Chaotic neural control

A. Potapov and M. K. Ali
Department of Physics, University of Lethbridge, 4401 University Drive W, Lethbridge, Alberta, Canada T1K 3M4

(Received 26 June 2000; published 29 March 2001)

We consider the problem of stabilizing unstable equilibria by discrete controls (the controls take discrete values at discrete moments of time). We prove that discrete control typically creates a chaotic attractor in the vicinity of an equilibrium. Artificial neural networks with reinforcement learning are known to be able to learn such a control scheme. We consider examples of such systems, discuss some details of implementing the reinforcement learning to controlling unstable equilibria, and show that the arising dynamics is characterized by positive Lyapunov exponents, and hence is chaotic. This chaos can be observed both in the controlled system and in the activity patterns of the controller.

DOI: 10.1103/PhysRevE.63.046215 PACS number(s): 05.45.−a, 87.18.Sn, 02.30.Xx

I. INTRODUCTION

In this work, we study the characteristics of dynamics generated by discrete controls of unstable equilibria, and also focus on neural network learning techniques for implementing such controls. It is well known that a properly designed control can suppress chaos. We prove that under certain circumstances control of an unstable fixed point can give rise to a chaotic attractor even in linear systems. The main problem is, how do we find a control that can stabilize an unstable fixed point? It turns out that there are methods in machine learning and artificial neural networks [1,2] that can solve this problem. In fact, examples of systems that can learn to perform such a control have been known for a long time, e.g., [3]. Nonetheless, the nature (regular or chaotic) of dynamics involved in such controls has not been studied. By analyzing dynamical regimes, we show that, under discrete controls, chaos necessarily appears in the process of achieving desired (meaningful) goals of the control. Such controls are of interest in the field of artificial neural networks. By applying the technique of reinforcement learning, we demonstrate that neural networks can be trained to realize these controls.

Although our goal in this work is not to explain how the brain works, we would like to digress briefly to some existing works on brain dynamics, because it may be associated with both chaos and control. Currently, we are not aware of any one claiming to know for sure whether the brain dynamics is chaotic, stochastic, or regular but complicated. There are experimental data [4] indicating that the brain dynamics *may* be chaotic. The question is, what is the role and origin of chaos if the brain does use chaos in what it does [4–7]? There is no definite answer to this question; only a number of hypotheses are available. Furthermore, most of these hypotheses lack support from experiments or model calculations.

The tasks of the brain include information processing and control. Some information processing or computing functions are now modeled by artificial neural networks (ANN), and almost none of the widely used ANNs require dynamical chaos as an *essential* element in their performance. Hence, it is worthwhile to study possible connections between chaos and control.

Among other activities, the brain and nervous system do solve problems of posture control and keeping the chemical balance of the body. We think that some essential features of such control mechanisms may be studied with models for stabilizing unstable mechanical or chemical equilibria. Such problems are typical of the theory of optimal control [8]. If the equation of motion and available control actions are known, this problem can be solved with the help of standard analytical or numerical methods [8,9]. However, such methods may not be used by the nervous system. Instead of solving equations, it perhaps learns from trial and error using certain types of experience such as pain or pleasure. In machine learning, such an approach is called ‘‘reinforcement learning’’ (RL) [2,1]. So we have studied model problems regarding chaotic stabilization of unstable equilibria with the help of neural networks that utilize RL techniques. Except for reminding that chaos can yield, under suitable conditions, desirable results, we will not delve any further into the subject of brain dynamics.

The paper is organized as follows. In Sec. II, we give details of the discrete control that we have used in this work. A discrete control implies that the control is chosen and applied at discrete moments of time, and it depends only on the current state of the controlled system. We prove that such a control of an unstable equilibrium can generate chaos. For simple problems, where the unstable manifold of a fixed point is one dimensional, such a control is obvious. For more complex problems, examples of which are presented in Sec. III, special methods of learning are necessary. The basic ideas of reinforcement learning are described in Sec. IV. Section V describes the neural network that we have used, numerical results that confirm chaotic behaviors in the resulting dynamics, and some problems related with application of RL. Section VI contains an illustration of how the chaos in a controlled object can generate complex temporal activity in the controlling neural network.

In short, our studies show that there are at least two possible relationships between control and chaos. Chaos may be a natural result of control. Also chaos may be used as a source of randomlike trials during learning. Earlier, there were attempts to utilize chaos instead of random signals in supervised learning of neural networks. Methods of reinforcement learning, which sometimes require special exploration techniques, seem better suited for utilizing chaos. Per-

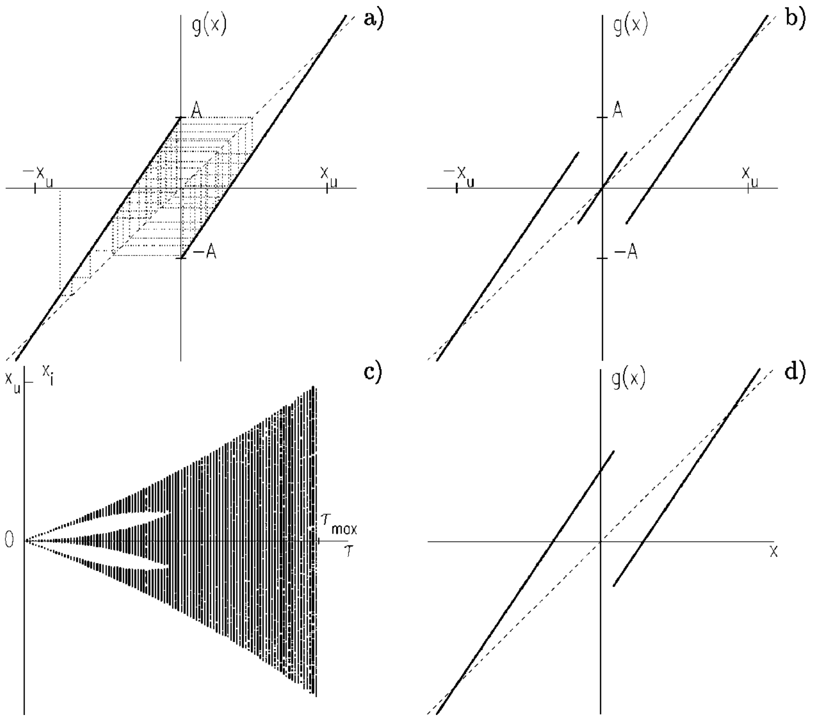

FIG. 1. Mapping resulting from discrete control of unstable fixed point in Eq. (1). (a) mapping (2) and an example trajectory; here $\langle f\rangle=0$; (b) increase in gradations of controlling force diminishes the attractor size, three regions correspond to $f=+f_0$, 0 and $-f_0$; (c) dependence of the size of the attractor on τ for the mapping in panel (a); (d) nonsymmetric mapping with unequal forces, which is used below in one of the versions of cart-pole control. Asymmetry makes $\langle f\rangle\neq 0$, which helps to control the cart position.

haps the brain also generates/utilizes chaos in a similar manner, but there is no evidence in favor or against such a thought.

