Differential Equations and Dynamical Systems,
Vol.9, Nos. 3&4, July & October 2001, pp. 259–319

Nonlinear dynamics and chaos in information processing neural networks

A.B. Potapov*and M.K. Ali†
Department of Physics, The University of Lethbridge,
Alberta, Canada T1K 3M4

Abstract

We consider a number of possible roles of complex dynamics and chaos in information processing by neural networks. First, we review the working principles of some well-known neural networks, and then discuss a number of approaches to utilization of chaos in neural networks. Our main goal is to present a novel view of the problem of chaos in information processing. We demonstrate that chaos emerges naturally in controls when a neural network forms a controlling part of a more complex system. We show that such neural networks can enhance efficiency by using chaos for explorations in a method known as Reinforcement Learning. A discussion on Hamiltonian neural networks is also included.

1 Introduction

Artificial neural networks (ANNs) are widely used now in many engineering applications and research problems [17, 18, 45, 46, 59, 58, 79, 95, 97]. Since ANNs are efficient tools for information processing, there is an ongoing quest for improving their performance, widening the areas of applications and finding new working principles. About ten tears ago, attempts were made to enhance the performance of ANNs on the basis of their complex or chaotic temporal behaviors (see e.g. [93, 34] for the review). The question is: can chaos be useful for information processing? There are arguments both in favor and against the question of usefulness of chaos. On one hand, activities of the brain demonstrate complex and possibly chaotic temporal behaviors, that suggests that maybe the brain uses chaos for sustaining life. On the other hand, neural networks that are currently in use for practical purposes are designed to be nonchaotic on the presumption that chaos is not needed for information processing (though chaotic information processing systems exist, e.g., [5]).

The purpose of this paper is to reflect on the underlying principles of operation of existing neural networks from the point view of nonlinear dynamics, to discuss a number of attempts to endow networks with chaotic behavior, and to present a new perspective of the role of chaos in neural networks. If we consider a neural network as an element of a larger system interacting with the world, then dynamical chaos can emerge in rather simple models. A number of such models are known, for example, in artificial intelligence. Moreover, systems interacting with their surroundings need a source of 'initiatives' to pursue exploration and learning from experience. Dynamical chaos *can* serve as a source of such initiatives. In this work, we also discuss Hamiltonian neural networks that have received little attention so far in information processing. Hamiltonian neural networks have the advantage that their quantum analogs can be studied.

*e-mail: alexei.potapov@uleth.ca
†e-mail: ali@uleth.ca

The paper is organized as follows. First, in Section 2, we discuss definitions and various viewpoints of neural networks. In Section 3, we consider working principles of some existing neural networks to explain why, in the opinions of users of these networks, complex dynamics and chaos are not necessary. In Section 4, we give reasons why chaos *may* be useful. In Sections 5, 6, and 7 we discuss different approaches to the problem of chaos and complex behavior in neural networks.

2 What is an artificial neural network?

In spite of the growing number of publications in the field, there is no consensus on the precise definition of ANNs. The reason for this is that there are too many types of them. Sometimes a more general term called "connectionism" is used for ANNs. The term connectionism means a methodology of making a complex system by a combination of connected elements that are similar or identical [29]. The basic nonlinear elements of an ANN are called formal neurons. Typically, a formal neuron receives a number of input signals x_i, sums them up, performs a nonlinear transformation of this sum $y = f(\sum x_i)$ and sends y as its output to other neurons or devices. The function f is called the "activation function". Quite often, f is a "sigmoid" function of the form $f(x) = 1/(1 + e^{-ax})$ or $f(x) = \tanh ax$. In the limiting case $a \to \infty$, f becomes a threshold function.

The connections among neurons are characterized by the weights w_{ij}. Some or all of the neurons may receive external input signals X_k with weights w'_{ik}. Therefore, a typical operation performed by a neuron in the network is a transformation $f(\sum_j w_{ij}x_j + \sum_k w'_{ik}X_k)$. After all the necessary internal computations are carried out, the output of the network consists of the states of some or all of the neurons. Therefore, an ANN has the following general features:

- It is an information processing tool that receives an input signal X (usually a multicomponent vector) and generates the output $Y = F(X)$ (Y may be a vector).
- It consists of a number of similar neurons that usually operate in parallel like the neurons of the real brain.
- It is capable of learning which is normally understood as adjusting the parameters w_{ij} to achieve certain desired properties of the mapping F.

ANNs with these general features are studied from different viewpoints of which the following four are the most important ones.

2.1 The viewpoint of statistics and data analysis

One of the problems that often arise in practice involves approximation of unknown functions from experimentally observed data points. That is, given the set of pairs of points $\{X_i, Y_i = \Phi(X_i)\}$, it is required to find an approximation $F(X)$ of $\Phi(X)$. Besides usual curve fitting, an astonishingly large number of problems such as pattern recognition, classification, prediction, and control can be reduced to this form. For example, X_i may be the set of sensor data, representing situations around a car, and Y_i may be the actions of an experienced human operator for safe driving. Good approximation $F(X)$ gives a control system the ability to drive a car.

Classical approximation theory is very well developed. However, it usually deals with one dimensional functions. When we need to approximate an unknown function of, say ten variables, classical methods such as polynomial approximations, are of little help. Approximations by compositions of sigmoid functions are more effective.

So, the general framework of the approach is as follows. We choose a certain form of the approximating function $F(X, w)$, and adjust w to minimize the error $\sum |Y_i - F(X, w)|^2$. The process of minimization will be the learning of the network. The process of learning essentially depends on the specific implementation of the network.

At the first glance, this seems very simple. But if one tries to build a network, one immediately faces a number of questions regarding the number of neurons to be taken, organizations of the connections to be followed, type of minimization to be used, and so on. The modern theory of neural networks can answer some of them on a rigorous basis, and others on the basis of a vast experience. The rigorous part is based on (a) Kolmogorov theorem on functions approximation by sigmoid-type functions, (b) a number of general approximation theorems which state that multilayer perceptrons (see below) are universal approximators, and (c) the methods of statistics. There is a very good description of statistical view of neural networks, e.g., in [16, 25, 81].

2.2 Computer science viewpoint

This direction is related mainly with logic, circuitry and computation capabilities of neural networks. The mapping $X \to Y$, provided by a neural network, can be considered as a way of performing computation. In this computational aspect, the main differences between neural networks and usual computers are that (1) ANNs can work in a highly parallel way, (2) instead of programming, ANNs can learn from examples, and (3) ANNs can recognize patterns even when only partial informations about the patterns are available.

Nonetheless, there is no gap in the history of neural networks and computers. The beginning of the history of ANNs often refers to the famous 1943 paper of McCulloch and Pitts [63]. This paper showed that simple networks of threshold elements (sometimes they are called McCulloch-Pitts neurons) can work as logical gates, and hence they can form a large network that can work as a universal computer.

Another interesting aspect related mainly with artificial intelligence is learning of a different type. For data analysis, if the output for a given input is known, the difference between the actual and desired outputs can be used to update the weights (supervised learning). In machine learning, for example in the problem of robot navigation, it is necessary to learn how to take the correct action at the current 'state' (the set of sensory data) when the answer is unknown. Only after a long sequence of trials and errors, it is possible to learn about the appropriate actions that should be taken. Such a learning is called *reinforcement learning* and it is based on reinforcement of reward (for success) and punishment (for failure). The first attempt to build a reinforcement learning neural network was made as early as 1954, but significant progress in developing the method of reinforcement learning was achieved during 1980-1990 [83]. An important feature of this type of learning involves integration of a neural network into a larger system, and the learning proceeds as an interaction of a learning agent with its environment. We shall return to this matter later in Sections 5 and 6.

An excellent account of the computer science viewpoint of neural networks is presented, e.g., in [7].

2.3 The biological viewpoint

Considerations of computing neural circuitry and limitations of the capabilities of computing devices inevitably send us back to the beginning of the theory of neural networks, that is, to biological analogies, cognitive science and theories of the brain (e.g., [78]). Or in other words, we have the biological viewpoint of neural networks. An excellent description of this viewpoint can be found, for example, in Freeman's review [34, 33].

Studies of the brain in psychology and physiology have a much longer history than that of neural networks. For biology, the technique of artificial neural networks is only one of many possible approaches. The main task of neural networks in the biological framework is to model and explain the results of physiological experiments, or to deepen our understanding of how the brain works. Moreover, such modeling helps test biological hypotheses that can not be verified any other way [34].

The whole field of neural network studies has been inspired by discoveries in neuroscience. The hope of modeling the principal brain functions was very high in 1950s. This was so because originally the complexity of the brain was underestimated. The first estimates gave the number of neurons as $\sim 10^3$ [34]. Later it was estimated that the brain contains about 10^9 neurons, and each of them might have up to 10^5 connections. So, the task of modeling the functions of the whole brain has been postponed to indefinite future.

The work on modeling biological structures began with a single neuron or its parts. It should be noted that a biological neuron is much more complex than a formal neuron of an ANN. For example, the model of electric signal propagation along axon ("output" structure of a neuron) includes two partial differential equations while a simple description of the neuron dynamics requires delay-differential equations (see e.g., references in [34]). On the next level of description, there were models of groups of neurons or small brain structures [34]. In this case the main task is to construct a network which would reproduce the features observed in real physiological experiments. Such models may be totally different compared to networks with approximation purposes. Also they are substantially more complex.

Important aspects of mathematical biology of neural networks include classification of various models, search for the most general model, and establishing the principles of their construction. A significant progress in this field has been made with the help of methods of bifurcation theory [53].

Numerous physiological experiments demonstrate the complexity of the behavior of the brain: most probably, it is chaotic [34, 8, 9, 38, 41]. But why? What are the advantages of chaos in information processing? Should data processing systems possess their internal dynamics and what would be the principles of their work? Such questions are akin to nonlinear dynamics.

2.4 The viewpoint of nonlinear dynamics

From the framework of nonlinear dynamics, the important issues are the general properties of system's dynamics. Nonlinear dynamics may be of little help for many specific details, but it does provide a basis for general view of various neural networks.

This point of view helps to classify all neural networks into two big categories: "functions" and "dynamical systems". Our main interest is in the latter category, although we will say a few words about the former one. Dynamical systems can be *conservative* or *dissipative.* All the dynamical neural networks that we know of are dissipative. Dissipation plays an important role in pattern recognition. It eliminates noisy or irrelevant features from the input data and transforms a distorted pattern into a "correct" one. That is, the process of pattern recognition is considered as convergence to attractors of dynamical systems. Conservative dynamical systems do not have attractors and, therefore, they can not implement this scheme of recognition. Nonetheless, in Sect. 7 we will consider an example of a conservative Hamiltonian dynamical system for pattern recognition. Such systems are of interest from the point of view of quantum information processing.

Attractors of dissipative dynamical systems represent the output (Y) of a neural network. The input X can be fed into it in two major ways: as initial data or as parameters of the dynamical system. Combinations of these ways are possible, but we do not know of any such networks. Therefore, nonlinear dynamics provides the following classification of all existing neural networks according to the method of building the mapping $X \to Y$.

1. *Direct functional representation*. The resulting output is obtained by a number of explicit transformations of the input. The stages of transformation involve the neurons and weights. Often such neural networks are simply multidimensional functions depending on parameters:

$Y = F(X, w)$, and the functional form of F is known (feed-forward nets). In principle, there can be loops in the function definition. For example, a part of the output can be fed back as input to generate an implicit function such as $Y = F(X, Y, w)$ or some other complex form (recurrent nets). The well known examples of functional networks are (a) multilayer perceptrons and (b) radial basis function networks [16, 45].

2. Dynamical system mapping I: X = initial state, Y = final state (attractor). The dynamical systems may be described by (i) differential equations (DE) in which space and time are continuous, (ii) mappings in which space is continuous and time is discrete or (iii) automata in which space and time are discrete. The final state Y of the mapping $Y = F(X)$ does not depend on X if X is in the basin of attraction of a given attractor. Therefore, this kind of networks may be interesting only when there are *multiple attractors*. For this reason we shall call this approach as one of *multiple-attractor network*. To define the mapping F properly, that is to teach the network, one should be able to place the dynamical system's attractors to desired places and to avoid spurious attractors. The best known types are Hopfield networks, brain state in a box, Haken's synergetic computer [49, 45, 27, 42].

3. Dynamical system mapping II: X =parameters of dynamical system, Y = its attractor. Again the dynamical system may be of different types, but in this case the *attractor is unique* subject to the input. Multiple attractors may spoil the performance. The role of the input is to manipulate this single attractor in a desired way. For this reason, we shall call this approach as one of *attractor manipulation* [45]. The well-known types of such networks are Grossberg's adaptive resonance maps, Hopfield-Tank combinatorial optimization network, and Freeman's models of olfactory system [18, 99, 51].

4. Hamiltonian neural networks. These networks have not received much attention in the context of information processing by neural networks.

From the point of view of nonlinear dynamics, the most interesting features of ANNs are their dynamical properties like complexity, stability, and predictability. In particular, a very interesting question is, can dynamical chaos improve the performance of neural networks or can it play an essential role in the operations of ANNs? Our attempt to address this question is the main purpose of this article. But, let us first consider in more detail the principles of operation of some important types of networks.

3 Basic types of ANN

In this section we briefly review several basic types of neural networks from the nonlinear dynamics viewpoint. Our goal is not to present an elaborate review of ANNs, and hence the choice of models and their discussions are limited.

3.1 Functional networks

Let us consider a simple example of approximating a function $Y = f(X)$ using its values $Y_i = F(X_i)$ at a number of points $X_i \in [X_{01}, X_{02}]$ with the sigmoid-type function $\tanh(x)$. Since hyperbolic tangent has values only in $[-1, 1]$, first we need to make sure that the values of $F(X_i)$ on $[X_{01}, X_{02}]$ also belong to this interval. This can be achieved by a linear transformation (data preprocessing) of Y_i, and we assume that already $Y_i \in [-1, 1]$. Then one has to choose the form of the approximating function $Y = F(X, \mathbf{w})$, where $\mathbf{w}$ is the set of parameters. For example, if the original dependence is close to being linear, one can try $Y = \tanh(w_1 X + w_2)$. Then one has to "teach" this approximation by examples, that is to find the best values of $\mathbf{w}$ from the known pairs X_i, Y_i. The parameters can be adjusted by minimizing the error function $E(\mathbf{w}) = \sum_i (Y_i - F(X_i, \mathbf{w}))^2$.

How does it relate to dynamics? The calculation of the result does not contain any dynamics at all. The process of minimization can take the form of a dynamical system, e.g., $\dot{\mathbf{w}} = -\nabla E(\mathbf{w})$, but any complexity in this process is considered as a shortcoming.

Nonetheless, dynamics comes into play as soon as one tries to use *implicit* functions, for example, $Y = \tanh(w_1 X + w_2 Y + w_3)$. Such implicit representations are more versatile, but they require a procedure to find Y. This is done with the help of the dynamics, e.g. $\dot{Y} = -Y + \tanh(w_1 X + w_2 Y + w_3)$. In this single equation no complex dynamics can exist, but in its multidimensional analogs complex dynamics may be possible. But still it is considered as a serious disadvantage since the main goal — obtaining a single number Y — remains unaccomplished.

