Computational capacity of time-recurrent networks

2002 J. Phys. A: Math. Gen. 35 2539

(http://iopscience.iop.org/0305-4470/35/11/302)

INSTITUTE OF PHYSICS PUBLISHING JOURNAL OF PHYSICS A: MATHEMATICAL AND GENERAL

J. Phys. A: Math. Gen. **35** (2002) 2539–2554 PII: S0305-4470(02)30145-8

Computational capacity of time-recurrent networks

S A Vakulenko

Institute for Problems of Mechanical Engineering, Russian Academy of Science, 199178, VO, Bolshoj pr. 61, Saint Petersburg, Russia

Received 23 October 2001, in final form 30 January 2002
Published 8 March 2002
Online at stacks.iop.org/JPhysA/35/2539

Abstract

Time-recurrent networks are considered. Synaptic plasticity is defined by a simple Hebb rule. It is well known that this Hebbian mechanism can support learning and memory.

We show that this plasticity is a computational instrument with large possibilities. In particular, the synaptic matrix can store different information, both dynamic and static. For example, the network can perform the Fourier and wavelet transformations and calculate probability distributions of unknown parameters. These networks can analyse and identify dynamics, calculate likelihood, study autoregression etc. They can resolve even more sophisticated problems, for example decoding fractal images.

PACS numbers: 05.90.+m, 02.50.−r, 87.10.+e, 87.18.Su, 89.75.Hc

1. Statement of problem. Main results

1.1. Description of model

To try to understand brain function, various models have been proposed. The important impetus in neural network research is due in part to the paper of John Hopfield [1]. In this paper he presented a model of neural computation that is based on the interaction of neurons. It can be used as an associative memory device [1] or a noise filter [2], or for optimization problems [3].

The aim of this paper is to consider time-recurrent networks based on the Hopfield model and possessing large computational capacities. These networks can support main fundamental statistical and computational algorithms.

Let us consider the following Hopfield model with discrete neuron states $x_i(t) \in \{0, 1\}$, $t = 0, h, \ldots$:

$$x_i(t+h) = \sigma\left(\sum_{j=1}^{N} K_{ij}x_j(t) - \theta_i\right), \qquad i = 1, 2, \ldots, N, \tag{1.1}$$

where N is the number of neurons in the network, θ_i are thresholds, h is a time delay, K_{ij} is a synaptic matrix and σ is the step function

$$\sigma(z) = 1 \quad (z > 0), \qquad \sigma(z) = 0 \quad (z \leqslant 0). \tag{1.2}$$

To simplify notation, let us introduce vectors $\boldsymbol{x} = (x_1, \ldots, x_N)$ and $\boldsymbol{K}_i = (K_{i1}, K_{i2}, \ldots, K_{iN})$, then equation (1.1) takes the form

$$x_i(t+1) = \sigma(\boldsymbol{K}_i \boldsymbol{x}(t) - \theta_i). \tag{1.1a}$$

We are going to study computational capacities of a global network consisting of many local units (1.1). These local networks can be connected through their thresholds $\theta_i(t)$. Suppose there exist local networks of two types. The networks of the first type perform 'preprocessing' while the networks of the second type store information in their synaptic matrices. These networks form a 'processor'. One can interpret the processor as a primitive model of cortex and the 'preprocessor' as an analogue of multilayered networks treating input signals.

Let us describe some natural restrictions to synaptic matrices of the preprocessing and processing parts. Experimental data show that the entries of the synaptic matrix take values from a discrete set. The idea that the entries K_{ij} have only a limited number of stable states began to develop as an alternative, more realistic description of the learning process [4, 5].

We shall suppose that for the 'preprocessing' part the synaptic efficacies can take three values: 0 (no connection), $\lambda > 0$ (ith neuron activates jth neuron) and $-\lambda < 0$ (ith neuron inhibits jth neuron). Another natural restriction is $K_{ii} = 0$. We assume

$$K_{ij} \in V_3 = \{0, \lambda, -\lambda\}, \qquad K_{ii} = 0 \tag{1.3}$$

where $i, j = 1, 2, \ldots, N$ and the value λ can vary depending on the local units.

We show that even under restrictions (1.3), preprocessors (1.1) possess two key properties:

(A) they can make logical operations ('and', 'or', 'not') and approximate any vector output;
(B) they can generate a large class of stochastic, periodic and chaotic dynamics and fractal patterns.

The case of symmetrical $\boldsymbol{K}$ has been studied analytically [1, 6–8]; asymmetrical networks have been investigated mainly by computer simulations [9–12].

As for the processor, we assume that either $K_{ij} = 0$ (no connection) or $K_{ij} = \lambda > 0$ (see the next section).

1.2. Hebb rule

We assume that the network is organized as follows. There exists an auxiliary part of the network which is responsible for filtration of input signals (for instance, visuals), for generation of some complicated patterns, time signals and auxiliary calculations. Another part of network (the processor) works by synaptic plasticity and supports memory and learning.

It is widely believed that synaptic plasticity is actually the basis underlying learning and memory. The location of storage, the engram of learning and memory must be found among those synapses which support changes in synaptic efficiency. A neuronal activity changes synaptic strength by long-term potentiation (LTP, see review in [13]) and long-term depression (LTD, see [14]). Beginning with seminal works [15] and [16] (who proposed a coincidence-detection rule in which the synapse linking two cell is strengthened if the cells are active at same time), many learning rules have been suggested (see for example [1, 2, 17]).

Recently the idea that synaptic efficacies have some stable states and stimulus arriving at the network provokes transitions between these discrete states has been developed to explain the learning process [18–20]. [21] described a molecular model of the synapse bistability.

Here we use a Hebb rule proposed by [4]. Suppose that, in the processor, the synaptic efficacies take only two stable values $K_{ij} = 0$ and λ. Let us denote $\xi(t) = x_i$ and $\eta(t) = x_j$ states of postsynaptic and presynaptic neurons respectively at the moment t where $t = 0, 1, \ldots$. The following transitions can occur:

(P) if $K = K_{ij} = 0$ then $K \to \lambda$ with a probability $a(\xi, \eta) = q_+\rho(\eta, \xi)$ (potentiation);
(D) if $K = \lambda$ then $K \to 0$ with a probability $b(\xi, \eta) = q_-\tilde{\rho}(\eta, \xi)$ (depression).

We suppose that $\rho = 1$ if and only if $\xi = 1$ and $\eta = 1$. In contrast, $\tilde{\rho} = 1$ if and only if ξ and η are different. Analytically,

$$\rho(\eta, \xi) = \eta\xi, \qquad \tilde{\rho}(\eta, \xi) = u\eta(1 - \xi) + v(1 - \eta)\xi, \tag{1.4}$$

where $u, v \in [0, 1]$. This Hebb rule depends on the parameters q_+, q_-, u, v. We shall call this rule the (P)–(D) Hebb one. If the synaptic efficacies are discrete, this rule is simplest, consistent with experimental data and basic neural concepts [13–16].