II. DISCRETE CONTROL CREATES CHAOS

Let us consider a dynamical system $\dot{x}=F(x)$, $x\in R^n$ with an unstable fixed point x_0 $[F(x_0)=0]$. We want to stabilize it by adding a control term *f*. The resulting system takes the form

$$\dot{x}=F(x)+f.$$

The task of the controller is to choose *f* so that the trajectory remains in the vicinity of x_0.

The resulting behavior of the system depends on the actual scheme of control: *f* may take any value between $f_{\min}$ and $f_{\max}$, or it can have values only from a discrete set; the controller may choose *f* continuously or only at some discrete moments of time. In typical problems of optimal control of technological systems [8], the control is usually continuous both in *f* and time. However, the so-called ''bang-bang control,'' with only extremal values of *f*, often proves optimal; one needs to find the switching curves or surfaces in phase space, where the value of *f* should change. For the brain, it is perhaps more appropriate to choose the discrete scheme. First, all signals are transmitted by discrete pulses. Second, during certain periods, the neurons are in such states that they are unable to process incoming signals.

In this paper we shall consider the case of discrete control with the following characteristics: the controlling action *f* can take only discrete values, usually from a finite set; the specific value of *f* is chosen by the controller at discrete moments of time $t_k=k\tau$, and after it has been chosen, it remains constant until the next decision is taken; the choice of action depends only on the state of the controlled system $x(t_k)$ at the moment t_k.

From this definition, it follows that the controller should split the phase space into domains according to the chosen value of *f*. Within each domain, *f* is constant. Therefore, almost everywhere $\partial f/\partial x=0$.

Discrete control of continuous-time dynamical systems has been used in several problems of machine learning, see e.g. [3,10]. We shall show that it may result in chaotic behavior of the controlled system, even if the latter is linear.

Let us consider the simplest unstable system

$$\dot{x}=\lambda x+f,\quad \lambda>0, \tag{1}$$

where *f* is the controlling ''force.'' Let us assume that, at some moment, the value $f=f(x(t_k))$ has been chosen. Then *f* remains constant until $t_{k+1}=t_k+\tau$, and on this interval, Eq. (1) can be solved analytically to obtain the expression for $x(t_{k+1})$. It is convenient to consider the mapping $x(t_k)\to x(t_{k+1})$. Let us denote $x_k=x(t_k)$, then $x_{k+1}=e^{\lambda\tau}x_k+(e^{\lambda\tau}-1)f/\lambda$.

The properties of this mapping depend on the choice of *f*. In the simplest case of binary control $f=\pm f_0$ [3,10], the control should return a trajectory back to the fixed point: $f=-f_0\mathrm{sgn}(x)$. This gives the following one-dimensional mapping:

$$x_{k+1}=g(x_k),\quad g(x)=\begin{cases}e^{\lambda\tau}x-A, & x\geqslant 0\\ e^{\lambda\tau}x+A, & x<0,\end{cases} A=\frac{(e^{\lambda\tau}-1)f_0}{\lambda}. \tag{2}$$

The plot of $g(x)$ is shown in Fig. 1(a). The mapping (2) has two unstable fixed points $\pm x_u$,

$$x_u = \frac{f_0}{\lambda}.$$

For initial data with $|x| > x_u$, iterations of x diverge. But for $|x| < x_u$, the trajectory always remains within the domain $-x_u < x < x_u$, provided

$$A < x_u, \quad e^{\lambda\tau} < 2. \tag{3}$$

In fact, the trajectory stays within the interval $-A \leqslant x \leqslant A$. Since $g'(x) = e^{\lambda\tau} > 1$ almost everywhere within this interval, there should be a chaotic attractor. Its dimension is 1 and the Lyapunov exponent is equal to λ. Therefore, the discrete ''bang-bang'' control adds a necessary *delayed feedback*, which creates chaos in a single differential equation.

The relations in Eq. (3) show that the control interval τ cannot be too large,

$$\tau < \lambda^{-1} \ln 2.$$

The smaller is τ, the closer is the trajectory to the point $x = 0$ (in the limit $\tau \to 0$ we have $A \approx \tau f_0 \to 0$—this is the situation of optimal continuous bang-bang control). At $\tau = \tau_{\max} = \lambda^{-1} \ln 2$, the value of A becomes equal to x_u, attractor ''collides'' with unstable fixed point and disappears via crisis [Fig. 1(c)].

It is interesting that, for too large τ, the trajectories can escape from the attractor only near the point $x = 0$. Therefore, to allow larger τ, one can change the mapping in the vicinity of the origin by introducing the third control value $f = 0$. For smaller τ values, the attractor shrinks closer to the desired state $x = 0$ [Fig. 1(b)]. Therefore, the efficiency of control increases with the number of available grades of f. Note that an increase in gradation does not change the value of the Lyapunov exponent, it is *always* equal to λ.

An equally simple problem is that of stabilizing a pendulum in its highest point. The equation of motion in the vicinity of the equilibrium point has the form

$$\ddot{x} = \omega^2 x + f. \tag{4}$$

It is convenient to transform the second-order ordinary differential equations (ODE) to a system of two equations of the first order: let $y = \omega^{-1}\dot{x}$, then $\dot{x} = \omega y$, $\dot{y} = \omega x + f/\omega$. Now, if we use the change of variables $u = x + y$, $v = x - y$, then these equations take the form

$$\dot{u} = \omega u + f/\omega, \tag{5}$$

$$\dot{v} = -\omega v - f/\omega. \tag{6}$$

The first equation describes motion along the unstable manifold of the saddle, while the second equation describes it along the stable manifold. As any special control of the second equation is unnecessary, the problem of inverse pendulum stabilization is reduced to Eq. (1), but the controlling force should depend on the sign (and maybe the value) of $\omega x + \dot{x}$. The discrete control described above brings up a chaotic attractor with Lyapunov exponents $\{\omega, -\omega\}$.

We can generalize these examples as stated in the following theorem.

Theorem 1. Let a linear dynamical system with the discrete control f be described by equation $\dot{x} = Ax + f$. Suppose the origin ($x = 0$) is an unstable equilibrium with the set of Lyapunov exponents $\lambda_1, \ldots, \lambda_n$ when there is no control ($f = 0$). If the control keeps the trajectories within a bounded domain, then the dynamics in this domain will be chaotic with a set of Lyapunov exponents identical to that of the equilibrium point without the control.

