Preface

This report is a result of a long-lasting search for simple and general approaches to modeling of neuronal behavior.

The original paper was submitted for publication in *Neural Networks* in early 2003. Valuable comments were given, and a revised, extended version was submitted in the end of 2003. Again, reviewer comments were given — one of the main criticisms was about the excessive length of the paper. Consequently, the paper was divided in three parts and resubmitted in June 2004. However, as one of the reviewers, for example, commented that *"I do not think linear approaches are interesting"*, it seems that the publication process will still take a long time.

However, it seems that the linear structures *are* extremely flexible and extensible, and the results of the papers are being implemented in theoretical and practical applications (see "`http://www.control.hut.fi/cybernetics`"). The papers are now published as a laboratory report.

[illegible]

Contents

Chapter 1

Analysis of Hebbian/Anti-Hebbian Learning

In this chapter it is shown how the very basic Hebbian/anti-Hebbian principles are only needed to implement principal subspace analysis without extra structural assumptions or nonlinearities. Stability of the neural structures is achieved by applying linear negative feedback. The resulting PCA scheme is compared against other neural principal component algorithms. The results are utilized towards a practical regression scheme, and it is shown how the same ideas can be utilized to implement a distributed sensor network.

1.1 Introduction

The research on artificial neural networks has departed from the original objectives — today, ANN's are seen only as computing devices, forgetting about the origins in artificial intelligence, when the operation of the brain was being studied and explained. Simultaneously, the models have become increasingly complex, so that efficient analysis methods do no more exist. For example, starting from the linear, intuitively appealing *Hebbian neuron* model, highly complex structures have been developed (for example, see [21], [14], [11]).

Have the limits of simple linear models been reached? Or could one still gain some physiological intuition from analysis of such simplistic models?

The intuition here is that *if one wants the models to be scalable to real life* *problems, the models have to be extremely simple.* In what follows, the goal is to explicitly stick to basic neurophysiological observations (Hebbian/anti-Hebbian principles).

Nonlinearities or extra structural assumptions are avoided. And, truly, it turns out that the analyzability that is reached when linear structures are employed facilitates compact formulations, new tools, and new intuitions.

In concrete terms, there are the following main parts in this report:

1. It is shown that direct implementation of Hebbian/anti-Hebbian learning with no nonlinearities, etc., implements *principal subspace analysis.* Further, if some additional structural constraints are implemented, *principal component analysis* and *regression* can be realized.

2. The compact mathematical framework is utilized by showing that there exist *optimality criteria* that correspond to the Hebbian/anti-Hebbian algorithm. This optimization framework is utilized to introduce extensions to the algorithm; for example, *sparse coding* and *self-organization* is implemented.

3. After the framework is defined, it is shown how the age-old dilemma of *quantitative vs. qualitative* can be effectively attacked. Truly, it seems that there might exist a fundamental duality between *structural* and *data-based representations.* Finally, a new generic framework is proposed for representing neural functions and cognitive functionalities alike.

1.2 Modeling of neurons

The behavior of a real neuron is extremely complicated, and when such neurons interact, the resulting behaviors are assumedly mindboggling. Perhaps some order still emerges from the chaos? Powerful conceptual tools are needed to master this complexity.

1.2.1 About system theory

System theory is the conceptual framework for capturing complex system behaviors. There are some ambitious, more or less obscure objectives that are used as guiding principles. Some of these principles are characteristic to good engineering work in general, and also in the field of neural networks research, they have already been exploited for a long time:

- **Modeling view.** Irrelevant application domain details are abstracted away: This means that rather than speaking of pulse trains, for example, one can concentrate on average activities in neurons. This starting point gives us a way to escape from the level of physical neurons to the level of information processing.
- **Search for structures.** The system boundaries are detected, inputs and outputs are distinguished, and the system behavior is studied reductionistically, conquering one phenomenon at a time.

As an illustration of the controversial nature of systemic thinking, quite opposite views have also been presented. Phenomenon-at-a-time approaches give no tools for attacking the system as a whole:

- **Holism.** The problems are seen in a wider perspective: Rather than studying single signals, for example, the whole system of signals is studied as a whole. What is more, rather than looking at some computations, one tries to see the underlying patterns.
- **Faith in interactions.** It may turn out that from the interactions between signals some unanticipated behaviour pops up. Networks of feedback structures result in dynamic fluctuations in the system, and understanding of such phenomena is crucial.

These are the basic ideas of *cybernetics* as originally presented in [51]. It is these facets of systems thinking that are explicitly concentrated on later.

When trying to be too ambitious, indeed, when pursuing General System Theory (see [5],[33]), the applicability of the above ideas necessarily becomes vague. To avoid sheer handwaving, an additional assumption is made here:

The neuronal structures that are studied are *essentially linear.*

The only justification for this assumption is based on intuition: We are not yet facing the End of Science; the neuronal system, after all, just *has to be* analyzable. Only for linear structures the *scalability* of the models can be reached, the discussions can be extended beyond toy domains, and properties of the whole system can be attacked using reductionistic approaches — the only ones we have. In Chapter 2 this linearity constraint is relaxed to make the studies better match reality.

In linear systems, the behaviors are straightforward, but they are still not necessarily trivial. There exist special ways to reach complex functionalities: *Dynamic structures* can be implemented, and, specially, it is *feedback* that can do wonders. Indeed, the behaviors can be too runaway, and for some more complex phenomena to emerge, yet another system theoretic basic intuition needs to be employed:

> The neuronal structures that are studied are *essentially stable.*

The loss of expressional power due to the linearity assumption, and the loss of intuitive appeal due to the stability assumption, are compensated in a system theoretic way: Rather than studying individual neurons, the whole grid of neurons is simultaneously taken into account. When applying system theoretical approaches, one is also facing a high-dimensional dynamic systems with plenty of interactions. To master such high-dimensional signal spaces, efficient tools for mastering parallel dynamic phenomena are employed: *Linear algebra* and *matrix calculus,* and the *theory of (linear) dynamic systems.* When the neurons are appropriately connected together, *self-stabilization* and *self-organization* takes place in the neuronal system.

1.2.2 Dynamics in a neuron grid

Neuron dynamics has been studied a lot, but when there are nonlinearities in the system structure, the analyses become extremely involved (for example, see [21], Chapter 14). Now, on the other hand, let us study what can be achieved when linear structures are assumed, and system theoretical tools are applied.

To achieve anything practical, the abstract ideas have to be grounded on concrete data. Assume that the m inputs to a neuronal (sub)system are collected in a vector u, and assume that the vector of neuronal activities in a grid of n neurons are denoted as x. The variables in u are assumed to have zero mean (towards the end of this report, this assumption is relaxed). During each time instant k there is a new input sample $u(k)$, and one would like to determine the corresponding "internal state" as determined by the neuronal activities.

One should not assume that, when given $u(k)$, the determination of $x(k)$ would be instantaneous. In what follows, it is studied what can be reached when the dynamic nature of the neuronal structure is explicitly taken into account. For this purpose, another (continuous) time variable t is needed. When the new input $u(k)$ is introduced, t starts from zero, and it is assumed that the adaptation of the continuous-time state vector $x_{\text{cont}}(t,k)$ continues until the steady state is reached.

As compared to traditional system theoretical approaches, now the problem is more challenging: Only the input data $u(k)$ is known, and one should somehow determine both the appropriate internal system structure and the internal state. Some assumptions are needed.

Assume that the *momentary change* in the neuronal activity is linearly proportional to the input activity and the current neuronal state. The whole grid of neurons can be simultaneously captured when matrix formulation is employed:

$$\frac{d\,x_{\text{cont}}}{d\,t}(t,k) = Ax_{\text{cont}}(t,k) + Bu(k). \tag{1.1}$$

The synaptic weights determining the connection strengths are collected in matrices A and B. In what follows, the dynamics is presented applying the discrete-time formulation to make the discussions directly compatible with algorithmic implementations. Approximating the derivative as

$$\frac{d\,x_{\text{cont}}}{d\,t}(t,k) \approx \frac{x_{\text{cont}}(t+h,k) - x_{\text{cont}}(t,k)}{h}, \tag{1.2}$$

and defining a discrete-time activity variable vector as $x(\kappa,k) = x_{\text{cont}}(\kappa h,k)$, one has

$$x(\kappa+1,k) = x(\kappa,k) + hAx(\kappa,k) + hBu(k). \tag{1.3}$$

In what follows, to avoid clumsy notations, either of the indices can be dropped if there is no risk of ambiquity. Assuming that the system is asymptotically stable, the state will converge to

$$\bar{x}(k) = \lim_{\kappa\to\infty}\{x(\kappa,k)\}. \tag{1.4}$$

1.2.3 Hebbian principles

To go deeper into the dynamics of the neuronal system, something more concrete needs to be assumed about the connections between the signals. For this purpose, it is the *Hebbian principle* that turns out to be fruitful:

> *If two neurons are activated at the same time, the connection between them is strengthened.*

This neuronal behavior was observed by the physician Donald O. Hebb some half a century ago [22]. This principle is among the only means that one has when trying to see the behavior of the neurons in a wider perspective. It gives intuition in what the neurons *try to do:* How do the neurons react to their environment, what kind of changes take place in neurons, or what is the neuronal learning principle. It gives a hint of how the low-level neuronal functions are connected to higher-level functionalities. It is interesting to see whether or not something emerges from such a simple principle.

To implement the Hebbian principle, it needs to be made more explicit. From now on, the following definition is utilized:

> **Hebbian law:** Synaptic connection between the neuron and an incoming signal becomes stronger if the signal and the current neuron activity correlate with each other.

According to this principle, the new value of the synapse weight between neuron i and input j, or r_{ij}, can be calculated as

$$r_{ij}(k+1) = r_{ij}(k) + \rho \bar{x}_i(k) u_j(k), \tag{1.5}$$

where ρ is the parameter determining the synaptic dynamics, and $\bar{x}(k)$ represents the steady-state neuron activity for the given input. As is known, simple Hebbian learning is unstable: Following the basic idea, the synaptic weights become higher and higher without limit. Typically in Hebbian algorithms, this instability is eliminated by introducing an additional nonlinearity (*Oja's rule,* see [42]). However, from the system theoretic point of view, stabilization of processes can be implemented also in linear terms by applying *negative feedback.* Only minor modification to the original formulation is needed:

$$r_{ij}(k+1) = r_{ij}(k) + \rho \bar{x}_i(k) u_j(k) - \frac{1}{\tau} r_{ij}(k). \tag{1.6}$$

Parameter τ is the time constant determining the rate of decay. Correspondingly, all connections from inputs to neurons can be expressed in matrix form as

$$R(k+1) = R(k) + \rho \bar{x}(k) u^T(k) - \frac{1}{\tau} R(k). \tag{1.7}$$

It needs to be recognized that this matrix notation only captures the $n \times m$ independent scalar expressions (1.6) in a compact form. If it is assumed that the system is (weakly) stationary, that is, the (second order) statistical properties of the data do not change over time, and if τ is large, one can solve for the steady-state value, so that the matrix of synaptic weights becomes (using the expectation operator in a somewhat sloppy way)

$$R = \rho\tau \, \mathrm{E}\{\bar{x}u^T\}. \tag{1.8}$$

That is, the matrix of R is the (scaled) *cross-correlation matrix.* Correlation matrices can be defined for signals in vectors x and u just in the same way, and in what follows, the correlation matrices are decomposed as

$$R_{\bar{x}\bar{x}} = \mathrm{E}\{\bar{x}\bar{x}^T\} = \begin{pmatrix} \mathrm{E}\{\bar{x}_1\bar{x}_1\} & \cdots & \mathrm{E}\{\bar{x}_1\bar{x}_n\} \\ \vdots & \ddots & \vdots \\ \mathrm{E}\{\bar{x}_n\bar{x}_1\} & \cdots & \mathrm{E}\{\bar{x}_n\bar{x}_n\} \end{pmatrix}, \tag{1.9}$$

and

$$R_{\bar{x}u} = \mathrm{E}\{\bar{x}u^T\} = \begin{pmatrix} \mathrm{E}\{\bar{x}_1 u_1\} & \cdots & \mathrm{E}\{\bar{x}_1 u_m\} \\ \vdots & \ddots & \vdots \\ \mathrm{E}\{\bar{x}_n u_1\} & \cdots & \mathrm{E}\{\bar{x}_n u_m\} \end{pmatrix}. \tag{1.10}$$

The expectation values have been calculated over the time index k. Even though the synaptic weights in the network have been expressed in a compact matrix form that makes it easy to present parallel operation, it should be noted that the structure of the matrices is such that the operation and adaptation of the neurons and their synaptic connections is purely local: Weights between neurons are determined by the corresponding output and input neurons alone.

It is now evident that, assuming that the synapses follow the streamlined Hebbian principle, the matrices in (1.3) can be selected as

$$A = -\mu \mathrm{E}\{\bar{x}\bar{x}^T\}, \tag{1.11}$$

and

$$B = \mu \mathrm{E}\{\bar{x}u^T\}. \tag{1.12}$$

Here, μ is a step size factor; its role is just to adjust the time scale in the adaptation algorithm. In any case, the same asymptotic behavior is reached however its value is selected assuming that the system remains stable. The overall system structure is shown in Fig. 1.1.

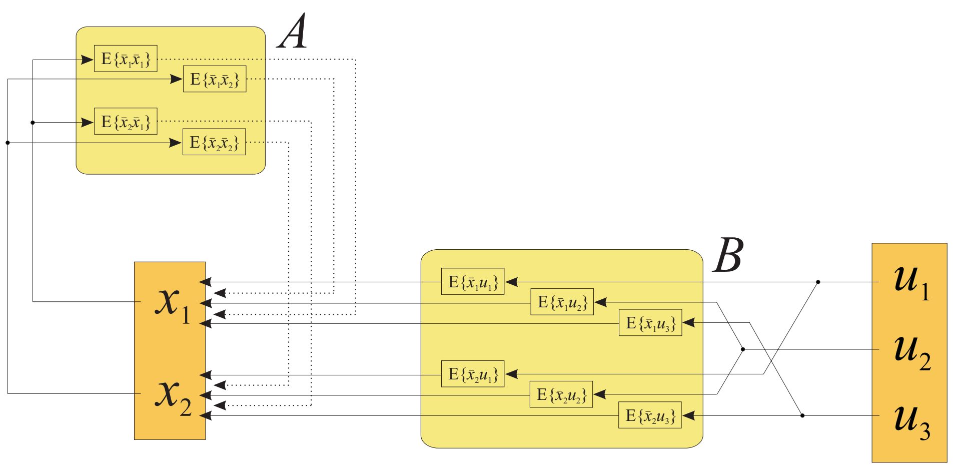


Figure 1.1: Local interactions as collected into matrices ($n = 2$ and $m = 3$)

The correlations-based structures in (1.11) and (1.12) are the same, but *signs are different.* The minus sign in (1.11) can be motivated exactly as in the case of a single synapse: Stability in the grid can be maintained applying linear dynamic structures if negative feedback is applied. One can define

> **Anti-Hebbian law:** Synaptic connection between two neurons becomes *weaker* if the neuronal activities correlate with each other.

The Hebbian laws can be interpreted so that the effects from the prior level are excitatory, but lateral connections between the same level neurons are *inhibitory;* or, actually, negative excitation (inhibition) becomes stronger.

Whereas the Hebbian learning has a long history, the term "anti-Hebbian learning" is newer, perhaps the most notable early studies being carried out in [17]. Hebbian and anti-Hebbian ideas have been applied more or less explicitly in all neural algorithms that carry out *principal component analysis* (see later): Hebbian learning searches for the maximum variation directions in the data, but without inhibitory effects from other competing neurons, there is no information to carry out organization among the neurons. In a grid of neurons, organization is carried out by implementing some kind of competitive learning.

1.2.4 Mathematics of Hebbian/anti-Hebbian learning

After the above assumptions are made, further analyses become possible. From (1.3) one has

$$x(\kappa+1,k)x(\kappa,k) - \mu h \mathrm{E}\{\bar{x}\bar{x}^T\}\, x(\kappa,k) + \mu h \mathrm{E}\{\bar{x}u^T\}\, u(k). \tag{1.13}$$

For a discrete-time system to remain stable, all of the system matrix eigenvalues have to be located within the unit circle (or, in the case of real eigenvalues, between -1 and 1; see [2]). This condition is fulfilled if $0 < \mu h < 2/\lambda_{\max}\left\{\mathrm{E}\{\bar{x}\bar{x}^T\}\right\}$, where $\lambda_{\max}$ denotes the largest eigenvalue of the matrix; if $\mu h = 2/\left(\lambda_{\max}\left\{\mathrm{E}\{\bar{x}\bar{x}^T\}\right\} + \lambda_{\min}\left\{\mathrm{E}\{\bar{x}\bar{x}^T\}\right\}\right)$, fastest possible convergence is reached (eigenvalues being nearest to the origin). When the steady state has been reached, there must hold

$$\bar{x}(k) = \bar{x}(k) - \mu h \mathrm{E}\{\bar{x}\bar{x}^T\}\, \bar{x}(k) + \mu h \mathrm{E}\{\bar{x}u^T\}\, u(k), \tag{1.14}$$

so that, when solved for $\bar{x}$,

$$\bar{x}(k) = \mathrm{E}\{\bar{x}\bar{x}^T\}^{-1}\mathrm{E}\{\bar{x}u^T\}\, u(k), \tag{1.15}$$

regardless of μ and h (assuming stability of the iteration). Inverse exists if the variables in x are not linearly dependent. There also exists a static linear mapping between $u(k)$ and $\bar{x}(k)$:

$$\bar{x}(k) = \phi^T u(k), \tag{1.16}$$

where the $m \times n$ matrix ϕ is defined as

$$\phi = \mathrm{E}^T\{\bar{x}u^T\}\mathrm{E}\{\bar{x}\bar{x}^T\}^{-1}. \tag{1.17}$$

Because the final neural activity, or $\bar{x}(k)$ corresponding to $u(k)$, is not known beforehand, determination of the covariance matrices becomes an iterative process.

Next, study the covariance of the neuronal state, as calculated from (1.15), taking the expectation of the sidewise outer products:

$$\mathrm{E}\{\bar{x}\bar{x}^T\} = \mathrm{E}\{\bar{x}\bar{x}^T\}^{-1}\mathrm{E}\{\bar{x}u^T\}\,\mathrm{E}\{uu^T\}\,\mathrm{E}^T\{\bar{x}u^T\}\mathrm{E}\{\bar{x}\bar{x}^T\}^{-1}. \tag{1.18}$$

Here it needs to be recognized that the above equation is justified only if the input is stationary and the covariance matrices have converged. Assuming that these assumptions hold, one can multiply the expression from left and from right by the covariance matrix $\mathrm{E}\{\bar{x}\bar{x}^T\}$, so that

$$\mathrm{E}^3\{\bar{x}\bar{x}^T\} = \mathrm{E}\{\bar{x}u^T\}\mathrm{E}\{uu^T\}\mathrm{E}^T\{\bar{x}u^T\}. \tag{1.19}$$

Applying (1.16), this becomes

$$\left(\phi^T\mathrm{E}\{uu^T\}\phi\right)^3 = \phi^T\left(\mathrm{E}\{uu^T\}\right)^3\phi. \tag{1.20}$$

Note that ϕ is non-trivial only when some compression takes place, that is, when $n < m$ (otherwise the identity matrix $\phi = I_n$ will do). What are the properties of the mapping matrix ϕ?

1.2.5 Principal subspace analysis

To understand the properties of the mapping represented by the matrix ϕ, one needs to study the properties of the eigenvalues and eigenvectors of the covariance matrix $\mathrm{E}\{uu^T\}$ (or *principal components*; see Appendices). It turns out that there holds

Theorem 1.

Assume that D is *any* invertible $n \times n$ matrix, and θ is *any* subset of n eigenvectors of $\mathrm{E}\{uu^T\}$. Then (1.20) is fulfilled by $\phi = \theta D$.

Proof.

Assume that the matrix ϕ in (1.20) is selected as

$$\phi = \theta D. \tag{1.21}$$

Here D is any orthogonal $n \times n$ matrix, so that $D^TD = DD^T = I_n$, and θ is a subset of distinct eigenvectors of $\mathrm{E}\{uu^T\}$. The left-hand side of (1.20) becomes

$$\begin{array}{rcl}\left(\phi^T\mathrm{E}\{uu^T\}\phi\right)^3 & = & \left(D^T\theta^T\Theta\Lambda\Theta^T\theta D\right)^3 \\ & = & \left(D^T\Lambda_\theta D\right)\left(D^T\Lambda_\theta D\right)\left(D^T\Lambda_\theta D\right) \\ & = & D^T\Lambda_\theta^3 D.\end{array} \tag{1.22}$$

The diagonal $n \times n$ matrix Λ_θ contains on the diagonal those eigenvalues λ_i that correspond to those eigenvectors that are selected in θ. Correspondingly, the right-hand side of (1.20) becomes

$$\begin{array}{rcl} \phi^T \left(\mathrm{E}\{uu^T\} \right)^3 \phi & = & D^T \theta^T \left(\Theta\Lambda\Theta^T \right) \left(\Theta\Lambda\Theta^T \right) \left(\Theta\Lambda\Theta^T \right) \theta D \\ & = & D^T \Lambda_\theta^3 D. \end{array} \tag{1.23}$$

This means that (1.21) gives a set of solutions fulfilling the constraint (1.69). The columns in ϕ are also linear combination of some of the principal components in the input data u. □

Theorem 1 gives the fixed points of the iteration (1.13). But some fixed points of the iteration can be unstable: When the iteration is carried out, and when the covariance matrices are updated accordingly, where will the process actually converge? This question is solved by

Theorem 2.

When the process (1.13) is continued until convergence, the basis θ spanning the subspace of the mapping matrix ϕ consists of the most significant principal component directions (that is, the n eigenvectors of $\mathrm{E}\{uu^T\}$ corresponding to the largest eigenvalues).

Proof.