1.3. Main results

Using some results of [4] and new approaches, we show the following (see sections 3.1 and 3.2).

(Ia) Given input $y(t)$, the network can calculate so-called z-transformation defined by

$$\hat{y}(t, z) = \sum_{\tau=0,1,\ldots} y(t - \tau)z^{\tau}. \tag{1.5}$$

(Ib) The network can fulfill the discrete wavelet transform defined by

$$\hat{y}(k) = \sum_{t} y(t)\Psi_k(t) \tag{1.6}$$

where Ψ_k is a special basis consisting of functions localized in t.

These transformations play a fundamental role in signal processing.

(II) The Hebb rule allows us to find important dynamic information. For example, let us observe time series $\boldsymbol{y}(t) = \bar{\boldsymbol{y}}(t) + \boldsymbol{e}(t)$ defined by a dynamical system with noise $\boldsymbol{e}(t)$. Our aim is to consistently recover a signal $\bar{\boldsymbol{y}}(t)$. This is the noise removing problem which has been discussed in a number of works (see [22,23] among others). Recently Lalley [22] showed that, in general, only averaged information on dynamics can be recovered. We show how our processor can find this information (see section 5).

(III) Networks (1.1) can effectively perform main statistical procedures, for example calculation of likelihoods and solution of regression and autoregression problems, or finding empirical distributions of unknown parameters (see section 3). The synaptic efficacies have, in this context, a fundamental probabilistic interpretation.

(IV) This (P)–(D) Hebb rule is flexible enough: this rule can be reduced to classical ones [1,2] working for associative memory devices and noise filters (section 4).

(V) Beginning with the pioneering ideas of Kolmogorov, Solomonoff, Loveland *et al* (see [24–26], also [27] for a review), the algorithmic approach to information theory develops, playing an important role in data compression problems.

To demonstrate that these networks can be a powerful tool for information compression, we show that the well known algorithm of fractal image recognition (see [29, 30]) can be implemented in our model (section 6). It is a more sophisticated problem than the previous ones and mathematical details can be found in the appendix.

This list can probably be continued.

One can formulate a hypothesis that the (P)–(D) Hebb rule [4] is a basic instrument sufficient to perform main known algorithms important for key biological functions such as signal processing, recognition, memory, data compression etc. This is possible only due to a combination of this rule with powerful preprocessing.

The biological plausibility of this model is obvious. It is well known that visual signals pass many layers before entering the cortex and, actually, biological preprocessing is a complicated procedure (see [28] for details).

Our models for the preprocessor and the processor are simplest consistent with the hypothesis on the discreteness of the synaptic efficacies. We shall not discuss here additional possibilities that can be obtained by a feedback between the preprocessor and the processor (an interesting idea proposed by the referee of this paper). Nonetheless, there is no doubt that this feedback must exist.

To conclude this introduction, let us note that here our goal is not to construct most computationally efficient schemes but to show that main known algorithms can be implemented in a biologically plausible and relatively simple model.

2. Some capacities of preprocessor and auxiliary relations

We formulate here the known results of multilayered network theory [32–34] and show that it holds under condition (1.3). We also recall some results [4] useful below.

2.1. Approximation of arbitrary 0–1 outputs

In this section we show that three-layered networks (1.1) satisfying (1.3) can solve any classification problems and approximate any vector outputs.

Denote by χ_A the characteristic function of a bounded subset A of $\boldsymbol{R}^n$. (Recall that $\chi_A(\boldsymbol{q}) = 1$ if $\boldsymbol{q} \in A$ and $\chi_A(\boldsymbol{q}) = 0$ otherwise.) Let $B(\boldsymbol{a}, \boldsymbol{b})$ be the box in $\boldsymbol{R}^n$ defined by $B(\boldsymbol{a}, \boldsymbol{b}) = \{\boldsymbol{q} : a_i < q_i < b_i\}$.

First let us observe that the characteristic function χ_B of any box $B(\boldsymbol{a}, \boldsymbol{b})$ can be calculated by a neural network consisting of two layers satisfying restriction (1.3).

To show this, we use the fact that our network can perform the logical operation 'and'. Namely, let us set

$$\chi_B(\boldsymbol{q}) = \sigma\left(\lambda \sum_{i=1}^{n} \sigma(q_i - a_i) + \lambda \sum_{i=1}^{n} \sigma(b_i - q_i) - \theta\right), \qquad \theta = \lambda\left(2n - \tfrac{1}{2}\right). \tag{2.1}$$

This formula can be interpreted as follows. At the moment t an input signal $\boldsymbol{q}(t)$ enters n independent neurons of the first layer with states $x_i, \tilde{x}_i$. As a result, at time $t + h$ their states become $x_i = \sigma(q_i - a_i), \tilde{x}_i = \sigma(q_i - b_i)$. The output of the first layer is an input for a single neuron of the second layer. Suppose that the summation $\boldsymbol{K}_i\boldsymbol{x}$ takes time h; then the procedure defined by (2.1) takes time $2h$. Here we introduce the factor λ to satisfy (1.3). An important detail: in general, we cannot set $\lambda = 1$ (see the next section).

Let us show now that the three-layered networks (1.1) (satisfying (1.3)) can approximate any 0–1 outputs $\chi_B(\boldsymbol{q})$ where B are arbitrary measurable bounded sets. For two bounded subsets B_1 and B_2 of $\boldsymbol{R}^n$, let us define the distance $\operatorname{dist}(B_1, B_2)$ between them by the integral $\int |\chi_{B_1} - \chi_{B_2}| \, \mathrm{d}^n q$.

Given $B \subset \boldsymbol{R}^n$, let us construct a special neural network as follows. For any $\delta > 0$, one can find a set $B_{\delta,N}$ which is an union of the boxes $B_k = B(\boldsymbol{a}^k, \boldsymbol{b}^k)$ (where $k = 1, 2, \ldots, N$) and such that $B_{\delta,N}$ approximates B: $\operatorname{dist}(B, B_{\delta,N}) < \delta$.

Let us consider now a network consisting of N neurons σ_k where $k = 1, 2, \ldots, N$. The state of the kth neuron is defined by

$$\sigma_k = \chi_{B_k}(\boldsymbol{q}) \tag{2.2}$$

where, in turn, χ_{B_k} can be calculated by a two-layered network (see (2.1)).

Let us use now the logical operation 'or':

$$\chi_{B_{\delta,N}}(\boldsymbol{q}) = \sigma\left(\lambda \sum_{k=1}^{N} \sigma_k(\boldsymbol{q}) - \frac{\lambda}{2}\right). \tag{2.3}$$

The right-hand side of (2.3) defines a three-layered network where each layer satisfies condition (1.3). The third layer consists of a single neuron that summarizes outputs of the second layer. Equation (2.3) shows that, even under restriction (1.3), the networks solve any classification problems.