The proof is obvious. According to the definition of a discrete control (see above), we have $\partial f/\partial x = 0$ almost everywhere. This means that the Lyapunov exponents of the controlled system will be defined by the linear system $\dot{u} = Au$. But this is just the original system without control and so the set of Lyapunov exponents is the same.

Similar arguments apply to linear mappings (systems with discrete time) with discrete controls.

We now consider nonlinear system with unstable equilibrium points. If the controlled trajectories of a nonlinear system stay sufficiently close to an equilibrium point, we have a case similar to a linear system discussed above. However, the Lyapunov exponents of the controlled and uncontrolled systems will generally be different. Only a weaker statement can be proved.

Theorem 2a. Let a nonlinear dynamical system with the discrete control f be described by $x_{k+1} = F(x_k) + f$, where F has the continuous derivative $DF(x)$. Let the origin ($x = 0$) be an asymptotically unstable equilibrium with the set of Lyapunov exponents $\lambda_1, \ldots, \lambda_n$, $\lambda_1 > 0$ when there is no control ($f = 0$). If the trajectories under control remain within a small enough ball centered at the origin, then the dynamics in it is chaotic with at least one positive Lyapunov exponent.

Let us denote $B = DF(0)$, then $\lambda_1, \ldots, \lambda_n$ are the Lyapunov exponents of the linear system $u_{k+1} = Bu_k$. If $|x| < \delta$, the continuity condition implies $|DF(x) - DF(0)| < \epsilon$ for some δ and ϵ. Therefore, we can write $DF(x) = B + \epsilon C(x)$ with some continuous and bounded $C(x)$ that vanishes at $x = 0$. By choosing δ small enough, it is possible to have ϵ as small as necessary. Now, the Lyapunov exponents of the controlled nonlinear system should be obtained from the linear system $v_{k+1} = (B + \epsilon C_k) v_k$. It is clear that, for small enough ϵ, there should be exponentially growing solution and there should be at least one positive λ, as in the case with $\epsilon = 0$. The proof of this statement is given in Appendix B.

If we consider $\dot{x} = F(x) + f$ as a system with continuous time, its Lyapunov exponents should be determined from $\dot{u} = DF(x)u$. As in the previous case, it is possible to write $DF(x) = A + \epsilon G(x)$, where $A = DF(0)$. Now, we can go from a differential equation to its ''stroboscopic map'' $x_{k+1} = x(t_k + \tau) = B_k x(t_k)$. Again due to continuity $B_k = B + \epsilon C_k$, $B = \exp(\tau A)$. This is a problem we have already considered; the only difference is that λ_i should be replaced by $\tau\lambda_i$. This proves the following theorem.

Theorem 2b. Let there be a nonlinear dynamical system described by $\dot{x}=F(x)+f$, where F has continuous derivative $DF(x)$ and f is the discrete control. Let the origin $(x=0)$ be an asymptotically unstable equilibrium with the set of Lyapunov exponents $\lambda_1, \ldots \lambda_n$, $\lambda_1>0$ when there is no control $(f=0)$. If the trajectories under control remain within a small enough ball centered at the origin, the dynamics in this domain is chaotic with at least one positive Lyapunov exponent.

The above theorems characterize the dynamics under control, but they do not show how to organize such a control. From the examples above we can see that discrete control does not convert an unstable fixed point into a stable one. Instead, it *shifts* its position. Then, moving *away* from the new unstable point (in the example above $x=\pm x_u$) may correspond to moving *towards* the original equilibrium ($x=0$). Such shifts should be made along the unstable and possibly central manifold of the original equilibrium. Therefore, the complexity of the stabilization problem depends on the dimensions d_u and d_c of these manifolds.

If $d_c+d_u=1$, as in the examples above, then the control strategy is obvious. But if the new position of the unstable point has to ''dance'' in a higher-dimensional space, then intuition rarely can give ready recipe for the control. Let us consider two examples where $d_c+d_u=2$ and 3.

III. EXAMPLES OF MORE DIFFICULT CONTROL TASKS

Note that the tasks considered in this section are difficult not from the viewpoint of the modern control theory [8], which we do not use here. The difficulty lies in the fact that there is no *obvious* control strategy for them. We want to see, how these tasks can be carried out by neural networks with reinforcement learning [11,3,12,1,2]. As sample cases, we have chosen one mechanical and one chemical system.

The mechanical problem is the so-called cart-pole balancing problem [3,2], Fig. 2 (sometimes called broomstick balancing [13]). There is a cart, which 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. Therefore, 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).

The control task is as follows: After every time interval τ, the controller receives the values of the cart-pole state variables $x, \dot{x}, \theta, \dot{\theta}$. The controller can apply a force equal to $\pm f$ to the cart, which 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 $[-x_{\max},x_{\max}]$. If either θ or x falls out of their limits the controller receives a failure signal, and then the cart is returned to the position $x=0$, and the pole is set at some small angle θ_0.

The parameters of the system were taken from [3]. They, along with the equations of motion (A7), are presented in

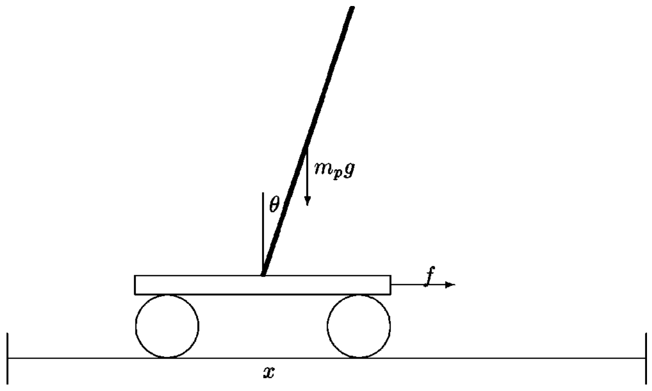

FIG. 2. The cart-pole balancing task. Controller should choose the proper direction for f after each time interval τ such that the angle θ for the pole will remain within $[-\theta_{\max},\theta_{\max}]$, and the cart never hits the ends of the track, $-x_{\max}<x<x_{\max}$. In the beginning, the cart is positioned at the middle of the track $x=0$ and the pole is set at some angle θ_0 that is within the admissible limits. In numerical calculations $\theta_{\max}=12°$, $x_{\max}=2.4$, and in most presented examples $\theta_0=-8°$ was used. The main conclusions, naturally, do not depend on θ_0, though for $|\theta_0|$ less than about 2.9° the control task becomes too easy — just the simple control strategy for an inverted pendulum solves the problem.