It turns out that it is beneficial to express the data in the coordinate system spanned by the eigenvectors of $\mathrm{E}\{uu^T\}$, that is, by the columns of Θ. This means that one can write

$$u(k) = \eta_1(k)\Theta_1 + \cdots + \eta_m(k)\Theta_m = \Theta\eta(k). \tag{1.24}$$

Because of the uncorrelatedness of the principal components, the covariance of the data vectors η in the new basis must be diagonal, $\mathrm{E}\{\eta\eta^T\} = \Lambda$. Correspondingly, the $n \times m$ matrix B can be represented as

$$B^T = \Theta_1 b_1 + \cdots + \Theta_m b_m = \Theta b, \tag{1.25}$$

where b_i are $1 \times n$ vectors revealing the relevances of the eigenvectors i when representing the n rows of B. Using these notations, one can write

$$\bar{x}(k) = -A^{-1}Bu(k) = -A^{-1}b^T\Theta^T\Theta\eta(k) = -A^{-1}b^T\eta(k). \tag{1.26}$$

When continuing with the adaptation analysis, the stationarity assumption has to be abandoned, that is, rather than speaking of actual covariances, one can only speak of sample covariances over a finite time interval. Assuming that the not-yet-converged matrix B has been presented by the matrix b_{old}, study what it will become after new data has been observed and the corresponding "virtual" data covariance structures are being updated. Using (1.24) and (1.26), one can write for the updated matrix

$$\begin{aligned} B_{\text{new}} &= \mathrm{E}\{\bar{x}u^T\} \\ &= \mathrm{E}\{-A^{-1}b_{\text{old}}^T\eta(k)\eta^T(k)\Theta^T\} \\ &= -A^{-1}b_{\text{old}}^T\mathrm{E}\{\eta\eta^T\}\Theta^T \\ &= -A^{-1}b_{\text{old}}^T\Lambda\Theta^T. \end{aligned} \tag{1.27}$$

Now, calculate the (unnormalized) correlation matrices between Θ and the old and new B matrices:

- Before the adaptation step:

$$B_{\text{old}}\Theta = b_{\text{old}}^T = \begin{pmatrix} \left(B_{\text{old}}^T\right)_1^T\Theta_1 & \cdots & \left(B_{\text{old}}^T\right)_1^T\Theta_m \\ \vdots & \ddots & \vdots \\ \left(B_{\text{old}}^T\right)_n^T\Theta_1 & \cdots & \left(B_{\text{old}}^T\right)_n^T\Theta_m \end{pmatrix}. \tag{1.28}$$

- After the adaptation step:

$$\begin{aligned} B_{\text{new}}\Theta &= -A^{-1}b_{\text{old}}^T\Lambda \\ &= -A^{-1}\begin{pmatrix} \lambda_1\left(B_{\text{old}}^T\right)_1^T\Theta_1 & \cdots & \lambda_m\left(B_{\text{old}}^T\right)_1^T\Theta_m \\ \vdots & \ddots & \vdots \\ \lambda_1\left(B_{\text{old}}^T\right)_n^T\Theta_1 & \cdots & \lambda_m\left(B_{\text{old}}^T\right)_n^T\Theta_m \end{pmatrix}. \end{aligned} \tag{1.29}$$

Here, $\left(B_{\text{old}}^T\right)_i^T$ denotes the i'th row in matrix B_{old}. In (1.29), he diagonal matrix Λ scales the columns in b_{old} by the eigenvalues; assuming that the process remains stable the emphasis among the elements in b_{old} can only be redistributed. This means that the relative weighting of the most significant eigenvectors grows in the first elements of b_{new} (assuming that there is ordering between the eigenvalues, $\lambda_1 \geq \cdots \geq \lambda_n > \lambda_{n+1} \geq \cdots \geq \lambda_m \geq 0$). The first part in the expression, or $-A^{-1}$, can only construct linear combinations of the rows in b, without affecting the "lateral" distribution of the weights. Thus, it turns out that the matrix B becomes better aligned with the n most significant eigenvectors of the input covariance matrix. When this process is iterated sufficiently many times, and when the data structures have converged, the matrix B assumedly spans the subspace determined by the most significant eigenvectors. This all means that Hebbian/anti-Hebbian learning spans the *principal subspace* of the input data. □

It has been known for a long time that the operation of Hebbian units have very much to do with principal components [42]. It turns out, however, that the cumbersome sequential extraction of covariance matrix eigenvectors (see [30], [47]) is not necessary in the Hebbian framework, but a parallel, linear, and neurally more streamlined process suffices. The complexity is now buried in the dynamic iterative structure where the covariances are determined by the neural states, and the neural states are determined by the covariances, respectively.

1.3 Neural regression

When studying Hebbian learning, it is customary to construct models with input only, that is, for data analysis purposes. To reach wider applicability of the models, it is beneficial to also include *synthesis,* or construction of output signals. It turns out that such *neural regression* can readily be implemented in the proposed framework.

1.3.1 Output neurons

Assume that one has done principal component analysis, and extracted the *latent variables* $\chi(k)$, or the "scores" of the input data when projected onto the basis determined by the n most significant principal components, collected in the matrix θ:

$$\chi(k) = \theta^T u(k). \tag{1.30}$$

Assuming that $n < m$, some information is lost in the projection — but it can be assumed that the most relevant dependencies among data are captured by the most significant principal components, and (hopefully) information is retained whereas noise is filtered. From this "internal image" $\chi(k)$ one can map the data to some external output $y(k)$. This kind of regression from input to output is called *principal component regression (PCR).* The final mapping from the latent variables onto the output space is carried out by the normal linear regression formula

$$\hat{y}(k) = \mathrm{E}\{y\chi^T\}\mathrm{E}\{\chi\chi^T\}^{-1}\chi(k). \tag{1.31}$$

In PCR, the latent variables are the scores for the most significant covariance matrix eigenvectors, that is, data is explicitly projected onto the principal components. However, it can be shown that the principal component basis does not need to be explicitly solved. Assume that one has $\phi = \theta D$, so that

$$\bar{x}(k) = \phi^T u(k) = D^T\theta^T u(k) = D^T \chi(k). \tag{1.32}$$

Using this data as latent variables, instead of using χ variables, one can express the estimate for output as

$$\begin{array}{rcl} \hat{y}'(k) & = & \mathrm{E}\{y\bar{x}^T\}\,\mathrm{E}\{\bar{x}\bar{x}^T\}^{-1}\,\bar{x}(k) \\ & = & \mathrm{E}\{y\chi^T\}D\,D^T\mathrm{E}\{\chi\chi^T\}^{-1}D\,D^T\chi(k) \\ & = & \mathrm{E}\{y\chi^T\}\,\mathrm{E}\{\chi\chi^T\}^{-1}\,\chi(k). \end{array} \tag{1.33}$$

The result is the same as in (1.31), meaning that there is no need to explicitly solve for the principal components to implement PCR; the principal subspace suffices.

The complete mapping from the input to the output can be explicitly written and decomposed as follows:

$$\hat{y}(k) = \mathrm{E}\{y\bar{x}^T\}\,\underbrace{\mathrm{E}\{\bar{x}\bar{x}^T\}^{-1}\,\underbrace{\mathrm{E}\{\bar{x}\bar{x}^T\}^{-1}\mathrm{E}\{\bar{x}u^T\}u(k)}_{\bar{x}(k)}}_{\bar{v}(k)}, \tag{1.34}$$

where the dummy vector $\bar{v}$ is defined as

$$\bar{v}(k) = \mathrm{E}\{\bar{x}\bar{x}^T\}^{-1}\bar{x}(k). \tag{1.35}$$

To implement neural regression in the proposed neural framework, one extra step is also needed: The covariance matrix $\mathrm{E}\{\bar{x}\bar{x}\}$ has to be inverted one more time to implement the least-squares mapping from the latent space into the output. Comparing (1.35) to (1.15), it is evident that this inversion can be accomplished by repeating the anti-Hebbian structure; that is, a new neuronal layer is introduced with $A' = -\mu\mathrm{E}\{\bar{x}\bar{x}^T\}$, $B' = I_n$, the input for this layer being $\bar{x}(k)$. This corresponds to an additional dynamic structure

$$v(\kappa+1,k) = v(\kappa,k) - \mu h\mathrm{E}\{\bar{x}\bar{x}^T\}v(\kappa,k) + \mu h\bar{x}(k). \tag{1.36}$$

After this, the output is calculated as

$$\hat{y}(k) = \mathrm{E}\{y\bar{x}^T\}\, \bar{v}(k) = C\bar{v}(k). \tag{1.37}$$

Looking at the formulas (1.36) and (1.37), one can see that perhaps the easiest way to explain regression is to define a special *output neuron.* As compared to the input neurons that have been studied this far, the operational principles have been inverted in the grid of output neurons:

- Whereas in the input neuron the variable value that is used for adapting the covariance matrices is the final value after convergence, in the output neuron it is the *initial input* that is used for adaptation.
- Whereas in the input neuron the matrix B operates on the incoming data, in the output neuron the matrix C operates on the *outgoing data* after the dynamic structure.

The structure of the output neurons, as compared to the input neurons, is depicted in Fig. 1.2. Smoothing of data, or filtering of noise is achieved if one selects $y = u$. The principal component approach gives the minimum for the average reconstruction error $\mathrm{E}\{\|u - \hat{u}\|^2\}$ among all models with the same number of latent variables.

The duality between input and output neurons is based on the strange analogy between the formula (1.15) and the structure of the general multilinear regression model:

$$\hat{u}(k) = \mathrm{E}^T\{\bar{x}u^T\}\mathrm{E}\{\bar{x}\bar{x}^T\}^{-1}\, \bar{x}(k). \tag{1.38}$$

It is evident that in this least-squares case there holds

$$\hat{u}(k) = \phi\bar{x}(k). \tag{1.39}$$

This means that if the system has converged, the estimate for u can directly be obtained by multiplying $\bar{x}$ by ϕ. The regression formula is derived by explicitly minimizing the expected error variances when mapping from $\bar{x}$ to u; in the current case the origin of the mapping matrix ϕ is completely different, being a consequence of of Hebbian/anti-Hebbian learning principle.

In multilinear regression, the covariance structure within the input data is first compensated by multiplying the data by the inverse covariance matrix, and only after that, the data is projected onto the output space. In

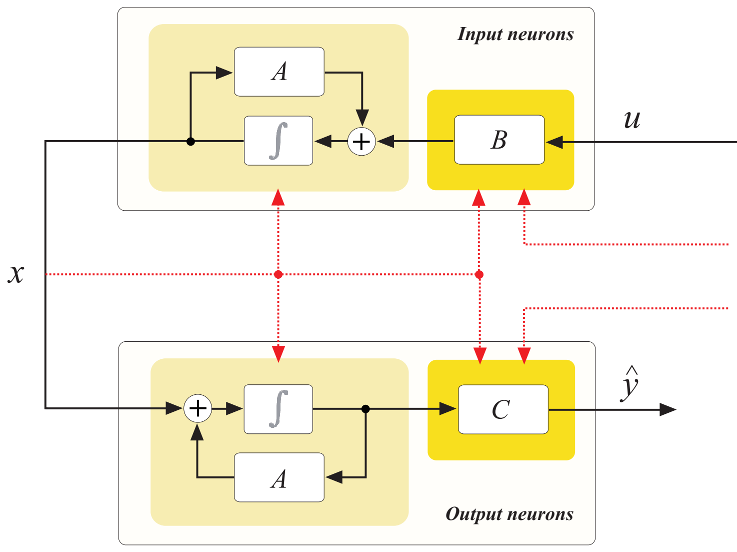


Figure 1.2: Input neurons vs. output neurons

Hebbian/anti-Hebbian learning, on the other hand, the data is normalized *after* the projection. This "postprocessing" makes the elements of $\bar{x}$ less correlated. In both cases the inverted matrix $\mathrm{E}\{\bar{x}\bar{x}^T\}^{-1}$ operates in the (low-dimensional) space of $\bar{x}$ rather than in the (high-dimensional) space of u, so that this matrix probably is better invertible.

1.3.2 Algorithmic implementations

The principal component analysis and regression can also be implemented using only Hebbian and anti-Hebbian imput and output neurons. However, for practical purposes, if such regression is to be applied for some technical purposes, the dynamic iteration process can be streamlined. One can directly employ the outcomes of the dynamic processes, implementing explicit matrix inversions, so that one has the following algorithm:

1. First update the covariance estimates ($0 \ll \lambda < 1$ being the forgetting factor)

$$\left\{ \begin{array}{rcl} R_{\bar{x}\bar{x}}(k) & = & \lambda R_{\bar{x}\bar{x}}(k-1) + (1-\lambda)\bar{x}(k-1)\bar{x}^T(k-1) \\ R_{\bar{x}u}(k) & = & \lambda R_{\bar{x}u}(k-1) + (1-\lambda)\bar{x}(k-1)u^T(k-1) \\ R_{y\bar{x}}(k) & = & \lambda R_{y\bar{x}}(k-1) + (1-\lambda)y(k-1)\bar{x}^T(k-1). \end{array} \right. \qquad (1.40)$$

2. The signal estimates can be found as

$$\left\{\begin{array}{lcl} \bar{x}(k) & = & R_{\bar{x}\bar{x}}^{-1}(k)R_{\bar{x}u}(k)u(k) \\ \bar{v}(k) & = & R_{\bar{x}\bar{x}}^{-1}(k)\bar{x}(k) \\ \hat{y}(k) & = & R_{y\bar{x}}(k)\bar{v}(k). \end{array}\right. \tag{1.41}$$

The covariances can be initialized, for example, as $R_{\bar{x}\bar{x}}(0) = \epsilon I_n$, where ϵ is a small constant, and similarly for the other matrices. Note that the above calculations are cross-referential, and the algorithm becomes highly iterative.

The inversion of the covariance matrix can be avoided when the update algorithm is written directly for $P_{\bar{x}\bar{x}} = R_{\bar{x}\bar{x}}^{-1}$ and the *matrix inversion lemma* (for example, see [39]) is applied:

$$\begin{array}{rcl} P_{\bar{x}\bar{x}}(k) & = & R_{\bar{x}\bar{x}}^{-1}(k) \\ & = & \dfrac{1}{\lambda}\Bigg(P_{\bar{x}\bar{x}}(k-1) - \\ & & \qquad \dfrac{P_{\bar{x}\bar{x}}(k-1)\bar{x}(k-1)\bar{x}^T(k-1)P_{\bar{x}\bar{x}}(k-1)}{\bar{x}^T(k-1)P_{\bar{x}\bar{x}}(k-1)\bar{x}(k-1) + \frac{\lambda}{1-\lambda}}\Bigg). \end{array} \tag{1.42}$$

Using this, the formulas are modified:

$$\left\{\begin{array}{lcl} \bar{x}(k) & = & P_{\bar{x}\bar{x}}(k)R_{\bar{x}u}(k)\,u(k) \\ \bar{v}(k) & = & P_{\bar{x}\bar{x}}(k)\,\bar{x}(k), \end{array}\right. \tag{1.43}$$

or, when written together,

$$\hat{y}(k) = R_{y\bar{x}}(k)P_{\bar{x}\bar{x}}^2(k)R_{\bar{x}u}(k)\,u(k). \tag{1.44}$$

The above straightforward approach only gives the principal subspace, the coordinate axes being linear combinations of the appropriate eigenvectors. For that reason, the additional layer with the vector v was needed. This could be circumvented if the actual principal components were employed rather than the principal subspace; this alternative is now elaborated on.

If one wants to explicitly solve for the eigenvectors, so that there would hold $\phi = \theta$, it suffices to make the state covariance matrix diagonal; in this case the latent variables are uncorrelated, and there must hold $\mathrm{E}\{\bar{x}\bar{x}^T\} = \Lambda$. To reach this, one can first note that if $n = 1$, there must hold $D = 1$, so that after convergence $\phi = \Theta_1$ necessarily represents the most significant eigenvector. This kind of individual adaptation can be carried out in the parallel environment if the incoming connections from other neurons are explicitly cut off, that is, if the corresponding row in the covariance matrix is zeroed (except the diagonal element). If the corresponding *column* is left intact, however, the correlations between this neuron and the other ones still have their effect, eliminating correlations; this means that the remaining neurons have to share the *remaining* orthogonal subspace. Continuing in the same manner, forgetting about the first neuron and its eigenvector, the second eigenvector can be extracted, etc. This means that if the covariance matrix is explicitly made *triangular,* the eigenvectors can be extracted one in each neuron. After convergence the $\bar{x}(k)$ vector represents the actual eigenvector scores $\chi(k)$, and, because of the uncorrelatedness, the covariance matrix is not only triangular but also diagonal.

Further, the regression structure can be simplified when the state variables are normalized to unit variance before output mapping: Assuming that the covariance matrix $R_{\bar{x}\bar{x}}$ (and also $P_{\bar{x}\bar{x}}$) is diagonal, the "whitened" data can be computed as

$$w(k) = \mathrm{E}_{\mathrm{diag}}\{\bar{x}\bar{x}\}^{-1/2}\,\bar{x}(k) = \begin{pmatrix} \sqrt{p_{11}} & & 0 \\ & \ddots & \\ 0 & & \sqrt{p_{nn}} \end{pmatrix} \bar{x}(k), \tag{1.45}$$

where p_{ii} denote the diagonal entries in $P_{\bar{x}\bar{x}}$. This inverse of the square root looks like a complicated construct; however, this normalization only scales the average signal levels back to unit variance.

For whitened data $w(k)$ where the covariance structure has been completely ripped off, covariance matrix being $\mathrm{E}\{ww^T\} = I_n$, the output mapping is simple:

$$\hat{y}(k) = \mathrm{E}\{yw^T\}\,\mathrm{E}\{ww^T\}^{-1}\,w(k) = \mathrm{E}\{yw^T\}\,w(k). \tag{1.46}$$

One only needs to match the whitened data against the output. The regression structure can be implemented as follows:

$$\left\{ \begin{array}{rcl} \bar{x}(k) & = & \mathrm{triang}\{P_{\bar{x}\bar{x}}\}(k)R_{\bar{x}u}(k)u(k) \\ w(k) & = & \mathrm{diag}\{P_{\bar{x}\bar{x}}\}^{1/2}(k)\bar{x}(k) \\ \hat{y}(k) & = & R_{yw}(k)w(k). \end{array} \right. \tag{1.47}$$

The operator "triang" zeroes the matrix elements above (or below) the diagonal; "diag" zeroes all but the diagonal elements. When the algorithm has converged, the rows of $P_{\bar{x}\bar{x}}R_{\bar{x}u}$ stand for the covariance matrix eigenvectors

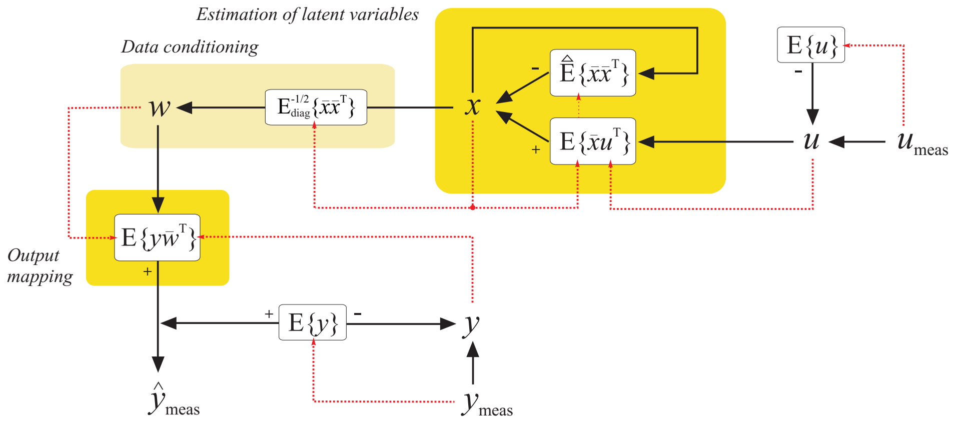

Estimation of latent variables
Data conditioning
Output mapping
w
$\mathrm{E}_{\mathrm{diag}}^{-1/2}\{\bar{x}\bar{x}^T\}$
x
$\hat{\mathrm{E}}\{\bar{x}\bar{x}^T\}$
$\mathrm{E}\{\bar{x}u^T\}$
$\mathrm{E}\{u\}$
u
u_{meas}
$\mathrm{E}\{y\bar{w}^T\}$
$\mathrm{E}\{y\}$
y
$\hat{y}_{\mathrm{meas}}$
y_{meas}
+
-

Figure 1.3: Streamlined implementation of PCR

Θ_i, and the elements on the diagonal of $R_{\bar{x}\bar{x}} = P_{\bar{x}\bar{x}}^{-1}$ represent the corresponding eigenvalues. The new regression structure is shown in Fig. 1.3. The internal connections between Hebbian units are nonsymmetric (denoted by the "triangle E" in the figure). Elimination of means is explicitly presented.

It is evident that in the beginning of the adaptation process the covariances are far from their final values. One could let the covariance adaptation process be time-variant to make the covariances better adaptible in the beginning of the learning; that is, the forgetting factor could be defined as

$$\lambda(k) = \lambda_\lambda \, \lambda(k) + (1 - \lambda_\lambda) \, \lambda_{\mathrm{final}}, \tag{1.48}$$

starting from some initial value $\lambda(0) \ll 1$ and tending towards $\lambda_{\mathrm{final}} \approx 1$, the "forgetting factor forgetting factor" λ_λ being adjusted appropriately to match the data properties.

1.3.3 Independent component analysis

Note that in vector z the data are whitened, that is, its covariance properties have been eliminated, the covariance matrix $\mathrm{E}\{zz^T\}$ being a unit matrix. This intuition makes it possible to extend the discussion towards *Independent Component Analysis (ICA)* (for example, see [28]). It turns out that if such whitened data are deformed, or "stretched" appropriately, the higher-order statistical properties of the original data can become reflected in its covariance structure — and this covariance structure can again be analyzed using the principal components of the modified data. This deformation of the data can be expressed as

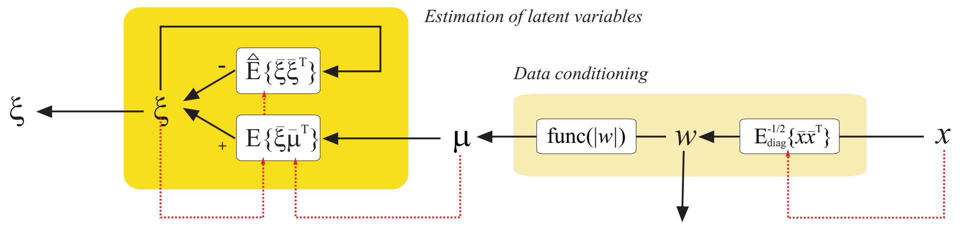


Figure 1.4: Extension towards Independent Component Analysis

$$\mu(k) = f\left(\|z(k)\|\right) z(k), \tag{1.49}$$

where $f(\cdot)$ is some function emphasizing the intended data properties. For example, defining $\mu(k) = \|z(k)\|\, z(k)$ makes it possible to capture the fourth-order properties of the data distribution or *kurtosis* (Fourth-Order Blind Identification method, or FOBI). This means that simple neuronal ICA can be implemented using a two-layer Hebbian/anti-Hebbian structure, having some additional nonlinearity between the layers (see Fig. 1.4).

However, it is well known that FOBI based approaches to ICA are not robust, and in practical applications iterative independent component extraction schemes are used instead. What is more, calculating $f(\|z\|)$ cannot be carried out in a distributed manner; all neuron outputs are needed to determine the value of this function, so that the intended locality of operations is lost, and the biological plausibility is further compromized. Rather than closer studying independent components here, *sparse components* will be studied in Chapter 2.

1.4 Experiments

It is well known that theoretical correctness and practical applicability can be very different things. Also in the case of Hebbian/anti-Hebbian algorithms, unanticipated behaviors emerge in simulations.

1.4.1 Comparisons

In this section typical behaviors of different kinds of neural PCA algorithms are presented[1]. The tested prototypical algorithms (as summarized in [14]) were Generalized Hebbian Algorithm (GHA), Adaptive Principal component

[1]The simulations in this section were carried out by Mr. Olli Haavisto

EXtraction (APEX) and its variable step length version (APEX2), and the proposed Hebbian/anti-Hebbian algorithm (HAH). There exist also novel and efficient algorithms in literature (for example, see [12] and [50]), but the above ones were selected as prototypical basic examples. Because the other methods are oriented exclusively to carry out PCA, the regression issues were not tested here. In the following, the notations are selected so as to match the above discussions.

- **GHA.** This algorithm explicitly extracts principal components one at a time, in the first neuron applying the *Oja's rule* (see [42]). In the latter neurons, the contributions of the earlier ones are first eliminated, and after that, the same principle is applied again. The assumed basic structure is

  $$\bar{x}(k) = Bu(k), \tag{1.50}$$

  where the matrix B is adapted elementwise as

  $$\begin{aligned} & B_{ij}(k+1) \\ & \quad = B_{ij}(k) + \beta \left(\bar{x}_i(k)u_j(k) - \bar{x}_i(k) \sum_{\iota \le i} \bar{x}_\iota(k) B_{\iota j}(k) \right). \end{aligned} \tag{1.51}$$

- **APEX.** Again, Oja's rule is utilized; now, however, the system structure is more complex, $\bar{x}$ being determined through a familiar-looking iterative expression

  $$\bar{x}(k) = Bu(k) - A\bar{x}(k), \tag{1.52}$$

  where the matrices are adapted as

  $$B_{ij}(k+1) = B_{ij}(k) + \beta \left(\bar{x}_i(k)u_j(k) - \bar{x}_i^2(k) B_{ij}(k) \right) \tag{1.53}$$

  and

  $$A_{ij}(k+1) = A_{ij}(k) + \beta \left(\bar{x}_i(k)\bar{x}_j(k) - \bar{x}_i^2(k) A_{ij}(k) \right). \tag{1.54}$$

- **APEX2.** Otherwise being identical with APEX, in this derivation the convergence rate is enhanced by having a time-variant adaptation factor in each neuron i (γ being some adjustable constant):

  $$\beta_i(k) = \frac{\beta_i(k-1)}{\gamma + \bar{x}_i^2(k)\beta_i(k-1)}. \tag{1.55}$$

- **HAH.** The presented Hebbian/anti-Hebbian algorithm is summarized here:

$$\begin{array}{rcl} \bar{x}(k) & = & \mathrm{triang}\{P_{\bar{x}\bar{x}}\}(k) R_{\bar{x}u}(k) u(k) \\ B(k) & = & P_{\bar{x}\bar{x}} R_{\bar{x}u}(k), \end{array} \tag{1.56}$$

where

$$\begin{array}{rcl} R_{\bar{x}u}(k) & = & \lambda R_{\bar{x}u}(k-1) + (1-\lambda)\bar{x}(k-1)u^T(k-1) \\ P_{\bar{x}\bar{x}}(k) & = & \frac{1}{\lambda}\Bigg(P_{\bar{x}\bar{x}}(k-1) - \\ & & \qquad \frac{P_{\bar{x}\bar{x}}(k-1)\bar{x}(k-1)\bar{x}^T(k-1)P_{\bar{x}\bar{x}}(k-1)}{\bar{x}^T(k-1)P_{\bar{x}\bar{x}}(k-1)\bar{x}(k-1) + \frac{\lambda}{1-\lambda}} \Bigg). \end{array}$$

When the algorithms have converged, the rows in B stand for the most significant principal components, so that $\hat{\theta} = B^T$.