2.2. Approximation of arbitrary vector output

Consider a continuous vector function $\boldsymbol{f}(\boldsymbol{q})$ defined on a bounded subset B of $\boldsymbol{R}^n$. This function can be approximated by a linear combination of the characteristic functions. Namely,

$$|\boldsymbol{f}(\boldsymbol{q}) - \boldsymbol{f}_N(\boldsymbol{q})| < \delta, \qquad \boldsymbol{q} \in B \tag{2.4}$$

where the ith component f_N^i of the vector function $\boldsymbol{f}_N$ is defined by

$$f_N^i(\boldsymbol{q}) = \sum_{j=1}^{N} f_j^i \chi_{B_j}(\boldsymbol{q}). \tag{2.5}$$

Let us calculate $\boldsymbol{f}_N$ using a three-layered network subjected to (1.3) where the first two layers generate sigmoidal signals, whereas the third (consisting of n neurons) generates continuous signals $\tilde{\theta}_i$.

To this end, we put

$$\theta_i(\boldsymbol{q}) = \lambda \sum_{j=1}^{N} \sum_{k=1}^{M} R_{jk}^i \sigma_{jk}(\boldsymbol{q}), \tag{2.6}$$

where N is taken from (2.5), a number M is large enough and the sigmoids σ_{jk} are defined by the relations $\sigma_{jk} = \chi_{B_j}$. Notice that these sigmoids can be calculated by means of two-layered networks (see (2.1)–(2.3)) and moreover (this is important) they do not depend on the index k.

Now we adjust coefficients R_{jk}^i by the following stochastic procedure. For $f_j^i > 0$, $R_{jk}^i = 1$ with a probability p_{ji} and $R_{jk}^i = 0$ with the probability $1 - p_{ji}$. Similarly, if $f_j^i < 0$, $R_{jk}^i = -1$ with the probability p_{ji} and $R_{jk}^i = 0$ with the probability $1 - p_{ji}$. Then, using the law of large numbers, we obtain that the mathematical expectation $\langle R_{jk} \rangle \to \pm p_{ji}$ as $M \to \infty$. To complete our procedure, we choose p_{ij}, λ and M such that $\lambda M p_{ij} = |f_j^i|$. Finally, we see that the three-layered networks with discrete efficacies (1.3) can simulate any maps $\boldsymbol{q} \to \boldsymbol{F}(\boldsymbol{q})$ but use a larger number of neurons than in the case of continuous synapses.

Notice that, in general, $\lambda \neq 1$. In fact, the number M must be large, $p_{ij} \gg M^{-1}$ and $f_j^i = \mathrm{O}(1)$, thus λ is small.

2.3. Generation of random signals and complicated dynamics

Different algorithms are based on white noises $X(t)$. Using the previous ideas one can generate the noise $X(t)$ with the prescribed mean and the deviation $\delta \equiv \text{const}$. It is natural to assume that in the network there is some white noise $S(t)$. To obtain X, the network performs a linear map $S \to c_1 + c_2 S = X$.

Now let us describe the construction of a recurrent network generating dynamics close to a given iterative dynamics $\boldsymbol{q}(t+1) = \boldsymbol{F}(\boldsymbol{q}(t))$. (It is well known that this dynamics can be periodic and even chaotic, see [35].) First, using results of section 2.3, we construct the three-layered network generating a map sufficiently close to $\boldsymbol{q} \to \boldsymbol{F}(\boldsymbol{q})$. This forward network

can be transformed to a time-recurrent one generating the prescribed dynamics. To do this, we suppose that the third layer transmits the output again to the first layer (this is based on formula (2.6)). We obtain then a time-recurrent network with iterative dynamics. Each iteration takes time $4h$. The synaptic matrix $\boldsymbol{K}$ and the thresholds θ of this network, in general, depend on the map $\boldsymbol{K} = \boldsymbol{K}(\boldsymbol{F}),\ \theta = \theta(\boldsymbol{F})$.

This construction can be essentially improved (see the appendix). A great network with a *fixed* synaptic matrix can generate a number of different dynamics depending only on the thresholds θ. To obtain given dynamics, we adjust an appropriate threshold. This property can be applied to compression of visual information (see section 6 and the appendix).

Finally, we conclude that our networks can generate different periodic and chaotic signals.

2.4. Probability distribution of synaptic efficacy

Here we describe results of [4] useful below. Consider synapse ij. For brevity we omit the indices and denote by $\xi(t)$ and $\eta(t)$ the states of the presynaptic and the postsynaptic neurons respectively and $K_{ij} = K(t)$ the synaptic efficacy at time t. The evolution of $K(t)$ is defined by a Markov chain with the two states, 0 and $\lambda > 0$. The corresponding 2×2 transition matrix M has the entries $m_{11} = 1 - a(t), m_{12} = a(t), m_{21} = b(t), m_{22} = 1 - b(t)$, where $a(t) = a(\eta(t), \xi(t)),\ b(t) = b(\eta(t), \xi(t))$. Denote by $G(T)$ the probability of the synapse being in the excited state. One can obtain [4] the following formula:

$$G(T) = \lambda \sum_{t=1}^{T} a(t) \prod_{s=t+1}^{T} \gamma(s) + e(T), \qquad \gamma(s) = 1 - a(s) - b(s), \tag{2.7}$$

where $e(T)$ is an exponentially small (as $T \to \infty$) correction depending on probabilities of the initial states.

2.5. Search for maximum (minimum)

Let us describe a preprocessor making a fast random search for the maximum of some function $f(\boldsymbol{y})$, where $\boldsymbol{y}$ ranges over a bounded domain B in $\boldsymbol{R}^{n_y}$. Consider m two-layered networks generating output signals $\xi_k \in \{0, 1\},\ k = 1, 2, \ldots, m$ by

$$\xi_k = \sigma\left(\lambda \sum_{i=1}^{m} \sigma(\theta_k - f(\boldsymbol{y}^i)) - \lambda\left(m - \tfrac{1}{2}\right)\right). \tag{2.8}$$

Here the inputs are vectors $\boldsymbol{y}^i \in B$ and numbers θ_k. The random search can be organized as follows. At each search step, one takes a random set consisting of m values $\boldsymbol{y}^i$ from B and for each k one puts $\theta_k = f(\boldsymbol{y}^k) + \epsilon$, where ϵ is small. Then $\xi_k = 1$ if and only if $f(\boldsymbol{y}^k) > f(\boldsymbol{y}^i)$ for all $i \neq k$: the neuron fired corresponds to an input giving the maximum. One step of such a search takes the time $2h$.