Appendix A. But the basic control mechanism and the main difficulty can be understood from the linearized equations near the fixed point $(x=\dot{x}=\theta=\dot{\theta}=0)$,

$$\ddot{x}+\frac{2l}{3}\ddot{\theta}=g\theta, \tag{7}$$

$$\ddot{x}+\frac{m_p l}{2(m_c+m_p)}\ddot{\theta}=\frac{f}{m_c+m_p}, \tag{8}$$

with the initial conditions $x(0)=\dot{x}(0)=\dot{\theta}(0)=0$, $\theta(0)=\theta_0\neq 0$. Here m_c is the mass of the cart, m_p and l are mass and length of the pole, g is the acceleration due to gravity, and friction terms are neglected. For the parameters used, $m_c=10m_p=1$, $l=0.5$, $g=9.8$ and $|f|=10$, so the factor at $\ddot{\theta}$ in the second equation is very small (1/44), and we get almost ''two-stage'' control: the force f accelerates the cart, while cart's acceleration works as a control force for the inverted pendulum.

It may seem that this problem is equivalent to the control of an inverted pendulum (Sec. II). One can exclude $\ddot{x}$ from Eq. (7), and the equation for θ coincides with Eq. (4), $\ddot{\theta}=\omega^2\theta+\phi$, where ω and ϕ can be found from Eqs. (7) and (8). But experiments show that the simplest strategy, when f depends only on the sign of $\omega\theta+\dot{\theta}$, works only when θ_0 is so small that it falls within the resulting chaotic attractor, $|\theta_0|\leqslant\theta_{0\max}\approx 2.9°$. Exact comparison of $\theta_{0\max}$ with A for the problem (1) is hard, since the variable x in Eq. (1) corresponds to $\theta+\omega^{-1}\dot{\theta}$ rather than θ. For $\theta_0>\theta_{0\max}$, while θ converges to the chaotic attractor, the cart acquires speed. This initial speed is not related with the current values of θ

or $\dot{\theta}$, and hence cannot be compensated by a ''pendulum'' control. As a result, after some time, depending on θ_0, the cart hits the end of the track.

It is easy to show that the linear problem described by Eqs. (7) and (8) has the eigenvalues $\{\omega,0,0,-\omega\}$. The two zero eigenvalues correspond to a translation degree of freedom. Therefore, the control should be organized in three dimensional subspace, and it is hard to organize it intuitively. Comparison with inverted pendulum shows the role of the center manifold.

The second control problem is related with chemical reactions. We have chosen the well-known system with oscillatory behavior, the Brusselator [14]. In specially chosen variables it has the form

$$\dot{x}=c-(b+1)x+x^2y,$$

$$\dot{y}=bx-x^2y.$$

Here *x*, *y*, *c*, *b* are the concentrations of reacting chemicals. It is supposed that *c* and *b* are kept constant, while *x* and *y* can vary with time. It is easy to check that this system has a fixed point $x=c$, $y=b/c$. If we fix $c=c_0=1$ and vary *b*, then for small $b<2$ the fixed point is stable. At $b=2$ the Hopf bifurcation occurs and oscillatory behavior can be observed. For the numerical experiment, we have chosen $b=b_0=3$. The amplitude of the limit cycle for this choice $b=b_0$, is about 3 in both *x* and *y*. The task was the following: the controller receives the current values of *x* and *y*. Then it can set the values of $c=f_1c_0$ and $b=f_2b_0$, where the factors f_1 and f_2 can be equal to 0.8, 0.9, 1.0, 1.1, or 1.2. These values of *c* and *b* remain unchanged during the next controlling period of the length τ. It is necessary to keep the values of *x* and *y* in the vicinity of the point (1,3). At $c=1$, $b=3$ this point is an unstable focus with the eigenvalues $\lambda_{1,2}=(1\pm\sqrt{3}i)/2$, therefore the unstable manifold is two dimensional. There is also no obvious way of control in this case.

To solve these model problems, we used the methods of reinforcement learning. Note that the solutions for the cart-pole problem were published long ago, but we are interested here in the analysis of the resulting dynamics.

IV. BASICS OF REINFORCEMENT LEARNING

In order to explain the learning algorithm, we have to say a few words about reinforcement learning (RL). Sometimes it is called ''learning with a critic.'' The main task of RL is to work out optimal sequence of control actions to achieve a goal based only on evaluative feedback, when there are no examples of successful control.

RL has been known for about 50 years, but rapidly increasing interest in it arose in the late 1980s and 1990s, when several approaches (theory of Markov decision processes or dynamical programming, eligibility traces, temporal difference learning) merged together forming the modern theory of RL [2,1]. Currently, it is not as widely known as, for example, neural networks. Its theory is under active development. Its recent applications include control of complex mechanical systems, navigation of robots, elevator dispatching, playing backgammon, optimal chaos control and others [2,1,15,16].

Now we discuss some basic concepts of RL. More details about RL can be found, for example, in the excellent introductory book [2] and a number of papers, e.g., [1]. The general scheme of RL includes (i) an *environment* which is characterized by its state *s*, (for example, a system to be controlled, such as the cart-pole) and (ii) an *agent*, (controller) who can perform action *a* (in the cart-pole example, the action is application of selected forces). 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 agent receives information about the next state s_{t+1} and a scalar signal r_t called *reward*. The reward may evaluate either the consequences of the last action or just the current state of the environment after a long sequence of actions. In the latter case the reward is delayed. For example, while the cart and the pole are within the specified limits, the reward is zero. When θ or *x* goes out of its limit, the reward is -1. Another example is the game of checkers or chess, where the reward is the outcome of the game. Negative rewards sometimes are called ''punishment'' or ''penalty.'' The rule, according to which the agent chooses its action $\pi(s,a)$ is called *policy*. Usually a policy 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* that brings maximal total reward during a large or infinite number of actions. In the latter case the sum $\Sigma_{k=1}^{\infty}r_k$ may be infinite, and the discounted total reward used is $\Sigma_{k=1}^{\infty}\gamma^{k-1}r_k$, $0<\gamma<1$ (usually $|r_k|$ is bounded).

Interaction of the agent and the environment is usually described in terms of the transition probabilities $P=P(a,s,s')$ from the state *s* to the state s' after the action *a*, and by the value of the reward $R=R(a,s,s')$ after such transition. The value of r_k in numerical experiments is always equal to a value of *R*. If all *P* and *R* are known, then the optimal policy can be found by the methods of *dynamic programming* (DP) [17,8,2] [sometimes the term *Markov decision process* (MDP) is used]. Within this approach, it is convenient to define the *value* $V^{\pi}(s)$ of each state *s* as the expected total discounted reward, if the agent starts to act from this state and follows the policy π. Similarly, one can define the value $Q^{\pi}(s,a)$ of each action *a* at the state *s*. The set of *V* or *Q* is a solution of a certain linear system of equations. Values of *V* and *Q* are closely related: if one knows all *V*, *P* and *R*, it is easy to calculate all *Q* and vice versa.