Simulations (see Figs. 1.5 to 1.12) were carried out using data sets having different dimensions and different degrees of freedom (that is, some of the eigenvalues in the data covariance matrix were zeros), and the noise level was also varied. In each case there were $K = 500$ data samples that were processed by the algorithms various times, the ordering being each time randomized.

The convergence of the algorithms was tested using two different cost criteria. The first criterion reveals how well the estimated principal components span the principal subspace:

$$J_1(k) = \frac{1}{K}\sum_{\kappa=1}^{K} \|u(\kappa) - \hat{\theta}(k)\hat{\theta}^\dagger(k)u(\kappa)\|^2. \tag{1.57}$$

Using the pseudoinverse $\hat{\theta}^\dagger$ gives minimum value for the criterion as soon as the appropriate subspace is spanned by the vectors in $\hat{\theta}$. However, the algorithms should converge so that $\hat{\theta}$ is an orthonormal matrix. That is why, the other criterion was defined as

$$J_2(k) = \frac{1}{K}\sum_{\kappa=1}^{K} \|u(\kappa) - \hat{\theta}(k)\hat{\theta}^T(k)u(\kappa)\|^2. \tag{1.58}$$

Both criteria have the same theoretical minimum, and it can be explicitly calculated by summing the variances in the excluded subspace directions:

$$J_{\min} = \sum_{i=n+1}^{m} \lambda_i. \tag{1.59}$$

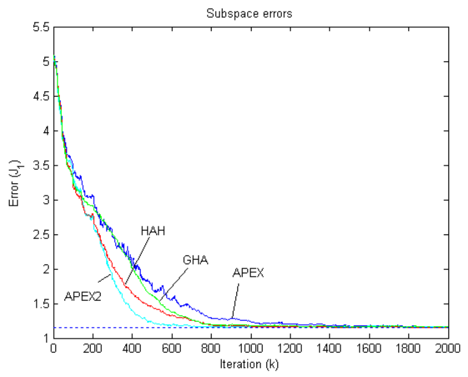


Figure 1.5: Experiment 1, criterion (1.57)

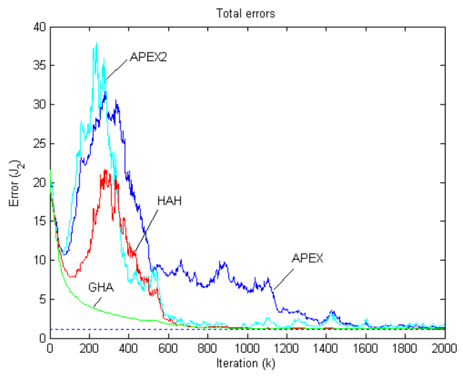


Figure 1.6: Experiment 1, criterion (1.58)

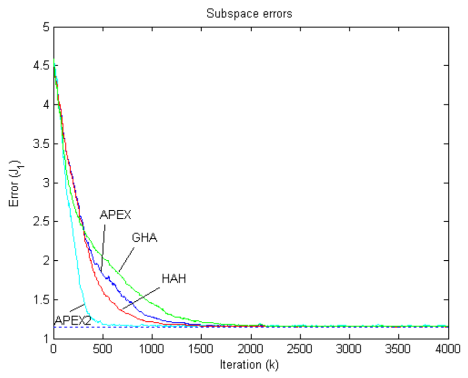


Figure 1.7: Experiment 2, criterion (1.57)

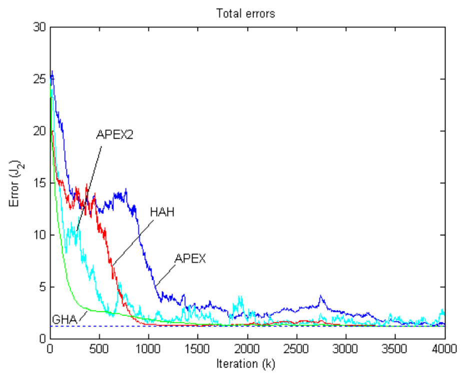


Figure 1.8: Experiment 2, criterion (1.58)

The algorithms were tested in four different cases:

	Data dimension	**Degrees of freedom**	**Number of neurons**	**Noise variance**	**Learning rate**
Exp. 1	$m = 8$	$d = 8$	$n = 4$	$\sigma^2 = 0.001$	$\beta = 0.005$
Exp. 2	$m = 8$	$d = 8$	$n = 4$	$\sigma^2 = 0.001$	$\beta = 0.003$
Exp. 3	$m = 5$	$d = 5$	$n = 4$	$\sigma^2 = 0.001$	$\beta = 0.001$
Exp. 4	$m = 20$	$d = 10$	$n = 5$	$\sigma^2 = 0.01$	$\beta = 0.005$

Because the HAH algorithm employs forgetting factor rather than learning rate, it was selected as $\lambda = 1 - \beta$. In APEX2, the step length varies according to a special adaptive strategy; in that case it is only the initial value $\beta(0)$ that is determined in the table.

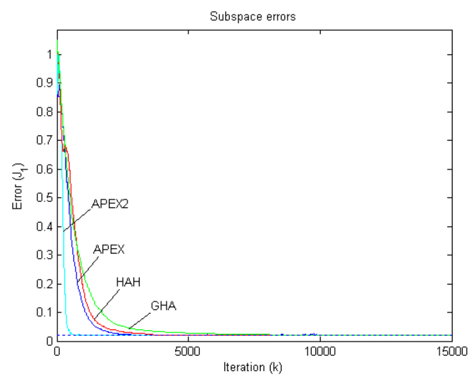


Figure 1.9: Experiment 3, criterion (1.57)

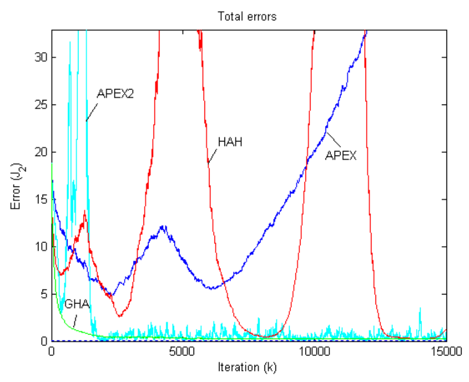


Figure 1.10: Experiment 3, criterion (1.58)

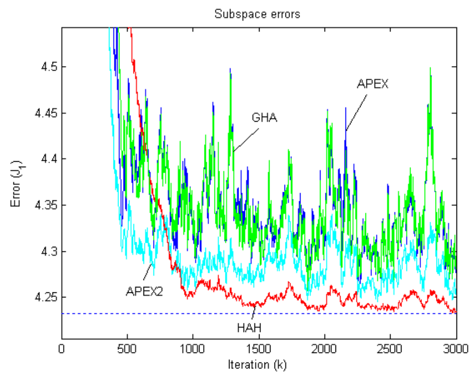


Figure 1.11: Experiment 4, criterion (1.57)

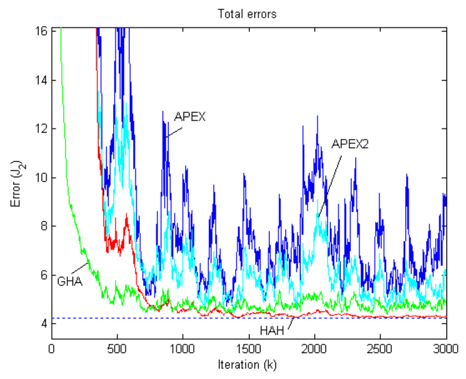


Figure 1.12: Experiment 4, criterion (1.58)

The simulations show that the proposed algorithm gives results that are well comparable with the established algorithms. Specially, when looking at the final error and the variation level in steady state, HAH algorithm seems to perform better than the other algorithms, at least in some cases. As there are fewer structural constraints in the new algorithm as compared to the older ones, the observed robustness is, indeed, a surprise. It seems that the HAH algorithm has problems if the theoretical minimum error is low, and if the least significant latent variables have low variation. In simulation 3 this method did not converge even though the step length was short; the subspace was spanned appropriately, but there were problems with finding the orthonormal basis. This problem is caused by the covariance matrix getting uninvertible; more intuition is gained in later in Chapter 2.

1.4.2 Application example: Distributed sensor network

The presented Hebbian/anti-Hebbian algorithm is not just another way to carry out linear principal component analysis. The key point here is that the algorithm is based on completely distributed operations with no central control. As it turns out, self-stabilization and self-organization of structures still pop up in the network even if the assumptions are slightly relaxed.

In today's systems, one would like to distribute the control of process components to reach better fault tolerance and independence of centralized hierarchies. The mainstream framework for distributed systems is the *agent* perspective [38]. Unfortunately, there is no solid theory available for such agent systems. Another approach to distributed networks has been studied, for example, in [3]; however, also these studies seem to remain on a rather descriptive level.

There exist many industrial processes where the systems have to be described in terms of partial differential equation models with distributed parameters. In principle, in such systems complete information about the process state cannot be measured, the state being infinite-dimensional. However, the new sensor technology still promises enhanced information about the process state: It is possible to place high numbers of sensors in the process, forming a *distributed sensor network.* The network of sensors could behave in an intelligent way, if only the principles of such distributed orchestration, or "ubiquitous computation", were known. It seems that research on distributed networks today very much concentrates on explicitly finding the global structure for the sensor system. If the structure is known, it is, of course, simple to implement applications in a traditional, globally controlled, centralized way. For example, having a global model available, one can implement a *Kalman filter* for measurement enhancement (see [2]). The problem here is that finding the global model is by no means a simple task.

Now, the sensors are thought to be like neurons, independent computing elements that can exchange information with each other. The above studied Hebbian/anti-Hebbian principles can directly be applied to implement a clever localized information exchange strategy.

As an application example, a heat diffusion process is studied; such a system is a typical example of partial differential equation models. Assume that a rod is being heated from the other end; the temperature is measured in three locations along the rod (considerable amount of uncorrelated measurement noise being disturbing the measurements). There are three sensors being located in a chain along the rod, sensor number 1 being nearest and sensor

3 being farthest from the heat source. The measurement vector consists of the current and three previous time step measurements, that is,

$$\nu_i(k) = \begin{pmatrix} T_i(k) \\ T_i(k-1) \\ T_i(k-2) \end{pmatrix}, \tag{1.60}$$

so that the actual input vector of measurements, when collected into a single vector, is

$$u(k) = \begin{pmatrix} \nu_1(k) \\ \hline \vdots \\ \hline \nu_n(k) \end{pmatrix}. \tag{1.61}$$

The measurement enhancement scheme here is based on *principal component filtering,* that is, the measurement data u is first projected onto the principal subspace, hoping that noise gets filtered, and after that the filtered measurement estimate $\hat{u}$ is reconstructed applying the regression scheme. Of course, only the newest temperature readings $\hat{T}_i(k)$ are of interest. Because there are three nodes, maximum of three latent variables can be constructed out from the nine input variables. Note that because the input is time series data, rather than implementing PCA, *subspace identification* is actually being carried out [44].

In Fig. 1.13, the original (discrete-time) measurements in node 1 are shown, together with the filtered estimates. Separate validation data was used that had the same statistical properties as the training data. First, it is clear that the local model that only has information about the local measurements in node 1, all connections between sensors being cut, becomes a finite-impulse response filter utilizing the local measurements only: The filtered result is a weighted average of the past three measurements. This means that the result resembles traditional "dummy" filtering, so that the smoothened responses are reached with the cost of delayed estimates. The global model, where all nodes are assumed to be connected to each other, and the chain model, where only the neighboring ones in the chain are connected, seem to be faster; indeed, it seems that no filtering takes place whatsoever. This is due to the structure of the measurements: Node 1 is the first one in the row, and no additional information can be received from the other nodes. Note that this structural fact has not been given *a priori;* it is the correlations among the signals only that dictate the emerging filter structures.

In Figs. 1.14 and 1.15, corresponding to the last and the middle sensors, respectively, the corresponding signals are shown. First, it can be noted

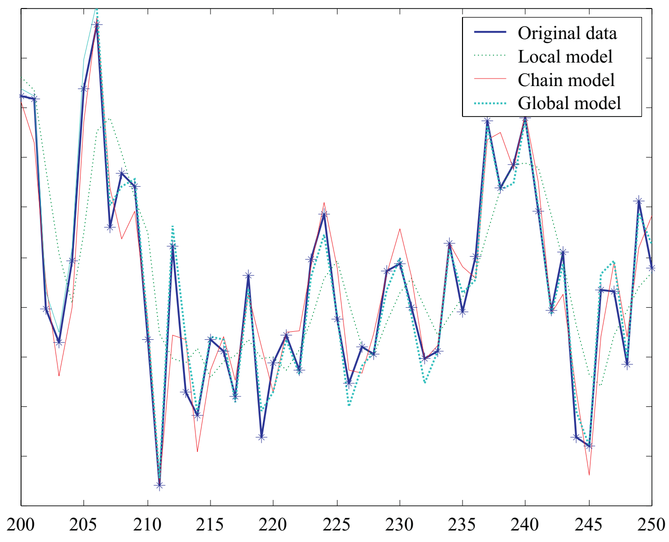


Figure 1.13: Original and filtered measurements in the first node

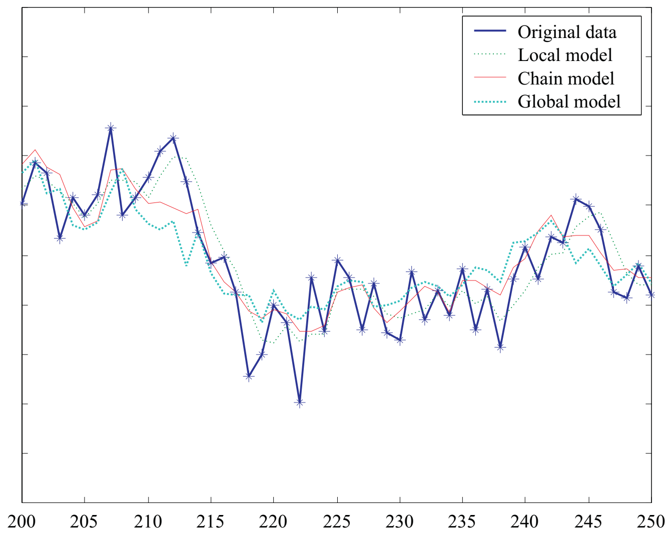


Figure 1.14: Original and filtered measurements in the last node

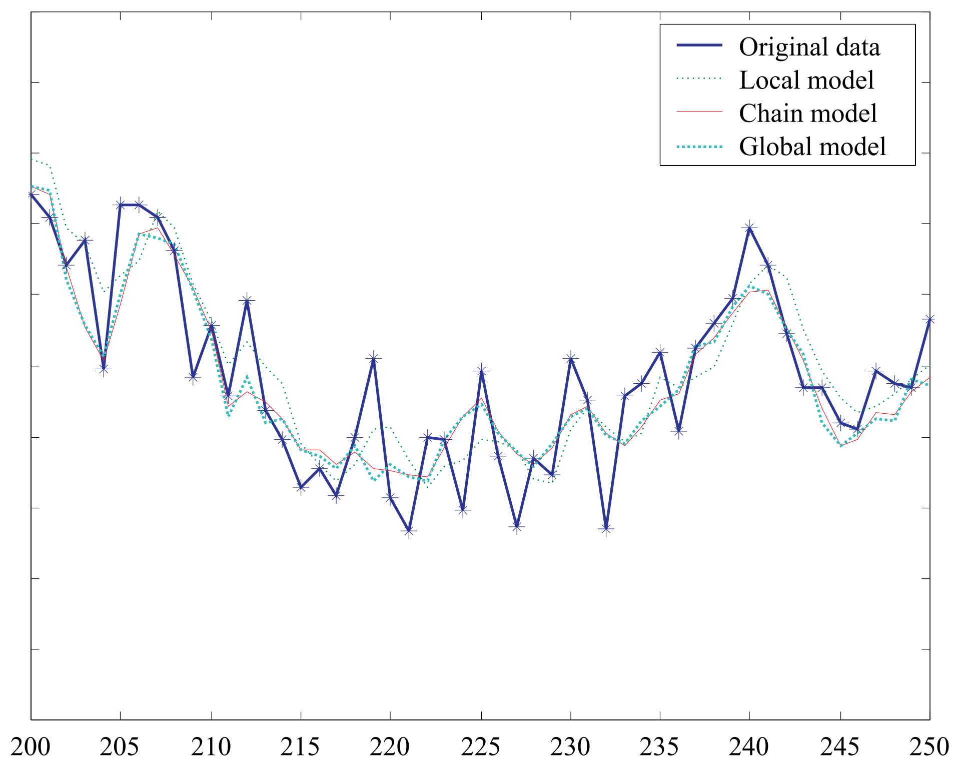


Figure 1.15: Original and filtered measurements in the middle node

that because of the low-pass nature of the process, the noise is automatically dampened, but it seems that the filtering schemes that are based on interactions among nodes can further attenuate the disturbances. Whereas the local filter always remains somewhat slow, the more “intelligent” schemes can anticipate the future behaviors. Even though node 3 is the last node in the row, receiving the most smoothened temperature values, it seems that the filtering results in node 2 are even better — indeed, it receives information from both directions.

And, looking at the middle sensor behavior in Fig. 1.15, what is most interesting is that it seems that the filtering scheme with incomplete information is *better* than that where all information is available (the “chain model” curve being slightly smoother than the “global model”). Indeed, the results can be more accurate when there is no excessive information available, but only the locally relevant information. This is an interesting result that cannot be foreseen when looking the network at the global level; it seems that the localized algorithms are *not* just implementations of global strategies, but the new approaches to distributed structures can deliver some real and practical added value to networked systems.

1.5 Conclusion

In this chapter it was shown that linear feedback structures are only needed to stabilize the neuronal dynamics; when the synaptic weights are adapted applying Hebbian and anti-Hebbian learning principles, the system caries out principal subspace analysis of the input data.

However, to implement more complicated models and interesting functionalities, it is clear that linear structures are not powerful enough. In Chapter 2, such issues are concentrated on, extending the linear structure.

Appendix 1A: About spaces and bases

A very brief summary of the basics of linear algebra is given here:

> The set of all possible real-valued vectors u of dimension m constitutes the *linear space* $\mathcal{R}^m$. If $\mathcal{S} \in \mathcal{R}^m$ is a set of vectors, a *subspace* spanned by $\mathcal{S}$, or $\mathcal{L}(\mathcal{S})$, is the set of all linear combinations of the vectors in $\mathcal{S}$. An (ordered) set of linearly independent vectors θ_i spanning a subspace is called a *basis* for that subspace.

The number of linearly independent vectors in the subspace basis determines the *dimension* of the subspace. The basis vectors θ_1 to θ_n of an n dimensional subspace can conveniently be represented in a matrix form:

$$\theta = \left(\; \theta_1 \; \middle| \; \cdots \; \middle| \; \theta_n \; \right). \tag{1.62}$$

Given a basis of a subspace, all points u_θ in that subspace have a unique representation. The basis vectors θ_i can be interpreted as coordinate axes in the subspace, and the weights of the basis vectors, denoted now z_i, determine the corresponding coordinate values (or *scores*) of the point:

$$u_\theta = \sum_{i=1}^{n} z_i \, \theta_i, \tag{1.63}$$

or in matrix form

$$u_\theta = \theta \, z. \tag{1.64}$$

If $m > n$, an arbitrary data point u cannot necessarily be represented in the new basis. Using the least squares technique an approximation can be found that minimizes the projection error:

$$\hat{z} = \left(\theta^T \theta \right)^{-1} \theta^T \, u. \tag{1.65}$$

If the basis vectors are *orthonormal* so that there holds $\theta^T \theta = I_n$, it is evident that (1.65) gives a simple solution:

$$\hat{z} = \theta^T \, u. \tag{1.66}$$

Appendix 1B: Principal components

Principal component analysis (PCA) is a mathematical procedure for determining a subspace that optimally captures the variation in the higher-dimensional data. The easiest way to determine the principal components is by analysis of the data covariance matrix. For understanding principal components, knowledge of *eigenvalues* and *eigenvectors* is necessary (for example, see [4]). The eigenvector Θ_i and the corresponding eigenvalue λ_i of the data covariance matrix fulfill

$$\mathrm{E}\{uu^T\}\,\Theta_i = \lambda_i \Theta_i. \tag{1.67}$$

There are m distinct eigenvectors and corresponding eigenvalues. If one collects the eigenvectors of $\mathrm{E}\{uu^T\}$ in the $m \times m$ dimensional matrix Θ, and the corresponding eigenvalues on the diagonal of the $m \times m$ dimensional Λ, so that

$$\Theta = \left(\begin{array}{c|c|c} \Theta_1 & \cdots & \Theta_m \end{array} \right) \quad \text{and} \quad \Lambda = \begin{pmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_m \end{pmatrix}, \tag{1.68}$$

the covariance matrix can be decomposed as

$$\mathrm{E}\{uu^T\} = \Theta\Lambda\Theta^{-1} = \Theta\Lambda\Theta^T. \tag{1.69}$$

This comes from the fact that the eigenvectors of the symmetric matrix are orthogonal. Because of the construction of the covariance matrix, the eigenvalues are real and non-negative; one can order the eigenvectors in the order of decreasing significance, so that $\lambda_1 \geq \cdots \geq \lambda_m \geq 0$. The eigenvalues λ_i reveal how much of the data variation takes place in the direction of the eigenvector Θ_i. This gives a practical way to compress data — select only the n most significant eigenvectors to constitute an orthogonal basis θ:

$$\theta = \left(\begin{array}{c|c|c} \Theta_1 & \cdots & \Theta_n \end{array} \right) \tag{1.70}$$

When the data in u is projected onto this subspace, so that $z = \theta^T u$, one receives the *latent variables* or *principal components* that are mutually uncorrelated and capture the variation in u in an optimal way.

Chapter 2

Implementing Sparse Coding

In this chapter it is shown how the Hebbian/anti-Hebbian learning principles can be extended to nonlinear neuron systems. The proposed algorithms can be interpreted as optimizing explicit optimality criteria: This interpretation offers new tools for analysis of the algorithm behavior. The optimality criterion is modified to implement sparse component analysis, and extensions towards self-organization are presented. As application examples, analysis of handwritten digits is carried out, and modeling of textual documents is illustrated.

2.1 Introduction

Hebbian/anti-Hebbian learning principle is a powerful concept. The discussions in Chapter 1 started from system theoretic considerations, and it was claimed that *linearity* and *stability* are the key issues when trying to make new structures emerge, and when trying to understand these structures. When these objectives are combined, it is *negative feedback* that can be applied to stabilize the system.

