berlin, Heidelberg, 1995. (Second Extended Edition 1997).

[41] Timo Koskela, Markus Varsta, Jukka Heikkonen, and Kimmo Kaski. Temporal sequence processing using recurrent SOM. In *2nd International Conference on Knowledge-Based Intelligent Engineering Systems*, volume 1, pages 290–297, Adelaide, Australia, April 1998.

[42] Michael C. Mackey and Leon Glass. Oscillations and chaos in physiological control systems. *Science*, (197):287–289, 1977.

[43] Thomas. M. Martinetz. Competetive hebbian learning rule forms perfectly topology preserving maps. In *ICANN'93: International Conference on Artificial Neural Networks*, pages 427–434, Amsterdam. Springer.

[44] Thomas M. Martinetz, Stanislav G. Berkovich, and Klaus J. Schulten. "neural-gas" network for vector quantization and its application to time-series prediction. *IEEE Transactions on Neural Networks*, 4(4):558–569, July 1993.

[45] Thomas M. Martinetz, Helge J. Ritter, and Klaus J. Schulten. Three-dimensional neural net for learning visuomotor coordination of a robot arm. *IEEE Trans. on Neural Networks*, 1(1):131–136, March 1990.

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

[47] Mihai Nadin. *Anticipatory Behavior in Adaptive Learning Systems*, chapter Not Everything We Know We Learned. Number 2684 in Lecture Notes in Artificial Intelligence. Springer, 2003.

[48] Ralf Otte and Karl Goser. New approaches of process visualization and analysis in power plants. In *Proceedings of WSOM'97, Workshop on Self-Organizing Maps, Espoo, Finland, June 4–6*, pages 44–50. Helsinki University of Technology, Neural Networks Research Centre, Espoo, Finland, 1997.

[49] Matti Pöllä, Tiina Lindh-Knuutila, and Timo Honkela. Self-refreshing SOM as a semantic memory model. In *Proceedings of International and Interdisciplinary Conference on Adaptive Knowledge Representation and Reasoning (AKRR'05)*, Espoo, Finland, June 2005. In press.

[50] Jukka T. Rantanen, Sampsa J. Laine, Osmo K. Antikainen, Jukka-Pekka Mannermaa, Olli E. Simula, and Jouko K. Yliruusi. Visualization of fluid-bed granulation with self-organizing maps. *Journal of Pharmaceutical and Biomedical Analysis*, 24(3):343–352, Jan 2001.

[51] Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

[52] Helge J. Ritter, Thomas M. Martinetz, and Klaus J. Schulten. Topology-conserving maps for learning visuo-motor-coordination. *Neural Networks*, 2(3):159–168, 1989.

[53] Anthony Robins. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, 7(2):123–147, 1995.

[54] Anthony Robins. Consolidation in neural networks and in the sleeping brain. *Connection Science*, 8(2):259–275, 1996.

[55] Robert Rosen. *Anticipatory Systems*. Pergamon Press, 1985.

[56] David E. Rumelhart, Richard Durbin, and R. J. Williams. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, chapter Learning internal representations by error propagation, Volume 1: Foundations, pages 318–362. MIT Press, Cambridge, MA, 1986.

[57] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2002.

[58] Yoshiyuki Sakamoto, M. Ishiguro, and G. Kitagawa. *Akaike Information Criterion Statistics*. Reidel Publishing Company, 1986.

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

[60] Robert Schalkoff. *Pattern Recognition: Statistical, Structural and Neural Approaches*. J. Wiley & Sons, 1992.

[61] J. Shao. Linear model selection by cross-validation. *Journal of the American Statistical Association*, (88):486–494, 1993.

[62] Yuri M. Shtarkov. Universal sequential coding of single messages. *Problems of Information Transmission*, (23):3–17, 1987.

[63] Panu Somervuo. Time topology for the self-organizing map. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN'99)*, volume 3, pages 1900–1905, Washington DC, USA, July 1999.

[64] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[65] M. A. Teixeira, G. Zaverucha, V. N. A. L. da Silva, and G. F. Ribeiro. Recurrent neural gas in electric load forecasting. In *IJCNN'99 International Joint Conference on Neural Networks*, volume 5, pages 3468–3473, July 1999.

[66] Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern Recognition*. Academic press, 1999.

[67] Jarkko Tikka, Jaakko Hollmén, and Amaury Lendasse. Input selection for long-term prediction of time series. In *Proceedings of the 8th International Work Conference of Artificial Neural Networks (IWANN'2005)*, Barcelona, Spain, June 2005.

[68] 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.

[69] A. Ultsch and H. P. Siemon. Kohonen's self organizing feature maps for exploratory data analysis. In *Proceedings of the International Neural Network Conference (INNC'90)*, pages 305–308, Dordrecht, Netherlands, 1990. Kluwer.

[70] Thomas Voegtlin. Recursive self-organizing maps. *Neural Networks*, (15):979–991, 2002.

[71] Paul J. Werbos. *Beyond regression: new tools for prediction and analysis in the behavioural sciences*. PhD thesis, Harvard University, Boston, MA, 1974.