8. EXTENSION OF KOHONEN'S MODEL

In the preceding chapters, motivated by the important role of topology-conserving maps in the brain, we considered a stochastic learning algorithm which, based only on a random sequence of sensory signals, generates such maps on a lattice of formal neurons. The aim of these maps was to represent the neighborhood relationships between the input signals in a two-dimensional, nonlinear projection as faithfully as possible.

8.1 Motor Maps

The brain cannot limit itself, however, to the representation of sensory input signals alone, but must also solve the complementary task of sending appropriate signals to the muscle system to react to the sensory input. The brain regions that are responsible for these tasks, such as the *motor cortex* and the *superior colliculus*, appear in many cases to be organized in a way similar to the sensory areas, *i.e.*, as maps that react to localized excitation by triggering a movement. This movement varies in a regular way with the focus of the excitation in the layer. Therefore, the layer can be considered as a *motor map* in which movement commands are mapped to two-dimensional locations of excitation (Lemon 1988).
By way of an abstraction that can also serve as a model for such motor maps, we consider in this chapter topology-conserving maps in which we additionally allow, as an extension of Kohonen's original learning algorithm, the storage of an *output* specific for each lattice point (Ritter and Schulten, 1986b, 1987, 1988). As we will see, the formation of a map then corresponds to the *learning of a control task*. The type of the output value can be, for example, a scalar, a vector, or a linear operator, *i.e.*, a tensor.
Several neural mechanisms for the storage of such outputs can be imagined. The information could be stored by a suitable selection of connections to subsequent motor neurons. Linear mappings could be realized by "de-inhibition" of local neuron groups. Furthermore, local "assemblies" of neurons also could

act as local associative memories for the storage of more complex information. Such a neural network would form an active storage medium that reacts to a local activation caused by an input signal with the response stored at the site of excitation. Stored data are arranged in such a way that similar data are stored in spatial proximity to each other. A thoroughly investigated example of such organization, involving the case of a vectorial output quantity, is the *motor map* in the so-called *superior colliculus*, a mounded, multi-layered neuron sheet in the upper region of the brain stem (Sparks and Nelson 1987). In this sheet there are neurons which, when excited, trigger the execution of rapid eye movements (*saccades*). Usually, such excitation is caused by neurons located in the more superficial layers that process sensory signals. Through electrical stimulations by inserted electrodes, this excitation can also be induced artificially. Such experiments demonstrate that the resulting change of the vector of view direction varies in a regular fashion with the location of excitation in the layer. In Chapter 9 we will discuss more closely this example of a motor map and the control of saccades.
In much the same way as maps are learned or are adaptively modifiable, the assignment of output values to lattice sites must be plastic. Consequently, a learning algorithm is also required for the output values. In the following chapter we will show that with an appropriate algorithm the learning of the output values can benefit substantially from the neighborhood-, and hence, continuity-preserving spatial arrangement of the values within a map.
In the following we again onsider a space V of input signals and a lattice A of formal neurons. As in Chapter 4, $\phi_{\mathbf{w}}$ denotes the mapping of V onto A that is specified by the synaptic strengths. In addition, for maps intended to be used for motor control tasks, an output value $w_{\mathbf{r}}^{(out)}$ is assigned to each neuron $\mathbf{r}$. (Here, as before, $w_{\mathbf{r}}^{(out)}$ can be a vectorial or tensorial value). Since the synaptic strengths, so far denoted by $\mathbf{w}$, determine the correspondence between *input signals* and neurons, we will denote them in this chapter by $\mathbf{w}^{(in)}$ so that they will not be confused with the recently introduced output values $\mathbf{w}^{(out)}$.
Together, all $\mathbf{w}_{\mathbf{r}}^{(out)}$ form a covering of the lattice with values in a second space U and extend the mapping $\phi_{\mathbf{w}}$ from V onto the lattice to a mapping Φ of V into the space U, given by

$$\Phi : V \mapsto U, \quad \mathbf{v} \mapsto \Phi(\mathbf{v}) := w_{\phi_{\mathbf{w}}(\mathbf{v})}^{(out)}. \tag{8.1}$$

Fig. 8.1 offers an illustration of this situation. In accordance with the motor maps formerly mentioned, we want to investigate mappings defined in this