First, these principles were applied to individual synapses, and it turned out that the connections between inputs and Hebbian neurons can be expressed in a matrix form as

$$B = \mu \mathrm{E}\{\bar{x} u^T\}. \tag{2.1}$$

Here $\mathrm{E}\{\bar{x} u^T\}$ is the correlation matrix; vector $\bar{x}(k)$ represents the steady state neural activation corresponding to the input vector $u(k)$, and μ is some parameter. Second, if the linearity and stability principles are applied to the whole neuron grid, it turned out that the anti-Hebbian connections can be characterized as

$$A = -\mu \mathrm{E}\{\bar{x}\bar{x}^T\}. \tag{2.2}$$

The overall model for the neural activity becomes

$$x(\kappa+1) = x(\kappa) + hAx(\kappa) + hBu, \tag{2.3}$$

where h is some step size factor. The steady state corresponding to an input $u(k)$ can be expressed as

$$\bar{x}(k) = A^{-1}B\, u(k) = \phi^T\, u(k). \tag{2.4}$$

Because $\bar{x}$ is dependent of the matrices A and B, and these matrices are dependent of $\bar{x}$, the overall system is deeply interconnected. When the system reaches stationary state, it turns out that the matrix ϕ that is defined through

$$\phi^T = \mathrm{E}\{\bar{x}\bar{x}^T\}^{-1}\mathrm{E}\{\bar{x}u^T\} \tag{2.5}$$

spans the *principal subspace* of the variations in the input data u. As it turned out, local neuronal adaptation results in global behaviors in the grid level.

However, one cannot go beyond the principal subspace just following the original linear Hebbian/anti-Hebbian intuitions, and the point of view has to be changed. But when the very stringent linearity constraint is relaxed, some new guiding principles are needed. In this context, further system theoretic intuitions are applied: The goal is to find a *higher-level* view of the neuronal processes. In concrete terms, this means that *cost criteria* are searched for.

As it turns out, the expressional power can be considerably enhanced when the system structure is relaxed, so that new technical tools for practical applications are found. It seems that the linear intuitions are still applicable, and what is most important, self-stabilization and self-organization properties remain also after the modifications.

2.2 Nonlinear Hebbian/anti-Hebbian learning

In Chapter 1, linearity made it possible to achieve analytically tractable results. However, linear networks truly *are* too restricted to carry out really interesting tasks. Now, this linearity is abandoned — but the goal is to change the system structure as little as possible.

2.2.1 “Transparent nonlinearities”

Rather than obeying (2.3), assume that the neuronal system can be described as

$$z(\kappa+1) = z(\kappa) + hX(z(\kappa))\, A_f\, x(\kappa) + hX(z(\kappa))\, B_f\, u, \tag{2.6}$$

with

$$x(\kappa) = f(z(\kappa)). \tag{2.7}$$

Comparing this to the original model formulation, it turns out that there are some extensions: First, there is the nonlinearity f that is assumed to be monotonous and invertible; second, there is the invertible matrix $X(z)$ whose role will be studied later. Assuming stability, in steady state there holds

$$\bar{z} = \bar{z} + hX(\bar{z})\, A_f\, f(\bar{z}) + hX(\bar{z})\, B_f\, u, \tag{2.8}$$

and one can solve

$$f(\bar{z}) = -\left(A_f\right)^{-1} B_f\, u. \tag{2.9}$$

If x is interpreted as the output of the neuron grid, there holds

$$x = f(\bar{z}) = \phi_f^T u. \tag{2.10}$$

Expression (2.10) shows that the output of the neuron grid is a linear function of u, despite the nonlinearity. This strange-looking result can be explained in terms of control theory: It is a well-known fact that negative feedback “smoothens” nonlinearities. When looking at the steady state only, the nonlinearity becomes completely transparent. Now, if one selects

$$A_f = -\mu \mathrm{E}\{\bar{x}\bar{x}^T\} = -\mu \mathrm{E}\{f(\bar{z}) f^T(\bar{z})\} \tag{2.11}$$

and

$$B_f = \mu \mathrm{E}\{\bar{x}u^T\} = \mu \mathrm{E}\{f(\bar{z}) u^T\}, \tag{2.12}$$

it turns out that exactly the same derivations as in Chapter 1 can be carried out, meaning that

$$\phi_f^T = \mathrm{E}\{\bar{x}\bar{x}^T\}^{-1} \mathrm{E}\{\bar{x}u^T\} = \mathrm{E}\{f(\bar{z}) f^T(\bar{z})\}^{-1} \mathrm{E}\{f(\bar{z}) u^T\} \tag{2.13}$$

again spans the subspace of the most significant principal components found in data u. This means that the nonlinear Hebbian/anti-Hebbian structure still implements principal subspace analysis — or, at least it *tries* to do that: In a nonlinear system, the process towards steady state makes the difference (see Fig. 2.1).

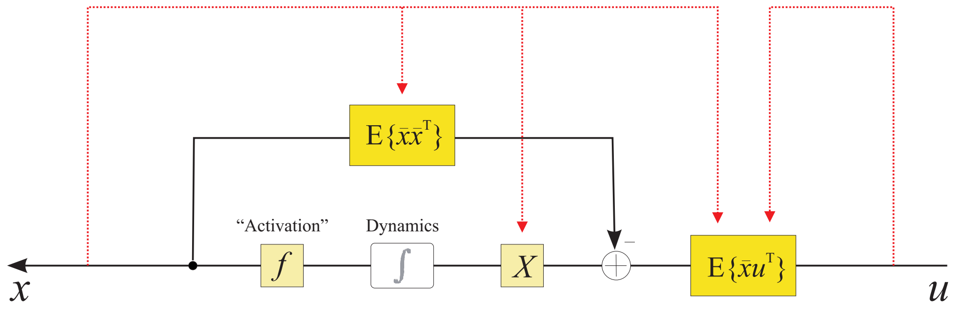


Figure 2.1: Outlook of a nonlinear Hebbian/anti-Hebbian neuron grid

2.2.2 Towards sparse coding

It is well known that linear PCA is well-motivated from the mathematical point of view, but not so well from the physical point of view. Natural data is often non-Gaussian, being composed of *independent* or *sparse* components (see [18], [43]). The cognitive machinery also seems to decompose sensory data into sparse components, that is, some neural units being active and some being inactive. This idea has been implemented for sensor signal processing, for example, in [32].

In this context, sparsity is interpreted so that some of the latent variables should be non-zero whereas other ones are strictly zero.

To implement sparse coding, some kind of nonlinearity seems to be necessary: The sparsity objective cannot be expressed in the linear/quadratic framework. There are infinitely many possibilities when selecting the nonlinear function form. System theory reveals that one should be extremely cautious, because, from the point of view of analysis and well-founded theory, nonlinearities open the *Pandora's box.* Extreme simplicity should be favored, but physical plausibility should also be taken into account. It turns out that a nice compromise between simplicity and expressional power is the *cut function* $f_{\text{cut}} : \mathcal{R}^n \to \mathcal{R}^n_+$ that can be defined elementwise as (see Fig. 2.2)

$$f_{\text{cut},i}(z) = \begin{cases} z_i, & \text{if } z_i \geq 0, \text{ and} \\ 0, & \text{otherwise.} \end{cases} \tag{2.14}$$

For technical reasons, the monotonous function formulation can be used:

$$f_{\text{cut},i}(z) = \begin{cases} z_i, & \text{if } z_i \geq 0, \text{ and} \\ \epsilon z_i, & \text{otherwise.} \end{cases} \tag{2.15}$$

In principle, vector elements below zero are cut to zero; the role of the small positive constant ϵ is to keep the function monotonous and (formally) invertible. This function form has the following advantages:

- **Physical plausibility.** No matter if the neuronal activity is based fundamentally on pulse frequencies, or chemical concentrations in synapses, the only structural constraint is that such signals *cannot be negative.*
- **Theoretical applicability.** The model is piecewise linear; sparsity is facilitated when negative variables automatically are inactivated, and do not affect the locally linear dynamics.
- **Pragmatic benefits.** There are no adjustable parameters; in larger systems, this considerably simplifies the tuning of the algorithm behavior.

As compared to the sigmoids and other function forms typically applied as activation functions in artificial neural networks, the proposed cut function is simpler, as there are no limitations from above. One can assume that neuron activities remain so low that linear part of their operation regime suffices. There are also problems with the selected function form: It is unsummetric, all variables being non-negative; this means that the variables cannot be zero-mean, and the correlations among variables are also non-negative (other covariance anomalies are studied in the following section). Even though systems with cut nonlinearity seem simple, it turns out that the dynamics of such systems can be very complex [26].

When the cut nonlinearity is applied, there is a qualitative difference between the *internal* and *external* activity of a neuron, or z_i and x_i, respectively. The internal activity value can become negative, and (negative) activity can cumulate, whereas the external or visible activity $f_{\text{cut}}(z_i)$ that is used also for adaptation of the data structures always remains non-negative. The positivity of the variables means that the covariance matrix $\mathrm{E}\{f_{\text{cut}}(\bar{z})f_{\text{cut}}^T(\bar{z})\}$ never can become diagonal; that is why, explicit (iterative) matrix inversion is needed when regression is implemented.

The external activity can still be continuous, not only binary (compare to [17]), so that this way of implementing sparse coding is closer to the original PCA, making it possible to implement continuous mappings.

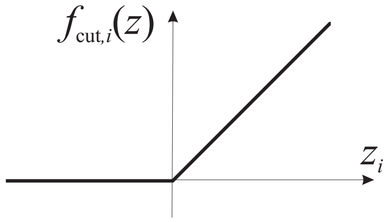


Figure 2.2: “Cut function”

In principle, the above scheme should work fine to achieve sparse coding. However, the emergence of sparse representations cannot explicitly be controlled: $f_{\mathrm{cut}}(\bar{z})$ seems to remain rather dense. If one wants to affect the process of sparsity emergence, some more analysis is needed.

2.2.3 Closer look at covariances

The introduced nonlinearity has its effects on the covariance matrices. For example, when only a subset of variables is used for regression, as is the case when implementing sparse regression, the covariance matrices,

$$A_f = -\mu \mathrm{E}\{f_{\mathrm{cut}}(\bar{z}) f_{\mathrm{cut}}^T(\bar{z})\} \tag{2.16}$$

and

$$B_f = \mu \mathrm{E}\{f_{\mathrm{cut}}(\bar{z}) u^T\}, \tag{2.17}$$

when calculated in the standard way, do *not* reflect the covariance structure among the active variables correctly: Always when a variable is employed, the corresponding diagonal entry in the covariance matrix is appropriately updated, whereas a non-diagonal entry is updated only when a *pair* of variables is *simultaneously* active. This means that for sparse data the diagonal is unjustly emphasized (of course, there is no problem if the variables are uncorrelated; but, when applying cut nonlinearity, they cannot be). In principle, one should record the individual covariance matrices for all sets of simultaneously active neurons; this is, however, unrealistic. It seems that there do not exist any compact, theoretically accurate solutions, and some practical shortcut is needed.

Rather than being defined as shown in (2.16), one can define the feedback matrix as

$$A' = -\mu \left(\delta V + \mathrm{E}\{f_{\mathrm{cut}}(\bar{x}) f_{\mathrm{cut}}^T(\bar{x})\} \right), \tag{2.18}$$

meaning that in addition to the standard covariance matrix, there is also another factor in the anti-Hebbian structure. Parameter δ is a scalar; to implement the above intuition about how the covariance matrix should be fixed, V can be defined, for example, as

$$V = \mathrm{E}\{f_{\mathrm{cut}}(\bar{x}) f_{\mathrm{cut}}^T(\bar{x})\} - \mathrm{diag}\left\{\mathrm{E}\{f_{\mathrm{cut}}(\bar{x}) f_{\mathrm{cut}}^T(\bar{x})\}\right\}, \tag{2.19}$$

that is, V is the covariance matrix with the diagonal zeroed. Adding this matrix to the covariance, multiplied by an appropriate δ, approximately compensates for the incompatibility of the neuron covariances.

As compared to sparse coding schemes based on *independent components,* for example, where sophisticated data distribution characteristics are utilizeed (see [28],[16]), now sparsity will be searched for applying straightforward correlation minimization techniques.

The proposed structure of the matrix V means that always when two variables are simultaneously active, there is some additional cost, whereas a single non-zero variable introduces no additional cost alone. This should lead to some variables dominating and others having low values. This objective is closely related to *varimax rotations* in factor analysis, and it should advance sparsity. However, some deeper theoretical analysis is needed; more intuition can be gained through an *optimality formulation.*

2.3 Optimality of Hebbian learning

Even though the above discussions gave rather unambiguous results, some issues were still left open. For example, how to select $X(z(\kappa))$ in (2.6) appropriately? As it turns out, higher-level views help to understand the neuronal processes.

2.3.1 Linear criterion

For a moment, forget about the above neuronal framework. Study an essentially quadratic, general optimality criterion to be minimized:

$$J(x) = \frac{1}{2} (u - \varphi x)^T W (u - \varphi x). \tag{2.20}$$

Matrix W is symmetric positive definite, compatible with the vector u, and φ is some $m \times n$ dimensional mapping matrix. This criterion has a unique minimum. To search for this minimum, the gradient can first be expressed as

$$\frac{dJ}{dx}(x) = \varphi^T W \varphi x - \varphi^T W u. \tag{2.21}$$

This can be solved for x in closed form:

$$\bar{x} = \left(\varphi^T W \varphi\right)^{-1} \varphi^T W u. \tag{2.22}$$

Alternatively, the steepest descent method for iteratively searching for the minimum can be formulated as

$$\begin{array}{rcl} x(\kappa+1) & = & x(\kappa) - \gamma \dfrac{dJ}{dx}(x(\kappa)) \\ & = & x(\kappa) - \gamma \varphi^T W \varphi\, x(\kappa) + \gamma \varphi^T W\, u. \end{array} \tag{2.23}$$

Comparing this to (2.3), there seems to exist some similarity. Indeed, to make the expressions truly identical, there should hold

$$\left\{ \begin{array}{rcl} \gamma \varphi^T W \varphi & = & -hA = \mu h \mathrm{E}\{\bar{x}\bar{x}^T\} = \mu h \phi^T \mathrm{E}\{uu^T\}\phi \\ \gamma \varphi^T W & = & hB = \mu h \mathrm{E}\{\bar{x}u^T\} = \mu h \phi^T \mathrm{E}\{uu^T\}. \end{array} \right. \tag{2.24}$$

These expressions are fulfilled if

$$\left\{ \begin{array}{rcl} \gamma & = & \mu h \\ \varphi & = & \phi \\ W & = & \mathrm{E}\{uu^T\}. \end{array} \right. \tag{2.25}$$

It also turns out that the Hebbian/anti-Hebbian learning can be formulated in the standard quadratic optimization framework, so that the algorithm (2.3) can be interpreted as the gradient descent algorithm. The algorithm tries to minimize the difference between u and φx; interpreting φ as containing some kind of (non-orthogonal) *features,* the algorithm tries to find the best combination of them, the matching errors being weighted by the matrix W.

This gives us yet higher-level view of what happens in the neuronal process: Rather than having to follow the iteration, one can directly concentrate on the pattern that would finally emerge out from the iteration. In (general) systems theory two views of looking at a system are distinguished: The *process view* and the *pattern view* (see [48]). In this perspective, the original approach of seeing the behavior of a system as an iteration, is an example of the process view. The opposite perspective, or trying to see beneath the complicated iterations, trying to perceive the emergent patterns, may open up new horizons.

For example, looking at the construction of the weighting matrix W in (2.25) it is easy to understand that directions having low variation are poorly reproduced: The errors in different directions are essentially weighted by the signal variances in those directions. The effects of this weighting were seen in the experiments in Chapter 1. On the other hand, the most significant directions in the data are excessively emphasized — this should make the algorithm efficient and robust if searching only for the most relevant principal components, and also the noise should be suppressed.

Note that the standard projection of data u onto the subspace basis φ can generally be defined by (2.20) with $W = I_m$. On the other hand, when matching data against a Gaussian distribution in the maximum likelihood sense, using the log-likelihood criterion, the covariance weighting is in an *inverse* way, so that $W = \mathrm{E}\{uu^T\}^{-1}$, making such mapping extremely sensitive to noise when the dimension of u is high. For yet other intuitions, see Appendices.

2.3.2 Nonlinear formulation

If the cost criterion is defined as

$$J'(z) = \frac{1}{2}\left(u - \varphi f(z)\right)^T W \left(u - \varphi f(z)\right), \tag{2.26}$$

its gradient will be

$$\frac{dJ'}{dz}(z) = \left(\frac{d}{dz}\left(f\left(z\right)\right)\right)^T \varphi^T W \varphi f\left(z\right) - \left(\frac{d}{dz}\left(f\left(z\right)\right)\right)^T \varphi^T W u, \tag{2.27}$$

and the gradient algorithm for finding the minimum would read

$$\begin{aligned} z(\kappa+1) &= z(\kappa) - \gamma\left(\frac{d}{dz}\left(f\left(z(\kappa)\right)\right)\right)^T \varphi^T W \varphi f\left(z(\kappa)\right) \\ &\quad + \gamma\left(\frac{d}{dz}\left(f\left(z(\kappa)\right)\right)\right)^T \varphi^T W u. \end{aligned} \tag{2.28}$$

For example, when the cut nonlinearity is selected, one has

$$\begin{aligned} z(\kappa+1) &= z(\kappa) - \gamma \left(\frac{d}{dz} \left(f_{\text{cut}}\left(z(\kappa) \right) \right) \right)^T \varphi^T W \varphi f_{\text{cut}}\left(z(\kappa) \right) \\ &\quad + \gamma \left(\frac{d}{dz} \left(f_{\text{cut}}\left(z(\kappa) \right) \right) \right)^T \varphi^T W u. \end{aligned} \tag{2.29}$$

Above, the Jacobian corresponding to the cut function is diagonal:

$$\frac{d}{dz} \left(f_{\text{cut}}(z) \right) = \begin{pmatrix} f_{\text{pos}}(z_1) & & 0 \\ & \ddots & \\ 0 & & f_{\text{pos}}(z_n) \end{pmatrix}, \tag{2.30}$$

where the elementwise derivatives are defined in terms of the unit step function (ignoring the discontinuity of the derivative in $z_i = 0$)

$$f_{\text{pos}}(z_i) = \begin{cases} 1, & \text{if } z_i > 0, \text{ and} \\ \epsilon, & \text{if } z_i < 0. \end{cases} \tag{2.31}$$

If the iteration in (2.29) converges, in steady state there will hold

$$\begin{aligned} \bar{z} = \bar{z} &- \gamma \left(\tfrac{d}{dz} \left(f_{\text{cut}}(\bar{z}) \right) \right)^T \varphi^T W \varphi f_{\text{cut}}(\bar{z}) \\ &+ \gamma \left(\tfrac{d}{dz} \left(f_{\text{cut}}(\bar{z}) \right) \right)^T \varphi^T W u, \end{aligned} \tag{2.32}$$

and, further, because of the monotonicity of the nonlinearity, the inverse of the Jacobian exists, and one can solve

$$f_{\text{cut}}(\bar{z}) = \left(\varphi^T W \varphi \right)^{-1} \varphi^T W u. \tag{2.33}$$

Note that because the structure is nonlinear, iteration cannot be avoided when searching for the steady state.

Proceeding as in the previous section, it is easy to show that expressions (2.29) and (2.6) are equal if one selects

$$\begin{cases} \gamma &= \mu h \\ \varphi &= \phi \\ W &= \mathrm{E}\{u u^T\} \\ X(z) &= \left(\tfrac{d}{dz} \left(f_{\text{cut}}(z) \right) \right)^T. \end{cases} \tag{2.34}$$

This way, it turns out that the other perspective gives insight in the neuronal process: If some of the neurons is inactive, $f_{\text{cut}}(z_i(\kappa)) = 0$, the internal (negative) activity does not cumulate. If this were not taken into account, variables could get stuck in zero level (however, see Sec. 2.3.4).

2.3.3 Refining the criterion

One can again reformulate the cost criterion (2.26). Extend the criterion as follows:

$$J''(x) = \frac{1}{2}\,\delta\, f_{\text{cut}}^T(z) V f_{\text{cut}}(z) + \frac{1}{2}\left(u - \varphi f_{\text{cut}}(z)\right)^T W \left(u - \varphi f_{\text{cut}}(z)\right). \quad (2.35)$$

The criterion is formulated in this way to emphasize the role of its components: The first term tries to affect the elements in $f_{\text{cut}}(z)$ directly, whereas the second term tries to keep the difference between u and $\varphi f_{\text{cut}}(z)$ low, in the same way as before. The gradient can be expressed as

$$\begin{aligned} \tfrac{dJ''}{dz}(z) = &\left(\tfrac{d}{dz}\left(f_{\text{cut}}(z)\right)\right)^T \left(\delta V + \varphi^T W \varphi\right) f_{\text{cut}}(z) \\ &- \left(\tfrac{d}{dz}\left(f_{\text{cut}}(z)\right)\right)^T \varphi^T W u, \end{aligned} \quad (2.36)$$

and the steepest descent algorithm becomes

$$\begin{aligned} z(\kappa+1) &= z(\kappa) - \gamma \frac{dJ''(z)}{dz} \\ &= z(\kappa) - \gamma \left(\frac{d}{dz}\left(f_{\text{cut}}(z(\kappa))\right)\right)^T \left(\delta V + \varphi^T W \varphi\right) f_{\text{cut}}(z(\kappa)) \\ &\qquad + \gamma \left(\frac{d}{dz}\left(f_{\text{cut}}(z(\kappa))\right)\right)^T \varphi^T W u. \end{aligned}$$

Following the above guidelines, this can be written as

$$\begin{aligned} z(\kappa+1) = &\; z(\kappa) \\ &- \gamma \left(\frac{d}{dz}\left(f_{\text{cut}}(z(\kappa))\right)\right)^T \left(\delta V + \mathrm{E}\{f_{\text{cut}}(\bar{z}) f_{\text{cut}}^T(\bar{z})\}\right) f_{\text{cut}}(z(\kappa)) \\ &+ \gamma \left(\frac{d}{dz}\left(f_{\text{cut}}(z(\kappa))\right)\right)^T \mathrm{E}\{f_{\text{cut}}(\bar{z}) u^T\} u, \end{aligned} \quad (2.37)$$

so that

$$f_{\text{cut}}\left(\bar{z}(k)\right) = \left(\delta V + \mathrm{E}\{f_{\text{cut}}(\bar{z}) f_{\text{cut}}^T(\bar{z})\}\right)^{-1} \mathrm{E}\{f_{\text{cut}}(\bar{z}) u^T\} u. \quad (2.38)$$

It turns out that, starting from the modified optimality criterion (2.35), the gradient algorithm corresponds to the Hebbian/anti-Hebbian structure with the modified covariance matrix A' in (2.18). It seems that also this heuristic modification of the original algorithm can be interpreted in terms of optimality.

Looking at the criterion (2.35), some conclusions can be drawn. If V were selected in a typical way, letting it be positive definite, *regularization* of the algorithm would result, that is, the latent variable values would be kept small. However, when selected as shown in (2.19), the matrix V is *not* positive definite. If δ were large enough, the (linear) criterion would not be bounded, and the algorithm tending towards the minimum would finally become unstable. However, the selected nonlinearity nicely stabilizes the system; note that the only route to instability is by letting some of the variables be negative — only in that case the cost could go down, the other variables growing without limit. When no negative variables can exist, growing variables always mean increasing cost. Higher weights to the additional criterion, or letting δ be high, means that competition among the neurons becomes more and more heated; some of the variables tend towards $-\infty$, ending in 0. This means that such weighting finally leads to sparse coding of data.

How to balance this competition among neurons? For improper values of δ, the coding easily becomes either too dense (δ being too small) or too local (δ being too large). A straightforward approach is to let the value of δ adapt according to the observed behavior of the neuron grid: If the average coding is too dense, the value is gradually increased during the adaptation process, and if the average coding is too sparse, it is decreased, until the intended degree of average sparsity is reached. It seems that robust behavior is reached if the sparsity level is selected so that (on average) about half of the variables are above zero.