Multilayer perceptrons generalize the above example. They have a number of "sigmoid" neurons with states denoted by x_i. Neuron i receives signals from other neurons through the connection weights w_{ij}, and it also receives the input signal X_i from outside the network. The computations performed by the network can be expressed as

$$x_i = f\left(\sum_{j=1}^{N} w_{ij} x_j\right) + X_i. \tag{1}$$

The function f is sometimes called activation function, and quite often it is a sigmoid function.

A generalization of an explicit function $Y = F(X, \mathbf{w})$ is a class of *feed-forward* networks for which $w_{ij} = 0$ for $j \geq i$. For such a network, the first neuron receives signal only from the outside, the second neuron receives signals from outside and possibly from the first one and so on. For the training of such networks, there is a method of error minimization called error backpropagation. In order to update the weights, one has to calculate the auxiliary error terms e_i for each neuron. It can be shown that e_i satisfy a linear system of equations with upper triangular matrix. Therefore, x_i are calculated from 1 to N, while e_i are calculated from N to 1.

Generalizations of implicit functions $Y = F(X, Y, \mathbf{w})$ are *recurrent* neural networks. For them there are no restrictions on the matrix w_{ij}. For solving Eq.(1), one has to apply some iterative scheme, that is, to transform the nonlinear system of equations into a dynamical system. For example, instead of $x = f(wx) + X$, consider the ODE system $\dot{x} = -x + f(wx) + X$ the fixed points of which give solutions of the nonlinear system. The learning scheme involving calculation of the error terms e_i can be generalized for such systems [74, 45], but the equations for e_i do not have such a simple form.

We give here a brief description of the backpropagation learning for the general form of perceptron (1). Let there be the set of M training examples, including the values of input and desired output: $(X^{(k)}, Y^{(k)})$, $k = 1, \dots, M$. Let us also denote by $x_i^{(k)}$ the value of i-th neuron when the input $X^{(k)}$ has been given and the solution of (1) has been obtained. For output neurons we define the error signal $E_i^{(k)} = x_i^{(k)} - Y_i^{(k)}$, for other non-output neurons $E_k = 0$. Then the total approximation error for the given set of parameters w_{ij} is

$$S = \sum_{k=1}^{M} \sum_{p=1}^{N} \left(E_p^{(k)}\right)^2.$$

The usual way of updating weights is the gradient descent minimization of S, that is

$$\delta w_{\alpha\beta} = -\epsilon \frac{\partial S}{\partial w_{\alpha\beta}} = -\epsilon \sum_{k=1}^{M} \sum_{p=1}^{N} E_p^{(k)} \partial x_{p,\alpha\beta}^{(k)}, \tag{2}$$

here we denote $\partial x^{(k)}_{p,\alpha\beta} = \frac{\partial x^{(k)}_p}{\partial w_{\alpha\beta}}$ for the sake of brevity. The values of $\partial x^{(k)}_{p,\alpha\beta}$ can be obtained by differentiating (1) with respect to $w_{\alpha\beta}$,

$$\partial x^{(k)}_{i,\alpha\beta} = f'_i \cdot \left(\delta_{\alpha i} x^{(k)}_\beta + \sum_{j=1}^{N} w_{ij} \partial x^{(k)}_{j,\alpha\beta} \right),$$

here f' is the derivative of f by its argument. If we denote $L_{ij} = \delta_{ij} - f'_i w_{ij}$, then we can express $\partial x^{(k)}_{j,\alpha\beta}$ through the inverse of L:

$$\partial x^{(k)}_{j,\alpha\beta} = \sum_{i=1}^{N} L^{-1}_{ji} f'_i \delta_{\alpha i} x^{(k)}_\beta = L^{-1}_{j\alpha} f'_\alpha x^{(k)}_\beta .$$

Then (2) takes the form

$$\delta w_{ij} = -\epsilon \sum_{k=1}^{M} \sum_{p=1}^{N} E^{(k)}_p L^{-1}_{pi} f'_i x^{(k)}_j = -\epsilon \sum_{k=1}^{M} e^{(k)}_i f'_i x^{(k)}_j , \tag{3}$$

where $e^{(k)}_i = \sum_{p=1}^{N} E^{(k)}_p L^{-1}_{pi}$ are the "auxiliary error signals". So, knowing $e^{(k)}$, we can easily calculate weight updates for all weights at once. e should satisfy the system of equations $L^T e^{(k)} = E^{(k)}$, where L^T is the transposed L. For feed forward networks L is lower triangle, L^T is upper triangle, and the system for e can be easily solved.

Often neurons in perceptrons are arranged in layers, and within a layer neurons are not connected to each other. It means that w has a block form with many zero entries. Layering leads to slightly bigger but computationally efficient formulas which can be found in many books. For recurrent perceptrons solving equations for x and e can be combined [74].

Applications of recurrent perceptrons may encounter some difficulties because (a) a solution may not exist, (b) if a solution exists it may not be unique, or (c) the iterative scheme may lead to some type of nontrivial behavior instead of convergence to a fixed point.

Another common type of "functional" network involves **radial basis functions**. One chooses a scalar *basis function* $\varphi(r)$ of one variable along with a number of nodes X_{0k}, $k = 1, \dots, M$. Then one seeks the approximation in the form $F(X) = \sum_{k=1}^{M} A_k \varphi\left(\|X - X_{0k}\|\right)$. The values of A_k can be found from minimization of the error function $E(A) = \sum \left(Y_i - F(X_i)\right)^2$. The solution exists for a large class of basis functions. Also in this case, dynamics is absent.

Nonetheless, the considered types of neural networks are closely related to nonlinear dynamics. They are used for approximating unknown equations of motion of dynamical systems from a time series. The time series gives the pairs of X_k (the current state of the dynamical system) and Y_k (the next state), and the task is to approximate the dependence $Y = F(X)$. Afterwards this approximation can be used for time series prediction [96] (there is even a special type of perceptrons for processing time series and a version of learning called "backpropagation through time") [45].

3.2 Multiple-attractor Networks

3.2.1 A simple example

Let us consider a gradient system with a double-well potential,

$$\dot{x} = x - x^3, \qquad x(0) = X.$$

If we take $Y = x(\infty)$ as the output, for $X < 0$ we get $Y = -1$, and for $X > 0$ we have $Y = 1$, so that $Y = \mathrm{sgn}(X)$. This dynamical system recognizes the sign of initial data and can classify them into two clusters. Or we can say that we have stored two "patterns" ($+1$ and -1, one bit) and the initial data X represent a "noisy" or "distorted" pattern. Then the system recognizes the input data as one of the two stored patterns.

All multiple-attractor networks work in a similar way: (1) stored patters correspond to attractors of the dynamical system; (2) classification or recognition of patterns is done according to the basins of attraction; and (3) convergence to one of the attractors is considered as the process of pattern recognition. This interpretation implies that the number of attractors should be greater than one.

3.2.2 Hopfield-type networks

These networks are the most well known multiple-attractor networks. The networks are fully connected. The equations of motion have the usual form

$$\hat{x}_i = f(\sum_j w_{ij} x_j + \theta_i), \tag{4}$$

where $\hat{x}_i$ stands for the value of x_i at the next time step. The important feature of the Hopfield [49, 50] model is *asynchronous* evolution: at every time step only one neuron changes its state while the states of the other neurons remain the same. Sometimes the term "Hopfield model" is applied to a wider class of model with synchronous evolution and partial connections.

In the simplest form the neurons are "binary", i.e. they can only take the values ± 1, and f is the sign function $f(u) = \ \mathrm{sign}(u)$ and $\theta_i = 0$.

Learning of this network is performed by storing the binary patterns $\xi^{(k)}$, $\xi_i^{(k)} = \pm 1$, according to the Hebbian rule,

$$w_{ij} = \frac{1}{N} \sum_{k=1}^{M} \xi_i^{(k)} \xi_j^{(k)}, \ i \neq j, \quad w_{ii} = 0, \tag{5}$$

$i, j = 1, \ldots, N$. This means that w_{ij} increases if $\xi_i^{(k)} = \xi_j^{(k)}$, and decreases otherwise.

An interesting feature of this network is the existence of an "energy" functional or Lyapunov function

$$E = - \sum_{i,j=1}^{N} w_{ij} x_i x_j .$$

It is easy to show that at every step E does not increase: if x_i remains the same, so does E. If x_i changes ($\hat{x}_i = -x_i$), then $\Delta E < 0$. Since E is bounded from below, the evolution will stop at an equilibrium point corresponding to local energy minimum.

The main weakness of this type of networks is low storage capacity. If the stored patterns are not specially chosen (orthogonal), usually the system is capable of correctly recognizing about $0.14N$ patterns. If one tries to store more patterns, then the resulting attractors cease to correspond to original patterns. To overcome the problem of false memories and to increase the storage capacity of the network several approaches were proposed, see e.g. [45, 37, 72, 66]. We shall return to this problem later, when we discuss possible role of chaos in neural networks.

Note that the Hopfield model uses a symmetric weight matrix. In case of a nonsymmetric W there is a possible oscillatory behavior, but at present it has not been successfully applied for pattern recognition purposes.

3.2.3 Brain state in a box

For this type of networks (see e.g. [43, 45]) the equations of motion have the form

$$\hat{x}_i = f(x_i + \beta \sum_{j=1}^{N} W_{ij} x_j),$$

where the function f has the form $f(x) = x$ for $x \in [-1, 1]$ and $f(x) = \mathrm{sgn}(x) = \pm 1$ if $x \notin [-1, 1]$, and the matrix W is positive definite. Therefore, the state of the system is always confined to the N-dimensional box $[-1, 1]^N$. This confinement is essential for the model performance. It can be shown, that the corners of the box $|x_i| = 1$ are stable attracting states provided $W_{ii} \geq \sum_{j \neq i} |W_{ij}|$ [45]. This condition guarantees stability of all box corners and also it is sufficient for the matrix W to be positive definite. But for matrices that arise in practice, only a few corners are usually stable. This stability has been the reason for successful applications of the BSB model for data clustering with each stable corner corresponding to a different cluster.

The matrix W can be obtained from the learning data set $x^{(k)}$ by the following iterative scheme

$$W_{k+1} = W_k + \epsilon(x^{(k)} - W_k x^{(k)}) x^{(k)T} = W_k(1 - \epsilon P_{x_k}) + \epsilon P_{x_k},$$

where $P_x = x x^T$ is the projection operator onto the x direction. If this procedure were applied infinitely many times to the same vector x, it will converge to the matrix W_∞, for which x would be an eigenvector with the eigenvalue $\lambda = 1$, that is $W_\infty x = x$. If among the training set it is possible to choose several most important dimensions, this procedure will find them, like the principal components analysis. After the matrix W has been obtained, the BSB iterations will exhibit the following results: stable corners correspond to clusters among the training data and enable us to classify subsequent vectors.

3.2.4 Kohonen networks or Self-Organizing Maps (SOM)

This network can be implemented both as multiple-attractor and attractor manipulation networks. The former implementation is slightly simpler, and so we shall consider it here. The main difference between this network and the others in this section is that its learning does not influence its dynamics. The learning is restricted only to preprocessing of initial data.

The network consists of N neurons with competitive dynamics: every active neuron tries to suppress the activities of other neurons, so that eventually only one neuron with the largest initial value $x(0)$ remains active, $x_k(\infty) = 1$, and all other $x_i(\infty) = 0$. An attractor consists of a string of $N - 1$ zeros and a single 1. The number k of this winning neuron marks the recognized pattern or cluster of input vectors. The specific form of the equations of motion is not very important. For example, the equations can have the form $\dot{x}_i = a x_i (1 - \sum_{j=1}^{N} x_j^2)$. The only purpose of the dynamics is to display, which of the neurons has received the largest input. Often the dynamics is omitted altogether and is replaced by the expression like $k = \arg\max_j x_j(0)$. An example of such a network with detailed description of dynamics can be found, e.g., in [10].

All the recognition and clustering capabilities of these networks lie in the process of preparing $x_i(0)$ from the network input X. The interneuron connections do not depend on the set of patterns to be recognized. But every neuron x_i has n input connections w_{ji}, $j = 1, \ldots, n$, $n = \dim X$, which are used for the preparation of $x_i(0)$. An example is $x_i(0) = R - \sum_j (X_j - w_{ji})^2$. It is convenient to consider the i-th column of the matrix w_{ji} as a vector $\mathbf{w}_i$, then $x_i(0) = R - \|X - \mathbf{w}_i\|^2$. It is obvious that the winner is the neuron that has $\mathbf{w}_i$ closest to X. If the vectors $\mathbf{w}_i$ are normalized, $\|\mathbf{w}_i\| = 1$, then $x_i(0) = R - X^2 - \|\mathbf{w}_i\|^2 + 2\,(X \cdot \mathbf{w}_i)$. Only the last term depends on i, and hence one can just set

$x_i(0) = (X \cdot \mathbf{w}_i)$ with the same result of recognition. This latter form is more "neural": the neuron receives the signals X_j through the connections with the weights w_{ij}.

If the network has to recognize up to N known patterns $X^{(i)}$, one can set $\mathbf{w}_i = X^{(i)}/\left\|X^{(i)}\right\|$. The network will check which of the known patterns is the closest match for the network input, and turn the corresponding neuron "on" setting it to 1. If one needs not only the indicator of the pattern, but also the pattern itself, it is possible to generate an n-dimensional output. One introduces the set of output connections $\mathbf{w}_i^{\text{out}}$ and obtains an output signal $Y = \sum_{i=1}^{N} x_i \mathbf{w}_i^{\text{out}}$. Since only one of x_i is nonzero, say, $x_k = 1$, the output of the whole network is equal to $\mathbf{w}_k^{\text{out}}$. If it is necessary to reproduce the stored pattern as in the Hopfield model, then it is necessary to set $\mathbf{w}_i^{\text{out}} = \mathbf{w}_i = X^{(i)}$. But the output may be set arbitrary $\mathbf{w}_i^{\text{out}} = Y^{(i)}$. Such a network performs the mapping $X \to Y^{(k)}$ provided $k = \arg\min_i \left\|X - X^{(i)}\right\|$. The basin of attraction for the variables x_i corresponds to domains in the space of inputs X. These domains are sometimes called Voronoi or Dirichlet cells: if X belongs to the k-th cell, then the closest of $X^{(i)}$ to it is $X^{(k)}$ [59].

A self-organized map is in fact the way of teaching such an architecture to classify input vectors into a number of self-organizing clusters, or, in other words, the adaptive way of creating the Voronoi cells. Suppose that there are M patterns $X^{(i)}$ which have to be subdivided into at most N clusters. Then one has to choose the input weights $\mathbf{w}_i$ in accordance with the data. It is done with the following iterative procedure. The patterns $X^{(i)}$ are successively presented to the network. After determining the winning neuron x_k, its input weights $\mathbf{w}_k$ are modified: $\hat{\mathbf{w}}_k = \mathbf{w}_k + \epsilon\left(X^{(i)} - \mathbf{w}_k\right)$. That is $\mathbf{w}_k$ is slightly shifted towards $X^{(i)}$. After repeating this procedure $\sim 10^4$ times the weights $\mathbf{w}_k$ usually form a good clustering for the set of the input patterns [59, 45].