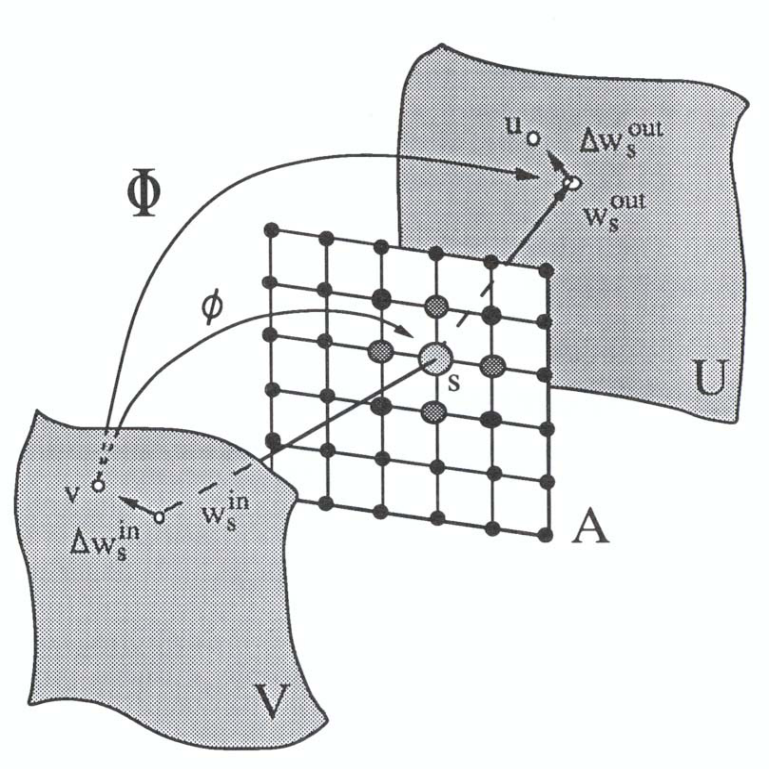

Abb. 8.1: The extended model with the inclusion of output values. Each formal neuron s of the neuron layer (lattice A) has, in addition to its pre-existing weight vector $\mathbf{w}_\mathbf{s}^{in}$, a vector $\mathbf{w}_\mathbf{s}^{out}$ of output values assigned to it. A learning step now requires, with each presentation of an input vector $\mathbf{v}$, the specification of a corresponding output value u. The adaptation of the output values $\mathbf{w}_\mathbf{s}^{out}$ is completely analogous to the scheme used for the "input side:" all neurons in the vicinity of the particular neuron selected by the input value shift their output vectors towards the specified output value u.

way, especially with regards to motor control tasks. In this case $\mathbf{v}$ represents the present state of the system which is to be controlled, and $\mathbf{w}_\mathbf{r}^{out}$ determines the required control action. This control action can specify a value for a *displacement*, a *force*, or a *torque* (Chapters 8, 9, and 13), or it can specify a *linear mapping* that determines these values in terms of the intended target state of the motion (Chapters 11 and 12).

8.2 Supervised and Unsupervised Learning

We begin with the simplest case, namely, that of a control task for which a sequence of correct *input-output pairs* $(\mathbf{v}, u)$ are available. Here, $\mathbf{v}$ denotes the system state and u the correct control action associated with that state (Ritter and Schulten 1987). This situation corresponds to *supervised learning* and requires the opportunity for observing the correct control, provided by a "teacher" for a sufficiently extended period of time. The synaptic strengths $\mathbf{w_r}$ and the output values $w_{\mathbf{r}}^{(out)}$ are initialized with values chosen without any a priori information. They can be initialized, for example, with random values. The purpose of the learning process is a gradual improvement of the initial mapping Φ with the goal of an increasingly better imitation of the teacher's actions by Φ. This can be realized by the following algorithm:

1. Record the next control action $(\mathbf{v}, u)$.
2. Determine the lattice site $\mathbf{s} := \phi_{\mathbf{w}}(\mathbf{v})$ whose input weight vector $\mathbf{w}_{\mathbf{s}}^{(in)}$ best matches the present system state $\mathbf{v}$.
3. Perform a learning step
$$\Delta \mathbf{w}_{\mathbf{r}}^{(in)} \quad = \quad \mathbf{w}_{\mathbf{r}}^{(in)} + \epsilon h_{\mathbf{rs}}(\mathbf{v} - \mathbf{w}_{\mathbf{r}}^{(in)})$$
for the input weight vectors $\mathbf{w}_{\mathbf{r}}^{(in)}$.
4. Perform a learning step
$$\Delta w_{\mathbf{r}}^{(out)} \quad = \quad w_{\mathbf{r}}^{(out)} + \epsilon' h'_{\mathbf{rs}}(u - w_{\mathbf{r}}^{(out)})$$
for the set of output values $w_{\mathbf{r}}^{(out)}$ and return to step 1.