To obtain an optimal policy, the method of *policy iteration* is used. If the agent chooses an action with the largest Q^{π} [instead of the current policy $\pi(s,a)$], then for this new policy π_1 all $V^{\pi_1}\geqslant V^{\pi}$ and $Q^{\pi_1}\geqslant Q^{\pi}$. This new policy in turn can be improved and so on. It has been proven that this process converges to an optimal policy with action values $Q^*(s,a)$. On the other hand, if one knows all $Q^*(s,a)$, then the policy itself can be easily found: every time one must choose the action with the largest Q^*. One of the RL algorithms, described below—the Q learning—estimates the values Q^* almost independently of the current policy π.

It should be noted that rigorous results correspond only to the case where the state s is Markovian, that is, all the transitional probabilities depend only on s and a, but not on the previous history. This allows one to write a system of equations for V or Q, which simply relate their estimates at two sequential time steps,

$$V^{\pi}(s)=\sum_{a} \pi(s,a)\sum_{s'} P(a,s,s')[R(a,s,s')+\gamma V^{\pi}(s')]. \tag{9}$$

Sometimes, in practice, methods based on MDP approach prove very efficient even in cases where states are non-Markovian.

The situation becomes more complicated when the transition probabilities are unknown, and the agent has to work out the best policy only *from its own experience* obtained from previous actions and rewards. Modern algorithms of RL can find such a policy in an efficient way. They use the concepts of values of states and actions from dynamical programming, along with some ideas from learning theories. There are two main classes of methods [2,1]: *Monte Carlo* , where V and Q are estimated as averaged reward received after corresponding state and action (it requires many repetitions), and *temporal differences* (TD), where the estimates of these values are received by iterations.

According to [2,1], TD methods are more popular. They are based upon the following simple recursive relation: When we are trying to estimate V *experimentally*, the estimates for different states are related as $V(s_t) =\sum_{k=0}^{\infty}\gamma^k r_{k+t+1}=r_{t+1}+\gamma V(s_{t+1})$, and therefore $\Delta_t=r_{t+1} +\gamma V(s_{t+1})-V(s_t)=0$. Note that for the *exact* V values, satisfying Eq. (9), only $\langle\Delta_t\rangle=0$, where averaging is taken by all possible actions a_t and all possible states s_{t+1} after them. Nonetheless, it has been found [2,1] that the current values of Δ_t, instead of the average values, can be efficiently used for V updates,

$$\hat{V}(s_t)=V(s_t)+\alpha_t\Delta_t. \tag{10}$$

Initial values for V may be arbitrary, e.g., $V=0$.

Similar updating scheme exists [2] for the action values,

$$\hat{Q}(s_t,a_t)=Q(s_t,a_t)+\alpha[r_{t+1}+\gamma Q(s_{t+1},a_{t+1})-Q(s_t,a_t)]. \tag{11}$$

It has been called the ''sarsa'' scheme because of the string $s_t,a_t,r_{t+1},s_{t+1},a_{t+1}$. The sarsa scheme provides Q estimates for the current policy, but it can be easily combined with policy improvement: decision regarding which action is to be taken, can be made from the latest Q values. Another well-known approach is ''Q learning.'' The relation for it is

$$\hat{Q}(s_t,a_t)=Q(s_t,a_t) +\alpha[r_{t+1}+\gamma\max_{a} Q(s_{t+1},a)-Q(s_t,a_t)]. \tag{12}$$

The resulting estimates of Q correspond to an optimal policy and there is no need for policy improvement (though, the current policy should make it possible to evaluate Q for all essential states and actions).

For the choice of α, usually there is no strong theoretical restriction. Proof of convergence for these methods [2,1] requires that α_t should slowly decrease with time so that $\Sigma_t\alpha_t=\infty$, $\Sigma_t\alpha_t^2<\infty$ (for example, $\alpha_t=\alpha_0/t$). Sometimes, the methods work well for nondecreasing α_t, which may even not be very small (for some problems even $\alpha=1$ is the best choice [1]).

The *rate* of convergence of RL methods (both TD and Monte Carlo) does not follow from the theory. Such results are empirical and based upon a large number of computer experiments. For any given system, the methods may behave differently. However, for working out a good policy, one may not need a *precise* estimate of Q, in fact, one needs only to know which Q is the largest. Experiments show [2,1] that TD methods usually provide such information faster.

Efficient implementation of these methods include two more ideas: eligibility traces, which come from learning theories [18], and exploration/exploitation tradeoff, which is a common problem in stochastic control.

The idea of eligibility traces was developed originally for model neurons to reproduce effects similar to conditional reflexes [18]. 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 $\Delta w_{ijt}\sim e_{ijt}r_t$, where r_t is a signal generated by ''essential'' event. Initially, all $e_{ij0}=0$. When 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 new resetting.

In the reinforcement learning, this idea has been implemented as an attempt to solve the problem of *credit assignment*. When the reward is a result of many actions a_t, $a_{t-1}\ldots$, one would like to know, which of these actions is responsible for r_{t+1}. How do we assign a correct credit to each a_t, so that we have the correct updating of value functions and improving policy? In general, the answer to this question may not exist, 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 (or state-action pair) there is a variable $e(s)$ [or $e(s,a)$]. It is set to 1 every time the environment is in the state s (and an action a is chosen). Then $e(s)$ exponentially decays to 0 unless the state returns to s. This helps updating simultaneously many V or Q values, which are relevant to the latest actions: $\hat{V}(s) = V(s) + \alpha_t e(s)\Delta_t$, or $\hat{Q}(s,a) = Q(s,a) + \alpha_t e(s,a)\Delta_t$. According to [2,1], eligibility traces may significantly accelerate RL especially in the case of delayed reward in which nonzero r appears after many actions.

The task considered may be *episodic*, that is, some of the states s_i are *terminal*. When the system falls into a terminal state, the process stops. For subsequent learning, another episode should begin. Note that the cart-pole balancing problem has been formulated as an episodic task. When the conditions for x or θ are violated, the system is considered to be in a failure state, after which it is reset. For episodic tasks, when a new episode begins, the eligibility traces also should be set to zero.

Another problem is related with the following contradiction: to increase performance it is necessary to follow the best policy, that is, to select actions with the largest Q estimate (such policy is called *greedy*). But to get correct Q estimates, it is necessary to try all or a large number of actions, most of which are inefficient. The learning systems must *explore* possible actions. This exploration is achieved usually by two methods: (i) Set high initial values for all states/actions. This encourages the learning system to try all states and actions, but only during some initial time interval. (ii) Follow nondeterministic policy, choosing nonoptimal actions with small probability ϵ (ϵ-greedy policy). Also, exploration is necessary when the environment is nonstationary. Then the optimal policy may change with time, and the learning agent should be able to find the new one.