We see that this non-iterative method is easy to implement in a parallel neural architecture and it is well adapted to online processing if the dimension n_y is relatively small, say, $n_y < 10$. We shall not discuss here more sophisticated algorithms of maximization.

3. Some computational possibilities of the network

3.1. Computation of z-transformation of signal

First we describe a linear representation of a continuous signal $S(t)$ by a set of 0–1 signals. Suppose the signal $S(t)$ fires an ith synapse η_i by the law $\eta_i = \sigma(\theta_i - S(t))$. Then $\eta_i = 1$

if and only if $S(t) < \theta_i$. Let the thresholds satisfy $\theta_1 < \theta_2 < \cdots$. Thus the output $X_{i,i+1}(t) = \eta_{i+1}(t) - \eta_i(t)$ is equal to 1 if and only if $\theta_i < S(t) < \theta_{i+1}$. Knowing $X_{i,i+1}(t)$ we can restore, for large N_0, the original signal by the formula

$$S(t) \approx \sum_{i=1}^{N_0} S_i X_{i,i+1}(t) \tag{3.1}$$

where $S_i = S(t_i)$, t_i are any points such that $X_{i,i+1}(t_i) = 1$. Since z-transformation is linear, it is now sufficient to show how the network performs z-transformations of signals taking values 0 and 1.

Let $y(t)$ be an input signal such that $y(t) = 1$ or 0. Let us choose the parameters (q_+, q_-, u, v) of the (P)–(D) Hebb rule as

$$q_+ = q_- = q, \qquad v = 1,\ q \in (0, 1). \tag{3.2}$$

Moreover, let us take the states of the postsynaptic and presynaptic neurons as follows:

$$\xi(t) = 1, \qquad \eta(t) = y(t) \tag{3.3}$$

Then, using relations (1.4), we see that $a(t) + b(t)$ is independent of t: $a(t) + b(t) = q(\eta(t)\xi(t) + (1 - \eta(t))\xi(t)) = q$. Besides $a(t) = qy(t)$. Substituting these relations into equation (2.7), one obtains

$$G(T) = \lambda q \sum_{t=1}^{T} y(t) z^{T-t} = \lambda q (y(T) + y(T-1)z + y(T-2)z^2 + \cdots), \qquad z = 1 - q, \tag{3.4}$$

that coincides with (1.5).

Finally, the main result of this section is as follows: if the presynaptic neuron is active respectively to an input signal $\xi(t)$ (in a 0–1 form) and the postsynaptic neuron is always active, then, for large times T, the corresponding averaged synaptic efficacy is z-transformation of the input signal (with $z = 1 - q$).

3.2. Computation of transformations of wavelet type

It is well known that the wavelet transform is very effective in signal processing and noise removal. There exists a possibility of obtaining transformation (1.6) under the assumption that $\Psi_k(t)$ are functions well localized in t at some $t = t_k \ll T$. The key idea is to use the property $\boldsymbol{B}$: the preprocessor generates any signals.

Following the previous approach, suppose that $y(t)$ is an input signal with the values 1 or 0.

Let us assume that the states of the postsynaptic and presynaptic neurons are defined by, instead of (3.3), the formula

$$\xi(t) = \sigma(R_k(t) - \theta_0), \qquad \eta(t) = y(t) \tag{3.5}$$

where R_k is a special signal; $\theta_0 \in \boldsymbol{N}(0, \delta)$ is a Gaussian random quantity. This relation means that $\xi = 1$ if and only if $R_k > \theta_0$.

Denote by $\langle X \rangle$ the average of X. We take q_-, q_+, u, v as above (see (3.2)), assuming that the constant q is small with respect to d, where the parameter d is the localization width of $R_k(t)$.

Observe that $\langle a(t) \rangle = qy(t)\mathrm{Prob}(R_k(t) > \theta_0)$. Now we apply the property $\boldsymbol{B}$: the preprocessor can generate a special signal $R_k(t)$ such that $\mathrm{Prob}(R_k(t) > \theta_0) = \Psi_k(t)$ for all $|t - t_k| < Cd$, where C is a large constant.

Then, by (2.7), we have for small q and z close to 1

$$\langle G(T)\rangle \approx \lambda \sum_{t=1}^{T} \langle a(t)\rangle = \lambda q \sum_{t=1}^{T} y(t)\Psi_k(t), \tag{3.6}$$

that coincides with (1.6).

Now we shall describe how the network can calculate the likelihoods and perform other statistical algorithms.

3.3. Computation of likelihood

Consider a random quantity X subjected to the normal distribution with zero mean and deviation δ. Our preprocessor can generate such X (see sections 2.3 and 2.4).

Let z be a fixed number. We put

$$\eta(X, z, \epsilon_0) = \sigma\left(\lambda\sigma(X - z) + \lambda\sigma(z + \epsilon_0 - X) - \frac{3\lambda}{2}\right). \tag{3.7}$$

Then the average of this double sigmoid is, for small ϵ_0,

$$\langle\eta\rangle = \text{Prob}((X > z) \text{ and } (X < z + \epsilon_0)) \approx \epsilon_0\phi(z, \delta) \tag{3.8}$$

where $\phi(z, \delta) = (\sqrt{2\pi}\delta)^{-1}\exp(-\frac{z^2}{2\delta^2})$ is the normal density. Recall that η from (3.8) can be computed by a two-layered network for the time $2h$ (see (2.1)).

Now let $z_1, z_2, \ldots, z_m$ be independent real quantities normally distributed according to $\boldsymbol{N}(0, \delta)$. Then the likelihood to observe given z_i is

$$L = \prod_{k=1}^{m} \phi(z_k, \delta). \tag{3.9}$$

To compute this product, we set

$$\eta = \sigma(\lambda(\eta_1 + \eta_2 + \cdots + \eta_m) - \lambda(m - 1/2)), \tag{3.10}$$

where η_k are defined by (3.8) with $z = z_k$. Clearly $\eta = 1$ if and only if all $\eta_i = 1$. Thus,

$$\langle\eta\rangle = \text{Prob}(\eta = 1) = \epsilon_0^m \prod_{k=1}^{m} \phi(z_k, \delta). \tag{3.11}$$

Finally, we see that our schema (3.8), (3.11) can be organized as a fast-working three-layered network. Using these results, we can now consider autoregression problems.

3.4. Networks resolving autoregression model

Actual biological processors successfully resolve autoregression problems. Indeed, a cat chasing a mouse very well foresees the mouse movements. Thus the cat processor must effectively analyse the mouse trajectory.