Steps 1–3 describe Kohonen's original algorithm for the formation of a topology conserving map on A. They are depicted in the front part of Fig. 8.1 where the different sizes and shades of the neurons in the vicinity of $\mathbf{s}$ indicate the shape and extent of the excitation zone as determined by the function $h_{\mathbf{rs}}$. The new step, step 4, changes the output values $w_{\mathbf{r}}^{(out)}$ assigned across A. This change is indicated in the rear part of Fig. 8.1. It occurs in an analogous manner to that of the learning step for the synaptic strengths $w_{\mathbf{r}}^{(in)}$, possibly with different learning step widths ϵ' and a different interaction function $h'_{\mathbf{rs}}$ as demanded by each situation. (This symmetric treatment of input values

$\mathbf{v}$ and output values u can be mathematically interpreted in such a way that now the algorithm creates a topology-conserving mapping of a subset Γ of the product space $V \otimes U$ onto A. Here the subset Γ is just the set of the input-output values $(\mathbf{v}, u)$ provided by the teacher; *i.e.*, it is the graph of the input-output relation that correctly describes the control law.)
The proposed process creates, in the course of the learning phase, a look-up table for the function Φ (see ((8.1)). The particular advantage of this process lies in the high adaptability of the table structure. The correspondence between table entries and input values is not rigidly specified from the outset, but instead develops in the course of the learning phase. This occurs in such a way that the table entries become distributed $(\mathbf{w}_{\mathbf{r}}^{(in)}, w_{\mathbf{r}}^{(out)})$ according to the required density of the control actions in the space $V \otimes U$. Regions in $V \otimes U$, from which control actions are frequently needed, are assigned a correspondingly higher number of table entries, resulting in a higher resolution of the input-output relation in these areas. Rarely or never used table entries are "redevoted." This facilitates a very economic use of the available storage space.
Due to the topology-conserving character of the assignment between value pairs and storage space, neighboring memory locations in the lattice A are usually assigned similar value pairs. In the case of a continuous relationship between input and output values, adjacent memory locations must, therefore, learn similar output values $w_{\mathbf{r}}^{(out)}$. Spreading the effect of learning steps for the output values $w_{\mathbf{r}}^{(out)}$ by virtue of the interaction function $h'_{\mathbf{rs}}$ into the vicinity of each selected storage location $\mathbf{r}$ represents a rudimentary form of generalization which accelerates the course of learning. If the function $h_{\mathbf{rs}}$ is given a long range at the beginning of the learning phase, then a large subset of all storage locations participates in each learning step, even if the learning steps for most of them are only approximately "correct." By this, a complete table, representing a good approximation to the correct input-output relation, can already develop after significantly fewer learning steps than there are storage locations. By gradually reducing the range of the function $h_{\mathbf{rs}}$ in the course of the learning phase the participation of storage locations at each learning step becomes increasingly more selective, so that eventually even fine details of the input-output relation can be learned.
In many cases, however, there is no teacher available, and the correct control actions must be found by the learning algorithm itself (*unsupervised learning*). The only source of information available in this case is a "reward function," which specifies at a given moment how well the control has mas-

tered the given task. To execute the above algorithm the system must now create, at each learning step, the component u that in the previous case was delivered by the teacher. In the absence of any further information this requires a stochastic search in the space U of the available values, with the aim to maximize the "reward" received at each step.

8.3 The "Pole-Balancing Problem"

To demonstrate the learning algorithm, defined in the previous section by steps 1–4, we consider as a prototypical problem the task of balancing a vertical pole in a gravitational field by appropriate movements of the base of the pole. This is a favorite problem for the investigation of learning algorithms, and it has already been studied in connection with previous neural network approaches (Barto and Sutton 1983). As in the present case, these studies were directed at learning the connection between movement states of the pole and the required control forces in the form of a look-up table. In contrast to the approach presented here, the goal of the former studies was not to obtain an optimal organization of this table by an adaptive distribution of its entries over the state space. Instead, the connection between the table entries and states was initially given in a rigid way, and the focus of interest was the investigation of learning rules which gradually set the table entries using as the only criterion of success the absence of a signal indicating that the pole has fallen over.

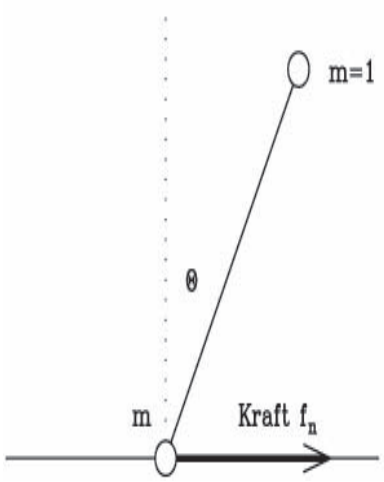

Abb. 8.2: Model of the pole in the simulation. The bottom end of the pole can slide without friction along a horizontal line. By an appropriate force f the pole is to be balanced in the position $\theta = 0^0$. Units are shown such that the mass at the top end of the pole, the pole length, and the acceleration due to gravity all have values of unity. The mass at the bottom end is m. The shaft is assumed to be massless.