The original simple algorithm in Chapter 1 that was based on linear Hebbian/anti-Hebbian principles has been essentially changed. The motivation for the extensions is given by the added functionality. It still seems that self-stabilization and self-organization properties are inherited in the new algorithm, thus offering a good framework for practical applications.

2.3.4 Additional extensions

Looking at formulas (2.37) and (2.38), it is evident that the fixed state of the iteration still has the same properties if one extends the algorithm as follows:

$$\begin{aligned}
z(\kappa+1) &= z(\kappa) \\
&\quad - \gamma N \left(\frac{d}{dz} \left(f_{\text{cut}} \left(z(\kappa) \right) \right) \right)^T M \left(\delta V + \mathrm{E}\{ f_{\text{cut}}(\bar{z}) f_{\text{cut}}^T(\bar{z}) \} \right) f_{\text{cut}}(z(\kappa)) \\
&\quad + \gamma N \left(\frac{d}{dz} \left(f_{\text{cut}} \left(z(\kappa) \right) \right) \right)^T M \mathrm{E}\{ f_{\text{cut}}(\bar{z}) u^T \} u.
\end{aligned}$$

If the matrices M and N are invertible, their effects will be cancelled, and it is still (2.38) that holds. However, because of the nonlinearities in the system, there can exist various minima, and the route towards the minimum can make a difference.

Let us define the conditioning matrices as follows: Let $M = I_n$, whereas N is defined elementwise as

$$N_{ij} = \mathrm{e}^{-\frac{1}{2} \left(\frac{d(i,j)}{\sigma} \right)^2}, \tag{2.39}$$

where $d(i,j)$ is determined by the physical distance between neurons i and j in the grid of neurons, and σ is the nominal width of the *neighborhood.* This additional structure can be used to model *spread of activation* among the neurons [1]. In the spirit of Kohonen's Self-Organizing Maps (SOM's), the neurons are thought to constitute a net where activation is diffused among the closely located neighbors (see [34]). It sounds plausible that there could exists some "crosstalk" among neighboring cells, so that some of one neuron's activity is transferred also to its immediate neighbors. The neighborhood can shrink during iterations, the value of σ decaying towards zero, so that the effects become more localized as adaptation proceeds.

The grid of neurons can have arbitrary topology. As an example of what the matrix N typically looks like, the following definition qualitatively describes

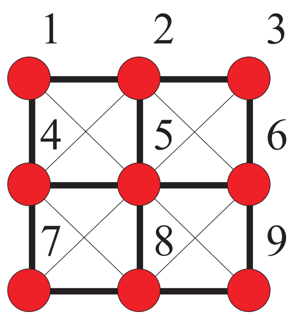


Figure 2.3: Grid of 9 neurons

the neighborhoods in the simple two-dimensional 9-neuron grid of Fig. 2.3:

$$N = \left(\begin{array}{ccc|ccc|ccc}
\bullet & \circ & & \circ & \cdot & & & & \\
\circ & \bullet & \circ & \cdot & \circ & \cdot & & & \\
& \circ & \bullet & & \cdot & \circ & & & \\
\hline
\circ & \cdot & & \bullet & \circ & & \circ & \cdot & \\
\cdot & \circ & \cdot & \circ & \bullet & \circ & \cdot & \circ & \cdot \\
& \cdot & \circ & & \circ & \bullet & & \cdot & \circ \\
\hline
& & & \circ & \cdot & & \bullet & \circ & \\
& & & \cdot & \circ & \cdot & \circ & \bullet & \circ \\
& & & & \cdot & \circ & & \circ & \bullet
\end{array}\right). \qquad (2.40)$$

Here the weights of the elements have been denoted by dots, heavier weights being larger and darker.

When the neighborhood effect is included in the adaptation algorithm, some kind of self-organization assumedly emerges. When self-organization is implemented as in Kohonen's original SOM, the operation is necessarily global, because the "winner" has to be selected among candidates. Now, on the other hand, the operation is completely local and decentralized. In this case there typically exist many winners at any time, so that one has sparse rather than SOM-style local coding, making it possible to have many (weighted) representatives for each input sample. This makes it possible to implement effective codings for input data. In addition to this, now the output is continuous-valued as compared to discrete-valued categorization in SOM, and regressions can be implemented with no added complexity.

Matrix N offers an efficient way to implement diffusion among neurons. Looking at (2.29), it is evident that if $z_i < 0$, adaptation of that variable freezes altogether. Diffusion from other neurons helps to avoid deadlocks.

The nonlinearity in the system means added structural complexity. However, there are also benefits: The emerging sparsity seems to automatically implement decomposition of variables. There is no need for explicit "triangularization" of the A matrix any more as in the linear case; the sparse structure emerges by itself. In the sparse coding case, it is also possible that $n > m$, that is, some kind of inflation of data space can take place; what is relevant is that only a few of all available neurons are active, and compression in this sense still takes place.

2.4 Applications

PCA can be interpreted also in the probabilistic framework [49]. For Gaussian data, the principal components represent the main axes of the distribution. A sparse model represents a *mixture model,* where only a subset of all available constructs are used at a time. This approach of modeling data using sparse principal components is also known as *multinomial PCA* [7].

Indeed, when applying the sparse coded Hebbian/anti-Hebbian algorithm, it is implicitly assumed that rather than constituting a single Gaussian data distribution, there can coexist various Gaussians. A traditional approach to implementing algorithms for extracting such locally computed PCA's is *expectation maximization,* where the data is first clustered, and only after that, the inner structures of the local distributions are modeled. This typically results in having mutually exclusive sets of features in the mutually exclusive clusters; now, on the other hand, the clusters can *share* common components (features) spanning their inner structures. The sets of active features at a time need not be orthogonal. Some examples are presented below.

2.4.1 Example: Features in handwritten digits

The presented algorithm was tested with handwritten digits [36]. The goal was to find features that are common to the digit patterns, and utilize this redundancy for PCA based compression, similarly as, for example, in [13]; in addition, the power of "sparse coded principal components" is illustrated.

The input data consisted of 8000 binary images with size of 32×32 pixels, so that $m = 1024$. No explicit *a priori* feature extraction was carried out (like analysis of stroke patterns); the pixelwise information was directly applied in the algorithm. The covariance matrices were initialized so that $R_{\bar{x}\bar{x}}(0) = 0.01 I_n$, and $R_{\bar{x}u}(0)$ had random entries with values between 0 and 0.0001. For each new input, the neuronal activities were initialized to $z_i(0) = 0.5$. In the simulations the parameters were selected so that $\lambda = 0.999$ and $\gamma = 0.1$. The iterations were started with $\delta = 0$, and this value was gradually increased until the desired level of average sparsity in $f_{\text{cut}}(\bar{z})$ was reached in steady state.

The grid of neurons was first 5×5, meaning that $n = 25$. The results in the neuronal grid are illustrated in Figs. 2.4 and 2.5 by plotting the contents of the columns in ϕ in the original two-dimensional form (note that because of the cut function, only *positive* features are modeled; if the *inverse* images $\mathbf{1} - u$ were simultaneously analyzed, so that $m = 2048$, also the *absence* of features could be utilized). In the former figure, the sparsity parameter was kept inactive, $\delta = 0$, and it turned out that the adaptation resulted in practically dense coding. Because there were no structural constraints, the actual principal components never emerged even though the principal subspace assumedly was found. In the latter figure, average of 5 neurons were simultaneously active in steady state. Data was not mean-centered; indeed, neuron 19 seems to stand for the mean value.

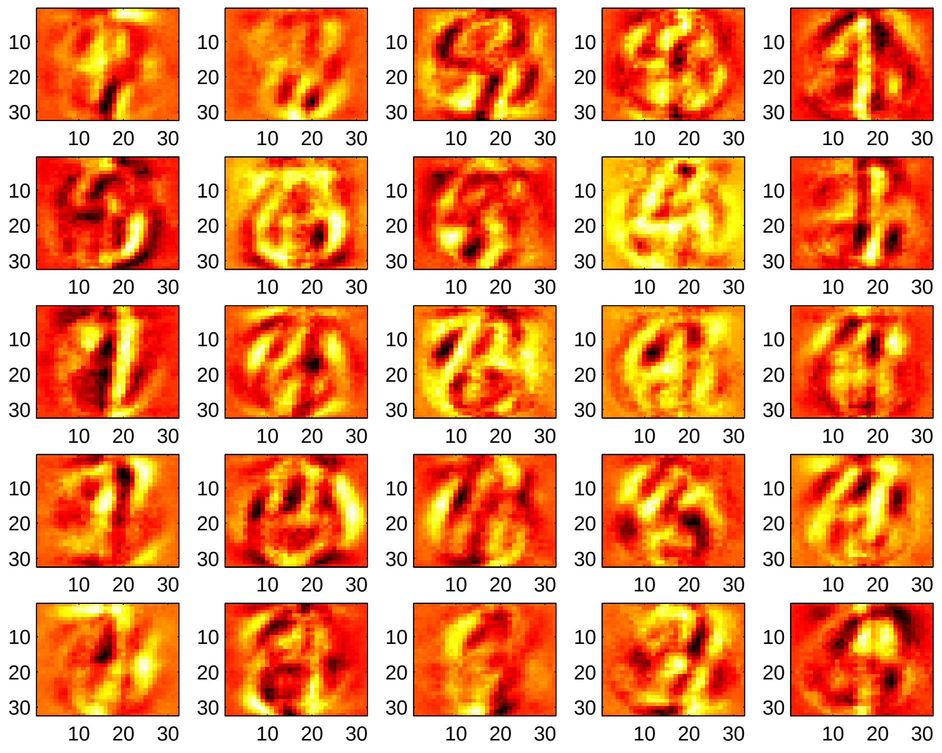


Figure 2.4: "Dense coding" of handwritten digits

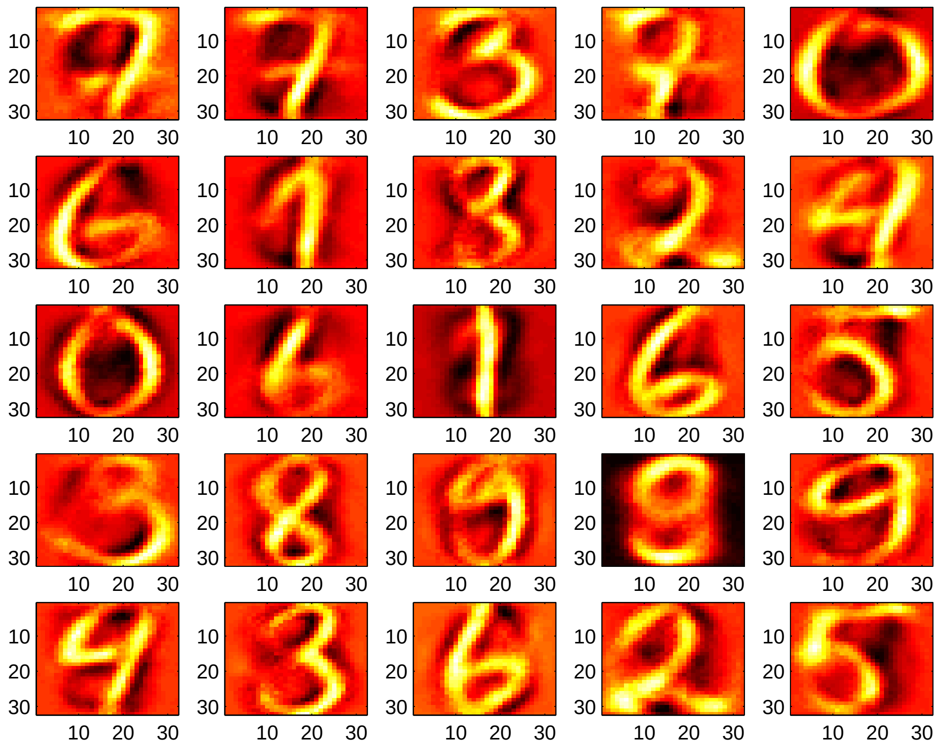


Figure 2.5: "Sparse coding" of handwritten digits

In the second experiment the grid was 7×7, meaning that $n = 49$, and the sparsity goal was 15. The results are shown in Figs. 2.6 and 2.7. In the larger neuron grid redundant and trivial features with no classification capability are manifested. The difference between the two figures is how the neuron contents is visualized: In Fig. 2.6, rows of $\left(\mathrm{E}\{f_{\mathrm{cut}}(\bar{z}) f_{\mathrm{cut}}^T(\bar{z})\}\right)^{-1} \mathrm{E}\{f_{\mathrm{cut}}(\bar{z}) u^T\}$ are shown, that is, the columns of φ. This visualization corresponds to a "dense" interpretation where the simultaneous contribution of all neurons is taken into account, that is, the sparsity and nonlinearity being forgotten. In Fig. 2.7, the rows of $\mathrm{E}\{f_{\mathrm{cut}}(\bar{z}) u^T\}$ alone are shown: This visualization corresponds to a "local" situation where only one neuron is studied at a time. It needs to be recognized that the actual sparse coding scheme cannot be compactly visualized in this way; actual reconstructions are somewhere between these two extremes. It is evident that because the contents in Figs. 2.6 and 2.7 are not essentially identical, there must hold $\mathrm{E}\{f_{\mathrm{cut}}(\bar{z}) f_{\mathrm{cut}}^T(\bar{z})\} \neq I_n$. This means that the neuronal activities are not uncorrelated. Comparing Figs. 2.6 and 2.7, it seems that characteristic features that make the neurons unique are better manifested when the neuronal interactions are taken into account.

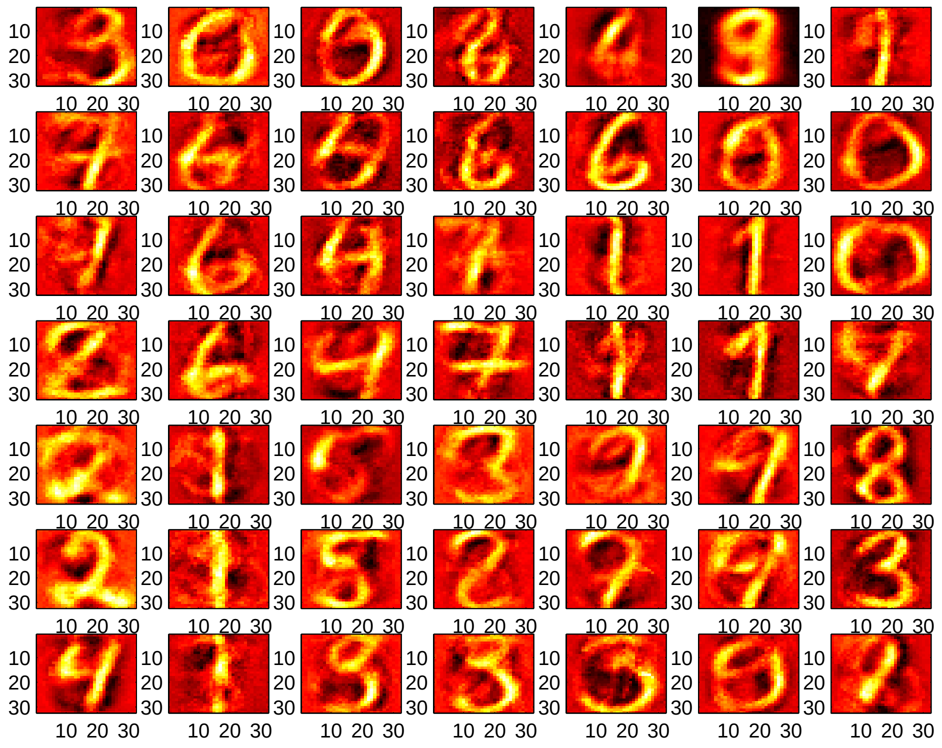


Figure 2.6: Visualization of 7×7 grid contents

In all the experiments the width of the neighborhood was $\sigma = 1$. In the smaller grids there is no visible self-organization among the neurons; there is too much variation in the data as compared to the grid capacity. It needs to be mentioned that Kohonen's SOM *cannot* efficiently be emulated by setting the sparsity level explicitly to 1: Whereas the Hebbian/anti-Hebbian structure nicely seems to balance the neuronal activities, the added nonlinearities and overemphasized sparsity objective with too much competition among the neurons may result in just few of the neurons remaining active at all.

Note that because $\mathrm{E}\{f_{\mathrm{cut}}(\bar{z})u^T\}u(k)$ remains constant during the iteration for $\bar{z}$, the high-dimensional matrix multiplications (this matrix having size $n \times 1024$) need not be repeated each time; the product can be calculated outside the iteration loop, once for each k.

2.4.2 Example: Structure in document collections

As the amount of available information has exploded, tools for Data Mining, and specially for Knowledge Mining, are becoming more and more impor-

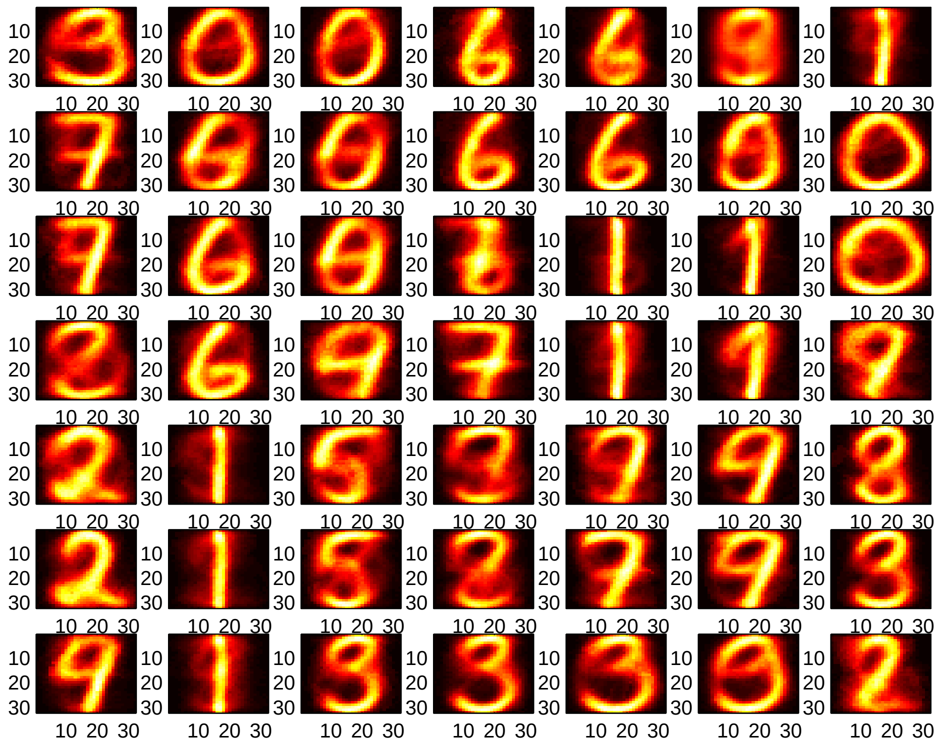


Figure 2.7: Another visualization, local effects only

tant (for example, see [40]). An automatic tool for finding a structure among a collection of textual documents, finding contextual relationships between texts could be invaluable for preliminary analyses of large bodies of knowledge. For example, assume that you are asked to get acquainted to some special field — first you go to a database and extract all documents with your keywords, but after that you let the algorithm look at the material and construct an overview. Instead of having hundreds of fragmentary documents, you would have some kind of table of contents giving "handles" into the documents. Other related applications could involve automatic modeling of news archives, and collaborative filtering. This kind of a service could be accessible through WWW, so that this tool could be seen as another route towards "Semantic Web".

There exist various applications of novel data distribution schemes for modeling of textual documents (for example, see [20], [41]). Data mining in textual knowledge bases can roughly be divided in two mainstream approaches: Either one tries to do "quantitative" analysis (for example, Latent Semantic Indexing LSI [19]), or one does "qualitative" analysis (different kinds of *cluster analysis* approaches). However, when studying the many-sided nature of complex data, it is evident that both of these two extremes alone are insufficient if trying to capture the essence of the data in an efficient way. What is needed is a framework for constructing integrated models with the capability of simultaneously representing the coarse and the fine structure buried in the information. Whereas the cluster centers are discrete and qualitative, the features modifying the cluster prototypes are continuous and quantitative. Again, it turns out that one should have a tool for finding *sparse coded linear latent structures* in the data.

To test the above modeling approach in a realistic environment, raw textual material from the `Inspec` database was used. First, all abstracts with the keyword "knowledge mining" were downloaded. The abstracts varied in length from 37 to 280 words, the overall number of words was about 2000, and the number of documents was $D = 162$ (this material was used to make the results comparable with [25]). The documents were presented using so called "fingerprint vectors" u: The fingerprints were vectors containing the word counts. The data dimension becomes high because each word has an entry of its own in the fingerprint vector, no matter whether that word is used in that specific document or not. What comes to the semantic contents, this kind of representation of the documents is, of course, extremely crude, but assuming that the terms in the document are more or less characteristic to the domain area, the interdependencies among the terms determine some kind of *contextual semantics* in the set of fingerprints. Linguistic analysis is now cut to minimum (for example, plural s's were eliminated). TFIDF weighting ("Term Frequency — Inverse Document Frequency") was applied to weigh the different words:

$$u_i \quad \leftarrow \quad u_i \cdot \log \frac{D}{D_i}.$$

Here, D is the overall number of available documents, and D_i is the number of such documents where the term number i is found. This weighting means that if a term is found in all documents, its weight will be zero; this can be motivated so that such unselective "stop-words" (like the words "the", "and", etc.) have no value when distinguishing documents from each other. Words that are found only once in the corpus material are also neglected. In the algorithm, all fingerprint vectors are normalized to unit length. This means that all training documents are assumed to have the same weight when used in model construction.

When the nonlinear Hebbian/anti-Hebbian algorithm version was run having $n = 9$ and $s = 4$, the process converged to a state where the sparse principal components were as shown in Fig. 2.8. In the figure, the 9 row

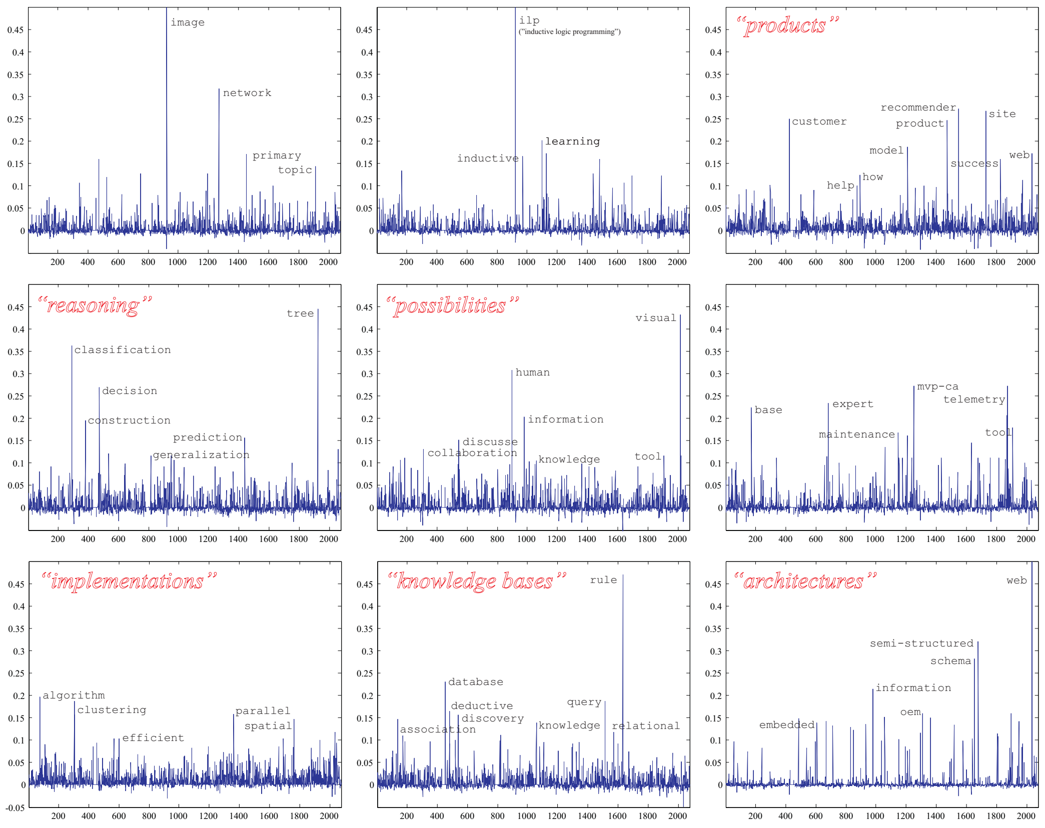


Figure 2.8: Document modeling with "generalized keywords"

vectors of $\left(\mathrm{E}\{f_{\mathrm{cut}}(\bar{z})f_{\mathrm{cut}}^T(\bar{z})\}\right)^{-1}\mathrm{E}\{f_{\mathrm{cut}}(\bar{z})u^T\}$ are shown as projected onto the word index axis; the elements in these vectors reveal how relevant the individual words are when explaining these "generalized keywords". Where appropriate, the generalized keywords have been labeled by human inspection. The documents can now be represented by a (sparse) weighted sum of such keywords. Note that some neurons seemingly have been allocated for representing sets of "outlier" documents for different reasons (for example, the document on "inductive logic programming" was stored various times in the database).