All these methods can be generalized for the case of continuous states and continuous actions [2,1]. This substantially increases capabilities of the methods, but at the same time creates new problems with $Q(s,a)$ or $V(s)$ approximation. For continuous actions, there is also the problem with the choice of the best action. In this paper we wanted to study some details of application of RL to dynamical systems and limited ourselves to the simplest implementations of RL.

It is important to note that, despite the relative simplicity of general idea, any specific implementation of RL may require a thorough accounting of the details. For example, performance of the methods sometimes depends on the proper choice of the parameters α, γ, λ, ϵ (examples are shown below). Also, if the system is non-Markovian (as it is often the case in practice), then there is no proof of convergence of the above technique. It is known that in such cases the results of Q learning may oscillate between several policies [1] (examples will be shown). Another possible source of oscillations is pointed out in [19]: convergence in the ''policy space'' may be rather complex and may resemble a stable focus. Convergence is accompanied by wandering between various ''neighboring'' policies.

We should also note that the described approach of value functions is the most general, but not the only one in the theory of RL. For other approaches see, e.g., [1,11,12,20,15] and references therein.

V. NEURAL CONTROLLING ARCHITECTURE AND NUMERICAL RESULTS

A. The problem of phase space partitioning

To apply a *discrete* version of RL to the problem of control of a dynamical system with *continuously* changing state, it is necessary to make a partitioning of its phase space $\Omega = \cup A_i$, $A_i \cap A_j = \varnothing$. In [2], it is called ''tile coding.'' Each domain A_i of this partition corresponds to one ''state'' s_i, which determines a controlling action. Then with the help of RL, it is necessary to find the best policy.

From the view point of the theory of Markov decision process, the main requirement is that this partitioning should be Markov. That is, the probability with which a trajectory jumps from partition A_i to A_j must not depend on its previous history. Such Markov partitions are used, e.g., in ergodic theory of dynamical systems [21]. But usually it is very hard to build such a partition and to prove whether it is Markov or not. It is known that the methods of RL may work for non-Markov partitions very well. On the other hand, non-Markov partitioning may be a cause for their poor performance.

The main problem with partitioning is that it should allow a good control policy. If it does, then most probably RL will find this policy. If it does not, nothing can help. The problem of partitioning seems to be one of the hardest, and it requires a good intuition, experience, and *a priori* information about the problem [1] (for example, in the problem of stabilizing a fixed point most important is the partitioning of the unstable and center manifolds). If the partitioning has to be very fine, then it may be better to use a different kind of coding. For example, one can use a continuous approximations of dependence $\langle$point of phase space$\rangle \rightarrow \langle$control action$\rangle$ [1,2] with the help of a neural network. But our model problems allow to work with rather coarse partitions, so we used the simplest approach. In [3,10] the partition of the cart-pole phase space has been done with the help of so-called ''boxes'' — rectangular cells of different sizes. But, as our experiments show, self-organizing neural network provides better partitioning.

One of the efficient mechanisms of partitioning is the use of vector quantizing (VQ) networks or self-organizing maps (SOM) [22–24]. These networks have a set of reference vectors $\mathbf{v}_k$ (input connections), and they split the whole space into so-called Voronoi or Dirichlet cells: the ith cell consists of the points $\mathbf{x}$, for which the vector $\mathbf{v}_i$ is the nearest of all $\mathbf{v}_k$ ($\mathbf{x} \in A_i$ if $\mathrm{argmin}_k |\mathbf{x} - \mathbf{v}_k| = i$). Such partition will depend on the number and location of the vectors $\mathbf{v}_k$, and on the metric used for calculation of distances.

So, if $\mathbf{x} \in A_i$, then the VQ network activates the ith neuron. We associate with it the values of $Q(s_i, a)$, corresponding to the state s_i and operations, related with the RL and the choice of control action (so this should be an ''intellectual neuron,'' which may represent a neurons ensemble or a separate network). This neuron receives current reinforcement signal, calculates the value of Δ_t, updates Q, chooses the next control action, and sends this information to other neurons and the corresponding f to the controlled system (Fig. 3).

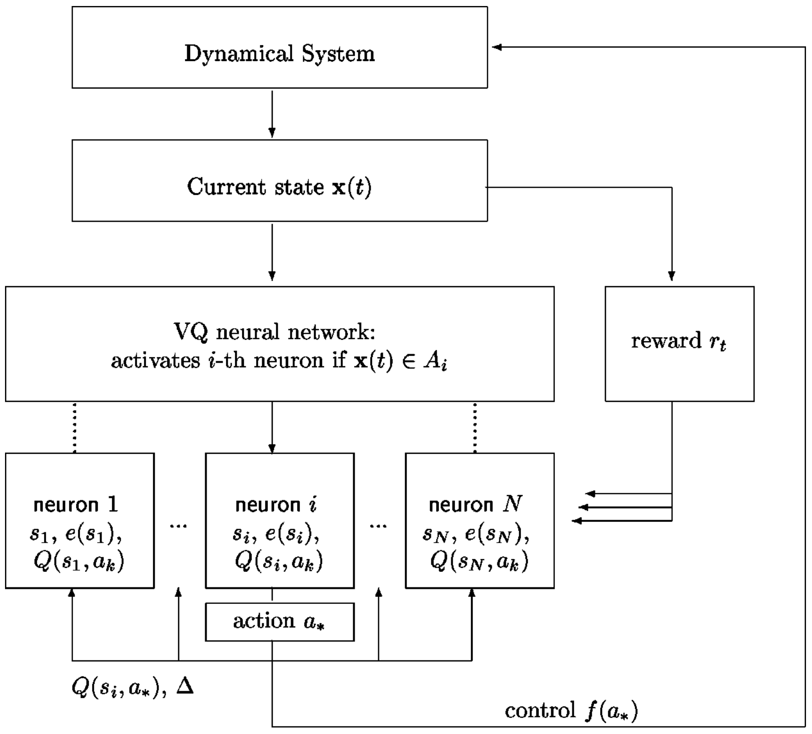
Dynamical System
Current state $\mathbf{x}(t)$
VQ neural network: activates i-th neuron if $\mathbf{x}(t) \in A_i$
reward r_t
neuron 1 s_1, $e(s_1)$, $Q(s_1, a_k)$
neuron i s_i, $e(s_i)$, $Q(s_i, a_k)$
neuron N s_N, $e(s_N)$, $Q(s_N, a_k)$
action a_*
$Q(s_i, a_*)$, Δ
control $f(a_*)$

FIG. 3. Scheme of the simplest neural controller with reinforcement learning. Information about current state **x** of dynamical system goes to vector quantizing network, which determines, which domain A_i it belongs to. Then the corresponding neuron is activated: it calculates the updating term Δ (11) or (12), updates $Q(s_i, a_k)$ if $e(s_i) \neq 0$, and chooses the next action a_* from the set of possible $\{a_k\}$. Then it sends the values of Δ (for current update) and $Q(s_i, a_*)$ (for calculating Δ next time) to other neurons, and the necessary control $f(a_*)$ to the dynamical system. The reward r is estimated from the current state **x** by a separate subsystem, e.g., a set of special sensors, then it goes to neurons for Δ calculating.