The motion of the pole is imitated by a computer simulation. A massless rod of unit length serves as the pole, with point masses of values m and unity at its bottom and top end, respectively. The motion of the pole is restricted to a vertical plane, and the bottom of the pole is confined to slide along the x-axis. The pole and its two degrees of freedom are presented in Fig. 8.2. For a gravitational field directed downward with unit strength, the equation of motion of the pole is

$$(m + \sin^2\theta)\ddot{\theta} + \frac{1}{2}\dot{\theta}^2 \sin(2\theta) - (m+1)\sin\theta = -f\cos\theta. \qquad (8.2)$$

Here, θ is the pole angle measured clockwise against the vertical, f is the horizontal force acting at the bottom end. The motion of the pole is simulated by the Runge-Kutta method using a time step of 0.1 in the units of Eq. (8.2).

8.4 Supervised Pole-Balancing

The first simulation demonstrates supervised learning. The Adoptation of a new movement often requires an adaptation period composed of partial, goal-oriented movements which must come together before the total course can be executed fluently and automatically. This indicates that in this phase a higher brain region might play the role of a teacher for a subordinate neural structure until the latter can execute the movement independently and without requiring conscious attention.

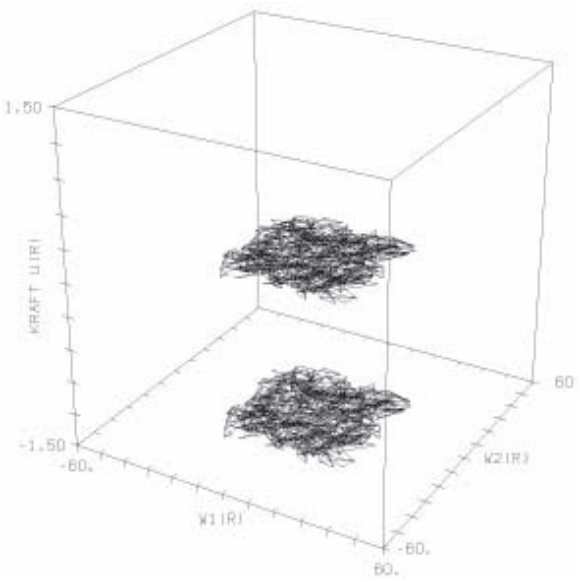

Abb. 8.3: Initial state of the simulation. For each lattice point $\mathbf{r}$ the stored output force $w_{\mathbf{r}}^{(out)}$ is recorded along the vertical axis above the point $(w_{\mathbf{r}1}^{(in)}, w_{\mathbf{r}2}^{(in)})$ of the two-dimensional space V of input signals. $(w_{\mathbf{r}1}^{(in)}, w_{\mathbf{r}2}^{(in)})$ represents a pair of successive inclinations of the pole, at the occurrence of which the lattice delivers $w_{\mathbf{r}}{}^{(out)}$ as a control force. In the beginning all forces are chosen to be zero, and to each lattice site $\mathbf{r}$ a point, randomly selected from the square $[-30°, 30°] \times [-30°, 30°] \subset V$, is assigned.

In the present simulation the neural network was chosen to be a lattice A of 25×25 formal neurons. Each formal neuron $\mathbf{r}$ is characterized by a two-dimensional vector $\mathbf{w}_{\mathbf{r}}^{(in)}$ and an output value $w_{\mathbf{r}}^{(out)}$. The vectors $\mathbf{w}_{\mathbf{r}}^{(in)}$ determine the correspondence between neurons and motion states of the pole,

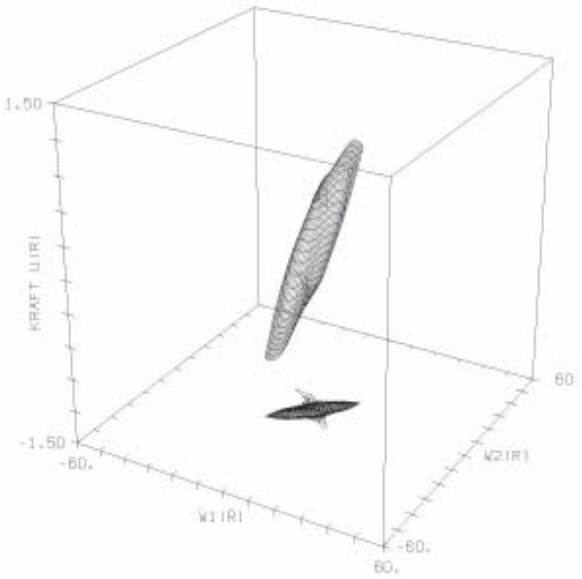

Abb. 8.4: After 200 learning steps, most neurons have become asssociated with input values that are concentrated in a narrow region along the diagonal $w_1^{(in)} = w_2^{(in)}$. In the vicinity of $(0^0, 0^0)$ the learned output values already correspond quite well to the actions of the teacher.