In such a simple version of the algorithm, neighboring neurons may not correspond to neighboring clusters. For this reason Kohonen proposed a simple procedure for clusters positioning among the neurons. All the neurons are arranged in one- or two-dimensional lattice and each of them assigned a neighborhood. Then the training weight modification is made not only for the winning neuron, but also for all its neighbors. This makes neighboring neurons (that is, their $\mathbf{w}_i$) to be closer. Then during the learning the neighborhood size diminishes until it includes only single neuron and (perhaps) its nearest neighbors. Thus the final distribution of neurons corresponds to the distribution of input data clusters. Often, when the input vectors X are two-dimensional, the results of learning are plotted as a distribution of the weight vectors $\mathbf{w}_i$ on the plane. It is interesting that if the neurons are arranged in a 1-D lattice, and the distribution of clusters is higher dimensional, then this 1-D lattice tend to fill higher-dimensional domain like fractal Peano curve. Because of this feature, Kohonen networks sometimes are called "topological maps".

Sometimes clustering of vectors is also called "vector quantizing". The applications of such adaptive clustering technique are numerous [59]. The most impressive application seems to be one in which a speech is automatically transformed into a typewritten text (the network has been taught to show which letter corresponds to input sounds) [60, 59].

This idea has been generalized by R. Hecht-Nielsen to self-organized vector mappings [47]. It has been called "counterpropagation network". In it, simultaneous learning of input and output weights is performed from the known pairs $X^{(i)}$, $Y^{(i)}$. The resulting mapping $X \to Y$ is piecewise constant within each Voronoi cell, and for X within a given cell Y is equal to the center of the corresponding cell in Y-space.

3.2.5 Synergetics computer and multiple-well potential systems

Synergetics computer has been proposed by H.Haken [42]. It is close to the continuous-time version of Hopfield network [50], but with some modifications. For storing patterns it also uses matrix formed according to the Hebbian rule. But instead of using a sigmoid function in equations of motion, higher order terms are added to the potential to stabilize the patterns.

Multiple-well systems also use higher-order terms to construct a dynamical system with stable fixed points at necessary places. This can be done e.g. by a proper choice of the potential in a form $U(\mathbf{x}) = \sum A_i V(\mathbf{x} - \mathbf{x}_i)$. The problem of mutual interference of states can be solved by a proper choice of all parameters. This way of constructing the recognizing dynamical system was proposed in the beginning of 90's. But it seems to have nothing in common with biology and real neurons, and slightly resembles the method of radial basis function.

We have described here only the most common types of multiple-attractor networks. For their "classical" versions, complex dynamics is not relevant, though it has been used in attempts to overcome some shortcomings like false memories of Hopfield networks.

3.3 Attractor manipulation networks

3.3.1 A simple example

Let us consider an equation

$$\dot{x} = -x + X.$$

This is a gradient system with the potential $U(x) = x^2 - Xx$. That means that the input changes the system potential. For any initial value $x(0)$ the final state is $x(\infty) = X$. Therefore, this system performs the mapping $Y = X$.

Certainly, other functional dependencies can be organized this way. For example, the system $\dot{x} = -x + \tanh(X)$ provides the sigmoid function output. Similar (and probably more interesting) results can be achieved by a potential with an infinite well,

$$\dot{x} = -\frac{ax}{1 - x^2} + X.$$

The resulting mapping is

$$Y = \frac{2X}{a + \sqrt{a^2 + 4X^2}},$$

that is, almost sigmoid function. Note, that for small a this function is close to the step function $Y = \mathrm{sign}(X)$, and therefore can be used for classification or recognition purposes, similar to multiple-attractor networks.

However, to be able to interprete $x(\infty)$ as a value of the function $F(X)$, the dynamical system for every X must have a single attractor, which is a fixed point. If there are many attractors, F may have several values for a given X, and if the attractor is not a fixed point, there will be no limiting value $x(\infty)$. But usually neural network designers try to avoid such situations, and use special theorems which guarantee the existence of a stable fixed point, e.g. the famous Cohen-Grossberg theorem [39]. Usually the existence of a Lyapunov function for network equations and absence of complex temporal behavior is considered as a great virtue.

Some of the networks can be implemented both as multiple-attractor and attractor manipulation networks, an example is the Kohonen network described in the previous section. In the latter case the input X has to act all the time, and be a set of parameters in the equations of motion. Another very famous example of attractor manipulation is S.Grossberg's adaptive resonance theory network and W. Freeman's model of olfactory system. But the latter shows complex dynamics that will be discussed in Section 4.

3.3.2 Adaptive Resonance Theory Networks (ART)

Like the Kohonen network, a basic ART network has a vector X as its input and a cluster indicator as its output. See [18] for a description of generalizations of the ART network. According to [18], proper functioning of such networks requires rather thorough tuning. As a result, what we describe here is simply a brief review of some basic ideas, and not a recipe for building an ART network.

The network has 3 layers: the input layer (where the signal X is fixed), the modified input layer F_1 (with elements denoted by x_i), and the output layer F_2 (with the elements y_j). As in the case of SOM, this network should represent a cluster, only one y_j should eventually be nonzero. The information flow is as follows: $X \to F_1 \leftrightarrow F_2 \to$output.

The equations of motion in the most convenient and compact form are presented in [80]. We shall cite the ART1 version, developed for clustering of binary input patterns, when components of X can take values only 0 or 1. The dynamics for F_1 is governed by the equation

$$\frac{dx_i}{dt} = -x_i + (1 - A_{11}x_i)\left(X_i + \sum_j T_{ij} f(y_j)\right) - (A_{12} + A_{13}x_i)\sum_j f(y_j).$$

The first term on the right-hand side guarantees that $x \to 0$ when no input is presented. The second term ensures that in the presence of an input, the growth of x is limited. The sum $X_i + \sum_j T_{ij} f(y_j)$ means, that x may gain large enough amplitude only if the input X matches the stored pattern in the top-down ($F_2 \to F_1$) connections T — the "expected input", corresponding to the present activity of F_2. The third term enables to suppress the present x pattern when F_2 layer is active if there is no matching between input and expectation. The function f is a sigmoid-type function: $f(x) = 0$ for $x \le 0$ and $f(x)$ grows and saturates at 1 for $x > 0$.

The equations for F_2 layer look similar,

$$\frac{dy_j}{dt} = -y_j + (1 - A_{21}y_j)\left(f(y_j) + \sum_i B_{ij} f(x_i)\right) - (A_{22} + A_{23}y_j)\left(\sum_{k \ne j} f(y_k) + r_j\right).$$

The second term, like in SOM, excites the nodes according to bottom-up connection matrix B_{ij} along with self-excitation, while the third term describes suppression of the neuron activity, and special reset term r, which will be discussed later.

The dynamics of learning for the matrices B and T is rather simple and resembles that of SOM:

$$\tau\frac{dT_{ij}}{dt} = f(y_j)\left(f(x_i) - T_{ij}\right), \qquad \tau\frac{dB_{ij}}{dt} = f(y_j)\left(\frac{f(x_i)}{1 + |F_1|} - B_{ij}\right),$$

here $|F_1|$ is the activity of the F_1 layer (the number of active neurons). These equations ensure that learning will occur only for connections, for which the neuron y_j is active, and the stored pattern will be that generated by F_1. This is the well-known Hebbian learning.

Now let us consider the operation of this network. An input pattern X activates the F_1 layer, and it, through the matrix B, activates some pattern in the layer F_2 with one of y_j in the on-state. The pattern stored in T which corresponds to $y_j = 1$ generates another input (top-down) to F_1. If these two inputs match, the network remains in this state and adjusts the matrices B and T. But if these patterns are not close enough, then the search mechanism is started. First, the reset variables r are activated,

$$\tau_r\frac{dr_j}{dt} = f(y_j)\left(Df(v|X| - |F_1|) - r_j\right).$$

Here v is the 'vigilance parameter', which quantifies the closeness of input and expected patterns, and $|X|$ and $|F_1|$ denotes the number of "on" elements in the layers X and F_1 respectively. After r_j has been activated, it suppresses the active F_2 neuron y_j (for the best network performance it should stay suppressed until search is ended). After that the top-down influence interrupts, pattern in F_1 restores and the network begins to activate another neuron in F_2. If its pattern stored in T also does not match, this node is also suppressed and the search continues until appropriate F_2 neuron is found. It may be a “free” neuron, for which no expected pattern exists, in this case a new cluster will start to form. Or the layer F_2 may be exhausted, then the network fails to classify the input.

S. Grossberg used the phrase *adaptive resonance* for this matching between F_1 and F_2 patterns. Here only an outline of this approach is mentioned. There are several implementations of this idea: ART1 for binary patterns, ART2 for analog patterns, and ART3 for the case in which the layers F_1 and F_2 are symmetric. In every implementation there are different versions. Note that proper network operation requires accurate tuning of all parameters, for detailed descriptions see [19, 20, 21, 39, 18, 80].

The interesting feature of ART1 and ART2 architectures is their ability to be activated from both sides, that is the layer F_2 can in principle be turned into the input layer. Then two ART2 modules can be combined such that the first module receives input and generates its category, while the second module receives category to its F_2 and generates analog output signal in its F_1. This approximation network has been called ARTMAP [22].

3.3.3 Hopfield-Tank optimization network

This network was proposed in [51]. The equations of motion for the network coincide with that for continuous-time Hopfield network described in [50],

$$\frac{dx_i}{dt} = -\frac{1}{\tau}x_i + \sum_{j=1}^{N} w_{ij} f(x_j) + I_i, \tag{6}$$

where f is a sigmoid-type function, w is the weight matrix and I_i is the firing threshold of neuron i. Equations (6) describe a steepest descent dynamics $\dot{x} = -\nabla E$ for some energy functional $E(f, w, I)$, and the final state should be a fixed point corresponding to minimum of E. Learning for this network means constructing the matrix w and the bias I in a special way, to place the minimum in a desired point of the phase space. The weaknesses of this method include the facts that it is hard to guarantee the absence of spurious attractors. Later in Section 4 we shall consider, how chaos helps to avoid spurious states.

For the traveling salesman problem (TSP), Hopfield and Tank proposed a special form of the energy function depending on the distances d_{XY} between the towns to be visited. Therefore, we come to the general form of parametric attractor manipulation with d_{XY} being the parameters. Due to the presence of false attractors, the solutions found by the network could not be optimal but, according to [51], close enough to it.

3.3.4 Other networks

There are other attractor manipulation networks. For example, the well-known Boltzmann machine falls into this category, since during learning some of its neurons are ”clumped” and play the role of parameters rather than dynamic variables. Freeman’s model of olfactory system [35] also falls into this category. Some networks of this type will be considered in the next section.

4 Complex dynamics as an attempt to improve ANN

Artificial neural networks appear to be universal tools for approximations, pattern recognition, etc. Nonetheless, difficulties may arise during their application. For functional networks, for example, problems arise when one tries to approximate very complex functional dependence: a too simple network can not give proper approximation, and a too complex one amplifies noise. This problem is called “overfitting” and it is usually solved by introducing a special “second level” structure of a network called the modular and ensemble networks [82].

For conventional dynamical neural networks considered above, the problems that arise most often are (1) convergence to wrong attractors (false memories), (2) too slow convergence to attractors and (3) failure to reproduce the activities of the real brain.

Dynamics indeed has a potential for information processing. For example, an interesting idea about its application has been proposed by A.Dmitriev et al. [4]: an image is stored as a long cycle of a dynamical system, and each step of the cycle corresponds to an image pixel.

Another type of networks with complex dynamics emerged from the analogy between neuron or small groups of neurons and an oscillator. Such a neural network is a lattice of coupled oscillators. It has been shown that information may be stored and processed in the phases of the oscillators. Such an oscillatory network may implement the principles of a Hopfield network [52] or other types of networks, e.g. [69, 48].

The idea that neural networks can work in chaotic regimes has also been proposed in a number of papers. A good review of the basic approaches in this field for early 1990s can be found in [34, 93], and we shall not repeat all that has already been reported. We shall describe several models focusing on the basic directions of studies in this field. In order to ascertain the usefulness of chaos, it is instructive to pay attention to the following properties of chaos [28, 40]:

1. *Local instability.* Local instability might be useful for preventing false memories and staying off trajectories that lead to undesirable locations in the phase space. This effect of chaos may be useful only during a transitional period while the trajectory converges to a ”true memory”. After such a transition, the dynamics should again become stable.

2. *Creation of information.* This may be the most attractive property of dynamical chaos. The fact that a chaotic system behaves unexpectedly can be used to search a new way for solving a problem. In other words, chaos can help in *exploration* of possibilities. However, modern neural network architectures can not use this property, because their desired behaviors are predictable: always the same output for a given input. There were attempts to use chaotic signals in algorithms for random minimization during learning of multilayer perceptrons, but deterministic methods had the upper hand. Neural networks capable of exploration during their learning are still to be found.

3. *Wandering along attractor.* This property is closely related to instability. There were attempts to use it as a tool for memory search with the help of chaotic attractor. The trajectory wanders between images, and the idea was to use this property for matching input data with the stored images. Several experiments have been done and preliminary results obtained, but the corresponding pattern recognition algorithm has not been developed.

4. *Resemblance with complex behavior of the brain.* A number of attempts have been made, mainly by Freeman and his colleagues [34, 31, 32, 99] to create a neural network with the structure resembling that of a part of the brain, mostly the olfactory bulb. The behavior of the brain may strongly change in time, and chaotic systems also can demonstrate a variety of behaviors. The behavior of models used resembled experimental signals from the brain.

5. Finally, chaos may be used without any special role: we can create a multiple-attractor system with periodic or chaotic attractors instead of a fixed point [10]. The kind or number of a chaotic attractor can be considered as a result of recognition, though a special decoding subsystem must be added to transform a chaotic signal into the deterministic network answer.

In the following section, we shall consider several neural networks with chaos.

4.1 Transient chaos vs false memories

4.1.1 Chaotic modifications of the Hopfield-Tank model

One of the drawbacks of the Hopfield-Tank model is that the dynamics can be trapped in local minima. To overcome this difficulty, a number of workers [87, 62] have used transient chaos and noise. Recently Kwok and Smith [62] have presented a unified approach to such neural networks. It is worth reporting one of the examples to illustrate the basic ideas involved.

Chen and Aihara [24] have used transient chaos in a network described by

$$\begin{aligned} x_i(t+1) &= kx_i(t) + \alpha\left(\sum_{j=1, j\neq i}^{N} w_{ij} f(x_j(t)) + I_i\right) - z(t)(f(x_i(t)) - I_0) \\ z(t+1) &= (1-\beta)z(t) \end{aligned} \tag{7}$$

where $z(t) \geq 0$ is the self-feedback term and $0 \leq \beta \leq 1$ is the damping term. These equations of motion can be obtained from the equations of motion similar to (6) by time discretization. Due to the last term with $z(t)$ the system can be chaotic. The dynamics starts with a large value of $z(t)$ to ensure the existence of chaos, and then $z(t)$ is reduced according to (7) and the system can converge to the attractor. So, the main purpose of using chaos in this type of work is to overcome the difficulties associated with spurious states.