One of the advantages of SOM networks is the ability to adjust the partitioning to the available set of ''experimental'' data points, or even reproduce their distribution density. Among numerous applications of such networks [22], there are applications in control, e.g., robot movements [22,15]. But standard algorithms of SOM learning are not good for our problem. For this reason we applied another type of VQ network, which can be called ''self-developing network.'' It is based upon the idea of network growth (see, e.g., [25]). The algorithm was very simple: we placed a new neuron at the current trajectory point when the nearest neuron to it proved farther than some threshold distance d_0. The values of Q for this new neuron are copied from its closest neighbor. Performance of this technique depends on the value of d_0, and on the choice of the metric in phase space. As our experience shows, it is not hard to find appropriate values by trial and error method. At the initial stages of learning, this approach leads to variable number of the states, but both sarsa and Q-learning methods can be easily adopted to it.

B. Cart-pole control

In the work [3], the ''boxes'' partitioning of phase space was taken from another paper on machine learning [10]. The allowed limits on x and θ were $|x| < x_{\max} = 2.4$ and $|\theta| < \theta_{\max} = 12°$, respectively. The partition was organized as a number of rectangular cells, formed by hyperplanes located at $x = \pm 0.8$, ± 2.4, $\theta = 0°$, $\pm 1°$, $\pm 6°$, $\pm 12°$, $\dot{x} = \pm 0.5$, $\pm \infty$, $\dot{\theta} = \pm 50$, $\pm \infty$. This gives a partition of $3 \times 6 \times 3 \times 3 = 162$ cells. The control technique was the simplest ''bang-bang'' — application of the force f equal to either $+f_{\max}$ or $-f_{\max}$ to the cart, where $f_{\max} = 10$. As it was shown in [3], the RL methods were able to find a good policy, and the pole was kept upright for a long time.

We repeated their results, and studied the problem in more detail, because we wanted to explore both the characteristics of the regime arising under control and the capabilities of the RL techniques. We used the Q learning (12) and sarsa (11), which did not exist when the work [3] was published.

As it can be expected, the control occurred in a chaotic regime. The estimated values of Lyapunov exponents [21] were very close to the eigenvalues of the linearized systems (7) and (8) $\{\omega, 0, 0, -\omega\}$, $\omega \cong 5.62$. The typical set of Lyapunov exponents obtained numerically for the original nonlinear model (see Appendix A) was $\{5.63, 0, 0, -5.63\}$ in good agreement with the linear analysis. They slightly varied for different control schemes, but the difference with the analytical estimates never exceeded 0.1.

From the linearized equations (7) and (8), one can get the equation for the pole alone, $\ddot{\theta} = \omega^2 \theta + \phi$, $|\phi| = 29.3$. The results of Sec. II give the estimate for the upper limit of the control interval as $\tau_{\max} = \omega^{-1} \ln 2 \cong 0.12$. But RL methods could not find a good policy for τ close to $\tau_{\max}$, because the cart rather quickly hit the end of the track. It proves that the control interval ($\tau = 0.02$) used in [3] is close to the practical upper limit for this control scheme. We also used this τ value.

To check the performance of RL algorithms, we wanted to find a coarser partition. With it, all possible policies can be tested to find the best one. Then the results of RL methods can be compared with this best policy. Note that the partition from [3] gives 2^{162}–10^{50} different policies. Hence, it is practically impossible to check whether a better one exists or not. As our results show, the partition indeed may be coarser, but the problem should be slightly nonsymmetric.

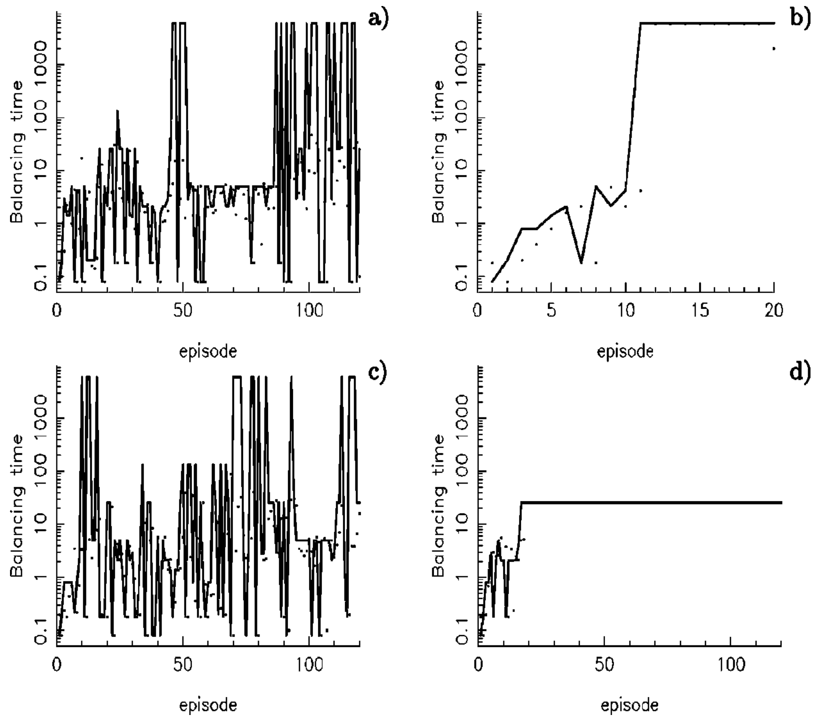

FIG. 4. The problem of cart-pole control, 12-cell boxes partition of phase space. Examples of Q-learning performance for different parameters. Shown is the balancing time for learned greedy policy after each episode. Testing of the balance time stopped after $t=6000$, and this value corresponds to the best policy. Dots show the duration of the corresponding episode. For all examples the initial position of the pole was the same $\theta_0=-8°$. (a) $\alpha=0.1$, $\beta=0.01$, $\gamma=\lambda=0.5$, $\epsilon=0.1$ (ϵ-greedy policy); (b) same as panel a, but $\epsilon=0$ (purely deterministic policy, exploration is due to system's chaoticity); (c) $\alpha=0.1$, $\beta=0.01$, $\gamma=0.95$, $\lambda=0.5$, $\epsilon=0.1$; (d) same as (c), but $\epsilon=0$. Therefore, policy randomness may both improve and spoil the performance. Optimal values of parameters do not follow from theory, but can be found in numerical experiments.