The results reveal that the nonlinear Hebbian/anti-Hebbian algorithm is a rather robust way to achieve sparse coding. It seems that the inherent self-stabilization and self-organization properties of the learning principles make it applicable in a wide variety of practical pattern recognition tasks. However, when the sparsity level is high, the covariance structure becomes too distorted; in such cases, it is better to explicitly extract the sparse components (see [25]).

2.5 Conclusions

In this chapter the linear Hebbian/anti-Hebbian neuron structure was extended to nonlinear neural systems. The motivation was to increase the expressional power of the model. It was recognized that there exist optimality criteria that can be used to interpret the algorithm behaviors.

How could the model be further extended? It seems that data-based approaches have their limits: Not all domains can be coded in data. The next challenge is to integrate declarative, etc., knowledge representations in the same structure. Can brain-like phenomena be implemented in a top-down rather than bottom-up manner, so that applying iteration some kind of relevant functionalities emerge? These issues are studied in Chapter 3.

Appendix 2A: Another optimality criterion

The optimality criteria are not unique; different criteria give different perspectives to the algorithm behavior. For example, study an optimality criterion of the form

$$J(\bar{x}(k)) = \frac{1}{2} \sum_{l=0}^{k-1} \left(u^T(l)u(k) - \bar{x}^T(l)\bar{x}(k) \right)^2 . \tag{2.41}$$

Given some $u(k)$ one tries to find $\bar{x}(k)$ so that the (unscaled) correlations with the prior vectors would match each other. The gradient becomes

$$\frac{dJ}{d\bar{x}}(\bar{x}(k)) = - \sum_{l=0}^{k-1} \bar{x}(l) \left(u^T(l)u(k) - \bar{x}^T(l)\bar{x}(k) \right) . \tag{2.42}$$

When one sets this to zero, one has

$$\sum_{l=0}^{k-1} \bar{x}(l)\bar{x}^T(l)\bar{x}(k) = \sum_{l=0}^{k-1} \bar{x}(l)u^T(l)u(k), \tag{2.43}$$

or

$$\frac{1}{k} \sum_{l=0}^{k-1} \bar{x}(l)\bar{x}^T(l)\, \bar{x}(k) = \frac{1}{k} \sum_{l=0}^{k-1} \bar{x}(l)u^T(l)\, u(k). \tag{2.44}$$

It turns out that if one defines the sample correlation matrices $R_{\bar{x}\bar{x}}(k) = \frac{1}{k} \sum_{l=0}^{k-1} \bar{x}(l)\bar{x}^T(l)$ and $R_{\bar{x}u}(k) = \frac{1}{k} \sum_{l=0}^{k-1} \bar{x}(l)u^T(l)$, one can write

$$\bar{x}(k) = \left(R_{\bar{x}\bar{x}}(k)\right)^{-1} R_{\bar{x}u}(k)\, u(k). \tag{2.45}$$

Comparing this to (2.4), it is evident that there exists yet another optimality criterion such that the operation of the neuron grid can be interpreted so that this criterion is minimized. Looking at (2.41), one can see that the past values $\bar{x}(l)$ essentially determine the new estimate. In this sense, the learning is "constructivistic": Optimum is not determined by the data alone but also by the history. Such an optimality criterion gives intuition of what the Hebbian/anti-Hebbian learning actually tries to accomplish. Specially, if the model is nonlinear and there exist local minima, dependency of the prior states in adaptation can considerably affect the results.

Appendix 2B: Another optimization approach

The optimality criteria can also be optimized in a variety of ways, not only applying the gradient algorithm. This is demonstrated below.

The second derivative of (2.41), or the Hessian, becomes

$$\frac{dJ^2}{d\bar{x}d\bar{x}^T}(\bar{x}(k)) = \sum_{l=0}^{k-1} \bar{x}(l)\bar{x}^T(l). \tag{2.46}$$

Now one can write the *Newton algorithm* for finding the zero point of the gradient:

$$\begin{aligned}
&\bar{x}(\kappa+1) \\
&\quad = \bar{x}(\kappa) - \left(\frac{dJ^2}{d\bar{x}d\bar{x}^T}(\bar{x}(\kappa))\right)^{-1} \frac{dJ}{d\bar{x}}(\bar{x}(\kappa)) \\
&\quad = \bar{x}(\kappa) - \left(\sum_{l=0}^{k-1} \bar{x}(l)\bar{x}^T(l)\right)^{-1} \left(-\sum_{l=0}^{k-1} \bar{x}(l)\left(u^T(l)\,u - \bar{x}^T(l)\bar{x}(\kappa)\right)\right) \\
&\quad = \left(\sum_{l=0}^{k-1} \bar{x}(l)\bar{x}^T(l)\right)^{-1} \sum_{l=0}^{k-1} \bar{x}(l)u^T(l)\,u \\
&\quad = \left(\frac{1}{k}\sum_{l=0}^{k-1} \bar{x}(l)\bar{x}^T(l)\right)^{-1} \left(\frac{1}{k}\sum_{l=0}^{k-1} \bar{x}(l)u^T(l)\right) u \\
&\quad = \left(R_{\bar{x}\bar{x}}(k)\right)^{-1} R_{\bar{x}u}(k)\,u,
\end{aligned}$$

meaning that the Newton iteration becomes a one-step process, regardless of the initial guess $\bar{x}(0)$ (this is expected because the model structure is linear). Again, the same formulation as in (2.4) is found; this means that rather than being a first-order algorithm, search for the best $\bar{x}$ is a very efficient process, being a one-step second-order process. However, the problem is that in the neuronal framework inversion of the matrix $R_{\bar{x}\bar{x}}$ has to be carried out iteratively.

Chapter 3

Synthesis of a Cognitive Model

It was shown in previous chapters that Hebbian/anti-Hebbian neuron structure offers a practical platform for sparse coding of complex data. In this chapter, it is shown how the proposed methodology makes it possible to integrate structural knowledge into the data-oriented framework as well, thus offering new intuitions into declarative and procedural as compared to associative information representations. A general cognitive model structure is suggested based on these experiences.

3.1 Introduction

It turned out that linear Hebbian/anti-Hebbian learning implements *principal subspace analysis,* and nonlinear Hebbian/anti-Hebbian learning can implement *sparse coding.* These results were achieved applying system theoretic intuitions and conceptual tools (see Chapters 1 and 2, respectively). The resulting algorihm was

$$\begin{aligned} z(\kappa+1) & \\ &= z(\kappa) - \gamma \left(\frac{d}{dz} \left(f_{\text{cut}} \left(z(\kappa) \right) \right) \right)^T \left(\delta V + \varphi^T W \varphi \right) f_{\text{cut}}(z(\kappa)) \\ &\qquad + \gamma \left(\frac{d}{dz} \left(f_{\text{cut}} \left(z(\kappa) \right) \right) \right)^T \varphi^T W u. \end{aligned} \tag{3.1}$$

Here, u is the input vector, z is the neuronal (internal) state, and $x = f_{\text{cut}}(z)$ is the system output, or external activity. Matrices V and W are for weighting, and parameters γ and δ for affecting the adaptation process. The matrix φ

contains the essence of what the neuron grid does to data, as presented in Chapter 2. The *cut nonlinearity* is defined elementwise as follows:

$$f_{\mathrm{cut},i}(z) = \begin{cases} z_i, & \text{if } z_i \geq 0, \text{ and} \\ 0, & \text{otherwise.} \end{cases} \tag{3.2}$$

The presented basic neural structure can be applied to many practically relevant pattern recognition tasks. However, there are problems when trying to extend the studies beyond the simple tasks:

- It has been recognized that many patterns *cannot* be detected from data alone. For example, *syntactic categories* cannot be justified in terms of observed statistical relationships only.
- It has been recognized that complex things *cannot* be coded as static patterns in the first place. Many representations are truly procedural or algorithmic, and many application domains contain causal structures by nature.

When trying to get forward from the data level, can system theoretic thinking help here; are there additional systemic intuitions available, or are they already exhausted?

It is claimed here that there *are* conceptual tools available. Indeed, these tools are not new ones; once again, the role of *feedback* is emphasized. It is an age-old paradox how all so different mental functionalities can be based on the same neural medium. The claim here is that the presented neural structure is a good platform for implementing recursive feedback structures; when simple things cumulate, when they are repeated over and over again, something qualitatively new can emerge that cannot be foreseen when looking at those single things alone.

As a starting point, in Chapter 1 the objective was modeling of real neuronal systems. In this sense, the test for validity of assumptions is how well the emergent "life-like" functionalities in a cognitive system can be explained in the assumed framework. As it turns out, many relevant functionalities can be implemented in a structure based on the proposed (nonlinear) Hebbian/anti-Hebbian neuron grids as connected in a (yet another) feedback structure.

3.2 Power of features

From the system theoretical point of view, perhaps the most annoying feature about the connectionistic approaches is their *conceptual unscalability:* Looking at the neuron level, one simply cannot understand how the higher-level cognitive functionalities could be explained. Still, these functionalities have to be based on the same kind of underlying building blocks. The question now is whether the framework that was proposed in Chapters 1 and 2 could perhaps be applied to understand the link between low-level and high-level phenomena.

3.2.1 Data vs. structural knowledge

If there exist some structural knowledge available, one would like to be able to directly implement this knowledge in the neuronal structures. Assume that knowledge is presented in the form of features φ_i, where $1 \leq i \leq n$. The goal is to find a representation for data vector u of dimension m so that it would be a weighted sum of the features, so that $u = \varphi^T \bar{x} = \varphi^T f_{\mathrm{cut}}(\bar{z})$, where φ contains the features as columns, and $\bar{x} = f_{\mathrm{cut}}(\bar{z})$ contains their (positive) weights. Because typically $n < m$, this problem cannot be exactly solved; a cost criterion can be written so that the match is defined in the error least squares sense: Best representation for u is given by $\bar{x} = f_{\mathrm{cut}}(\bar{z})$ that minimizes the criterion

$$J_{\mathrm{struct}}(z) = \frac{1}{2}\delta f_{\mathrm{cut}}^T(z) V f_{\mathrm{cut}}(z) + \frac{1}{2}\left(u - \varphi f_{\mathrm{cut}}(z)\right)^T \left(u - \varphi f_{\mathrm{cut}}(z)\right). \tag{3.3}$$

The features in φ need not be orthogonal. This is a pattern matching problem, and studying the derivations in Chapters 1 and 2, it is evident that the weight vector $\bar{x}$ can be directly solved by using the algorithm (3.1). If one wants to stick to the Hebbian/anti-Hebbian intuitions, one can define the system matrices explicitly as follows:

$$A = -\mu\left(\delta V + \varphi^T \varphi\right), \tag{3.4}$$

and

$$B = \mu\,\varphi^T. \tag{3.5}$$

Applying the iteration (3.1), having $W = I_m$, the input neurons match the data against (non-orthogonal) features, and the output neurons implement regression from the obtained latent basis to the output — even if these features were not optimal for representing the data, and even if the regression were not either optimal.

So how can the neuronal structure be "programmed" to carry out the matching process? Assume there exists a "free" neuron i (activity level being denoted z_i) available to be allocated for storing the feature number i. First, it must be so that the matrix B from input to that neuron is φ_i^T. If there is a positive weight in some location in φ_i, there will be an excitatory connection from that input to the neuron, and if there is negative weight, the connection is inhibitory. The rule can thus be readily hardcoded in the neuronal structures. On the other hand, the feedback matrix is essentially of the form $\varphi^T\varphi$: This is constructed by computing the correlations between each pair of rules and storing them in the matrix. Also, adding a new feature vector in the system is simple: If the set of features is not orthogonal, the new feature just has to be conditioned against the prior features by calculating its "overlap" with them.

Note that there is a *dualism* between data and structure: When structure is known rather than data, the training of the network is still carried out essentially in the same way, that is, by calculating correlations. Rather than calculating data covariances, one determines the "knowledge covariance" $\varphi^T\varphi$.

It is clear that associative models based purely on correlations cannot be used to model more complicated domains of knowledge — for example, *exclusive or* structures are the traditional counterexample. However, the assumed sparsity in the representations efficiently circumvents this problem: The neurons compete, and just a subset of them can be active at a time, each of such subsets possibly representing a separate case among candidates. This all means that whenever knowledge can be expressed in the assumed form (as a sparsely coded weighted sum of features), matching of data against such knowledge can be directly implemented.

From now on, the formulation (3.3) is also assumed to convey the essence of structural knowledge. However, this assumption sounds very restrictive. How could any cognitively relevant mental tasks, like reasoning, be represented in such an optimization based framework? As shown in [27], it turns out that many non-trivial cognitive functionalities *can* be expressed in this form. Truly, the framework is rather versatile, and to illustrate its potential, some examples are presented below.

3.2.2 Backward reasoning

As an example of high-level cognitive functionalities, study *logical reasoning,* or *backward inference.* This means that, given some state of affairs and the deduction rules, one tries to infer the simplest set of necessary preconditions explaining the current state. The claim here is that (a subset of) logic programming problems can be expressed and solved in the proposed feature-based framework. First, study proposition logic in general.

The modern logic programming formalisms like Prolog are based on the *Aristotelian syllogism,* meaning that a set of inference steps can be simplified:

$$\begin{array}{rl} & \mathtt{A} \rightarrow \mathtt{B} \\ \wedge & \mathtt{B} \rightarrow \mathtt{C} \\ \hline \Rightarrow & \mathtt{A} \rightarrow \mathtt{C}. \end{array}$$

Using the properties of the logic connectives, it can be shown that the logical content remains unchanged if the above reasoning is written in the following form:

$$\begin{array}{rl} & \neg\mathtt{A} \vee \mathtt{B} \\ \wedge & \neg\mathtt{B} \vee \mathtt{C} \\ \hline \Rightarrow & \neg\mathtt{A} \vee \mathtt{C}. \end{array}$$

The operator "$\vee$" denotes logical OR operation; "$\wedge$" is logical AND, and "$\neg$" means negation. Typically, the rules in a rule base are of a more complex form, so that, for example, $\mathtt{A}_1 \wedge \cdots \wedge \mathtt{A}_n \rightarrow \mathtt{B}$; however, using *de Morgan rules,* they can also be brought to the same disjunctive form: One has $\neg(\mathtt{A}_1 \wedge \cdots \wedge \mathtt{A}_n) \vee \mathtt{B}$, and, further, $\neg\mathtt{A}_1 \vee \cdots \vee \neg\mathtt{A}_n \vee \mathtt{B}$.

To implement this kind of reasoning in a mathematical form, one can first notice that the logical values of syllogisms can easily be operated on — the table below is consistent with the original deduction, showing only the logical truth values:

$$\begin{array}{crrr} & \mathtt{A:} & \mathtt{B:} & \mathtt{C:} \\ & -1 & 1 & \\ + & & -1 & 1 \\ \hline = & -1 & & 1 \end{array}$$

Here 1 stands for "true" and -1 for "false"; value 0 meaning "not used" or "irrelevant". This kind of coding seems to miss the structure and uniqueness of logical expressions. For example, study the following distribution of truth values:

$$\begin{array}{rrrr} \mathtt{A:} & \mathtt{B:} & \mathtt{C:} & \mathtt{D:} \\ \hline -1 & -1 & 1 & 1. \end{array}$$

In logical terms this can be interpreted as

$$\mathtt{A} \wedge \mathtt{B} \rightarrow \mathtt{C} \vee \mathtt{D},$$

but just as well, for example, as

$$\mathtt{A} \wedge \neg\mathtt{C} \rightarrow \neg\mathtt{B} \vee \mathtt{D}.$$

There always exist various alternative interpretations. However, studying such expressions closer, it turns out that they all are *logically equivalent:* Such a distribution of logical values among `A`, `B`, `C`, and `D` that makes one of the expressions hold, also fulfills the other ones. The only difference is caused by the intuitive feel of causality built in the expressions — but, indeed, the formulas can work in both directions, depending on which of the variables are known and which are to be determined.

It turns out that logic expressions and manipulations with them can be coded in linear algebra. However, there are some complications: In the numeric formulation, when rule vectors refer to variables, the truths are being "exhausted" as they are utilized — this is not the case when operating with true logical values. To circumvent this, assume that truth values can be numeric values other than the nominal -1, 0, or 1. The non-binary values simply have to be interpreted so that any positive value stands for "true", and negative values stand for "false".

Using the *resolution principle,* reasoning in the Prolog systems goes as follows (for example, see [8]). Assume that one wants to know whether some logical clause (expression) can be deduced, given a consistent knowledge base. One first inserts the *negated goal clause* among the other clauses, and starts applying elimination steps of the type shown above. If it turns out that the negated expression contradicts the other rules, that is, an "empty clause" can be deduced, the goal is reached: It must be so that the non-negated clause is deducible from the axioms in the assumed closed universe.

To implement this intuition in mathematical terms, some assumptions are needed. Let the vector φ_i now stand for the rule number i as represented in the above numeric form: References to the logical entities are represented by -1's or 1's, depending on whether they are negated or not. Further, let the vector u represent the observed state of the "world". It contains truth values for the state variables as expressed in the above mathematical framework; that is, u contains -1's, 0's, and 1's. The "internal image" corresponding to the observed world, or $\bar{x}$, contains the "activities" or "relevances" of the rules applying in that situation. This means that the effect of the combined rules, or the sum of the rule vectors, can be expressed as φx.

Following the resolution principle, the sum of the negated goal vector and the applied rule vectors should be a zero vector. After reasoning there should also hold $-u_{\text{goal}} + \varphi x = \mathbf{0}$, or, when written in another way, $\varphi x = u_{\text{goal}}$. This starts looking familiar. And, indeed, there are some additional points that help to match the current application with (3.3):

- **Continuity of the objective.** Rather than setting hard limits, it is beneficial to define a cost criterion that tries to keep the difference between u and φx small.

- **Logical structure of the clauses.** If a rule vector φ_i is multiplied by a *negative* number, the logical contents of the rule changes altogether, and its validity is lost. That is why, the rule activations in $\bar{x}$ must always remain non-negative.

- **Sparsity of the solutions.** If there are various alternative inference results, only one of them is to be selected, not some kind of linear combination of them.

Luckily, all these constraints can easily be fulfilled when implementing reasoning in the proposed cost criterion based framework. This means that searching for the minimum of (3.3) solves the reasoning problem (assuming that the solution exists).

When implementing *predicate calculus,* so that expressions can contain variables, the situation is much more complicated, and complete congruence between the formalisms cannot be reached: The universe of literals and their Skolem functions need to be enumerated and listed (see [27]). However, for unary predicates the situation is simple — predicates are operated on very much like the propositions are.

Next, a simple example is presented to clarify the above discussion. Assume that the knowledge base consists of the following expressions:

```
P(a)
P(b)
¬P(c)
P(x) → Q(x)
```

The rules need to be rewritten:

`a`	Literal `a`
`b`	Literal `b`
`c`	Literal `c`
$\mathtt{a} \rightarrow \mathrm{P}$	Expression 1
$\mathtt{b} \rightarrow \mathrm{P}$	Expression 2
$\mathtt{c} \rightarrow -\mathrm{P}$	Expression 3
$\mathtt{P} \rightarrow \mathrm{Q}$	Expression 4

Every literal and every expression needs the "activity variable" of its own. Collected together, one has the variable vector

$$x = \begin{pmatrix} x_{\mathtt{a}} & x_{\mathtt{b}} & x_{\mathtt{c}} & x_{\mathtt{a}\rightarrow\mathtt{P}} & x_{\mathtt{b}\rightarrow\mathtt{P}} & x_{\mathtt{a}\rightarrow\,-\mathtt{P}} & x_{\mathtt{P}\rightarrow\mathtt{Q}} \end{pmatrix}^T .$$

The rule base can be written in a table form as

	a	b	c	P	Q
Literal `a`	1				
Literal `b`		1			
Literal `c`			1		
a→P	−1			1	
b→P		−1		1	
c→ −P			−1	−1	
P→Q				−1	1

To write the matrices in (3.4) and (3.5) appropriately, one can recognize that the matrix φ can be constructed from the above table by simply transposing it:

$$\varphi = \begin{pmatrix} 1 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 1 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{3.6}$$

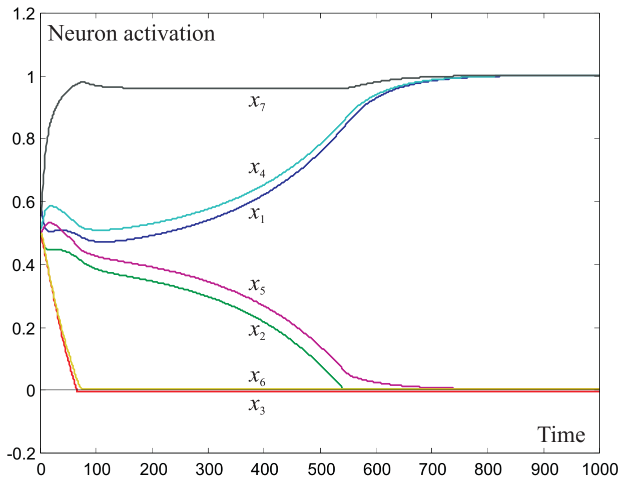


Figure 3.1: "When `Q`?"

The weighting matrix in the optimality criterion (3.3) can be selected as

$$V = \left(\begin{array}{c|c} \mathbf{1}_3 - I_3 & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} \end{array}\right) = \left(\begin{array}{ccc|cccc} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right). \tag{3.7}$$

Almost all entries in the matrix V are zeros, because no matter how many times the rules are applied, there is no cost for them. The upper left corner, on the other hand, means that if various literals are simultaneously active, it decreases the validity of the solution; if there is only one active literal, there is no cost. Loosely speaking, V implements "exclusive or" within the model structure. If the goal is achievable, zero cost can be reached.