and the output values $w_{\mathbf{r}}^{(out)}$ are the forces to be applied. In this example the teacher has to deliver, for each motion state $(\theta, \dot{\theta})$, a force $f^L(\theta, \dot{\theta})$ that ensures the intended pole balance. In the simulation this force was chosen as

$$f^L(\theta, \dot{\theta}) = 5 \sin \theta + \dot{\theta}. \tag{8.3}$$

The pole was controlled by choosing a force f_n at equidistant time instants $t_n = n\Delta t$, $\Delta t = 0.3$ and applying this force at the bottom of the pole until time t_{n+1}. At each time instant t_n the learning steps 1-4 were executed for

$$(\mathbf{v}, u) = (v_1, v_2, u) := \Big(\theta(t_n), \theta(t_{n-1}), f^L(\theta(t_n), \dot{\theta}(t_n))\Big). \tag{8.4}$$

The force f_n for the next time step was determined according to

$$f_n = \alpha(t_n) w_{\phi_{\mathbf{w}}(\mathbf{v})}^{(out)} + (1 - \alpha(t_n)) f^L(\theta(t_n), \dot{\theta}(t_n)). \tag{8.5}$$

Here $w_{\phi_{\mathbf{w}}(\mathbf{v})}^{(out)}$ is the force proposed by A at the time t_n, and $\alpha(t)$ is a function that increases gradually from $\alpha = 0$ to $\alpha = 1$. Consequently, the control is

initially done exclusively by the teacher, but it is gradually taken over by A until eventually at $\alpha = 1$, the neural net has completely replaced the teacher. At the end of the learning phase, the map has learned to imitate the behavior of the teacher. One may object that by this procedure no new information has been gained because the solution of the balancing problem was already provided by the teacher. Nonetheless, this objection denies that an essential result of the learning process lies in the coding of the information in a new way, namely, as an optimized look-up table. The latter obviates recalculation of the correct control actions previously required at each time step; now, only a simple table look-up is required. In all cases where a recalculation is more wasteful or slower than a simple table look-up, a gain in efficiency will result.

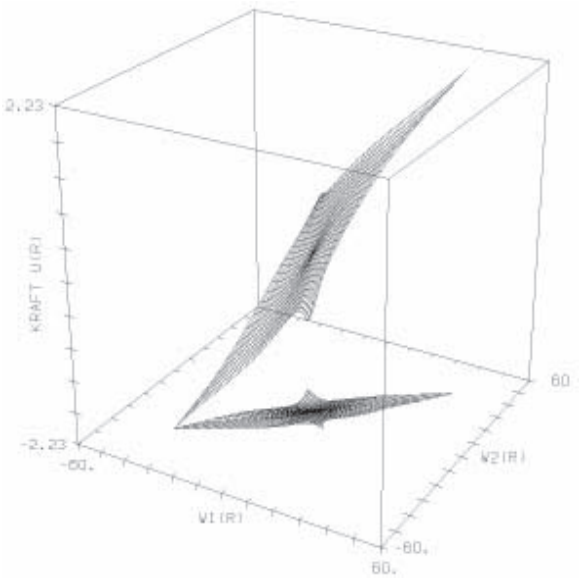

Abb. 8.5: The stage reached after 1000 learning steps at the end of the simulation. All neurons have attached themselves to the part of the graph which is most important for the pole balancing, in accordance with the input-output relation delivered by the teacher. Since ${w_1}^{(in)}$, ${w_2}^{(in)}$ stands for sequential pole orientations, points with a small difference ${w_1}^{(in)} - {w_2}^{(in)}$ correspond to most of the pole states. The region of the represented differences is broader in the vicinity of the point $(0^0, 0^0)$ since near this point higher velocities occur on average than at the turning points. Also, the resolution is particularly high near $(0^0, 0^0)$ because forces are requested most frequently for these pole states.

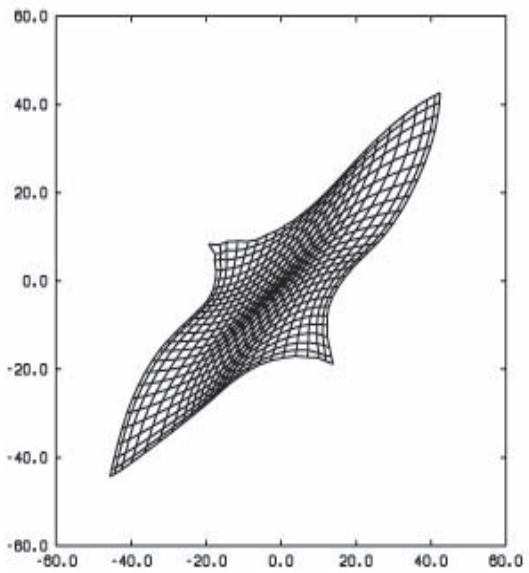

Abb. 8.6: This graph depicts, in the ${w_1}^{(in)} - {w_2}^{(in)}$-plane, the mapping between lattice locations and those angle pairs ${w_1}^{(in)}$,${w_2}^{(in)}$ which are important for the balancing process.