Let us consider a general nonlinear autoregression model

$$\boldsymbol{q}(t + \tau) = \boldsymbol{F}(\boldsymbol{q}(t), \boldsymbol{a}) + \epsilon(t) \tag{3.12}$$

where $\boldsymbol{F}$ is a fixed nonlinear map (that defines the mouse dynamics), τ is a discrete time step, $\epsilon(t)$ is the white noise with known deviation δ and zero mean and $\boldsymbol{a}$ is a unknown vector of parameters. Our problem is to correctly define these parameters in frameworks of the given model (the problem of model choice is very difficult, see a brief discussion in section 6). Our hypothesis about $\epsilon(t)$ is quite standard.

To discriminate the parameters, it is natural to use the maximum-likelihood principle. Suppose we observe actual values of the process $\boldsymbol{y}(t\tau), t = 1, \ldots, T_0$ where τ is a time step. Then the probability $p(\boldsymbol{y}|\boldsymbol{a})$ of observing this time series is the following likelihood:

$$L(\boldsymbol{a}) = \prod_{t=1}^{T_0} \prod_{j=1}^{n} \phi(z_j(t), \delta), \tag{3.13}$$

where

$$z_j(t) = y_j((t+1)\tau) - F_j(\boldsymbol{y}(t\tau), \boldsymbol{a}), \qquad t = 1, \ldots, T_0. \tag{3.14}$$

The mappings $\boldsymbol{y} \to F_j(\boldsymbol{y})$ can be calculated by a three-layered network for the time $3h$ (see section 2). Let us set $\tau = 3h$. As has been shown in the previous section, L from (3.13) can be computed by a two-layered network (which obtains $z_j(t)$ as input signals). Therefore, the likelihood (3.13) can be calculated by a forward network with five layers where $\boldsymbol{y}(t)$ is an input.

To obtain online an optimal $\boldsymbol{a}$, one can use the preprocessor described in section 2.5.

3.5. Random search

Consider the following question: does there exist an input $\boldsymbol{q}$ such that $f(\boldsymbol{q}) < c$, where f is a given scalar function?

The Hebbian processor can resolve this problem using a random search. Let a preprocessor generate $\theta = f(\boldsymbol{q})$ by (2.6). Define the states of the presynaptic and postsynaptic neurons by

$$\xi(t) = \sigma(c - f(\boldsymbol{q}(t))), \qquad \eta(t) = 1. \tag{3.15}$$

Moreover, let us choose the parameters of the Hebb rule by

$$q_+ = 1, \qquad q_- = 0, \tag{3.16}$$

and suppose that the initial value of the synapse $K(0) = 0$. Now we take a random input $\boldsymbol{q}(t)$, $t = 1, 2, \ldots$. If there exists a value t_0 such that $f(\boldsymbol{q}(t_0)) < c$, the efficacy K becomes 1 at moment t_0 and conserves this value up to the final moment $t = T$. Otherwise $K(T) = 0$. Thus, the mean $G(T) > 0$ if and only if the value $\boldsymbol{q}$ exists.

4. Memory and learning

First we shall show that this approach reveals a fundamental probabilistic interpretation of the synaptic efficacies.

Suppose that a random sequence $O_1, O_2, \ldots, O_m$ of some independent objects (patterns, time signals etc) is exposed (where $m \gg 1$). It is natural to assume that the preprocessor can find some 'features' $A_1, A_2, \ldots, A_s$ of these objects. For example, we can connect with each feature A_i an output neuron η_i of the preprocessor and suppose that $\eta_i = 1$ (the neuron is fired) if and only if an object O observed possesses the feature A_i. We shall also use an additional neuron η_0 that is always fired.

Notice that this approach is applicable to both qualitative and quantitative analysis of objects: to estimate, for example, a continuous parameter $\boldsymbol{a}$ of an input signal, we can decompose the domain of possible values of $\boldsymbol{a}$ into small subdomains. Thus, after such quantization, a quantitative parameter can be replaced by a family of qualitative features (properties).

Let us consider now two neurons η_i and η_j and calculate the mean value of the synaptic efficacy $\langle K_{ij} \rangle = G_{ij}$. As the Hebb rule parameters, first we take

$$q_+ = q_- = q, \qquad v = 0, \qquad u = 1, \qquad q \in (0, 1). \tag{4.1}$$

Denote by $a_{ij} = a(\eta_i, \eta_j)$ and $b_{ij} = b(\eta_i, \eta_j)$ the quantities introduced above in section 1.2. Let us calculate the averages $\langle a_{ij}\rangle$ and $\langle b_{ij}\rangle$ that we obtain after observation of our object sequence $O_1, O_2, \ldots, O_m$.

Clearly, for large sequences, the mean $\langle a_{ij}\rangle$ is

$$\langle a_{ij}\rangle \approx q\mathrm{Prob}(\eta_i(t) = 1, \qquad \eta_j(t) = 1) = q\mathrm{Prob}(A_i A_j), \tag{4.2}$$

i.e. it is proportional to the probability of observing, in our family of objects, the features A_i and A_j together.

Using (4.1) and (1.4) we obtain that

$$\langle a_{ij}\rangle + \langle b_{ij}\rangle = q\langle \eta_i\eta_j + (1-\eta_i)\eta_j + (1-\eta_j)\eta_i\rangle = q\langle\eta_i\rangle \approx q\mathrm{Prob}(A_i). \tag{4.3}$$

Consider (2.7) and average it over all possible realizations (setting $T = m$). Since the terms in (2.7) are independent for different times, we obtain (repeating calculations [4])

$$\langle K_{ij}(T)\rangle \approx \lambda\frac{\langle a_{ij}\rangle}{\langle a_{ij}\rangle + \langle b_{ij}\rangle} = \lambda\frac{\mathrm{Prob}(A_i A_j)}{\mathrm{Prob}(A_i)}, \qquad (T = m \gg 1). \tag{4.4}$$

We see that this mean is proportional to the conditional probability $\mathrm{Prob}(A_j|A_i)$ that the patterns have the feature A_j under the condition that they possess the feature A_i. If we take, as a presynaptic neuron, neuron η_0 (instead of $\eta_i,\ i > 0$), the corresponding value is the probability of having feature A_j.

Thus, the networks can compute empirical probabilistic distribution of the parameters, find correlations etc. This memory organization helps to analyse the structure of the objects. For example, if $G_{ij} = \langle K_{ij}\rangle$ and G_{ji} both are close to zero, this means that features (properties) A_i and A_j are, in a sense, almost uncorrelated. If G_{ij} is close to 0 but G_{ij} is close to 1, this means that property A_i 'almost always' entails A_j. One can assume that the memory organization, naturally associated with variant (4.1) of the (P)–(D) Hebb rule, is a treelike structure, when, to classify objects, we use a hierarchic system of features.