Another network with transient chaos for optimization problems has been proposed in [23]. The network is again a mapping, but now with a delay

$$x_i(t+1) = \sum_{j=1}^{N} w_{ij} f(x_j(t)) + I_i + g(x_i(t) - x_i(t-1)), \tag{8}$$

where f is again a sigmoid function and $g(x) = axe^{-b|x|}$. In a stationary state the last term vanishes and therefore the fixed points of (8) coincide with that of usual Hopfield-Tank model (6).

The term g makes the equations of motion more complex and ensures the chaotic wandering. Due to it the spurious stable states of the original Hopfield-Tank network in numerical simulations became unstable, though trajectory sometimes spent some time near them. To ensure convergence to the global minima, authors of [23] proposed special control scheme for parameters a and b. The experiments showed that the network successfully solved the TSP problem.

Another modification of the Hopfield-Tank model with additive chaos or noise was studied in [44].

4.1.2 The Hopfield-type network with transient chaos

This network was proposed in [55]. The idea is to replace a simple threshold neuron with an one-dimensional dynamical system $x_{t+1} = f(x_t, \mu)$ — a neuron with its own internal dynamics.

The resulting network of mappings is controlled by the system's energy E via the parameter μ. If the energy is high, the dynamics of the mappings is chaotic, and when the energy becomes low, the trajectory of the mapping tends to one of two fixed points, close to ± 1. Therefore, during transient phase the system's dynamics is chaotic, and close to the energy minimum it turns stable. The equations of motion for each "neuron" has the form

$$x_i(t+1) = f(x_i(t), E_i), \tag{9}$$

where

$$F(x,E) = \{K(E)\,(x + |E|)\} \,\mathrm{mod}2 - 1, \qquad K(E) = 2(1+|E|)^{-1}. \tag{10}$$

(chaotic behavior for $|E| < 1$ and regular otherwise). The parameter E is called the local energy

$$E_i(t) = \lambda \sum_{j=1}^{N} w_{ij} x_j(t).$$

The matrix w, as in the original Hopfield model, is constructed according to the Hebbian rule (5). The definition of the local energy E_i involves the parameter λ, which describes neuron interactions: when $\lambda = 0$ — the network splits into N independent mappings, and when $\lambda = \infty$, one obtains the usual Hopfield model.

Numerical experiments show, that for large values of λ the system behaves as the Hopfield model, but for moderate values (for the specific example considered in [55], $\lambda \leq 5$) there is a difference: almost all the false memories become unstable, and the system either remains chaotic for a very long time or converges to one of the stored images. So, in this case chaos helps in getting rid of false memories.

4.1.3 Coupled map lattice with nonstationary synchronous clusters

These networks were described in [54]. They are based upon the globally coupled map lattice with the equations of motion

$$x_i(t+1) = (1-\epsilon_i) f_i(x_i(t)) + \frac{\epsilon_i}{N} \sum_{j=1}^{N} f_j(x_j(t)), \tag{11}$$

$$f_i(x) = \alpha_i x^3 - \alpha_i x + x, \qquad 2 \leq \alpha_i \leq 4. \tag{12}$$

There are two versions of this system: α-version, when $\epsilon_i = \epsilon$, α_i are different, and ϵ-version, when all $\alpha_i = \alpha$, ϵ_i are different. Bifurcation diagram for the mapping (12) resembles that for the logistic map. The encoding and decoding are very simple, all positive values $x > 0$ are associated with the value $+1$, and negative values with -1.

The idea of the CML-based pattern recognition arose due to the fact that in the CML (11) with $\alpha_i = \alpha$ and $\epsilon_i = \epsilon$ there is a domain of parameters (α, ϵ), where the network splits into synchronized clusters corresponding to periodic behavior. For these parameters the system possesses rather high "information preservation", that is initial data strongly influence the subsequent behaviors, and there is high correlation between $x(t)$ and $x(0)$. For larger α or smaller ϵ in chaotic state this property is lost.

The idea is to define a local energy functional $E_i = -x_i \sum w_{ij} x_j$ and use chaos instead of annealing to break unwanted correlations. The matrix w is formed according to the Hebbian rule using the patterns $\xi^{(k)}$ to be stored, $w_{ij} = \sum_k \xi_i^{(k)} \xi_j^{(k)}$. Then in the α-version, the dynamics of x is augmented by the dynamics of α:

$$\alpha_i(t+1) = \begin{cases} \alpha_i(t) + (\alpha_i(t) - \alpha_{\min}) \tanh(\beta E_i) & \text{every 16 steps,} \\ \alpha_i(t) & \text{otherwise.} \end{cases}$$

The value of $\alpha_{\min}$ corresponds to the clustering phase. This causes diminishing of α, that is entering more ordered phase when the local energy is low enough.

Computer experiments show that this system works as an associative memory, and its memory capacity is about $0.18N$. Similar characteristics are obtained for the ϵ-version of the algorithm.

Another version of pattern recognition with the transition chaos$\rightarrow$order has been proposed in [67]

We note that this network does not use chaos for recognition. It uses chaos only for some short transient period although the normal dynamics of the system is non-stationary.

4.1.4 What if transient never ends or a novelty filter

There is always a possibility that chaotic transient states, under some combinations of input parameters, can give rise to a chaotic attractor or a very long transient period. Skarda and Freeman [85] supposed that such a state can mean "I don't know", that is, a neural network faces something that has not been learned (see also [55]). Such a state, like the inconsistency in bottom-up and top-down patterns of ART maps, can be used in principle as a novelty filter to initiate the learning phase. To our knowledge examples of networks with such filtering have not been published.

4.2 Chaos in memory search

When a trajectory moves along a chaotic attractor, it moves sequentially from one part to another. If we associate various parts of the attractor with different patterns, then the trajectory will wander between them. In principle, this wandering can be used for the recognition or association purposes: if a trajectory spends most of its time near one of the patterns, then the latter can be considered as "recognized", and if in the sequence of visited patterns there are stable combinations, those patterns may be considered as "associated" with one another. Note that sequences of patterns can be stored into a Hopfield-type networks. There is a possibility that chaos may help vary these combinations to learn new ones or to allow one pattern to participate in a number of associations simultaneously. These ideas have not been implemented completely, but some preliminary results have been obtained.

To study the linking of stored memories with one another, Tsuda [93] proposed a model that basically resembles a Hopfield-type network. Initially, patterns are stored by the Hebbian rule, but afterwards the connection matrix is dynamically modified. It has been shown that association of patterns with one another takes place, but there is not enough control over the process.

Another example of a model with associative dynamics in chaos has been proposed in [1] and references therein. The equations of motion had the form

$$x_i(t+1) = k_f x_i(t) + \sum_{j=1}^{N} w_{ij} f(x_j(t) + y_j(t))$$

$$y_i(t+1) = k_r y_i(t) - \alpha f(x_j(t) + y_j(t)) + a_i,$$

where $0 \le x, y \le 1$, $f(x) = 1/(1 + e^{-x/\epsilon})$, and patterns are stored in the network with the help of the Hebbian rule, $w_{ij} = \sum_k (2\xi_i^{(k)} - 1)(2\xi_j^{(k)} - 1)$.

The results of numerical experiments show that trajectories of the system visit neighborhoods of the stored patterns, and after a coarse-graining transformation some of the states reproduce the stored patterns. But the network does not give any *definite* pattern that can be considered as the resulting output. The presented results show some dependence on the behavior of initial conditions, but it is not clear, whether they correspond to (i) different attractors, (ii) transient processes or (iii) insufficiently long trajectories to establish a good statistics.

The chaotic wandering between stored patterns has also been studied in [26] and called "chaotic memory scanning".

A different idea for using chaotic regimes for recognition purposes has been proposed in [86, 87, 88, 89]. It has been observed, that patterns can be recognized by the shortest time required for full or phase synchronizations of an unknown pattern with known ones. The strength of this approach is that the procedure is general and can be applied to chaotic as well as non-chaotic neural networks. In this approach, one does not need fixed points or stationary final states for pattern recognition. The weak point is that the dynamics of known patterns needs to be run in parallel with that of the unknown pattern. It has also been found that virtual basins of attraction can be created around fixed points by root finders. What is good about this approach is that it utilizes the dense set of periodic orbits of chaotic NNs and thereby increases the capacity enormously. The work that needs to be done is to find ways to map patterns to state variables of the network.

4.3 Chaos instead of a fixed point

As we have mentioned in the beginning of Section 4, the multiple-attractor and attractor manipulation networks can be implemented with attractors other than fixed points. One of the simplest ways to construct a multiple-attractor network is described in [10]. For attractor manipulation networks there is no such simple example. This may be because a complex bifurcation structure of chaotic attractors may cause severe difficulty in distinguishing attractors for different parameter values. Nonetheless, one of the most famous chaotic networks belongs to the attractor manipulation class. This is the Freeman's model of olfactory bulb.

The studies of olfactory system have been the primary goal of W. Freeman and his colleagues for several decades, see [99, 31, 32, 30, 85] and references therein. After some years of biological studies of olfactory bulb, they concluded that studying only the structure of neurons and their connections is not enough to understand the neural mechanisms responsible for olfaction. For this reason they developed a number of mathematical models for information processing in olfactory bulb. The dynamics of the models are in qualitative agreement with the experimental EEG measurements, and is chaotic.

The model is rather complex. Each "memory unit" is described by about 10 second-order differential equations, which describe the "specialization" of neurons within every unit. All equations have similar form $\ddot{x}_i + A\dot{x}_i + Bx_i = G_i$, where the right hand side terms are different for different types of neurons [99]. Some of the G_i include input terms X, while others have delayed input (dependent on past values of x). There are neurons responsible for connections with other memory modules, and for them G_i include the term $\sum K[i,j]Q(x[j])$, where Q is a sigmoid-like function and $x[j]$ is similar "connection" neuron from the j-th memory module.

Information in this network is stored into the connections $K[i,j]$. They can take only two values, $K_{\min}$ and $K_{\max}$. Initially, all connections are set to $K_{\min}$, and to store pattern, for which modules i and j are both "active", the corresponding $K[i,j] = K[j,i]$ is set to $K_{\max}$ (Hebbian rule).

The network works as follows: When there are no external signals, the network oscillates chaotically on an attractor. If an external stimulus is presented to the network, the system stabilizes onto individual parts of the attractor. To explain the main idea, one can consider a multilobed attractor such as the Lorenz attractor which has two lobes as the network's "basal activity state". By applying an input, the network can be stabilized onto one of the two lobes. The dynamics would still be chaotic but confined to one of the lobes only. Then the presence of the trajectory at this lobe can be decoded into the network output, e.g. with the help of local time averages.

Other chaotic models of olfactory bulb also exist, e.g. [6], though they are not in very good agreement with biological experiments.

4.4 Recurrent neural networks as generators of chaos

There are works that consider neural networks just as models for some biological phenomena or convenient forms of dynamical systems. There is no question of computation, approximation or associative memory, e.g. [34, 2, 3, 61, 70] and some other. This class of neural networks falls out of our scope. Indeed, it is not hard to construct a neural network, e.g., in a form of recurrent perceptron (Section 3), with chaotic dynamics. Such works do not answer the question about the role of chaos in information processing. Nonetheless, if there *is* such a role, those neural networks may serve as a convenient generators of chaos [75].

4.5 What's wrong with chaotic networks?

We have mentioned only some of the works related to chaotic neural networks just to illustrate major directions of studies. A common feature of all chaotic networks is that they are not used in practical applications. The only exception is the experiment described in [100]. In contrast, multilayer perceptrons are widely used, while chaotic networks have remained only as objects of theoretical studies for about 15 years. What is the reason for this? From our point of view, the reason is the way in which neural networks are currently used. Current uses of neural networks may be called "isolated computations". A neural network's task is only to generate definite, always the same output for a given input. Chaotic dynamics, which is unstable, can only make such a computation unreliable. Therefore, there is no apparent room for chaos in such a scheme.

In contrast with such neural networks, the brain always works as a part of the body. It is involved in continuous processing of information coming from the outside world, and it guides the body to perform actions that change the environment. Therefore, the brain operates as a part of a closed loop: brain — actions — world — sensing — brain. It is possible that accounting for the *embodiment* of the brain can explain the advantages of operating in chaotic mode. Also chaos may just emerge as a consequence of positive feedback in the aforementioned loop.

Note that the problem of "embodiment of intelligence" has been intensively discussed in Artificial Intelligence during the last 15 years, see [73] for a review. The related approach called "behavior based robotics" or "embodied cognitive science" led to a number of efficient practical solutions and new theoretical concepts. Moreover, with examples of small robots controlled by a rather simple neural networks, it has been shown that closing the loop through the world may lead to a very complex, probably chaotic behaviors [73].

Another natural source of complex temporal behaviors may be special implementations of neural networks: using ensembles of coupled oscillators to perform computation, e.g., [52, 48, 69]. In Section 7 we shall discuss this point in more details.

5 Closing the loop: chaos in a combination of controlling neural network and controlled system

As we said in Sect. 3, 4, one of the simplest ways to obtain chaos is to take a functional network, say a feed-forward multilayer perceptron approximating equations of motion of a chaotic system, and feed its output back to the input. The result is a chaotic dynamical system. However, it does not perform useful information processing. A useful task can arise if we place between output and input of a network a system that needs controlling (often called the "plant"). In this section we present some rather simple examples that demonstrate how chaos can appear in a controller coupled with a controlled system. Chaotic signals can be registered at any part of the combined controller-plant system. This notion may partially explain the appearance of chaos in the activities of the real brain.

Here we are referring only to chaos emerging in the course of information processing. In the next section, we will consider the importance of chaos in the learning process of a controller-plant system.

Let us consider a dynamical system

$$\dot{x} = \lambda x + f, \qquad \lambda > 0. \tag{13}$$

For $f = 0$ it has an unstable equilibrium point $x = 0$. Suppose that we can control this system by applying the "force" f at the discrete moments of time $t_k = \tau k$. Then the force remains the same until the next switching. The absolute value $|f| = f_0$ always remains the same, we can only change its direction. Our goal is to keep the trajectory in the vicinity of the point $x = 0$. So, at every t_k

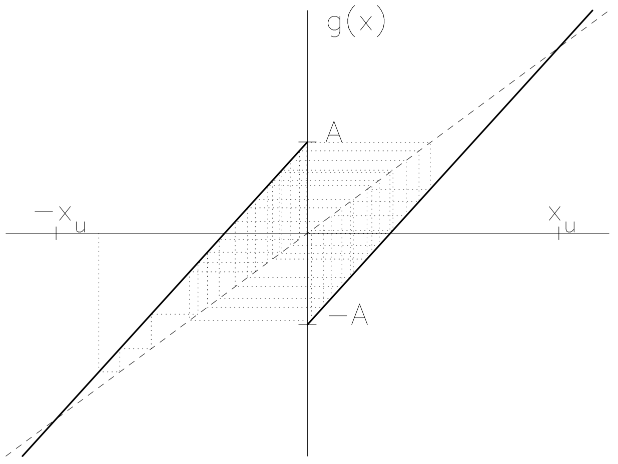

Figure 1: Mapping resulting from discrete control of unstable fixed point and an example trajectory.

we know $x(t_k)$, and we have to make a decision regarding the direction at which the force should be applied.