Two simulations are shown in Figs. 3.1 and 3.2, having step size $\gamma = 0.1$ and sparsity factor $\delta = 1$. In Fig. 3.1, it is assumed that one wants to know "When would `Q` be true?", given the above rule base, and in Fig. 3.2, the query is "When would `Q` be false?". The inputs in these two cases, or the

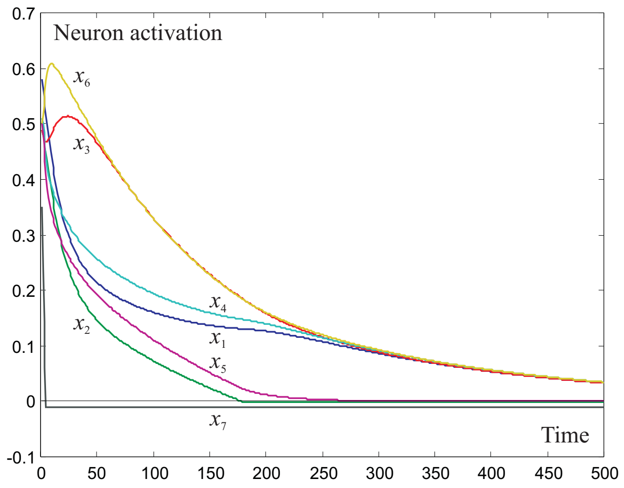


Figure 3.2: "When ¬`Q`?"

transposed goals are

$$u_{\mathrm{pos}} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \qquad \text{and} \qquad u_{\mathrm{neg}} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ -1 \end{pmatrix}. \tag{3.8}$$

In the beginning of the iteration, variables x_i start from 0.5, exception being $x_1(0) = 0.6$. This means that `a` is preferred to `b`, even though both have equal logical properties. It seems that variable #1 remains above zero in the steady state, meaning that "`a` makes `Q` true (through `P`)". Note that logically just as valid solution would be "`b`" — however, because of the initial bias (0.1) in `a`, this solution is preferred to the alternative. In technical terms, it turns out that if both variables were equally weighted, a saddle point in the cost landscape would be found, and, because of the zero derivative, long time could be wasted before either of the solutions would be (randomly) selected.

In the other case, on the other hand, one would like to know how to assure that `Q` will be false; using the given rules alone, such a conclusion can never be drawn, and all variables tend to zero. In the former case, the error $u - \varphi\bar{x}$ has zero length, meaning that the solution has been reached, whereas in the latter case, the error is non-zero.

If there exist various non-zero entries in u_{goal}, or if there should be negative entries (assuming that the input entries are also truncated; see later), it may be that the determined numerical combination of numeric values cannot be exactly spanned by the features even though the distribution of logical values would indeed be reachable. In such a case it is better to augment the rule matrix and define a new predicate that stands specially for the query:

$$\varphi' = \left(\begin{array}{ccc|c} \varphi_1 & \cdots & \varphi_n & -u_{\text{goal}} \\ \hline 0 & \cdots & 0 & 1 \end{array} \right), \qquad \text{and} \qquad u_{\text{new}} = \left(\begin{array}{c} 0 \\ \vdots \\ 0 \\ \hline 1 \end{array} \right). \tag{3.9}$$

Now it is easy to search for the solution when one selects the new goal vector u_{new} as shown above, so that this new rule alone is emphasized.

3.3 Modeling sequential processes

Above, it turned out that in some cases the intuitively causal processes of reasoning can be implemented as static pattern recognition tasks. However, in less structured cases, the step-at-a-time reasoning cannot be avoided.

3.3.1 Declarative knowledge

The backward reasoning above was formulated as a pattern matching. How about a truly *sequential* task, how could such reasoning be implemented in the presented framework? As an example, a *production system,* or *forward chaining* is studied. All reasoning rules are assumed to be either of the conjunctive form `IF` $\mathtt{P}_1$ `AND ...AND` $\mathtt{P}_i$ `THEN` $\mathtt{P}_{\mathtt{new}}$, or of the disjunctive form `IF` $\mathtt{P}_1$ `OR ...OR` $\mathtt{P}_i$ `THEN` $\mathtt{P}_{\mathtt{new}}$. It is now assumed that for each $\mathtt{P}_{\mathtt{new}}$ there exists (at most) one such rule. If there is need for more complex rules, where disjunctive and conjunctive structures are combined, intermediate dummy variables need to be introduced.

To implement such rules in a numerical form, let us now assume that value 1 means "true" and 0 means "false". Then the disjunction of two expressions, $\mathtt{P}_1$ `OR` $\mathtt{P}_2$, can be implemented as

$$u_{\mathrm{P}_1 \vee \mathrm{P}_2} = f_{\text{cut}}(u_{\mathrm{P}_1} + u_{\mathrm{P}_2}), \tag{3.10}$$

where $u_{\mathtt{P}}$ denotes the logical value of the proposition `P`. Correspondingly, the conjunction of expressions, $\mathtt{P}_1$ `AND` $\mathtt{P}_2$, can be implemented as

$$u_{\mathrm{P}_1 \wedge \mathrm{P}_2} = f_{\text{cut}}(u_{\mathrm{P}_1} + u_{\mathrm{P}_2} - 1). \tag{3.11}$$

A negation of an expression P, or ¬P, can be implemented as

$$u_{\neg\mathrm{P}} = 1 - u_{\mathrm{P}} = f_{\mathrm{cut}}(1 - u_{\mathrm{P}}). \tag{3.12}$$

This means that a single reasoning rule can be implemented as

$$u_{\mathrm{P}_{\mathrm{new}}} = f_{\mathrm{cut}}(\sum_j \pm u_{\mathrm{P}_j} + c), \tag{3.13}$$

where c is some constant. The rule number i determining the value of u_{P_i} can be expressed in a vector form as

$$u_{\mathrm{P}_i} = f_{\mathrm{cut}}\left(\varphi_i^T u + c_i\right) = f_{\mathrm{cut}}\left(\varphi_{i,\mathrm{aug}}^T u_{\mathrm{aug}},\right) \tag{3.14}$$

where φ contains the variable references in the condition part of the rule, that is, there is "1" in the vector if the corresponding expression in u is referred to in the rule, and "-1", if the expression is negated. The value of c_i is determined as follows: If the rule is disjunctive, $c_i =$#`neg`, equalling the number of negative expressions in the rule; if the rule is conjunctive, $c_i = 1-$#`pos`, where #`pos` is the number of non-negated expressions.

Augmentation of the vectors is needed for "grounding" the truth level. The augmented vectors are

$$\varphi_{i,\mathrm{aug}} = \left(\begin{array}{c} c_i \\ \hline \\ \varphi_i \end{array}\right), \qquad \text{and} \qquad u_{\mathrm{aug}} = \left(\begin{array}{c} 1 \\ \hline \\ u \end{array}\right). \tag{3.15}$$

If there are various rules, the matrix φ_{aug} contains them as its columns. One reasoning step for the whole set of rules can be expressed as

$$u(k+1) = f_{\mathrm{cut}}\left(\varphi_{\mathrm{aug}}^T u_{\mathrm{aug}}(k)\right). \tag{3.16}$$

For example, if there is just one rule `IF P`$_1$ `AND P`$_2$ `THEN P`$_3$, the corresponding "reasoning system" becomes

$$\left(\begin{array}{c} 1 \\ \hline u_{\mathrm{P}_1}(k+1) \\ u_{\mathrm{P}_2}(k+1) \\ u_{\mathrm{P}_3}(k+1) \end{array}\right) = f_{\mathrm{cut}}\left(\left(\begin{array}{r|ccc} 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 1 & 1 & 0 \end{array}\right)\left(\begin{array}{c} 1 \\ \hline u_{\mathrm{P}_1}(k) \\ u_{\mathrm{P}_2}(k) \\ u_{\mathrm{P}_3}(k) \end{array}\right)\right). \tag{3.17}$$

Starting from some initial state, the values in vector u change appropriately. If the rule base is more complex, the calculation is iterated until the vector u converges. Note that some kind of *integrators* are additionally needed, so that the acquired values are not lost; this is studied below.

However, there is a complication: In practice, the variables u_i can have values higher than 1, and if this is the case, the results of calculations do not correspond to appropriate logical values. To circumvent this problem, the variables need to be truncated to unit level before applying the rules. Using the cut function, this can be accomplished for a scalar variable as

$$u_{\text{trunc}} = 1 - f_{\text{cut}}(1-u) . \tag{3.18}$$

Introducing an "inversion matrix"

$$B = \left(\begin{array}{c|ccc} 1 & 0 & \cdots & 0 \\ \hline 1 & -1 & & \mathbf{0} \\ \vdots & & \ddots & \\ 1 & \mathbf{0} & & -1 \end{array} \right) , \tag{3.19}$$

this repeated inversion can be extended to (augmented) vectors:

$$u_{\text{trunc;aug}} = B f_{\text{cut}}(B u_{\text{aug}}) . \tag{3.20}$$

To implement the reasoning process in the same framework, one first needs two inversions for conditioning of the variable levels, and after that, the appropriate rule matrix can be applied. One of the inversion matrices can be combined with the rule matrix:

$$C = \varphi_{\text{aug}}^T B = \left(\begin{array}{c|ccc} 0 & 0 & \cdots & 0 \\ \hline c_1 + \sum_j \varphi_{1,j} & & -\varphi_1^T & \\ \vdots & & \vdots & \\ c_{m-1} + \sum_j \varphi_{m-1,j} & & -\varphi_{m-1}^T & \end{array} \right) . \tag{3.21}$$

The final iteration looks like

$$u(k+1) = f_{\text{cut}}\left(u(k) + C f_{\text{cut}}(B u(k))\right) \tag{3.22}$$

where the input vector u_{in} is put in u at time $k=0$. Above, integration is included in the iteration structure to maintain the current truth values of the variables; the unlimited growth of the variable values in the iteration is prevented when the vectors are additionally normalized (see later).

The forward chaining can also be implemented using matrix multiplications and applying cut functions; the structure seems complicated, though. What is more, it is not the original iteration structure (3.1) that is employed here. However, this complexity is just an illusion; and it turns out that the original neuron structure can be extended in a rather reasonable way. These issues are studied later (see Sec. 3.4.1).

As compared to the backward chaining example, now, rather than being rule activities, the variables x_i are quite redundant, needed only to implement truth value normalization, and one operates on the truth values of the expressions directly. Again, values above zero mean "non-false"; in a sense, *fuzzy logic* is being implemented.

In a rule system with no cyclic structures, matrix C can be written in a triangular form with zero diagonal. After a finite number of steps, no more changes take place in the vector u (assuming vector normalization; see later). It needs to be noted that the "straight-forward chaining" can be extended in this framework: Allowing non-binary weights, and fuzzy logical truth values, the resolution results can also be recirculated in the system. This means that C no more needs to be triangular; finding a stable result becomes an infinite process that hopefully converges.

This kind of production systems based on sequential processing are very versatile, as will be shown next.

3.3.2 Algorithmic representations

Nonlinear dynamic systems with feedback structure can have very complicated dynamics [6]. As shown in [26], *all computable functions* (functions that can be characterized by some algorithm) can be written in the form of a discrete-time dynamic system

$$\mathrm{x}(k+1) = f_{\mathrm{cut}}\left(\mathrm{A}\mathrm{x}(k)\right). \tag{3.23}$$

To make notations of [26] match the discussions ahead, the above expression will be written in the form

$$u(k+1) = f_{\mathrm{cut}}\left(u(k) + Cu(k)\right), \tag{3.24}$$

so that now $C = \mathrm{A} - I_n$.

Without going into details, it can be said that there exists a simple but universal programming language so that all algorithms coded in that language can be translated into matrix C. The vector $u(k)$ contains the program *snapshot,* that is, the program counter and the variable values are stored in that vector. Just a simple example is presented below.

Assume that the *parity function* is to be realized, so that $Y(X) = 0$ if $X \in \mathcal{N}$ is even, and $Y(X) = 1$ if X is odd. Using the tailored description language this can be coded as

```
1   VAR X = X     % Input variable
2   VAR Y = 0     % Output variable
3   IF X > 0      % Entry point
    THEN X SUB 1 Y ADD 1 GOTO 6
    ELSE GOTO END
6   IF X > 0
    THEN X SUB 1 Y SUB 1 GOTO 3
    ELSE GOTO END.
```

The syntax of the program code is simple and more or less self-explanatory (see [26]). For example, in the first conditional structure on line 3, if there still holds $X > 0$, one sets $X \leftarrow X - 1$ and $Y \leftarrow Y + 1$, and puts "6" in the program counter, meaning that the next instruction to be executed is in line 6; otherwise, the program terminates. The mnemonic `END` has been used to denote jump "outside" the program, that is, the program counter vanishes altogether, and the evaluation halts.

In a matrix form this algorithm can be implemented as shown below. The basic idea in the compilation of the language is that each program row spans a new dimension in the "snapshot space". One matrix row is allocated for all variables, and similarly for all program rows; conditional branches exhaust altogether *three rows.* The initial program snapshot, or the state $u(0)$ before iteration, is also shown below: Originally, u_1 contains the X value to be analyzed, and $u_3(0) = 1$ is the program counter.

$$C_{\texttt{parity}} = \begin{pmatrix} 0 & 0 & 0 & -1 & 1 & 0 & -1 & 1 \\ 0 & 0 & 0 & 1 & -1 & 0 & -1 & 1 \\ 0 & 0 & -1 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ -1 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \end{pmatrix}$$

with

$$u(0) = \begin{pmatrix} X \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

After the iteration (3.24) has converged to a fixed point $u(\infty)$, the final result Y is obtained as the second element u_2 of the state vector (because Y is declared on the second program line). It turns out that regardless of the value $X \in \mathcal{N}$, this system remains always stable; the time it takes to converge to a fixed state is linearly dependent of the value of X. For example, if $X = 3$, the following snapshot sequence results. The process seems to freeze in a state where $u_2 = Y = 1$, so that "3 is odd".

$$\begin{pmatrix} 3 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \to \begin{pmatrix} 3 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \to \begin{pmatrix} 2 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \to \begin{pmatrix} 2 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \to$$
$$\begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \to \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \to \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \to \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} \to \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \to \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \to \cdots$$

The behavior of the above "parity process" is visualized in Fig. 3.3. The output Y (entry $u_2(\infty)$) has been plotted after convergence as the initial values $u_1(0)$ and $u_3(0)$, or the "input" X and the "program counter" are continuousaly varied. In the figure, black stands for zero level, lighter colors denoting higher values. Note that only in discrete points (dotted points in the figure) the parity value is defined in the traditional sense, the values "1" and "0", or "true" and "false", alternating along that line. The continuity

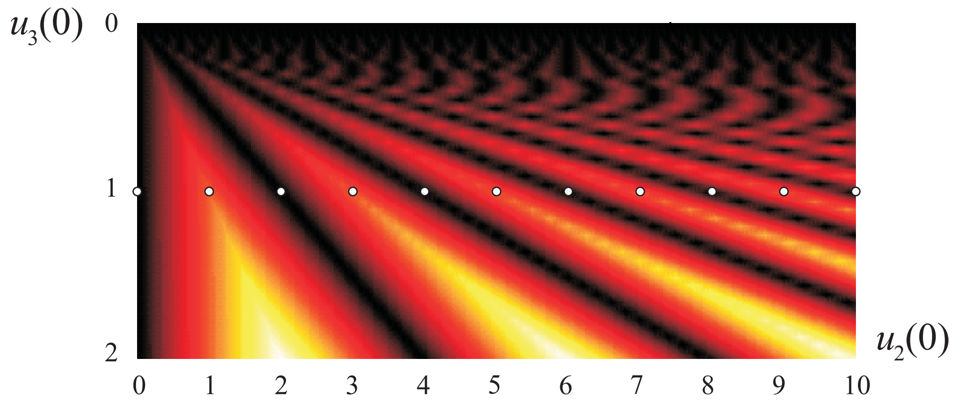


Figure 3.3: "Generalized parity function" (see text)

and (piecewise) differentiability of the algorithm implementation sounds fascinating: In principle, it is possible to adapt the algorithm structures along the gradient direction to better implement the mappings between X and Y. However, this seems not to be realistic: In complex algorithm implementations there are many local minima; what is more, there seem to exist very abrupt behaviors in the mapping functions.

The presented approach to implementing algorithms does not perhaps have very much practical relevance; it is just a proof that anything, indeed, that can be somehow implemented, can also be implemented in principle using simple discrete-time dynamics boosted with the cut nonlinearity, if the structures are sufficiently piled on top of each other. In what follows, these studies are summarized.

3.4 Higher-level views

The goal now is to study whether the presented functionalities could be implemented in the proposed Hebbian/anti-Hebbian neuron structure. Inevitably, the original framework (single grid of neurons) needs to be extended; when wondering of the infinite possibilities how the framework could be modified, it is again system theory that seems to offer intuition. Remember that one of the central tools is the idea of feedback: *Connect the inputs and outputs of the original open-ended system.*

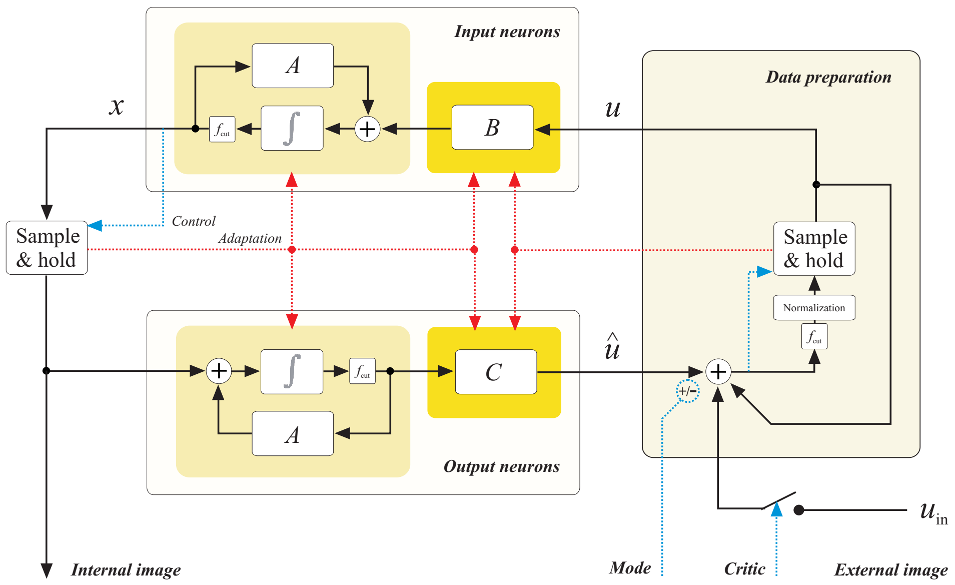


Figure 3.4: Cognitive basic block

3.4.1 Mental model

The remaining loose ends are now also connected by applying feedback, recirculating the analysis results back to the input (see Fig. 3.4).

In the structure of Fig. 3.4, two neuronal structures are combined. The input neuron structure implements sparse analysis, and the output neuron structure implements sparse regression (see Chapter 1). The input data u is stored in the input buffer, and the latent variables $\bar{x}$ are stored in the internal buffer. The processes are synchronized so that new samples in the buffers are taken whenever the appropriate signals have converged. Fresh input u_{in} is stored in the input buffer, or, if this is missing, the residuals are recirculated. The input is normalized to unit variance before each iteration, so that the contents of the buffer is (ϵ being a small scalar, added here to avoid pathological cases)

$$u \leftarrow \frac{u}{\|u\| + \epsilon}. \tag{3.25}$$

Note that the data cannot have zero mean because of the positivity assumption. Still, there is no mean centering; it is also implicitly assumed that there always holds $\bar{x} = 0$ when $u = 0$.

The dynamics in Fig. 3.4 is implemented in a continuous-time form, applying integrators; formulations are simpler in this way, and stability issues can concerning the internal dynamics can be avoided. Note that rather than using ideal integrators, it is reasonable to apply *limited integrators,* so that negative internal activity does not cumulate.

A fundamental question that arises is whether the feedback in the overall system should be *positive* or *negative.* Intuitively speaking, negative feedback tries to somehow equalize the system properties, whereas positive feedback makes the structures differentiate themselves, so that some kind of self-organization can emerge. Positive feedback often results in instability; however, now there is the normalization of the data included in the loop, so that such problems do not become acute. Both alternatives are available, which one to choose?

It turns out that *both* of the feedback alternatives are equally useful. There is a control signal α for switching between the two modes ($\alpha = 1$ for positive, and $\alpha = -1$ for negative feedback):

1. During the *learning* phase, the feedback is *negative,* so that $u'(k+1) = u(k) - \hat{u}(k)$ (before truncation and normalization). That is, differences between the pattern and the already captured features, or the unmodeled phenomena in the input, are emphasized whereas the already captured ones are ignored during the successive iterations. In this sense, automatic *attention control* takes place.

2. During the *recall* phase, the feedback is *positive,* so that $u'(k+1) = u(k) + \hat{u}(k)$ (before truncation and normalization). That is, some kind of an average between the prior image and its reconstruction is calculated. An incomplete image in the buffer is associatively completed: This means that unknown data can be reconstructed, or noisy data can be filtered.

In both cases, iteration is continued until some stopping criterion is reached (for example, convergence of the input buffer contents). When learning from data, the iteration could be controlled, for example, by the pattern recognition capability: Only if the classification is incorrect, the residual $u(k+1) = u(k) - \hat{u}(k)$ is recirculated, so that the fine structure between patterns can be learned. Because of the underlying linearity, different patterns (and different resolutions of patterns) can coexist in the same structures. But these patterns can also interact: Specially, when there is sparse coding, patterns on all levels compete with each other, only the most relevant becoming manifested.

Some outside *critic* is needed to decide when a new pattern is to be concentrated on. The sampling in the buffers, on the other hand, can be autonomous: Always when the incoming signals have converged, a new sample can be taken.

However, it is also possible to apply the presented scheme if the signals do *not* converge: Assume that there is a train of fresh input vectors u_{in} coming with short constant time intervals. Assuming that there are no (crucial) zero-crossings taking place during those intervals, the whole system can be described as dynamic linear variable structure system:

$$\left\{ \begin{array}{rcl} x(k+1) & = & A_x(k)x(k) + Bu(k) \\ v(k+1) & = & A_v(k)v(k) + x(k) \\ \hat{u}(k+1) & = & Cv(k+1). \end{array} \right. \tag{3.26}$$

Because the input is constant during time interval k, and because the set of active neurons behaves linearly (inactive ones being ignored altogether), the matrices $A_x(k)$ and $A_v(k)$ are constant during that interval, representing the discretized versions of the continuous-time matrix A being integrated over the interval (zero-order hold formulas apply; see [2]). This means that dynamic succession of input patterns can be captured, and some kind of continuum, or *causality* among patterns, can (perhaps) be modeled.

Comparing the presented overall structure to the functionalities that were discussed above, one can see that all of them can be implemented in this framework:

- **Declarative rules.** Select $\alpha = 1$, $A = I_n$, and let B and C be the inversion matrix and combined inversion/rule matrix, respectively, as defined in Sec. 3.3.1.
- **Algorithmic procedures.** Select $\alpha = 1$, $A = B = I_n$, and let C be the matrix describing the algorithm, as defined in Sec. 3.3.2.
- **Programmed features.** Select $\alpha = 1$, $A = \delta V + \varphi^T \varphi$, and let $B = C^T = \varphi^T$. Matrix V is determined obeying the domain area semantics.
- **Data-based models.** Select $\alpha = 1/-1$, $A = \delta V + \mathrm{E}\{f_{\text{cut}}(\bar{x}) f_{\text{cut}}^T(\bar{x})\}$, and let $B = C^T = \mathrm{E}\{f_{\text{cut}}(\bar{x}) u^T\}$, as presented in Chapters 1 and 2.

Let us study closer one of the additional structures that were introduced in addition to the original Hebbian/anti-Hebbian neuron structure, namely, the *normalization* block. This normalization means that the overall activity level is kept constant. This seemingly very innocent operation facilitates many fundamental functionalities. For example, if $u_{\text{in}} \gg 1$, the new input always outweighs the earlier data in the buffer, because the buffer contents are scaled down proportionally; separate initialization structures or procedures are thus not needed.

Normalization keeps the data bounded also when positive feedback is applied: Indeed, when the reconstruction is sufficiently many times added to the input buffer, and when the data is normalized, the original input gradually changes into a *virtual input.* And when negative feedback is applied, on the other hand, analysis beyond the nominal PCA can be carried out. To facilitate analyses, for a moment, assume linearity in the structures: Remember that the Hebbian/anti-Hebbian learning emphasizes the directions with highest data variation, that is, it extracts the most significant principal components. Assuming that the most relevant principal components have already been extracted, these most dominant principal components are deflated from the data $u - \hat{u}$; when the data is scaled up, in this new data the less visible principal components are equally visible during the next iteration, and they can also be learned.