Let us show that the (P)–(D) Hebb rule is sufficiently flexible and can be reduced to the classical rules [1] that also have a simple interpretation. Indeed, instead of (4.1) let us set now

$$q_+ = q = \frac{1}{m}, \qquad q_- = 0. \tag{4.5}$$

Then, in (2.7), one has $\gamma(s) = 1$ and $a(t) = a_{ij}(t) = q\eta_i(t)\eta_j(t)$. Here the 'time' t is actually the pattern index $\mu,\ \mu = 1, 2, \ldots, m$. Introducing notation $\eta_i(\mu) = \eta^\mu$, we obtain the classical Hebb rule

$$\langle K_{ij}(m)\rangle = \lambda m^{-1}\sum_{\mu=1}^{m}\eta_i^\mu\eta_j^\mu. \tag{4.6}$$

Therefore, $\langle K_{ij}\rangle$ is proportional to the probability that the objects possess simultaneously features A_i and A_j.

Thus, we see that this network can work in a different ways depending on the Hebb rule parameter choice, and, in particular, as a classical attractor neural network. Notice that the work [4] showed that this (P)–(D) Hebb rule allows us to recognize random patterns of a simple structure without a supervisor.

5. Analysis of dynamical information by network

The following problem is also important from the biological point of view. Suppose the network obtains a time input signal in the vector form $\boldsymbol{S}(t)$ where $\boldsymbol{S} \in \boldsymbol{R}^n$. Furthermore, let data $\boldsymbol{S}(t)$ be trajectories of a noised dynamical process

$$\boldsymbol{S}(t+1) = \boldsymbol{Q}(\boldsymbol{S}(t)) + \epsilon(t)$$

where $\boldsymbol{Q}$ is a nonlinear map and $\epsilon(t)$ is a white noise. It is impossible to remove a true trajectory from the observed one even if the white noise $\epsilon(t)$ is small [22]. However some averaged characteristics can be found. To describe these, first let us recall some fundamental notions of the dynamical system theory.

There exist different natural approaches to the noise removal problem; in particular, one can approximate the process by some Markov chain with discrete states. Recall this fundamental construction. Partition the compact phase space X into a finite number of nonempty connected sets $A_1, A_2, \ldots, A_p$. To form our Markov model, we identify each set A_i with the ith state of our Markov chain. We construct an $n \times n$ transition matrix with entries P_{ij} interpreted as follows: P_{ij} is the probability that a typical point in A_i moves into A_j under one iteration.

Another important dynamical characteristic is an invariant measure on the attractor called the Bowen–Ruelle–Sinai (BRS) measure. The BRS measure of a set A is the frequency with which orbits visit this set, namely

$$\nu(A) = \lim_{T\to\infty} T^{-1} \sum_{t=1}^{T} \phi(A, t) \tag{5.1}$$

where $\phi(A, t) = 1$ if the point $\boldsymbol{y}(t)$ lies in A and $\phi = 0$ otherwise.

The BRS measure (5.1) is a characteristic stable under noise. This measure can be found [22].

Question: how can the network obtain $\nu(A)$ (where A is a compact domain in our phase space)?

We shall show that information on ν and P_{ij} can be written in the synaptic matrix by the Hebb rule.

Suppose that the process is ergodic. Let us show how the Hebb rule allows us to calculate ν.

To find an approximation of ν, we split the phase space into a finite set of compact subdomains A_k. One can assume (see [23] for details) that, for large T, a good approximation of $\nu_k = \nu(A_k)$ is the following quantity:

$$\mu_k(T) = T^{-1} \sum_{t=0}^{T} \phi(A_k, t) \tag{5.2}$$

where $\phi(A_k, t) = 1$ if the point $\boldsymbol{S}(t)$ in A_k and $\phi = 0$ otherwise.

To calculate (5.2) we assume that the states $\xi(t)$ and $\eta(t)$ of the postsynaptic and presynaptic neurons are defined by

$$\xi(t) = 1, \qquad \eta(t) = \phi(A_k, t). \tag{5.3}$$

(To obtain $\eta(t)$ in this form, we use a preprocessor, see section 2.1.) Let us take the Hebb parameters by (3.2) and calculate $G(T)$. First we notice that $a(t)+b(t) \equiv q$ and $\gamma(t) \equiv 1-q$. Thus we obtain

$$G(T) = \lambda q(\phi(A_k, T) + \phi(A_k, T-1)z + \phi(A_k, T-2)z^2 + \cdots), \qquad z = 1-q. \tag{5.4}$$

Since our random process $\boldsymbol{S}(t)$ is ergodic, the following asymptotic holds:

$$\alpha_k(t, T) = \sum_{\tau=1}^{t} \phi(A_k, T-\tau) = \mu_k t + \mathrm{o}(t), \qquad (t \to \infty) \tag{5.5}$$

where μ_k are defined by (4.2), the correction $\mathrm{o}(t) \ll t$ for large t. (Indeed, $\alpha_k(t, T)$ is the frequency with which our process visits A_k.) To use relation (5.5), one can apply to (5.4) the Abel formula [36], assuming that q is small (and thus z close to 1). As a result, we obtain

$$G(T) = \lambda q(1-z)(\alpha_k(1, T) + \alpha_k(2, T)z + \alpha_k(3, T)z^2 + \cdots). \tag{5.6}$$

Clearly, when $z = 1 - q$ with a small q, in this sum the main contribution is given by terms $\alpha_k(t, T)$ for large t. Thus we can substitute into (5.6) asymptotic (5.5), that gives

$$G(T) \to \lambda\mu_k \qquad (T \to \infty). \tag{5.7}$$

Similarly one can organize calculation of the transition frequency P_{ij} (see for definition section 1.3). To end this, we put

$$\xi(t) = 1, \qquad \eta(t) = \sigma(\lambda(\phi(A_i, t) + \phi(A_j, t+1) - \tfrac{3}{2})). \tag{5.8}$$

The analysis is similar and we omit it.

6. Data compression. Recognition of fractal patterns and decoding fractals

The previous sections were mainly connected with the probability theory, statistics and the statistical information theory.

However, a simple example from the book [37] shows that the brain analyses the information and performs data compression. Thus, to understand organization of neural memory, we need also the algorithmic information theory [24–27]. The example is the following. It is clear that it is simpler to remember the telephone number 345-67-89 than the number 749-36-58 because the first sequence is increasing.

The main idea is that the 'complexity' of an object O_{in} relative to given objects $O_1, \ldots, O_p$ (that can be stored in memory, for example) is the minimal length of an algorithm that constructs O_{in} from given O_k, or, otherwise, the minimal description length. For example, it is well known now that many real fractal images and scenes actually admit a short description (see [29–31]). So, their complexity is actually less than one may think observing these images.

We are going to demonstrate that the networks considered can serve as information compressors. We illustrate this by an example, showing that the standard algorithms of fractal recognition [30] can be implemented in this neuronal architecture. These algorithms have practical applications to image compression and recognition, particularly when we must handle online images depending on time.