This is a simple problem. Let us write $x_k = x(t_k)$. Since $f_k = f(x_k)$ remains constant until t_{k+1}, (13) gives: $x_{k+1} = e^{\lambda\tau}x_k + \left(e^{\lambda\tau} - 1\right)\frac{f_k}{\lambda}$. It is easy to check that the choice $f_k = -f_0\mathrm{sgn}(x)$ solves the problem, and we obtain the following one-dimensional mapping

$$x_{k+1} = g(x_k), \qquad g(x) = \begin{cases} e^{\lambda\tau}x - A, & x \geq 0 \\ e^{\lambda\tau}x + A, & x < 0 \end{cases}, \qquad A = \frac{\left(e^{\lambda\tau} - 1\right)f_0}{\lambda}. \tag{14}$$

The plot of $g(x)$ is shown in Fig. 1. It can be seen that the trajectory remains near the unstable point provided $|x(0)| < f_0/\lambda$. Since $dg(x)/dx = e^{\lambda\tau} > 1$, a chaotic attractor is born with the Lyapunov exponent equal to λ.

The control of the system can be performed by a "network" with a single threshold neuron that receives input x_k and generates the signal ± 1, showing the direction of the force. As the attractor is chaotic, the sequence of the neuron outputs will look random. The source of this randomness is a discrete control of an unstable equilibrium.

This example explains the main idea, but it has two obvious shortcomings: (i) there is no learning, and (ii) there is no true need for the use of a neural network. Let us consider more complex examples related with discrete controls. Numerous examples of such problems can be found, e.g., in the literature on machine learning [65].

The easiest generalization is an inverted pendulum that need to be kept close to its highest point. However, it can be shown that this problem reduces to the above example — the unstable manifold of the fixed point of the inverted pendulum is one-dimensional.

A more interesting problem is that of cart-pole balancing, one of the well-known benchmark problems in machine learning [64, 12, 46]. There is a cart that can move along the line from $-x_{\max}$ to $x_{\max}$. A pole is attached to the cart with one end such that it can rotate in the vertical plane parallel to the line of motion of the cart. If the pole is set almost vertical, while falling, it moves the cart. If one pushes the cart, the push affects the pole dynamics as well. That is, by moving the cart, one can change the position of the pole. The state of the cart-pole system is determined by x (coordinate of the cart), $\dot{x}$ (velocity of the cart), θ (inclination angle of the pole from the vertical), and $\dot{\theta}$ (angular speed of the pole), see Fig. 2. The task of control is as follows: After every time interval τ, the

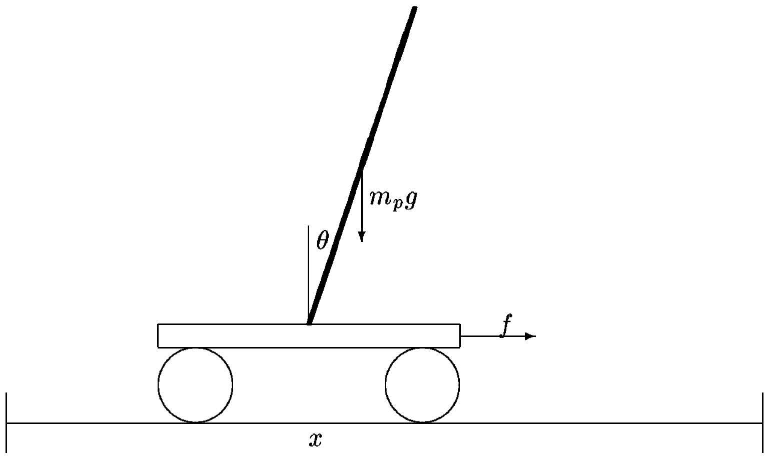

Figure 2: The cart-pole balancing task. Controller should choose the proper direction for f after each time interval τ such that the angle θ for the pole will remain within $[-\theta_{\max}, \theta_{\max}]$, and the cart never hits the ends of the track, $-x_{\max} < x < x_{\max}$. In the beginning the cart is positioned at the middle of the track $x = 0$ and the pole is set at some angle θ_0 which is within the admissible limits.

controller receives the values of the cart-pole state variables x, $\dot{x}$, θ, $\dot{\theta}$. The controller can apply a force equal to $\pm f$ to the cart that acts during the next τ-interval. The task is to keep the angle θ within the limits $[-\theta_{\max}, \theta_{\max}]$, and the position of the cart x within $[-x_{\max}, x_{\max}]$.

At the first glance this problem seems equivalent to the control of the inverted pendulum. However, if one tries to apply the same control algorithm, the cart very soon hits the end of the track. So, it is necessary to control the cart position as well, but there is no "obvious" control policy that should be learned. In the next section, we consider the details of such learning. Here, we only mention that the policy for neural network controller has been obtained [77], and the resulting regime is chaotic with one positive Lyapunov exponent. We registered (see Fig. 3) activity patterns from several neurons of the controller and calculated their autocorrelation functions. They look chaotic, though the neural network architecture (a Kohonen-type network) does not possess any complex dynamics. Similar results were obtained for another model control task — stabilization of an unstable chemical equilibrium [77].

So, chaos may arise in complex feedback loops where a neural network plays the role of a learning controller. One may ask: can chaos be useful for the information processing or for learning? The results of our models show that the answer is yes, but to explain it we need to describe the idea of reinforcement learning.

6 Chaos and reinforcement learning

6.1 What is reinforcement learning?

In most books on neural networks two types of learning are considered, supervised and unsupervised learning. If for every training input X the correct output Y is known, and this knowledge can be used for updating the weights of the network, the learning is called supervised learning or learning with a teacher. Examples of supervised learning include all functional and most dynamical networks. If learning proceeds without a teacher that can provide the correcting signals, then it is called unsupervised learning. For examples of unsupervised learning see, for example, Kohonen and ART networks.

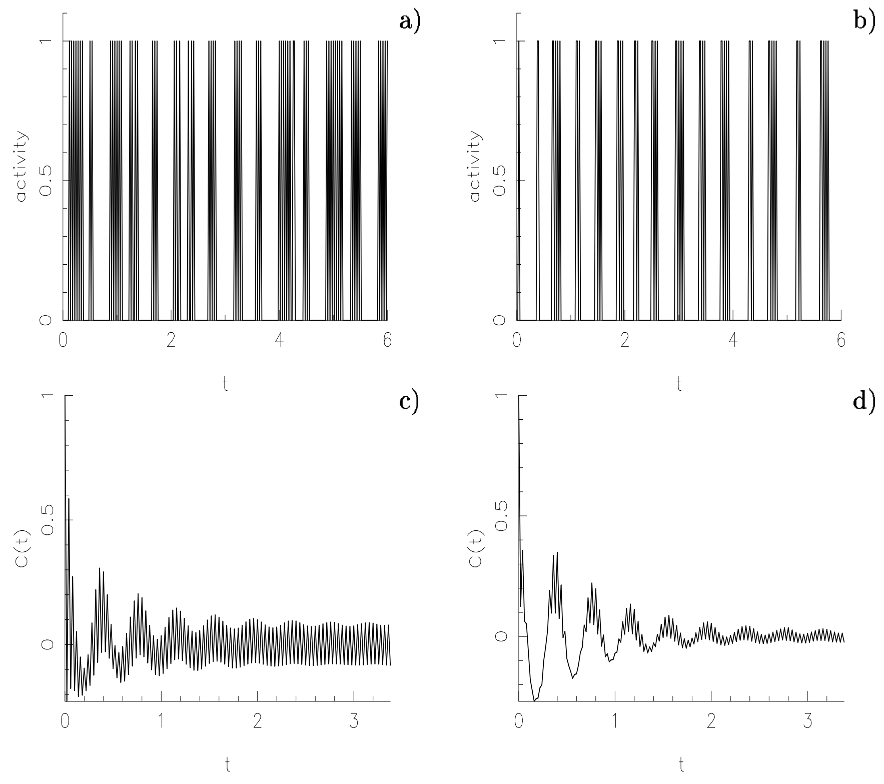

Figure 3: "Encefalogram" for the cart-pole control. Panels a and b show the temporal activity for two of the neurons of the controlling network (1 when control was performed by this neuron and 0 otherwise). Panels c and d show corresponding autocorrelation functions. Plots show both determinism and randomness.

In addition to these two types of learning, an intermediate situation is possible when some evaluation of the network performance can be done, but the correct answer is unknown. Usually, such an evaluation of performance comes in the form of a scalar 'reward' r: for success $r > 0$, for failure $r < 0$, and for neutral case $r = 0$. The corresponding learning is called reinforcement learning or learning with a critic. This type of learning is rarely used in traditional neural networks. However, it is very valuable in situations where one knows the final goal but does not know the correct way to achieve it. An example is the cart-pole balancing problem in which nobody can tell what direction of force the controller must choose at an instant although it is easy to detect a failure when the pole falls or the cart reaches the end of the track. It may be pointed out that the reward may be delayed. An example is the game of chess: the reward comes only after many moves, and it is often hard to tell which moves were right and which ones were wrong. The problem of determining the role of every action in attaining the final outcome is called the "credit assignment problem".

It is quite likely that highly organized living organisms make use of some kind of reinforcement learning. These organisms have a rather sophisticated source of reward signals such as the feelings of pain and delight. The existence of such a reward system is advantageous to these organisms. For mobile robots, similar sources of reward are also used [73], they are called "value systems".

In view of the present trends in artificial intelligence and autonomous robots, methods of reinforcement learning are expected to become increasingly important. The trouble is that there is no general theory for such a learning: the concept is not very specific and can be applied to different situations and systems. However, much progress has been made in a class of problems known as Markov Decision Processes (MDP) or Dynamical Programming (DP).

The formal description of MDP includes (i) an *environment* which is characterized by its state s, (for example, a system to be controlled, e.g., the cart-pole, where $s = \{x, \dot{x}, \theta, \dot{\theta}\}$), and (ii) an *agent*, (controller) who can perform action a (in the cart-pole example, the action is the application of the force $\pm f$). For the sake of simplicity, we shall assume that both s and a are discrete. The agent receives information about the state s_t at time t, and undertakes an action a_t, which influences the environment. After each action the environment changes its state, the agent receives information about the next state s_{t+1} and a scalar reward signal r_{t+1}. The rule, according to which the agent chooses its action $\pi(s,a)$ is called *policy*. Usually, $\pi(s,a)$ is the probability of taking the action a. Deterministic choice means that for only one of the actions the probability is nonzero.

The task of the agent is to work out an *optimal policy* which brings maximal total reward during a large or infinite number of actions. In the latter case the sum $\sum_{k=1}^{\infty} r_k$ may be infinite, and hence the *discounted* total reward $\sum_{k=1}^{\infty} \gamma^{k-1} r_k$, $0 < \gamma < 1$ (usually $|r_k|$ is bounded) is used.

Up to this point, the situation looks very general. The restricting assumption for theoretical study is that the dynamics of the environment is Markovian. Let $P = P(a, s, s')$ be the probability of change of state of the environment from s to s' after the action a has been made, and the value of reward in this case be $R = R(a, s, s')$ (r_k is always equal to R with the proper arguments). It is assumed that the transition probability P and reward R do not depend on previous states of the environment, on the previous actions of the agent or on time. Therefore, the dynamics of the environment can be described in terms of controlled Markov chain. Later, we shall also assume that this chain is ergodic.

If all P and R are known, then it is possible to estimate the expected discounted reward for the state s, action a and the selected policy π by averaging rewards for all possible paths along the Markov chain,

$$Q^{\pi}(s,a) = \sum_{s'} P(a,s,s') \left(R(a,s,s') + \gamma \sum_{a'} \pi(s',a') \sum_{s''} P(a',s',s'') \left(R(a',s',s'') + \ldots \right) \right).$$

The $Q^{\pi}(s,a)$ are called *action values.* Since we have assumed that π, P and R do not depend on time, the same Q^π appear on the right hand side, so that

$$Q^\pi(s,a) = \sum_{s'} P(a,s,s') \left(R(a,s,s') + \gamma \sum_{a'} \pi(s',a') Q^\pi(s',a') \right). \tag{15}$$

This is a linear system of equations for Q^π, which can be solved by standard methods.

Why are the action values Q^π so important? With their help, one can improve the present policy unless an optimal policy bringing the biggest reward is found. After all the Q^π have been found, it seems natural to modify the policy π so that, in every state, the action with the largest value is chosen, that is, $a = \arg\max_{a'} Q^\pi(s,a')$. This gives the new policy π_1. It has been proven [14], that $Q^{\pi_1}(a,s) \geq Q^\pi(a,s)$. Then it is possible to do the same thing with $Q^{\pi_1}(a,s)$ and find the next policy π_2 and so on. This process of policy iteration converges to an optimal policy π_* with the corresponding $Q^*(s,a)$. There may be several optimal policies with the same action values. There are very efficient numerical algorithms for policy iteration [14, 83].

The algorithms for Reinforcement Learning (RL) [83, 57] are designed for the case where all the assumptions of the dynamic programming approach are valid, but the transition probabilities P and the rewards R are *unknown.* This means that we can not find an optimal policy using the equation (15). The task resembles that of the cart-pole control: if we do not know the equations of motion, we can only act by a trial and error method. How could we learn to balance the pole?

The idea of the methods of reinforcement learning is to estimate the same action values Q, but from time averages instead of ensemble averages (15). Hence we need ergodicity. When an agent chooses the action a_t in the state s_t at the moment t and on the next step receives the reward r_{t+1}, the latter can be used for updating the estimate of $Q(s_t,a_t)$. If the number of such trials is large, then the estimates may become accurate enough and may be used for policy improvement. In fact, for improving the present policy one does not need accurate estimates of Q, all that is necessary is to know the a for which $Q(s,a)$ is the largest. This information can often be obtained during a reasonable time.

There are two main classes of methods [83, 57]: *Monte-Carlo method* in which the Q are estimated as averaged reward received after corresponding state and action (it requires many repetitions), and *Temporal Differences method* (TD) in which the estimates of these values are received by iterations. Monte Carlo methods usually require many repetitions and converge rather slowly. We will not consider them here. According to [83, 57], TD methods are more popular and usually they converge much faster.

TD methods are based upon the following simple recursive relation: When we are trying to estimate Q *experimentally*, the estimates for different states are related as $Q(s_t,a_t) = \sum_{k=0}^{\infty} \gamma^k r_{k+t+1} = r_{t+1} + \gamma Q(s_{t+1},a_{t+1})$, and therefore $\Delta_t = r_{t+1} + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,,a_t) = 0$. If $\Delta_t \neq 0$, it means that the present estimates have to be modified. So, the methods look as follows. One uses time-dependent estimates $Q(s,a,t)$ of action values. The initial guesses $Q(s,a,0)$ can be chosen arbitrarily. Then one chooses a policy π, and after every step modifies the action values iteratively:

$$Q(s,a,t+1) = Q(s,a,t) + \alpha_t e(s,a,t)\Delta_t. \tag{16}$$

This relation contains two new factors — the learning rate α_t and the so-called *eligibility traces* $e(s,a,t)$, which we shall describe a little later. For the evaluation of the correction term Δ there are two major methods.

a) "Sarsa" (the name reflects the sequence $s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}$ used for Δ evaluation)

$$\Delta_t = r_{t+1} + \gamma Q(s_{t+1},a_{t+1},t) - Q(s_t,a_t,t).$$

It provides the estimates of Q for the current policy. Nonetheless, if one uses the current values of Q for the policy formation, e.g., at the moment t chooses the action a with the largest current $Q(s_t,a,t)$

(usually the choice is more complex, we shall discuss it later), then it proves that the method admits policy improvement as well.

b) "Q-learning",

$$\Delta_t = r_{t+1} + \gamma \max_a Q(s_{t+1}, a, t) - Q(s_t, a_t, t).$$

This method converges to the action values Q^* for the optimal policy π^* almost independently of the current policy π. Knowing Q^* it is easy to obtain the optimal policy π^* itself.