The other data used in our simulations were $\sigma(n) = 6 \cdot (0.5)^{n/n_f}$, $h_{\mathbf{rs}} = h'_{\mathbf{rs}} = \exp[-\|\mathbf{r}-\mathbf{s}\|^2/\sigma(n)^2]$, $\epsilon(n) = \epsilon'(n) = 0.5 \cdot (0.04)^{n/n_f}$, $n_f = 1000$ and $m = 1$.
The learning phase consisted of a sequence of trials. At the beginning of each trial the pole was released from rest at an initial, randomly chosen angle $\theta \in [-30°, 30°]$. The pole was then balanced by the forces f_n either until the pole fell over, (Criterion: $|\theta(t_n)| > 60°$), or until 100 time steps had passed. Subsequently, the simulation was continued with a new trial until a total of $n_f = 1000$ time steps were completed.
Figures 8.3–8.5 show the development of the mapping Φ in the course of the simulation. Φ is displayed as a mesh surface in the $w_1^{(in)}$-$w_2^{(in)}$-$w^{(out)}$-space. At the beginning random pair values $w_1^{(in)}$-$w_2^{(in)}$ from the subset $[-30^0, 30^0] \times [-30^0, 30^0]$, together with a force $w_{\mathbf{r}}^{(out)} = 0$, were assigned to each neuron $\mathbf{r}$ (Fig. 8.3).
After 100 time steps most of the neurons have become associated with angle pairs in the vicinity of the diagonal $w_1^{(in)} = w_2^{(in)}$ and have roughly learned the corresponding forces (Fig. 8.4). Finally, after 1000 time steps, the result depicted in Fig. 8.5 is obtained. Now all neurons are associated with a

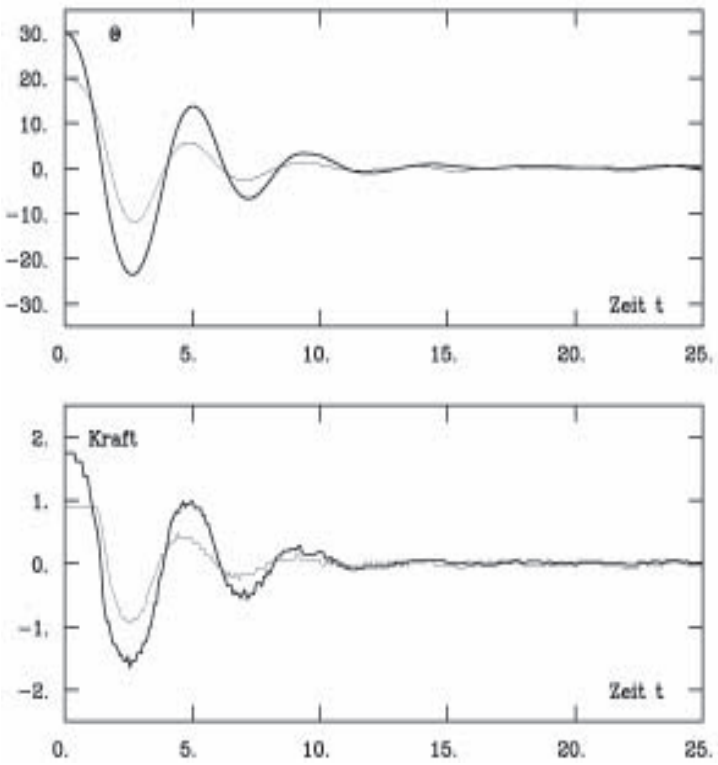

Abb. 8.7: Balancing performance after 200 (light curves) and after 1000 learning steps (heavy curves). The curves in the top diagram show the time course of the pole angle θ after 200 learning steps. The pole was released from angles of 20^0 and 30^0, after 200 and 1000 learning steps, respectively. The bottom diagram displays the time course of the force that was applied by A.

narrow region near the diagonal $w_1^{(in)} = w_2^{(in)}$. This region represents, under the chosen coding of each state by two sequential angles, those states of the pole that are particularly important for the balancing problem (Fig. 8.6). The representation by neurons is especially dense in the vicinity of $(0°, 0°)$, because for these values forces are most frequently requested during the balancing. Figure 8.7 displays the learned balancing behavior after 200 steps (light curves) and after 1000 steps (heavy curves). The top diagram depicts the time course of the pole angle $\theta(t)$, which is monitored after release of the pole from rest and using the control furnished by A in the absence of

a teacher, *i.e.*, for $\alpha = 1$. After 200 learning steps an initial angle of 20° away from vertical is handled well; at the end of the simulation even a trial starting with $\theta(0) = 30°$ succeeds. The bottom diagram shows the response of A for both cases.