In this section we recall only main ideas of the approach [30]; mathematical details can be found in the appendix.

To describe generation of fractal patterns, we shall study special random maps. Consider a family of maps $\boldsymbol{q} \to \boldsymbol{F}^i(\boldsymbol{q})$, $\boldsymbol{q} \in X$ where X is a compact in $\boldsymbol{R}^n$. Let, moreover, each map $\boldsymbol{F}^i$ be a contraction. Such a family is called an iterated function system (IFS) [29]. We can use this IFS to construct a mapping W from the space of compact subsets of X into itself. Namely, let us define

$$W(B) = \bigcup_{i=1}^{m} f_i(B), \qquad B \subset X. \tag{6.1}$$

Then W is a contraction map with respect to the Hausdorff metric (see the appendix). In this setting W admits a unique fixed point; that is, there is exactly one nonempty compact set $A \subset X$ such that $W(A) = A$. This set is the attractor of the IFS.

We can use IFS to compress visual information presented as 0–1 (black–white) images (a point $\boldsymbol{q} \in \boldsymbol{R}^2$ is black, if $\boldsymbol{q} \in W$). To achieve this goal, one often applies IFS consisting of contracting linear maps. For example, the well known Kantor set [35] is an attractor of the IFS on the interval $[0, 1]$ defined by the two maps $f_1 : x \to x/3$ and $f_2 : x \to (x + 2)/3$. If a fractal image can be written to memory as an attractor of some IFS, much information can be saved. In fact, to write the Kantor set in memory in details immediately, bits by bits, we must use a number of bits. Using the IFS allows us to remember only the number of linear applications and their parameters.

An implementation of IFS in the networks considered can be made by results of section 2.3 and the appendix. It essentially exploits the fact that the preprocessor can generate a number of complicated dynamics.

However, a number of questions remain without answers. A typical very difficult question is the following. This approach is based on an *a priori* hypothesis that fractals are attractors of corresponding IFSs. However, there exist many different models of fractal objects [31]. If an algorithm of the fractal coding is known *a priori*, up to unknown parameters, it is not difficult to find these parameters. However, how do we find this algorithm itself? Otherwise, how does the network form its model of the object? How does the network find the shortest model description?

These problems are very complex and we shall not discuss them here (see e.g. [27, 38]).

7. Concluding remarks

The main conclusion of this paper is the following: synaptic plasticity based on the Hebb rule from [4] (see section 1.2) is a powerful instrument allowing us to support many fundamental statistical and computational algorithms. This rule is the simplest consistent with some experiments and theoretical approaches. The main idea of this approach is to use multilayered networks as an effective preprocessor. Thus this model can be described as a combination of multilayered perceptrons and attractor neural networks.

We have not discussed here some important algorithms such as effective Monte Carlo Bayes approaches (for example, Metropolis and simulated annealing algorithms) [39] or the hidden Markov chains. They can probably also be implemented in our model.

Acknowledgments

This work was supported by the Russian Fund of Basic Research (grant 00-01-00477) and grants 'Tempra' and PAST. I am grateful to the University of Lyon for hospitality.

The author express his great gratitude to N Brunel, J P Nadal and V Volpert for interesting discussions that stimulated this paper. The author also very much thanks two anonymous referees for their interesting comments on the first version of the work and Klebanov for great help in mathematical statistics.

Appendix

To implement in our model the fractal decoding algorithm, we need the following auxiliary instruments: (a) an algorithm of the comparison of two sets (which, for example, can be interpreted as two visual images); (b) the method of fractal generation.

(a) Let us define the distance between the subsets A and B in $\boldsymbol{R}^2$ by the Hausdorff metric

$$d_H(A, B) = \max\{d(A, B), d(B, A)\}, \qquad d(A, B) = \max_{\boldsymbol{x}\in A} \min_{\boldsymbol{y}\in B} d(\boldsymbol{x}, \boldsymbol{y}). \tag{A.1}$$

Actually we are dealing with discrete data. Let us represent the plane $\boldsymbol{R}^2$ as a great two-dimensional lattice, say, consisting of $M = m \times m$ points. Thus we have M different points $\boldsymbol{x}^i$.

Let us now describe a procedure checking the condition $d(A, B) < \epsilon$. We check the following equivalent condition: for each $\boldsymbol{x}^i \in A$ there exists a point $\boldsymbol{y}^j \in B$ such that $d(\boldsymbol{x}^i, \boldsymbol{y}^j) < \epsilon$.

Our network uses (1) a preprocessor calculating the distance $d(\boldsymbol{x}, \boldsymbol{y})$ between two points and (2) two preprocessors with 0–1 outputs η_A and η_B computing the characteristic functions $\chi_A(\boldsymbol{x})$ and $\chi_B(\boldsymbol{y})$.

Now we construct a three-layered network as follows. The first two layers generate the signals

$$\eta_i = \sigma\left(\lambda \sum_{j=1}^{M} \sigma(\epsilon - d(\boldsymbol{x}^i, \boldsymbol{y}^j)) - \frac{\lambda}{2}\right) \tag{A.2}$$

(operation 'or'). These signals enter for the third layer consisting of a single neuron

$$\xi = \sigma\left(\lambda \sum_{i=1}^{M} \eta_i - \lambda \frac{M-1}{2}\right) \tag{A.3}$$

(operation 'and'). It is clear that $\xi = 1$ only if all $\eta_i = 1$, and thus, for each $\boldsymbol{x}^i \in A$ there exists a point $\boldsymbol{y}^j \in B$ such that $d(\boldsymbol{x}^i, \boldsymbol{y}^j) < \epsilon$.

Notice that this calculation will be fast (it takes time $3h$) only if the degree of connectivity N_C of our network (1.1) is more than M (i.e. each neuron is connected with a maximum of $N_C > M$ neighbours).

(b) Let us show that a network of great size and with a *fixed* synaptic matrix can generate a number of dynamics (depending on the thresholds). More precisely, given a finite family of maps $\boldsymbol{F}^i(\boldsymbol{q})$, where $i = 1, 2, \ldots, m$, one can find networks simulating all these maps. The synaptic matrix $\boldsymbol{K}^i \equiv \boldsymbol{K}$ is the same for all the maps, while θ are, in general, different. To simplify, let us assume that we have a scalar input $\boldsymbol{q} = q \in [0, 1]$ and that the output F lies in the same interval [0, 1].

Let us illustrate the main idea of this construction by an example. Consider two maps $q \to f_1(q)$ and $q \to f_2(q)$ from [0, 1] to [0, 1] (here the continuous input $q \in [0, 1]$). Let us construct a network simulating these two maps. The input and output of this network are continuous and lie in [0, 1].