For the choice of the learning rate α, there is no strong theoretical restrictions. Proof of convergence for the RL methods [83, 57] requires that α_t should slowly decrease with time such that $\sum_t \alpha_t = \infty$, $\sum_t \alpha_t^2 < \infty$ (requirements of the stochastic approximation theory), for example, $\alpha_t = \alpha_0/t$. Sometimes, the methods work well for nondecreasing α_t that may not be very small. For some problems $\alpha = 1$ is the best choice [57].

Now let us consider eligibility traces. Originally, they appeared in the model of neural learning and were necessary to reproduce effects similar to conditional reflexes [84]. These reflexes relate two events, "neutral" and "essential" that are separated by a time interval. The idea was that neurons and connections responsible for processing neutral events, for some time after them remain eligible for changes. Essential events can cause these changes. As a result, after several repetitions, a neutral event may cause the same effect as an essential one. The corresponding mathematical model of neurons, besides the usual connection weights between neurons w_{ijt}, includes values e_{ijt}, called eligibility traces. The updating rule for the weight takes the form $\Delta w_{ijt} \sim e_{ijt} g_t$, where g_t is a signal generated by an "essential" event. Initially, all $e_{ij0} = 0$. When the neutral event occurs a connection becomes eligible for changes, the corresponding e is set to 1 (or increased by 1). Then it exponentially decays $e_{t+1} = \lambda e_t$, $\lambda < 1$. Therefore, for some time after resetting e, the corresponding connection can change, but later such a possibility vanishes until a new resetting.

In the reinforcement learning, this idea has been implemented as an attempt to solve the mentioned problem of credit assignment. When the correction term Δ is calculated, it would be desirable to update Q not only for the last state s_t, but also for the previous states, since the last reward is partially due to the previous actions. To do this, one has to assign a "credit" factor for those states. In a general situation there is no recipe for this, but for ergodic Markov chains the effect of each action (or being in a specific state) decays exponentially. Therefore, at least for some problems, exponentially decaying eligibility traces may partially solve the problem of credit assignment. The corresponding class of RL algorithms has been called TD(λ), where λ stands for the decay rate of the eligibility traces. For each state-action pair, there is a variable $e(s, a, t)$, $e(s, a, 0) = 0$. Then, as soon as the environment is in the state s_t and the action a_t is chosen, the correspondent $Q(s_t, a_t)$ becomes eligible for changes, and we set $e(s_t, a_t, t) = 1$ (or in some versions $e(s_t, a_t, t) = e(s_t, a_t, t) + 1$). For all other state-action pairs eligibility exponentially decay, so finally

$$e(s, a, t) = \begin{cases} 1, & (s, a) = (s_t, a_t), \\ \lambda e(s, a, t-1), & (s, a) \neq (s_t, a_t). \end{cases}$$

The learning may be separated into episodes, for example for the cart-pole problem, after the fall of the pole or hitting the end of the track the cart is returned back in the middle and the pole into the near-vertical position. In such cases eligibility traces should be reset also.

According to [83, 57], eligibility traces may significantly accelerate RL, especially in the case of delayed reward in which nonzero r appears after many actions. As for the choice of the decay rate λ, as well as the discounting parameter γ, there is no theoretical limitations besides the inequalities $0 \leq \lambda < 1$, $0 < \gamma \leq 1$. Some practical recommendations about the choices of α, λ and γ are presented in [90]. But the performance of the methods may essentially depend on their values, so for every specific problem some tests may be necessary. Applications of reinforcement learning are numerous, among them control of complex mechanical systems, navigation of robots, elevator dispatching, playing backgammon, optimal chaos control and others, see [83, 57, 92, 36, 97].

Now, how do all these relate to neural networks and complex dynamics? Relations with neural networks are numerous. First, the methods of reinforcement learning involve a number of functions, approximation of which is required for some versions of RL [57, 83]: $\pi : s \to a$, $Q : s \times a \to R^1$, $R, P : s \times s \times a \to R^1$. Such an approximation can be done with the help of one of the approximating networks, e.g., a multilayer perceptron. Second, often the state s changes continuously. The corresponding methods of dynamic programming and reinforcement learning exist, but they are more complex than those for discrete cases. For this reason the space of possible states is subdivided into a number of domains, and each domain is considered as a single discrete state. This procedure of state space quantizing can be naturally done with the help of vector quantizing neural networks, such as Kohonen's or ART [18, 59]. For example, a neural controller may look like a two-stage Kohonen's network. It receives s as its input, each neuron corresponds to a discrete state. After the competition, the neuron corresponding to the present state turns on and sends a signal to the second network through a number of output connections with the weights equal to Q. In the second winner-takes-all network each neuron corresponds to a separate action. After a competition an action with the largest Q is selected. Therefore, neural networks are good tools for RL methods, sometimes even the term "neurodynamic programming" is used [15].

Relation with complex dynamics and chaos also exists. First, as it has been mentioned in the previous section, a neural controller with reinforcement learning integrated with an unstable dynamical system can learn to stabilize it in a chaotic state. Fig. 4 shows the example of learning for the cart-pole problem. Second, chaos may be necessary for the controllers implementing RL methods for obtaining a good policy with *exploration.* Let us consider the problem in more details.

6.2 Reinforcement learning, exploration and chaos

Let us now discuss policy making, that is, choosing criteria for taking action at a given state. In the dynamic programming, when information about the environment is complete, the action values Q^* for optimal policy can be calculated, the optimal way is to use actions with the largest Q^*. But, in learning by a trial and error method, this approach may not be optimal, because only estimates and not the true values of Q^* are available. Even if some of the Q are estimated accurately, there always remains a possibility that the best action might not have been tried. This means that the true Q^* may still remain unknown. If an agent always uses only the best known actions, then it may never learn the optimal policy. Hence, at times, it is necessary to try some non-optimal actions. The agent should *explore* the possibilities by trial and error. On the other hand, if the agent takes too many bad actions, its performance will be poor. Therefore, there is the *exploration-exploitation dilemma* [83, 91]. When we described Q-learning, we said that it provides Q^* values for the optimal policy for *almost* any current policy π: the current policy must give the possibility to explore the best actions.

When an agent always uses the action with the largest Q estimate, the policy is called *greedy.* It has been shown that following the greedy policy without exploration may lead to eventual non-optimal policy [83]. On the other hand, in many examples following a greedy policy is satisfactory. The necessary exploration may arise just because of wandering between various policies while the Q estimates converge. It is also possible to stimulate exploration by setting the initial guesses for Q high ("optimistic" initial values), inspiring the agent to try most of the actions during learning.

To make sure that the exploration takes place, a policy called ϵ*-greedy* is often used. Suppose that in a state s there are n possible actions. Then we choose the best known action with probability $1 - \epsilon$, and all other actions with probability $\epsilon/(n-1)$. When the estimates of Q have almost converged to the optimal policy, exploration will only spoil the performance because of non-optimal actions. Therefore,

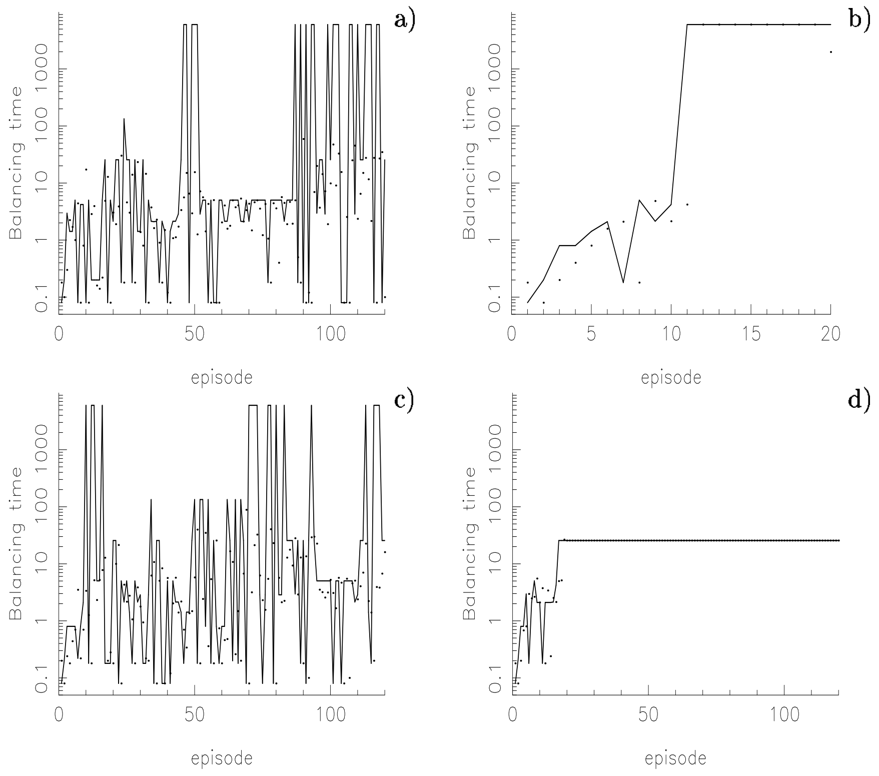

Figure 4: Examples of reinforcement learning (Q-learning) for the cart-pole control. The learning is subdivided into episodes. Episode ends either if x or θ fall outside admissible limits (then the controller receives negative reinforcement signal) or if control is successful until $T = 6000$. The latter means that the controller can keep the pole for a very long (probably infinite) time. The duration of each episode is shown by dots. During learning the policy may be ϵ-greedy (panels a and c), and failure may be only due to random action. After each episode the learned greedy policy also was tested, and the duration of the tested episode is shown by solid line. Panels correspond to different values of the parameters of Q-learning. For the good choice (panel b) learning is efficient. Panels a and c show, that for some parameter values the learned policy may oscillate: controller finds a good policy, then forgets it, then finds again.

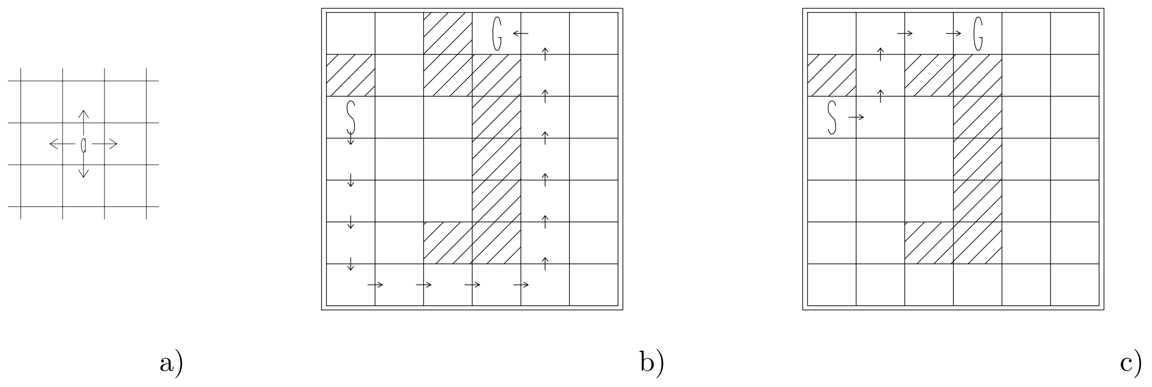

Figure 5: The testing problem for chaotic policy — a shortcut maze. (a) The agent can perform 4 actions in every state: up, down, left, right, if there is no free cell in the specified direction, the agent stays where it is. (b) Original environment, the agent starts from the "S" cell and should reach the goal "G". The shaded cells are not available for the agent, and the shortest way takes 15 steps. At every step it receives the reward $r = -1$, unless the goal is reached; then it receives the reward $r = +1$ and returns to the starting point. (c) After $t = 2010$ one cells opens, and the shorter 5-steps way appear. Exploration is necessary to "discover" this short path.

it is recommended that ϵ is reduced gradually with time [83].

Now let us suppose that the environment is nonstationary, but the changes are rare or small, and the model of Markov chain is still applicable on time intervals large enough for Q convergence. Due to nonstationarity the optimal policy may change with time. Then following the greedy policy may be dangerous: there is no more exploration due to initial values or policy convergence, and therefore a new optimal policy may never be found. Under such circumstances special exploration, e.g., in the form of an ϵ-greedy policy may become important. This can be shown with an example of navigation in a gridworld, a typical test problem for the reinforcement learning [57, 83].

In the example of navigation in a gridworld, the environment consists of a number of cells, and the agent can move from a cell to one of its neighbors, Fig. 5a. The possible actions are up, down, left, and right. If there is no neighboring cell in a chosen direction or if this cell is blocked (dashed cells in Fig. 5), then the agent stays where it is. The agent begins its motion from the starting cell marked "S", and receives a reward $r = -1$ after every action unless it reaches the goal cell "G". If the agent reaches the cell G, it receives the reward $r = 1$, and instantly goes back to the starting cell. So the agent's goal is to find the shortest path from "S" to "G" to receive the least punishment. The number of states s here is equal to the number of free cells minus one ("G"), and in every state 4 actions are available. Events that take place in the journey from "S" to "G" form an *episode*. The duration of the current episode T may be considered as a measure of the quality of the learned policy. We characterize the current ϵ-greedy policy by the averaged ratio of minimal possible duration $T_{\min}$ and T: $q = \langle T_{\min}/T \rangle$, the averaging is done over 100 different runs with the same ϵ. For the choice of actions we numbered them, and sometimes the results depended on the order. For this reason the results are presented for all 24 possible orderings of four actions.