What comes to declarative rules, normalization causes no essential complications. The nominal "truth level" is grounded to the augmented variable, and when the data is scaled, this level just fluctuates without affecting the reasoning results. In the algorithm case, analogously, the unit level is grounded to the program counter, and normalization does not affect the system dynamics as long as the program counter is scaled by the same amount as the variables are (note that the actual variable values are obtained by dividing the scaled values by the program counter).

3.4.2 Relations to cognitive science

There is yet another characteristic facet of system theory that makes it powerful. In system theoretic studies, *intuition* is valued, and one tries to see analogues and connections between disciplines. Simultaneously, this kind of scent of heuristics also makes such discussions vulnerable to attacks from traditional theories. Above, it was not only neuron level phenomena that were studied, but also the cognitive ones. Indeed, such issues are elaborated on here, and some more or less justified comparisons are made to the state-of-the-art *cognitive science* (for example, see [31]).

There exist many fundamental concepts in cognitive science, among the most prominent perhaps being *long-term memory* and *short-term memory* (or *working memory*). The volatile short-term memory contains references to long-term memory representations, often called *chunks,* or basic units determining the input pattern decompositions. In the absence of alternative conceptual frameworks these chunks have traditionally been thought of as being strictly symbolic constructs [9]. The cognitivist theories recognize (among other things) the essence of *constraints* when studying cognitive capabilities: For example, it has been recognized that the short-term memory capacity is limited, it can only store some 4 to 9 independent entities at a time.

Looking at the proposed mental model structure, it is the internal buffer that corresponds to short-term memory; long-term memory is distributed in the matrices A, B, and C. The sparsity of the coding corresponds to the limited short-term memory capacity: Only a subset of indices referring to long-term memory are simultaneously active. The input buffer is the *sensory buffer,* where the original data is stored; manipulation of that data corresponds to the process of filtering the observation through the existing memory structures in the constructivistic sense to reach the "mental image". Whereas the input buffer corresponds to the observed state of the world, the internal buffer corresponds to *perceived* state of the world.

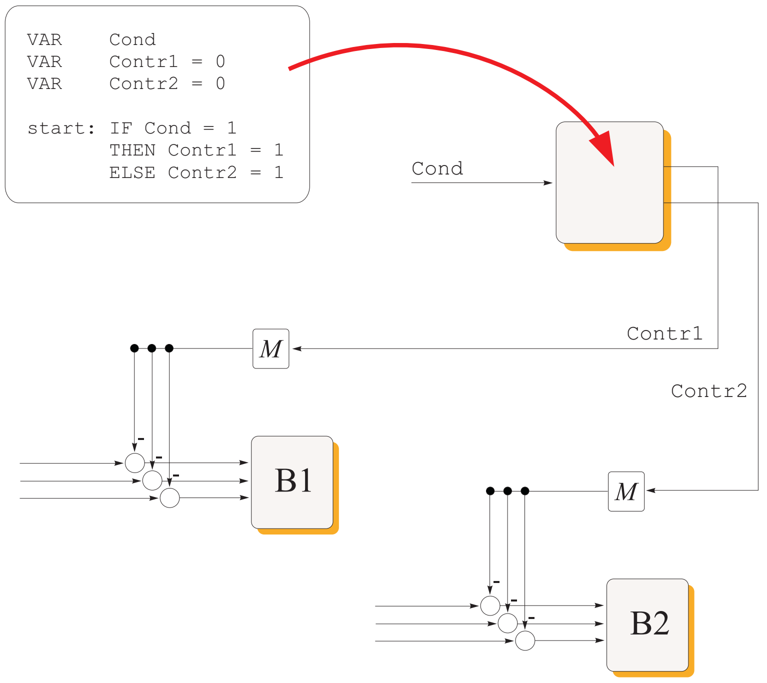
The idea of *mental images* is a useful concept: Originally, mental imagery was studied exclusively in the context of concrete visual scenes (see [35]). However, the nature of the mental imagery is not agreed upon [45], and parallel "mental views" seem like a good approach to discuss expertise in general — the expert has internalized a sophisticated set of mental images governing the problem area. The constructs describing the domain field, or chunks, can be subsymbolic as well as symbolic. As shown above, the specialized imagery consisting of the domain-specific constructs constitutes a "filter" that preprocesses the observation data, creating a compact subjective internal representation of the situation at hand.

It is now assumed that the structure in Fig. 3.4 constitutes the *cognitive basic building block* for implementing the different mental faculties. Such blocks can be combined, so that the internal images from some prior processing phases can be used as input data for subsequent processing. One block can thus combine various modalities or input channels, introducing some kind of hierarchy among the blocks. The structures among processing levels are not strictly hierarchic, because the dependencies can be cyclic, and the hierarchies can be tangled. Cognitive functionalities assumedly emerge from massive iteration in the feedback loops also on the higher levels of abstraction. One needs to master controlling and timing among subprocesses — How could this be explained in the proposed framework? The principle is visualized in Figs. 3.5 and 3.6. The control structures are algorithmic, and they can be implemented as shown in Sec. 3.3.2; when denoted control variables are toggled between 0 and 1, separate sub-blocks can be activated. Look at Fig. 3.5: If the signal `Cond` equals 0, the block B1 is deactivated; if it equals 1, block B2 is deactivated (M being a large constant). On the other hand, one needs mechanisms to collect binary information from the sub-blocks to launch control functions; this kind of quantization is shown in Fig. 3.6, where a comparator is implemented ("Is x larger than threshold T?"). This means that a hard-wired system with control structures can be implemented using the proposed building blocks; however, the question remains how such control structures could autonomously be organized without external intervention.

One of the most interesting paradoxes in cognitive science is the *shift from novice to expert* (see [10], [15]). Somehow the mental representations change from *declarative* to *associative;* the procedural data processing changes gradually towards parallel pattern matching. The associations can be modeled using correlations — and, indeed, the proposed framework can offer here new intuitions. If the preprogrammed matrices are used for manipulation of input data, the relationships between the buffer contents in $\bar{x}$ and u can be used for simultaneous adaptation of the matrices. The sequential, multiple-step analysis processes can thus be gradually transformed into an associative, one-step process, the solution starting to emerge immediately when the input is applied.

However, it needs to be noted that everything cannot be implemented in the associative framework, even though the sparsity of the representations extends the possibilities beyond the strictly linear models. An easy way to see this is to note that because there can only exist positive entries in $\bar{x}$, its covariance matrix cannot contain negative entries — even though the optimal $\varphi^T\varphi$ should sometimes contain also negative ones (see Sec. 3.2.2). This means that a data-based system can *never* converge to such a structure in the proposed framework.

The presented approach to explaining mental phenomena is very simplistic, and its plausibility can be criticized. However, it seems that when simple structures cumulate, the emerging complexity may start looking "intelligent". The potential of the "numeric chunks" was studied in [24], where the extracted correlation structures were used for modeling chess configurations; the behaviors of such models were very expert-like. For example, the errors in piece recall were rather plausible, and there was graceful degradation in the observed model performance.



```
VAR     Cond
VAR     Contr1 = 0
VAR     Contr2 = 0

start: IF Cond = 1
       THEN Contr1 = 1
       ELSE Contr2 = 1
```

Figure 3.5: From logic domain to continuous variables

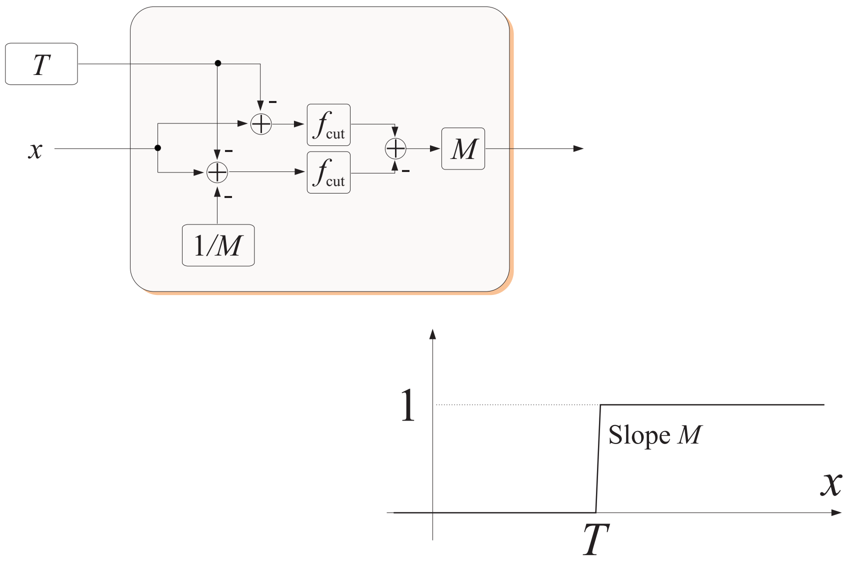


Figure 3.6: From continuous variables to logic domain

3.5 Discussion

In this chapter, the behavior of a neuron grid was studied from the *holistic* point of view. It turned out that cybernetic *self-stabilization* was reached when the neurons were connected to each other and *feedback* was applied. Further, it was recognized that if structural constraints are introduced, or if there is nonlinearity in the system, some kind of *order* emerges in the system. Finally, when the last feedback loop was closed, a complete *cognitive model* was reached.

There exist various frameworks that have been proposed for modeling of mental phenomena, most notably perhaps ACT-R [1] and SOAR [37], starting from some specific architectural or functional premises. The basic problem is that such models often need to be extended to include additional functionalities, and the general-purpose models finally become increasingly complex, thus making them intuitively less appealing. Looking at the today's models of cognition, one can see that they contain functionally very specialized blocks, various separate control units coordinating their operation, transfer of data between storages being carried out in a very computer-like way. Still, there is only the same, fundamentally non-digital neural structure available — in a such framework it is very difficult to explain how different mental faculties could have adapted rather than being hard-wired. Connectionist models that have been proposed typically fail to answer the systemic questions (for example, see [29]).

It should be remembered that one is just constructing models, not claiming that there should exist some fundamental correspondence between the model and the reality. However, as compared to traditional engineering disciplines, system theory is just a step nearer to philosophy and metaphysics: It not only tries to explain behaviors, answering the *how* questions, but it also tries to answer the *why* questions, trying to find general principles governing the system behavior. There is a huge leap between the *how*'s and the *why*'s; the successful derivations above partly rely on the favorable starting point, the intuitively appealing Hebian principles. One started with an *intentional* assumption concerning the neuronal behavior: It was assumed that neurons *try to* maximize correlation or match between some quantities. It turned out to be a fruitful starting point; however, now the problem setting has been transformed and one has the possibility of making still more ambitious and general assumptions about the goals of the neuronal system:

- **Optimality.** It was noticed that there exist optimality criteria embracing the Hebbian/anti-Hebbian learning principle, so that the criteria are optimized when the iterations converge. Looking at the process from this perspective, seeing the infinite process as a finite pattern, helps in gaining intuition, making it possible to easily modify the system behavior.

- **Balance.** The role of (negative) feedbacks can be emphasized by saying that the neural system tries to react so that the changes affecting the system become compensated. If there is some excitation in the sensory buffer, the neuronal feedbacks do their best to eliminate this disturbance. In a sense, this idea follows the *Le Chatelier Principle* familiar from Chemistry: As a reaction to changes in the environment, the steady state of the system changes so that the environmental changes become compensated (at least to some extent).

- **Symmetry.** It seems that in the converged Hebbian/anti-Hebbian structure, there exists symmetry in many levels: First, the matrix A is symmetric, as well as $C = B^T$; in the input/output structure, the two branches are also symmetrical. And, as explained in Sec. 3.2.1, data and structural knowledge are also symmetric in some sense. Perhaps the degree of symmetry in the system could be used as a measure of how far the system is from the final fully associative state? Note that, as compared to particle physics, for example, symmetricity is interpreted here in a somewhat intuitive way (indeed, actual symmetry groups are far from visible symmetries).

- **Simplicity.** Last but not least, the age-old measure for model validity is its simplicity. It can be claimed that in the proposed framework maximum amount of functionality is reached with minimum amount of complexity; in the spirit of *Ockham's razor,* there must also be some *truth* in the proposed model? As compared to conventional cognitive models, the number of adjustable parameters, or degrees of freedom, is now minimum — indeed, there are *none* of them (except for the number of neurons). This number of parameters is another aspect of scalability: If dozens of such blocks are combined to implement some more complicated functionality, the number of adjustable parameters must not explode.

Whether or not such high-level ideas can be taken as starting points in further modeling of neuronal and cognitive systems is an open question. However, it is evident that the boundaries of understanding of mental phenomena are not yet at hand. Perhaps the main obstacle here is *lazy thinking,* or the reluctance of humans to adopt fresh ideas. Take two examples from opposite ends of the continuum:

1. *"Mental phenomena cannot ever be captured using reductionistic approaches."* Another way to put this is *"Human brain cannot study its own operation."* Divine explanations have been forgotten in physics, why could they not be forgotten also in physiology? At least when studying limited mental faculties, like different manifestations of intelligence in narrow fields, why should one assume that there must exist some mind, or soul, or consciousness, that is running the machinery? As was seen above, global, non-trivial results emerge when the individual actors just try to follow some simple adaptation principles with no centralized control.

2. *"When computing machinery develops sufficiently it automatically becomes intelligent."* It has been claimed that in some twenty years (why is it always twenty years in AI?) the computing capacity in a computer is comparable to that of a human brain. However, faster and faster processing only means faster and faster cumulation of numerical errors. The key point is that *correct things* have to be done by the algorithms: It does not matter how long it takes, sparks of relevant behavior can easily be detected if they are to emerge from the chaos eventually. Indeed, the claim here is that today's computer already *is* powerful enough to implement truly interesting functionalities.

3.6 Conclusions

In this chapter it was shown how the simple neural structures that were intended for data modeling tasks can be extended to carry out declarative reasoning tasks and algorithmic computation. It turned out that also in this task the system theoretic intuitions (feedback, etc.) could be utilized.

System theory is the framework for attacking systems in a holistic way, and it offers conceptual tools for analyzing emergent phenomena. In Chapters 1 and 2, the discussions started in a bottom-up way, studying individual neurons; the emergent properties (optimality of the neuron grid behaviors) inspired the extension to the more abstract levels, towards cognitive functionalities. Is there still something that would be offered by system theoretic thinking? Indeed, when the road bottom-up has been studied, one can start analyses in *top-down* direction: The general intuitions gained can be applied in *other* more or less similar domain fields.

Such analogues can truly turn out to be fruitful. The neural system is an example of *cybernetic systems* (see [51]) — and there are dozens of interesting cybernetic domains falling short of efficient analysis practices. The experiences with neural systems have been applied in a variety of fields, and, indeed, the results are interesting (see `http://www.control.hut.fi/cybernetics`).

Bibliography

[1] Anderson, J.R. (1983). **The Architecture of Cognition.** Cambridge, MA: Harvard University Press.

[2] Åström, K.J. and Wittenmark, B. (1997). **Computer controlled systems.** Upper Saddle River, NJ: Prentice–Hall.

[3] Barabasi, A.-L. (2002). **Linked: The New Science of Networks.** Cambridge, MA: Perseus Books.

[4] Basilevsky, A. (1994). **Statistical Factor Analysis and Related Methods.** New York, NY: John Wiley & Sons.

[5] von Bertalanffy, L. (1969). **General System Theory — Foundations, Development, Applications** (revised edition). New York, NY: George Braziller.

[6] Blondel, V.D. and Tsitsiklis, J.N. (2000). A survey of computational complexity results in systems and control. *Automatica,* **36**, Issue 9, pp. 1249–1274.

[7] Buntine, W. (2002). Variational Extensions to EM and Multinomial PCA. In *Machine Learning: Proceedings of ECML 2002, 13th European Conference on Machine Learning* (eds. Elomaa, T., Mannila, H., and Toivonen, H.), Lecture Notes in Computer Science 2430. Berlin: Springer.

[8] Chang, C.-L. and Lee, R. (1973). **Symbolic Logic and Mechanical Theorem Proving.** New York, NY: Academic Press.

[9] Chase, W.G. and Simon, H.A. (1973). The mind's eye in chess. In *Visual Information Processing* (ed. Chase, W.). New York, NY: Academic Press.

[10] Chase, W.G. and Ericsson, K.A. (1982). Skill and working memory. In *The Psychology of Learning and Motivation* (ed. Bower, G.H.). New York, NY: Academic Press.

[11] Cichocki, A. and Amari, S. (2002). **Adaptive Blind Signal and Image Processing: Learning Algorithms and Applications.** New York, NY: Wiley.

[12] Cichocki, A., Kasprzak, W., and Skarbek, W. (1996). Adaptive Learning Algorithm for Principal Component Analysis with Partial Data. *Proc. Cybernetics and Systems,* **2**, pp. 1014–1019.

[13] Costa, S. and Fiori, S. (2001). Image compression using principal component neural networks. *Image and Vision Computing,* **19**, pp. 649–668.

[14] Diamantaras, K.I. and Kung, S.Y. (1996). **Principal Component Neural Networks: Theory and Applications.** New York, NY: Wiley.

[15] Elio, R. and Scharf, P.B. (1990). Modeling novice-to-expert shifts in problem-solving strategy and knowledge organization. *Cognitive Science,* **14**, pp. 579–639.

[16] Fiori, S. (2003). Neural Independent Component Analysis by "Maximum-Mismatch" Learning Principle. *Neural Networks,* **16**, No. 8, pp. 1201–1221.

[17] Földiák, P. (1990). Forming sparse representations by local anti-Hebbian learning. *Biological Cybernetics,* **64**, pp. 165–170.

[18] Földiák, P. (2002). Sparse coding in the primate cortex. In *The Handbook of Brain Theory and Neural Networks* (second edition), ed. M.A. Arbib. Cambridge, MA: MIT Press.

[19] Foltz, P.W. (1990). Using Latent Semantic Indexing for Information Filtering. *Proceedings of the Conference on Office Information Systems,* Cambridge, Massachusetts, pp. 40–47.

[20] Glymour, C., Madigan, D., Pregibon, D., and Smyth, P. (1997). Statistical Themes and Lessons for Data Mining. *Data Mining and Knowledge Discovery,* **1**, pp. 11–28.

[21] Haykin, S. (1999). **Neural Networks — A Comprehensive Foundation.** Upper Saddle River, NJ: Prentice–Hall.

[22] Hebb, D.O. (1949). **The Organization of Behavior: A Neuropsychological Theory.** New York: John Wiley & Sons.

[23] Honkela, T., Kaski, S., Lagus, K., and Kohonen, T. (1996). *Newsgroup Exploration with WEBSOM Method and Browsing Interface.* Helsinki University of Technology, Laboratory of Computer and Information Science, Report A32.

[24] Hyötyniemi, H. and Saariluoma, P. (1999). Chess — Beyond the Rules. In **Games, Computers and People**, ed. Timo Honkela. Finnish Artificial Intelligence Society, Helsinki, Finland, pp. 100–112.
(Available at `http://www.control.hut.fi/hyotyniemi/publications/`)

[25] Hyötyniemi, H. (2002). HUTCH Model for Information Structuring. In *Proceedings of Finnish Artificial Intelligence Conference (STeP'02),* Oulu, Finland, December 2002, pp. 241–255.
(Available at `http://www.control.hut.fi/hyotyniemi/publications/`)

[26] Hyötyniemi, H. (2002). **On the universality and undecidability in dynamic systems.** Helsinki University of Technology, Control Engineering Laboratory.
(Available at `http://www.control.hut.fi/hyotyniemi/publications/`)

[27] Hyötyniemi, H. (2002). Studies on Emergence and Cognition — Part 1: Low-Level Functions, and Part 2: High-Level Functionalities. In *Proceedings of Finnish Artificial Intelligence Conference (STeP'02),* Oulu, Finland, December 2002, pp. 286–312.
(Available at `http://www.control.hut.fi/hyotyniemi/publications/`)

[28] Hyvärinen, A., Karhunen, J., and Oja, E. (2001). **Independent Component Analysis.** New York, NY: John Wiley & Sons.

[29] Jackson, S.A. (1996). **Connectionism and Meaning: From Truth Conditions to Weight Representations.** New Jersey, NJ: Ablex Publishing.

[30] Karhunen, J. and Oja, E. (1982). New Methods for Stochastic Approximation of Truncated Karhunen-Loeve Expansions. In *Proceedings of the 6th International Conference on Pattern Recognition,* Munich, Germany, October 1982, pp. 550–553.

[31] Kellogg, R.T. (1995). **Cognitive Psychology.** London: SAGE Publications.

[32] Klein, D.J., König, P., and Körding, K.P. (2003). Sparse Spectrotemporal Coding of Sounds. *EURASIP Journal on Applied Signal Processing,* **7**, pp. 659–667.

[33] Klir, G.J. (2001). **Facets of Systems Science** (second edition). Ifsr International Series on Systems Science and Engineering, Vol. 15. New York, NY: Plenum Press.

[34] Kohonen, T. (2001). **Self-Organizing Maps** (third edition). Springer Series in Information Sciences, Vol. 30. Berlin: Springer.

[35] Kosslyn, S.M. (1980). **Image and Mind**. Cambridge, MA: Harvard University Press.

[36] Laaksonen, J.: *Subspace Classifiers in Recognition of Handwritten Digits.* Acta Polytechnica Mathematica; Mathematics, Computing and Management in Engineering series No. 84, Espoo, 1997.

[37] Laird, J.E., Rosenbloom, P.S., and Newell, A. (1987). Chunking in SOAR: The anatomy of a general learning mechanism. *Machine Learning,* **1**, pp. 11–46.

[38] Lesser, V., Ortiz, C.L., Jr., and Tambe M., eds. (2003). **Distributed Sensor Networks — A Multiagent Perspective.** Boston, MA: Kluwer Academic Publishers.

[39] Ljung, L. (1986). **System Identification: Theory for the User.** Upper Saddle River, NJ: Prentice–Hall.

[40] Mannila, H. (1997). Methods and Problems in Data Mining. *Proceedings of the International Conference on Database Theory,* Delphi, Greece, January 1997.

[41] Myllymäki, P., Silander, T., Tirri, H., and Uronen, P.: Bayesian Data Mining on the Web with B-Course. *Proceedings of the 2001 IEEE International Conference on Data Mining,* San Jose, USA, November 2001, pp. 626–629.

[42] Oja, E. (1992). Principal components, minor components, and linear neural networks. *Neural Networks,* **5**, pp. 927–935.

[43] Olshausen, B.A. and Field, D.J.: Sparse coding with an overcomplete basis set: A strategy employed by V1? *Vision Research,* Vol. 37, 1997, pp. 3311–3325.

[44] van Overschee, P. and de Moor, B. (1996). **Subspace Identification for Linear Systems: Theory — Implementation — Applications.** Boston, MA: Kluwer Academic Publischers.

[45] Pylyshyn, Z. (1981). The imagery debate: Analogue media versus tacit knowledge. *Psychological Review,* **88**, pp. 16–45.

[46] Salton, G. and Buckley, C. (1988). Term Weighting Approaches in Automatic Text Retrieval. *Information Processing and Management,* Vol. 24, No. 5, pp. 513–523.

[47] Sanger, T.D. (1989). Optimal unsupervised learning in a single-layer linear feedforward neural network. *Neural Networks,* **12**, pp. 459–473.

[48] Simon, H.A. (1996). **Sciences of the Artificial** (third edition). Cambridge, MA: MIT Press.

[49] Tipping, M.E. and Bishop, C.M. (1999). Mixtures of Probabilistic Principal Component Analyzers. *Neural Computation,* **11**, pp. 443–482.

[50] Wang, Z., Lee, Y., Fiori, S., Leung, C.-S., and Zhu, Y.-S. (2003). An Improved Sequential Method for Principal Component Analysis. *Pattern Recognition Letters,* **24**, No. 9–10, pp. 1409–1415.

[51] Wiener, N. (1948). **Cybernetics: Or Control and Communication in the Animal and the Machine.** New York, NY: Wiley.