8.5 Unsupervised Pole-Balancing

In the following, second simulation, the balancing of the pole is learned without a teacher by means of a "reward function" *(unsupervised learning)*. In contrast to the teacher the reward function no longer directly specifies a suitable force, but only indicates "how well" the resulting state complies with the given task. An appropriate reward function for the balancing task is, *e.g.*,

$$R(\theta) = -\theta^2. \tag{8.6}$$

This function "rewards" vertical ($\theta \approx 0°$) orientations of the pole. As before, the lattice site $\mathbf{s}$ is selected by the pole motion at each time step. Since the teacher is no longer present, a force f_n for a particular step is now determined by the selected lattice site $\mathbf{s}$ only. The learning goal is to find a force which yields a maximal increase of the reward function R towards the end of each time step. In the absence of any further information this can be accomplished in the most general way by a stochastic search process.[1]
Every time that $\mathbf{s}$ is chosen (step 2), instead of the most recently found output value $w_{\mathbf{s}}^{(out)}$ for f_n, the modified value

$$f_n := w_{\mathbf{s}}^{(out)} + a_{\mathbf{s}} \cdot \eta \tag{8.7}$$

is used. Here η is a Gaussian random variable with zero mean and unit variance, and $a_{\mathbf{s}} > 0$ is a new parameter that determines the mean value of the search step for lattice location $\mathbf{s}$. The adaptation steps 3 and 4 are executed with $u = f_n$ only when, at the end of a time step, the increase ΔR of the reward function that has been caused by f_n exceeds the averaged

[1] More efficient searching methods can be applied when certain types of additional information are available. For example, one could use the differentiability of R and could replace the stochastic search method by the gradient descent method, which avoids searching steps in the "wrong" direction. Because we are less interested in variations of the detailed design for each application and would rather focus on a representation of the method's general structure, we will stay with the generally applicable stochastic search method.

increase $b_{\mathbf{s}}$ gained so far at lattice site $\mathbf{s}$. The consequence of this is that A learns only from "actions" which lead to an improved performance, and thus A continually improves itself.
The mean increase of the reward function $b_{\mathbf{s}}$, necessary in addition to $a_{\mathbf{s}}$, has to be updated after each selection of $\mathbf{s}$ by using the actual increase ΔR that has been reached. This can be most simply accomplished by the instruction

$$b_{\mathbf{s}}^{new} = b_{\mathbf{s}}^{old} + \gamma(\Delta R - b_{\mathbf{s}}^{old}). \tag{8.8}$$

The effect of the latter procedure is a low-pass filtering and a corresponding "smoothing" of the most recent updates of the reward function with a time constant given by γ^{-1}.
At the beginning the average search step width $a_{\mathbf{s}}$ should be sufficiently large for each lattice location in order to rapidly find an approximately correct force. Each time an $\mathbf{s}$ is selected that has the opportunity to act, $a_{\mathbf{s}}$ is diminished. Therefore, the number of search steps for each lattice site is reduced as more experience is gained, and the stored output value can gradually converge. Because the neighboring lattice sites $\mathbf{r} \neq \mathbf{s}$ are also involved in every adaption step of $\mathbf{s}$ (steps 3 and 4), the corresponding $a_{\mathbf{r}}$ are reduced in the same way as the $a_{\mathbf{s}}$. In analogy with steps 3 and 4, this can be achieved by including the following additional step.

5. Adaptation rule for the search step widths:

$$a_{\mathbf{r}}^{\text{new}} \quad = \quad a_{\mathbf{r}}^{\text{old}} + \epsilon'' h''_{\mathbf{rs}}(a - a_{\mathbf{r}}^{\text{old}}).$$