If the environment remains unchanged, no special exploration is necessary, both sarsa and Q-learning with greedy policy find optimal or almost optimal policy. Exploration can only spoil the performance, Fig. 6a, so we used a nonstationary environment [76]. After 2010 steps, the environment changes: the uppermost closed (dashed) cell in Fig. 5b opens, Fig. 5c, and in the new environment the shortest path takes only 5 steps instead of 15. Adaptation to the new environment can not be achieved without special exploration in the algorithms. Now there is an optimal degree of exploration,

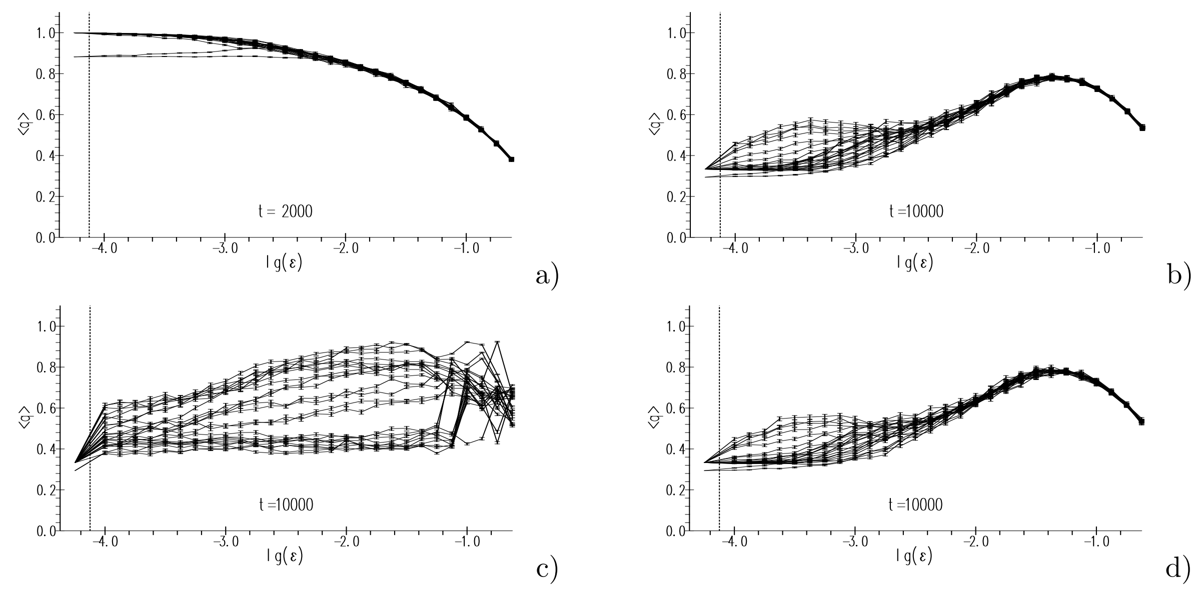

Figure 6: Results for exploration tests in Fig. 5. Different curves correspond to various numberings of the four actions. Error marks show the error estimate after averaging over 1000 runs. The leftmost point, separated by dotted vertical line, corresponds to $\epsilon = 0$. (a) For stationary environment random exploration only spoil the performance: the greater ϵ the less is q. (b) After change in the environment exploration may increase average performance: there is an optimal noice level for which performance is the best. (c) Nonchaotic exploration, a trajectory of logistic map at a_∞ is used for action selection, regularity leads to strong dependence of the results on numbering scheme. (d) Chaotic exploration, every 8-th point of the robust chaotic heuron. The results are as good as for random exploration. Therefore, chaos may be used for exploration purposes.

plots in Fig, 6b has a maximum.

Exploration requires randomness in the algorithm. A learning system must possess some source of the random signal. And here we come to *the natural place of dynamical chaos in the process of learning.* Chaos may be a source of random-like decisions in policies with exploration. To test this hypothesis we repeated the previous experiment with random, regular and chaotic policies. To choose actions for ϵ-greedy policy, we used the sequence of pseudorandom numbers as well as numbers generated by nonchaotic and chaotic dynamical systems. Fig. 6c,d. For a chaotic sequence without noticeable correlations (every k-th number or data from k independent systems) the results were as good as those with random numbers.

Among chaotic systems, the best results were obtained for the robust chaotic neuron, the mapping $x_{t+1} = |\tanh s(x_t - c)|$ [11, 75]. Therefore, neural networks can both be used for implementing the methods of reinforcement learning and at the same time be a source of exploration.

7 Hamiltonian neural networks: pattern recognition without dissipation

Recently, we have seen a feverish activity in the field of quantum computing and quantum information processing, see e.g. [56, 68, 98, 94] for a review. There is also a possibility that quantum mechanisms are used by the brain [71]. At the moment, it is hard to predict what quantum devices will appear in the future. However, one thing is sure that quantum computers are and will remain Hamiltonian systems. The importance of quantum-classical correspondence has been realized since the very beginning of quantum mechanics. Although, there are difficulties in the pursuit of this correspondence, the quest for better understandings of this correspondence will remain valid and desirable as long as we have quantum and classical mechanics as separate entities for describing natural phenomena. None of the neural networks that we have described above qualifies for quantum analysis since they are not Hamiltonian systems. This has led us to look into Hamiltonian neural networks.

Hamiltonian neural networks (HNNs) can be linear, nonlinear, chaotic, regular, autonomous or non-autonomous. Obviously, the field is wide. The simplest HNN is a linear autonomous Hamiltonian dynamical system. This is a conservative system, nothing comes in or goes out of the system. In contrast, the neural networks that we have discussed in previous sections are dissipative. In traditional neural networks, dissipation plays an important role. It works as a filter to eliminate unnecessary (noisy) information from the input data in pattern recognition. The very existence of attractors is based on dissipation. Thus, for pattern recognition with autonomous HNN, we need to resort to techniques that are different from the traditional ones.

In this section we would like to describe a number of ideas, which can be used for the development of Hamiltonian neural networks. We shall describe an implementation of a subspace projection technique and a winner-take-all architecture. Together they can make a Hamiltonian pattern recognition network.

A linear Hamiltonian dynamical system with oscillatory behavior can be written in a form $\dot{z} = (U + iW)z$, where U is an antisymmetric and W a symmetric real matrix. Here we use $z = x + ip$ as a complex N-dimensional vector. For simplicity, let us set $U = 0$. Then this system is equivalent to N oscillators with the frequencies ω_k — eigenvalues of W. Each oscillatory mode corresponds to the eigenvector e_k of W. These oscillatory modes are called "normal modes". When we present the initial data z_0, they are decomposed into a set of those modes, $z(t) = \sum_k (z_0, e_k)e^{i\omega_k t}$, the amplitude of each mode being proportional to the projection of the input pattern onto e_k.

If we can construct a matrix W such that the e_k are the stored patterns, then the amplitudes of modes correspond to the degree of resemblance of the input pattern with each of the stored patterns.

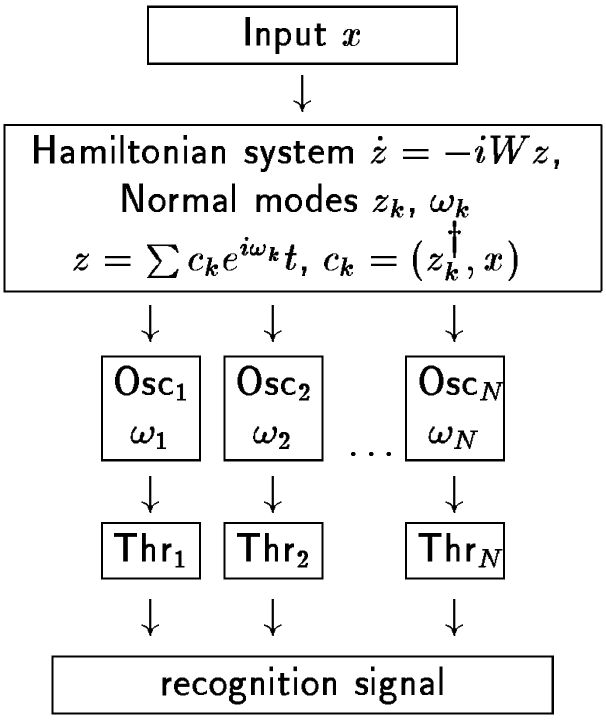
Input x
Hamiltonian system $\dot{z} = -iWz$,
Normal modes z_k, ω_k
$z = \sum c_k e^{i\omega_k t}$, $c_k = (z_k^\dagger, x)$
Osc$_1$ ω_1
Osc$_2$ ω_2
...
Osc$_N$ ω_N
Thr$_1$
Thr$_2$
Thr$_N$
recognition signal

Figure 7: Scheme of Hamiltonian neural network. Input x is used as initial data for the Linear Hamiltonian system. The latter decomposes it into oscillatory modes z_k. The modes excite oscillators Osc with the frequencies ω_k equal to that of the modes. Amplitude of the oscillators grow with the rate $\sim |c_k|$ due to resonance. Threshold detectors Thr determine, which of the amplitudes overcome a threshold value first and generate a recognition signal, defining the winning mode. Learning of this system is related with calculating of the matrix W.

Therefore, the mode with the largest amplitude (greatest resemblance) can be considered as the recognized pattern.

The mode selection can be done with the help of the effect of resonance in excited oscillator. One takes N oscillators with frequencies ω_k equal to those of the normal modes, and excites them with the signal from the Hamiltonian system. Since all the oscillators are in resonance, their amplitude will grow, and the largest rate of growth will be for the oscillator that corresponds to the most excited mode. Hence it is necessary to find which of the amplitudes passes some threshold value first. This event marks the required oscillator and hence can produce the recognition signal. Note that a convenient form for such an oscillator may be a system with a double-well potential. Initial data correspond to the particle at the bottom of one well, and its presence in the other well generates a recognition signal (Fig. 7).

The only remaining problem is to find out how to store a set of generically non-orthogonal patterns into orthogonal normal modes without distorting their form. This problem can be solved with the help of hidden dimensions.

Suppose that the vectors ξ_1, ..., $\xi_M \in R^{n_1}$, representing the patterns to be stored, are not orthogonal but linearly independent. We want to make them orthogonal without changing the patterns themselves. Let us show that it can be done by adding $L \geq M$ new dimensions. The new components of the extended vectors are denoted by v_k. It is convenient to consider them as vectors in R^L. So the task is to make new orthogonal vectors $\zeta_k = \{\xi_k, v_k\} \in R^n$, $n = n_1 + L$. The vectors v_k should satisfy the equations

$$(\xi_k \cdot \xi_j)_{n_1} + (v_k \cdot v_j)_L = A_k \delta_{kj}, \quad k, j = 1, \ldots, M \tag{17}$$

Here $(\cdot)_{n_1}$ and $(\cdot)_L$ denote dot products in the subspaces of ξ and v respectively. Let us show that for A_k sufficiently large the solution for this problem exists.

It is convenient to consider the vectors v_k as columns of an $L \times M$ matrix V. Let the latter has the form

$$V = \begin{pmatrix} d & v_{12} & v_{13} & v_{14} & & v_{1M} \\ & d & v_{23} & v_{24} & & v_{2M} \\ & & d & v_{34} & & \vdots \\ & & & d & & \vdots \\ & & 0 & & \ddots & v_{M-1,M} \\ & & & & & d \\ & & & & & \end{pmatrix}.$$

Then the conditions (17) for $k = 1$ take the form

$$(\xi_1 \cdot \xi_j) + d \cdot v_{1j} = A_1 \delta_{1j},$$

therefore

$$v_{1j} = -(\xi_1 \cdot \xi_j)/d, \quad j = 2, \ldots, M,$$

$A_1 = |\xi_1|^2 + d^2$. So, we have chosen the first column and the first row of V and have made the vector $\zeta_1 = \{\xi_1, v_1\}$ orthogonal to all other ζ_k, irrespective of the choices of v_k. Then we make the rest of the vectors orthogonal to ζ_2:

$$(\xi_2 \cdot \xi_j) + v_{12} v_{1j} + d \cdot v_{2j} = A_2 \delta_{2j},$$

and

$$v_{2j} = -\frac{(\xi_2 \cdot \xi_j) + v_{12} v_{1j}}{d}, \quad j = 2, \ldots, M,$$

$A_2 = |\xi_2|^2 + v_{12}^2 + d^2$. After repeating this procedure $M - 1$ times we find all the components of v_k. This proves the orthogonality of the resulting extended vectors ζ_k.

Test calculations show that this scheme of pattern recognition works and one can store up to N linearly independent images and correctly recognize them. The presented pattern recognition scheme can be considered as a new implementation of a subspace classification technique [59].

8 Conclusions and outlook

We have presented an overview of dynamical neural networks that perform information processing. There is an input X data and an output $Y = \varphi^t(X)$, where φ is a solution of differential or difference equations of the network dynamics. Except for the tasks of modeling a time series [45], one is interested in invariant sets of φ, that is, attractors. Many different techniques can bring up the same result, therefore one looks for the most efficient one. Efficiency, however, is a relative attribute, it depends on currently available tools of operation. Nonetheless, we must say that the scheme, initial data $\rightarrow$ attractor (or parameters $\rightarrow$ attractor) is not a universal one, since information can come gradually in time. For example, as they say, "if it quacks like a duck (time evolution), walks like a duck (time evolution), and has web feet (time invariant property), then it is the duck". Sometimes such tasks can be transformed into usual ones by recording temporal process and then presenting it as a high-dimensional vector, but usually similar problems now are treated by Artificial Intelligence techniques rather than neural networks. One asks: how can the area of applicability of neural networks be extended and chaos made useful in their performance?

We considered a number of successful applications and interesting approaches to possible future applications of dynamical chaos in neural networks in Section 4. However, they do not extend the area of applicability of existing neural networks. If one is interested in a definite answer (time independent Y) from a network, what can be the role of chaos that gives unpredictable answers? From the point of view of a definite final answer, an isolated chaotic network is not very useful, except for the role of a specific "lubricant" which prevents sticking to a wrong attractor.

However, if a neural network forms a part of a global system, there can be substantial benefit. Let us call a network an 'embodied network' if it forms a controlling part of a complex autonomous system. Embodied neural networks perform new types of tasks that cannot be done by isolated ones: an embodied network can learn through the interaction with the 'world'. An example is reinforcement learning considered in Sect. 6. For these tasks, sometimes it does not matter whether the output of an isolated network is predictable or unpredictable. Often an autonomous agent does not have an exactly specified goal; it simply has some criteria for success. As we have shown in the text, there is at least one possible niche for chaos in the interactive learning. Maybe others also exist.

We can conclude that embodied neural networks are a new and interesting field of studies from the viewpoint of nonlinear dynamics. The only problem is that one needs a controlled system to study an embodied network. Chaotic systems such as logistic map, Lorenz system, and Henon map have helped formulate most of the concepts of contemporary nonlinear dynamics. We hope that a number of "standard" subsystems and control tasks for studies of embodied networks will appear in the future. Most probably such studies will be related with robotics and the direction called "Artificial Life" (see, e.g., [73] for review). Small autonomous robots or "creatures" controlled by simple neural networks have been modeled and built for a number of years. Typically, these artificial autonomous creatures do not learn. A necessary configuration of neural network arises with the help of evolutionary or genetic algorithms. The resulting behavior of the creatures often looks chaotic. However, to our knowledge, such behaviors have not been studied in detail. Now there is a challenge of creating creatures which could both evolve and learn. And it is likely that dynamical chaos will be a typical state for neural networks needed for such tasks [13], or that such networks will need a chaotic subsystems. Embodied neural networks and artificial life seem to be a natural home for nonlinear dynamics and chaos.

Dynamical systems cover the vast field of natural phenomena, while currently known neural networks fill only a small part of it. We have microscopic, mesoscopic and macroscopic systems. The concept of information processing by neural networks will extend to areas other than what is known at the present time. An immediate extension is expected to happen in quantum computation that is based on Hamiltonian neural networks. In the text, we have presented a simple example of classical Hamiltonian neural network. Currently, very little is known about Hamiltonian neural networks. We anticipate more activities in this direction.

8.1 Acknowledgments

This work has been supported through a grant to M.K.A. from the Defence Research Establishment Suffield under the Contract No. W7702-8-R745/001/EDM. The work was carried out using computing facilities provided by Multimedia Advanced Computational Infrastructure (MACI).

References

[1] Adachi M., Aihara K. Associative dynamics in a chaotic neural network. *Neural Networks*, **10** (1997) 83–98.