By these maps, we define the map f from [0, 2] to [0, 1] as follows: if $q \in [0, 1]$ then $f(q) = f_1(q)$; if $q \in [1, 2]$, one sets $f = f_2(q - 1)$. There exists a network simulating this discontinuous map (see section 2). The corresponding input lies in [0, 2] while the output lies in [0, 1]. The synaptic matrix and the thresholds are fixed, but our construction is not yet complete since the input lies in [0, 2] (instead of [0, 1]).

However, this network can be replaced by two networks with the same synaptic matrix, the input and the output from [0, 1], and different thresholds. In fact, the argument $q \in [1, 2]$ can be shifted into interval [0, 1] by a shift of the thresholds. Indeed, formula (2.1) shows that the shift of the argument q is equivalent to a shift of thresholds a_i and b_i.

Implementation of the fractal decoding algorithm

Now we describe how a sufficiently large neural network can solve the fractal decoding problem. An approach to this problem is based on the so-called collage theorem [29]. Suppose that a set A' is found such that

$$d_H\left(A', \bigcup_{i=1}^{m} f^i(A)\right) < \epsilon_0. \tag{A.4}$$

Then for small ϵ_0 this set is sufficiently close to the actual attractor A, namely

$$d_H(A', A) < c\epsilon_0, \tag{A.5}$$

where c is a constant depending on our IFS.

Consider a family of maps $\boldsymbol{F}^i$, $i = 1, 2, \ldots, m$. Choosing a multi-index $I = \{i_1, i_2, \ldots, i_l\}$ (where $l \leqslant m$) one has the IFS formed by the maps $\boldsymbol{F}^{i_s}$, $s = 1, \ldots, l$. Each such IFS (and thus the corresponding fractal) can be obtained according to (b), by a choice of thresholds in our network.

If the number l of the maps in the IFS is *a priori* restricted by some value $r < m$ (many realistic pictures can be obtained with small $l = 2, 3, 4$, see [29]), the maximal number M of possible fractals is polynomial: $M = m^r$. Notice that some fractals can be trivial or similar to others, thus actually $M < m^r$, but all the fractals can be generated in a great network (1.1).

Consider now the following problem: given set A', close to a fractal from this family of fractals, to find IFS that generate it.

The solution is based on the results of part (a) (see (A.1)–(A.3)). We can construct preprocessors that calculate the characteristic functions of the given set A' and the set B that is the union of the images $f^{i_s}(A)$ for $i_s \in I$. Using this, we can organize an algorithm checking condition (A.4) as follows.

Let us take a sufficiently small ϵ_0. Furthermore, step by step, we sort out different multi-indices I. This corresponds to an exhaustive search by adjusting different thresholds (or, which is the same, different inputs) of the network. At each step, the network checks (A.4) by the preprocessors described above.

The efficiency of this neuronal implementation depends on the connectivity of the network. Suppose for instance that the connectivity N_C has order 10^4. Then this algorithm works effectively for planar images of size 100×100. Thus, for large images, the procedure must be complicated. Following [30], we can split a large picture into small local parts, 100×100, and apply the method to each part.

References

[1] Hopfield J J 1982 *Proc. Natl Acad. Sci. USA* **79** 2554
[2] Fontanari J F and Meir R 1989 *Phys. Rev.* A **40** 2806
[3] Hopfield J J and Tank D W 1985 *Biol. Cybern.* **52** 141
[4] Brunel N, Carusi F and Fusi S 1998 *Network: Comput. Neural Syst.* **9** 123
[5] Amit D J and Fusi S 1992 *Network: Comput. Neural Syst.* **3** 443
[6] Gardner E 1988 *J. Phys. A: Math. Gen.* **21** 257
[7] Gardner E 1989 *J. Phys. A: Math. Gen.* **22** 1969
[8] Derrida B, Gardner E and Zippelius A 1987 *Europhys. Lett.* **2** 337
[9] Cessac B 1995 *J. Physique I* **5** 409
[10] Sompolinsky H, Crisanti A and Sommers H-J 1988 *Phys. Rev. Lett.* **61** 259
[11] Derrida B 1989 *Helv. Phys. Acta* **62** 512
[12] Parisi G 1986 *J. Phys. A: Math. Gen.* **19** L675
[13] Bliss T V P and Collingridge G L 1993 *Nature* **361** 31
[14] Artola A and Singer W 1993 *Trends Neurosci.* **16** 480
[15] Hebb D O 1949 *Organization of Behaviour* (New York: Wiley)
[16] Konorski J 1948 *Conditional Reflexes and Neuron Organization* (Cambridge: Cambridge University Press)
[17] Tsodyks M V 1990 *Mod. Phys. Lett.* B **4** 713
[18] Amit D J and Fusi S 1992 *Network: Comput. Neural Syst.* **3** 443
[19] Amit D J and Fusi S 1994 *Neural Comput.* **6** 957
[20] Amit D J and Brunel N 1995 *Network: Comput. Neural Syst.* **6** 359
[21] Zhabotinsky A M 2000 *Biophys. J.* **79** 2211
[22] Lalley S P 1999 *Ann. Stat.* **27**
[23] Froland G and Aihara K 2000 *Int. J. Bifurcation Chaos* **10** 103
[24] Solomonoff R A 1964 *Inf. Control* **7** 1
[25] Loveland D A 1966 *Trans. Am. Math. Soc.* **125** 497
[26] Kolmogorov A N 1968 *IEEE Trans. Inf. Theory* **14** 662

[27] Vitanyi P and Li M 1996 *Information, Statistics and Induction in Science* vol 282, ed D L Dowe, K B Korb and J J Oliver (Melbourne: World Scientific)
[28] Grossberg S 2000 *Not. AMS* **47** 1361
[29] Barnsley M F 1988 *Fractals Anywhere* (Orlando, FL: Academic)
[30] Barnsley M F 1996 *Not. AMS* **43** 657
[31] Massopust P R 1994 *Fractal Functions, Fractal Surfaces and Wavelets* (San Diego, CA: Academic)
[32] Cybenko G 1989 *Math. Control Signals Syst.* **2** 303
[33] Hecht-Nielsen R 1987 *IEEE Int. Conf. on Neural Networks* vol 11 (San Diego, CA: SOS) p 11
[34] Funahashi K and Nakamura Y 1993 *Neural Networks* **6** 801
[35] Lichtenberg A J and Lieberman M A 1984 *Regular and Stochastic Motion* (Berlin: Springer)
[36] de Brein N G 1958 *Asymptotical Methods in Analysis* (Amsterdam: North-Holland)
[37] Norman D A 1982 *Learning and Memory* (San Francisco: Freeman)
[38] Judd K and Mees A I 1995 *Physica* D **82** 426
[39] Tanner M A 1993 *Tools for Statistical Inference: Methods for the Exploration of Posterior Information and Likelihood Functions* (New York: Springer)