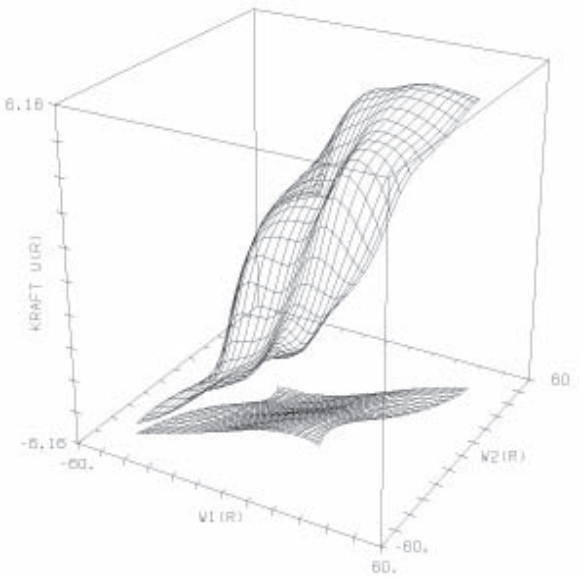

Abb. 8.8: The association between pole motion states and control forces as it has been learned after 3000 learning steps without a teacher by using the reward function R (displayed as in Fig. 8.3-8.5). Again the states along the diagonal of the $w_1^{(in)}$, $w_2^{(in)}$ plane, i.e., with a small difference $w_1^{(in)} - w_2^{(in)}$, are represented particularly well. For deviations from the vertical position the learned forces increase more strongly than in the simulation presented in Fig. 8.5 and lead to very rapid corrective adjustments of deviations from vertical.

Here a is equal for all steps and defines the threshold towards which the search step widths should converge over long periods of time. If one intends for $w_{\mathbf{r}}^{(out)}$ to converge, one chooses $a = 0$. If one wishes a residual plasticity to remain for the capability of later readaptation under slow changes, one assigns a small corresponding positive value to a. The additional parameters ϵ'' and $h''_{\mathbf{rs}}$ (step 5) are varied analogously to ϵ and $h_{\mathbf{rs}}$, and ϵ' and $h'_{\mathbf{rs}}$, respectively.

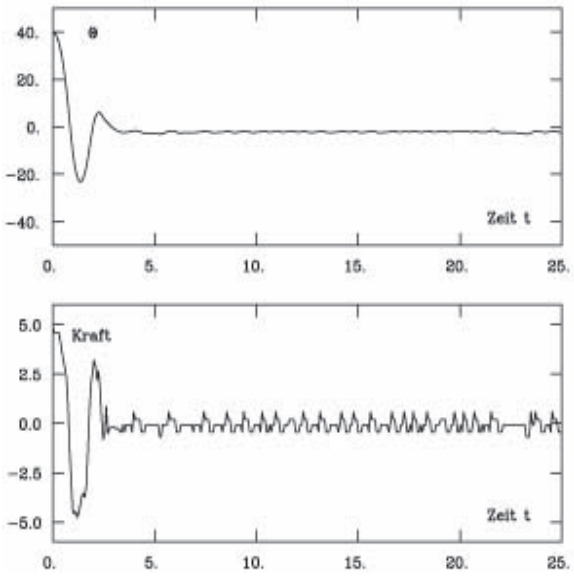

Abb. 8.9: Pole-balance after 3000 learning steps using the reward function R. The top diagram shows how the pole angle θ changes in time after the pole has been released from a rest position $\theta = 40°$. The bottom diagram shows the corresponding control force that was applied.

For the simulation we chose $h''_{\mathbf{rs}} = h'_{\mathbf{rs}} = h_{\mathbf{rs}}$ where $h_{\mathbf{rs}}$ has the values of the simulation in Section 8.4. $\sigma(n)$ and $\epsilon(n)$ were chosen as in Section 8.4 but with $n_f = 3000$. Other quantities in the simulation were $\epsilon' = 0.2$, $\epsilon'' = 0.005$, $a = 0$, $\gamma = 0.05$. At the beginning of the simulation all averaged search step widths were set to $a_{\mathbf{r}} = 1$. The initial estimators $b_{\mathbf{r}}$ for the changes of the reward function were set to zero.
Starting with the same initial state as for supervised learning (Fig. 8.3) and with the parameters just given, the connection between pole motion states and control forces $w^{(out)}$ evolved as depicted in Fig. 8.8 in the course of 3000 learning steps. Again the preponderance of values along the $w_1^{(in)} - w_2^{(in)}$ main diagonal can be seen, particularly in the vicinity of $(0°, 0°)$. The resulting dependence of the control force on the pole positions evolves qualitatively as before, but has a steeper slope at deviations from $(0°, 0°)$, indicating a very rapid correction of any deviations from vertical. This is demonstrated in Fig. 8.9 where one can observe the time variation of the pole angle θ (top diagram) and the learned forces f (bottom diagram) that occur after the

pole has been released from an initial angle of 40° from vertical.
Problems similar to the pole-balancing problem also arise in connection with upright walking, rendering the pole-balancing solution interesting for biological organisms as well as in robotics. Of presumably even higher interest, as far as applications are concerned, is the control of arm motions, especially under visual supervision. Chapters 10 and 11 are devoted to these issues. In the next chapter, however, we will first consider an issue with a biological background, namely the control of rapid eye movements that serve to center a visual object on the retina and, thus, solve the task of "visual grasping." This latter task is taken up in Chapter 9 which then leads naturally into issues in robotics.