[2] Aihara K., Takabe T., Toyoda M., "Chaotic neural networks", *Phys. Lett. A* **144**, 333–340 (1990).

[3] Albers D.J., Sprott J.C., Dechert W.D., "Routes to chaos in neural networks with random weight", *Int. J. Bifurc. Chaos* **8**, 1463–1478 (1998).

[4] Andreyev Yu.V., Dmitriev A.S., Chua L.O., Wu C.W., "Associative and random access memory using one-dimensional maps", *Int. J. Bifurc. Chaos* **2**, 483–504 (1992).

[5] Andreyev Yu.V., Dmitriev A.S., Starkov S.O., "Information processing in 1-D systems with chaos", *IEEE Trans. on Circuits and Systems* **44**, 21–28 (1997).

[6] Aradi I., Barna G., Érdi P., Gröbler T. Chaos and learning in the olfactory bulb. *Int. J. Intelligent Systems* **10** (1995) 89–117.

[7] Arbib M.A. *Brains, machines, and mathematics.* New York : Springer-Verlag, 1987, xvi+202 p.

[8] Babloyantz A., Destexhe A., Low-dimensional chaos in an instance of epilepsy. *Proc. Natl. Acad. Sci. USA*, **83**, 3513 (1986).

[9] Baird B., Nonlinear dynamics of pattern formation and pattern recognition in the rabbit olfactory bulb. *Physica D*, **22**, 150 (1986).

[10] Baird B., Eeckman F. A normal form projection algorithm for associative memory. in: [43], p.135–166.

[11] Banerjee S., Yorke J.A., Grebogi C., "Robust chaos", *Phys. Rev. Lett* **80**, 3049–3052 (1998).

[12] Barto A.G., Sutton R.S., Anderson C.W., "Neuronlike adaptive elements that can solve difficult control problems" *IEEE Trans. on Systems, Man, and Cybernetics* **SMC-13**, 834–846 (1983).

[13] Barton S.A., "Two-dimensional movement controlled by a chaotic neural network", *Automatica* **31**, 1149–1155 (1995).

[14] Bellman R., *Dynamic Programming* (Princeton University Press, Princeton, 1957).

[15] Bertsekas D.P., Tsitsiklis J.N., *Neuro-Dynamic Programming* (Athena Scientific, Belmont, 1995).

[16] Bishop C.M. *Neural networks for pattern recognition.* Oxford: Clarendon Press, 1995, xvii+482pp.

[17] Carpenter G.A. Neural network models for pattern recognition and associative memory. *Neural Networks* **2** (1989) 243–257; also [18], 2–33.

[18] Carpenter G.A., Grossberg S. (eds.) *Pattern recognition by self-organizing neural networks.* MIT Press: Cambridge, MA, 1991, 691p.

[19] Carpenter G.A., Grossberg S. A massively parallel architecture for a self-organizing neural pattern recognition machine. *Computer Visions, Graphics and Image Processing*, **37** (1987) 54–115; also [18], 316–382.

[20] Carpenter G.A., Grossberg S. ART2: self-organization of stable category recognition codes for analog input patterns. *Applied Optics*, **26** (1987) 4919–4930; also [18], 398–423.

[21] Carpenter G.A., Grossberg S. ART3: hierarchical search using chemical transmitters in self-organizing pattern recognition architectures. *Neural Networks* **3** (1990) 129–152; also [18], 453–499.

[22] Carpenter G.A., Grossberg S., Reynolds J. ARTMAP: supervised real-time learning and classification of nonstationary data by a self-organizing neural network. *Neural Networks* **4** (1991) 565–588; also [18], 503–544.

[23] Chang-song Z., Tian-lin C., Wu-qun H. Chaotic neural network with nonlinear self-feedback and its application in optimization. *Neurocomputing*, **14** (1997) 209–222.

[24] Chen L., Aihara K., "Chaotic simulated annealing by a neural network model with transient chaos", Neural Networks, Vol. 8, pp. 915-930, 1995

[25] Cherkassky V., Mulier F. *Learning from data. Concepts, theory and methods.* NY etc: Wiley, 1998.

[26] Dmitriev A.S., Kuminov D.A. Chaotic scanning and recognition of images in neuron-like systems with learning. *J. of Communications Technology and Electronics*, **39** (1994) 118–127.

[27] Domany E., van Hemmel J.L., Schulten K. (eds.) *Models of neural networks I.* Berlin: Springer, 1995.

[28] Eckmann J.-P., Ruelle D., "Ergodic theory of chaos and strange attractors" Rev. Mod. Phys. **57**, 617–656 (1985).

[29] Farmer D.J. A rosetta stone for connectionism. *Physica D*, **42** (1990) 153–187.

[30] Freeman W.J. Qualitative overview of population neurodynamics. in: *Neural Modeling and Neural Networks* (Pergamon Studies in Neuroscience, No 11) F. Ventriglia, ed., Pergamon, 1993, 185–215.

[31] Freeman W.J. Role of chaotic dynamics in neural plasticity. in: J. van Pelt, M.A. Corner, H.B.M. Uylings, F.H. Lopes da Silva (eds.) *Progress in Brain Research*, **102**, Springer, 1994.

[32] Freeman W.J. Simulation of Chaotic EEG patterns with a dynamic model of the olfactory system. *Biol. Cybern.*, **56** (1987) 139–150.

[33] Freeman W.J., "The physiology of perception" *Scientific American* **264**/2, 78–85 (1991).

[34] Freeman W.J. Tutorial on neurobiology: from single neurons to brain chaos. *Int. J. Bifurc. Chaos*, **2** (1992) 451–482.

[35] Freeman W. J., Yao Y., Burke B., Central pattern generating and recognizing in olfactory bulb: a correlation learning rule. *Neural Networks*, **1**, 277 (1988).

[36] Gadaleta S., Dangelmayr G., "Optimal chaos control through reinforcement learning." Chaos **9**, 775–788 (1999).

[37] Gardner E. The space of interactions in neural network models. *J.Phys. A*, **21** (1988) 257–270.

[38] Glanz J., Sharpening the senses with neural "noise". *Science* **277**, 1758 (1997).

[39] Grossberg S. Nonlinear neural networks: principles, mechanisms, and architectures. *Neural Networks* **1** (1988) 17–61; also [18], 36–109.

[40] Guckenheimer J., Holmes P., *Nonlinear Oscillations, Dynamical Systems and Bifurcations of Vector Fields* (Springer, NY, 1983).

[41] Guevara M.R., Glass L., Mackey M.C., Shrier A., "Chaos in neurobiology" IEEE *Trans. on Systems, Man and Cybernetics* **SMC-13**, 790–798 (1983).

[42] Haken H. *Information and self-organization.* Berlin:Springer-verlag, 1988.

[43] Hassoun M.H., ed. *Associative Neural Memories*, NY, Oxford: OUP, 1993, 350p.

[44] Hayakawa Y., Marumoto A., Sawada Y., "Effects of the chaotic noise on the performance of a neural network model for optimization problems", *Phys. Rev. E*, **51**(4), R2693–R2696 (1995).

[45] Haykin S. *Neural networks: a comprehensive foundation.* Macmillan: N.Y. *etc.*, 1994.

[46] Hecht-Nielsen R. *Neurocomputing.* Reading, MA etc.: Addison-Wesley, 1990, xiii+433pp.

[47] Hecht-Nielsen R. Counterpropagation networks. *Applied optics*, **26** (1987) 4979–4984; also [18], 264–276.

[48] Ho C.Y., Kurokawa H., A learning algorithm for oscillatory cellular neural networks. *Neural Networks* **12** (1999) 825–836.

[49] Hopfield J.J. Neural networks and physical systems with emergent collective computational abilities. *Proc. Natl. Acad. Sci. USA*, **79** (1982) 2554–2558.

[50] Hopfield J.J. Neurons with graded response have collective computational properties like those of two-state neurons. *Proc. Natl. Acad. Sci. USA*, **81** (1984) 3088–3092.

[51] Hopfield J.J., Tank D.W. "Neural" computation of decisions in optimization problems. *Biol. Cybern.*, **52** (1985) 141–152.

[52] Hoppensteadt F.C., Izhikevich E.M., Oscillatory neurocomputers with dynamic connectivity. *Phys. Rev. Lett.* 82 (1999) 2983–2986.

[53] Hoppensteadt F. C., Izhkevich E. M., *Weakly Connected Neural Networks* (Springer-Verlage, New York, 1997).

[54] Ishii S., Fukumizu K., Watanabe S. A network of chaotic elements for information processing. *Neural Networks*, **9** (1996) 25–40.

[55] Izhikevich E.M., Malinetskii G.G. A possible role of chaos in neurosystems. *Sov. Phys.-Dokl. (USA)*, **37** (1992) 492–495.

[56] Jibu M., Yassue K., in: *Rethinking Neural Networks*, ed. K. H. Pribram, Lawrence Erlbaum Associates, Inc., Publishers , Hillsdale New Jersey, USA (1993)

[57] Kaebling L.P., Littman M.L., Moore A.W., "Reinforcement learning: A survey" *J. Artificial Intelligence Research* **4**, 237–285 (1996).

[58] Kosko B. *Neural networks and fuzzy systems.* Englewood Cliffs: Prentice Hall, 1992, xxvii+449pp.

[59] Kohonen T. *Self-organizing maps.* Berlin:Springer, 1997, 448p.

[60] Kohonen T. The “neural” phonetic typewriter. *Computer* **21** (1988) 11–22; also [18], 239–259.

[61] Kürten K.E., Clark J.W., “Chaos in neural systems”, *Phys. Lett. A* **114**, 413–418 (1986).

[62] Kwok T., Smith K.A., A unified framework for chaotic neural -network approach to combinatorial optimization”, *IEEE Trans. Neural Networks* **10** (1999) 978–981 (and references therein).

[63] McCulloch W.S., Pitts W., A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys.* **5** (1943) 115–133.

[64] Michie D., Chambers R.A., “Boxes: an experiment in adaptive control” in: E. Dale, D. Michie, eds. *Machine Learning 2* (American Elsevier, N.Y., 1968), p.137–152.

[65] Miller W.T., Sutton R.S., Werbos P.J., eds. *Neural Networks for Control* (MIT Press, Cambridge, London, 1990).

[66] Morita M. Associative memory with nonmonotone dynamics. *Neural Networks*, **6** (1993) 115–126.

[67] Nakagawa M., A chaos associative model with a sinusoidal activation function. *Chaos, Solitons and Fractals* **10** (1999) 1437–1452.

[68] Nielsen M.A., Chuang I.L., *Quantum Computing and Quantum Information.* (Cambridge University Press 2000).

[69] Nishii J., A learning model for oscillatory networks. *Neural Networks* **11** (1998) 249–257.

[70] Pasemann F., “A simple chaotic neuron”, *Physica D* **104**, 205-211 (1997).

[71] Penrose R. *Shadows of the mind.* Vintage, 1995.

[72] Personnaz L., Guyon I., Dreyfus G., *J. de Physique Lettres* **46** (1985) L359–L365.

[73] Pfeifer R., Scheier C., *Understanding Intelligence* (The MIT Press, Cambridge MA, 1999).

[74] Pineda F.J. Generalization of backpropagation to recurrent neural networks. *Phys. Rev. Lett.*, **59** (1987) 2229–2232.

[75] Potapov A., Ali M.K., “Robust chaos in neural networks”, *Phys. Lett. A* **277** (2000) 310–322.

[76] Potapov A.B., Ali M.K., Learning, Exploration and Chaotic Policies. *Int. J. Modern Physics C* **11** (2000) 1455–1464.

[77] Potapov A.B., Ali M.K., Chaotic neural control. *Phys. Rev. E* **63** (2001), April.

[78] Pribram K.H.. *Brain and perception.* Hillsdale: Lawrence Erlbaum Associates, 1991, xxix+388p.

[79] Rumelhart D.E., McClelland J.L., eds., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* (vol. 1, 2). The MIT Press, 1986.

[80] Ryan T.W., Winter C.L. Variations on adaptive resonance. *IEEE First International Conference on Neural Networks*, M. Caudill and C. Butler, eds., San Diego: IEEE, 1987, 767-775; also [18], 385–396.

[81] Sarle, W.S., ed. *Neural Network FAQ.*
URL: `ftp://ftp.sas.com/pub/neural/FAQ.html`.

[82] Sharkey A.J.C., ed. Combining artificial neural nets. Ensemble and modular multi-net systems. NY:Springer, 1999, 304p.

[83] Sutton R.S., Barto A.G. *Reinforcement learning.* Cambridge and London: A Bradford Book and MIT Press, 1998, xviii+322pp.

[84] Sutton R.S., Barto A.G., "Toward a modern theory of adaptive networks: expectation and prediction" *Psychological review* **88**, 135–170 (1981).

[85] Skarda C. A., Freeman W. J., How brains make chaos in order to make sense of the world. *Behav. Brain Sci.*, **10**, 161 (1987).

[86] Tan Z., Ali M.K., Pattern recognition in a neural network with chaos. *Phys. Rev. E*, **58**, 3649 (1998).

[87] Tan Z., Ali M.K., Associative memory using synchronization in a chaotic neural network. *Int. J. Modern Physics C* **12** (2001).

[88] Tan Z., Ali M.K., Pattern recognition with stochastic resonance in a generic neural network. *Int. J. Modern Physics C* **12** (2001).

[89] Tan Z., Hepburn B.S., Tucker C., Ali M.K. Pattern recognition using chaotic neural networks. *Discrete Dynamics in Nature and Society*, **2** (1998) 243–247.

[90] Tesauro G., Practical issues in temporal difference learning. *Machine Learning* **8** (1992) 257–277

[91] Thrun S.B., "The role of exploration in learning control", in: D.A. White, D.A. Sofge, eds., *Handbook of Intelligent Control. Neural, Fuzzy, and Adaptive Approaches* (Van Nostrand Reinhold, NY, 1992).

[92] Touzet C., "Neural reinforcement learning for behaviour synthesis." *Robotics and Autonomous Systems* **22**, 251–281 (1987).

[93] Tsuda I., Dynamic link of memory — chaotic memory map in nonequilibrium neural networks. *Neural Networks*, **5**, 313 (1992).

[94] Ventura D., Martinez T., Quantum Associative Memory. xxx.lanl.gov, quant-ph/9807053 (1998).

[95] Wang X.-J., Rinzel J., *Handbook of Brain and Neural Networks*, ed. M. Arbib, (MIT Press, Cambridge, 1995).

[96] Weigend A.S., Gershenfeld N.A., eds., *Time series prediction: forecasting the future and understanding the past.* Addison-Wesley, 1994.

[97] White D.A., Sofge D.A., eds. *Handbook of Intelligent Control. Neural, Fuzzy and Adaptive Approaches* (Van Nostrand Reinhold, N.Y., 1992).

[98] Williams C.P., Clearwater S.H., *Exploration in Quantum Computing.* (Springer-Verlag, New York 1998).

[99] Yao Y., Freeman W.J. Model of biological pattern recognition with spatially chaotic dynamics. *Neural Networks*, **3** (1990) 153–170.

[100] Yao Y., Freeman W.J., Burke B., Yang Q., Pattern recognition by a distributed neural network: an industrial application. *Neural Networks*, **4**, 103 (1991).