1. Coarse boxes partition

As we mentioned in Sec. III, the problem with the cart position control arises because, for large initial angles $\theta(0)$, the cart acquires some speed during the convergence of θ to the attractor. With a large number of partition cells the cart sometimes gets more pushes to the right or to the left and this helps compensate for initial and all subsequent impulses. But similar effect can be easily obtained if the controlling forces acting to the left and right are made unequal, say, $-f_{\max}$ and $+0.9f_{\max}$, the partition of unstable manifold is chosen to be unsymmetrical. This way it is easier to control the motion of the cart. It is necessary only to switch between the "left" and "right" modes at proper times. This can be done by partitioning the center manifold: the important factors are the signs of x and $\dot{x}$ (where the cart is and where it is moving to).

The unstable manifold corresponds to the variable $u=\omega\theta+\dot{\theta}$, and for unsymmetrical partitioning we set the threshold values of u equal to ±10, $\pm\infty$. For the center manifold, we used only zero threshold for both x and $\dot{x}$. This gives only $3\times2\times2=12$ cells and $2^{12}=4096$ control policies. It is easy to find an optimal policy just by trying them all without any learning. Thus, we can check how good is the solution obtained with the help of RL.

The "balancing times" t_B, for most *deterministic* policies, depend on the initial angle $\theta(0)$, but qualitatively, the distributions of t_B are similar. For $\theta(0)=-8°$, for example, 4030 policies have $0.08\leqslant t_B\leqslant5.02$ (more than half give the worst result). The three best results are: one policy with $t_B=25.60$, 64 policies with $t_B=133.58$, and the optimal policy with $t_B>10^6$ (most probably, it is infinite). For $\theta(0)=-3°$, three best results were 29.62 (1 policy), 139.58 (64 policies), and also one optimal policy with most likely $t_B=\infty$. For positive $\theta(0)$, when large negative θ values are not needed, one more policy becomes optimal. Therefore, for a random choice of policy, the probability to find a good one with $t_B>100$ is 2%, and that for the optimal policy is less than 0.05%. To show the importance of symmetry breaking, we made similar tests for symmetric control with $f=\pm f_{\max}$, and the best t_B was about 23.

When applying RL methods to this problem, we tested a number of different values of α, γ, ϵ, and λ, and the two methods, Q learning and sarsa. In both methods, we used the ϵ-greedy policy based on the latest Q estimates.

The problem naturally splits into episodes, the end of each episode being the fall of x or θ outside their limits (terminal state in terms of RL). While the cart and the pole remain within specified limits, the controller receives the reward $r=0$, and at the terminal state the reward $r=-1$. Therefore, nonzero reward comes only rarely, but learning goes on all the time: TD methods change the values of Q such that they become consistent with the trajectory of the dynamical system. The value of α_t slowly diminished with time as $\alpha_t=\alpha_0/(1+\beta t_{learn})$, where t_{learn} is the total learning time, including the duration of all previous episodes.

The quality of the current Q estimates (and hence the current greedy policy) has been verified experimentally by evaluating the corresponding t_B value. We estimated t_B at the end of each episode (this time is not included in t_{learn}), when x or θ goes out of their limits and the controller receives nonzero reward $r=-1$. The process of learning then can be shown as a sequence of t_B values. Sometimes, these t_B values stabilized at certain number, and sometimes they oscillated. The examples are shown in Figs. 4–6. Note that usually t_B does not coincide with the episode duration even if the policy is deterministic ($\epsilon=0$). During the episode the values of Q are updated, and the current greedy policy may change.

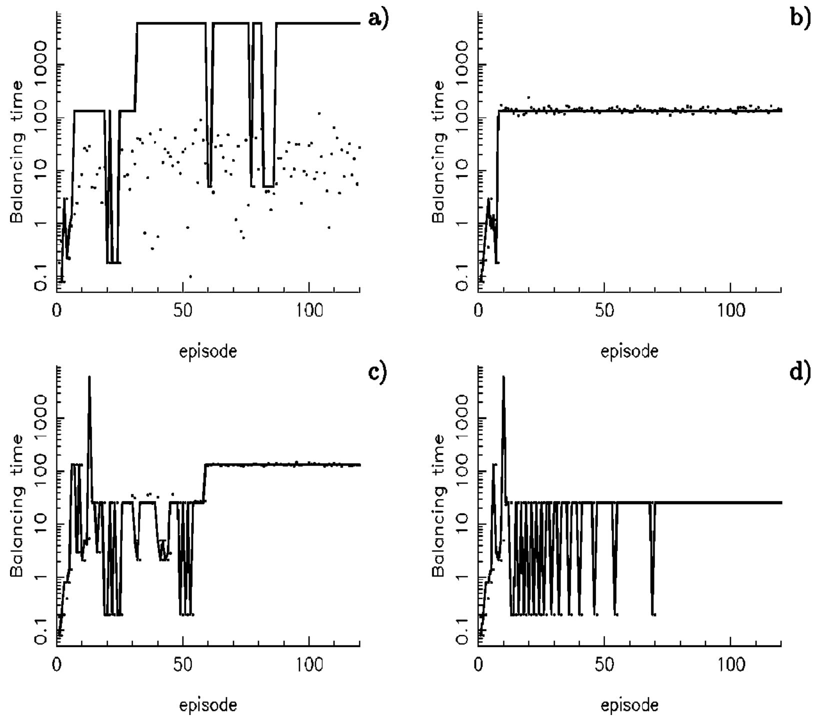

FIG. 5. The problem of cart-pole control, 12-cell boxes partition of phase space. These examples show that sometimes performance of Q learning may be poor, probably, because of non-Markovian partitioning of phase space. It is known that for partially observed Markov process Q learning may oscillate [1]. $\alpha=\beta=0.01$, $\gamma=\lambda=0.5$, $\theta_0=-8°$, $\epsilon=0.1$ (a), 0.01 (b), 0.001 (c), 0 (d).

The total number of test runs was about 1000 for every method. The learning period was limited: we had the number of episodes $\leqslant 500$, the duration of each episode $\leqslant 6000$ (both in learning and testing, $t_B=6000$ in the figures corresponds to the optimal policy), and the total learning time $\leqslant 50\,000$. To characterize the RL performance, we used the following values: (1) the best t_B value $T_{\max}$ found during learning: (2) the average balancing time T_A for the period after a good policy with $t_B > 100$ has also been found. Even though this value depends on the limit of episode duration, it is instructive.

For Q learning, the best policy has been found at least once ($T_{\max}=6000$) in 43% of all runs with the average value of $T_A=1230$, in 51% of runs $T_{\max}=133$, and only in 6% a good policy has not been found at all. For sarsa, similar result was 69% ($T_A=469$), 29% and 2%.

To find out the role of the exploration, we compared only the results for $\epsilon=0$. They were 29% ($T_A=1813$), 67%, and

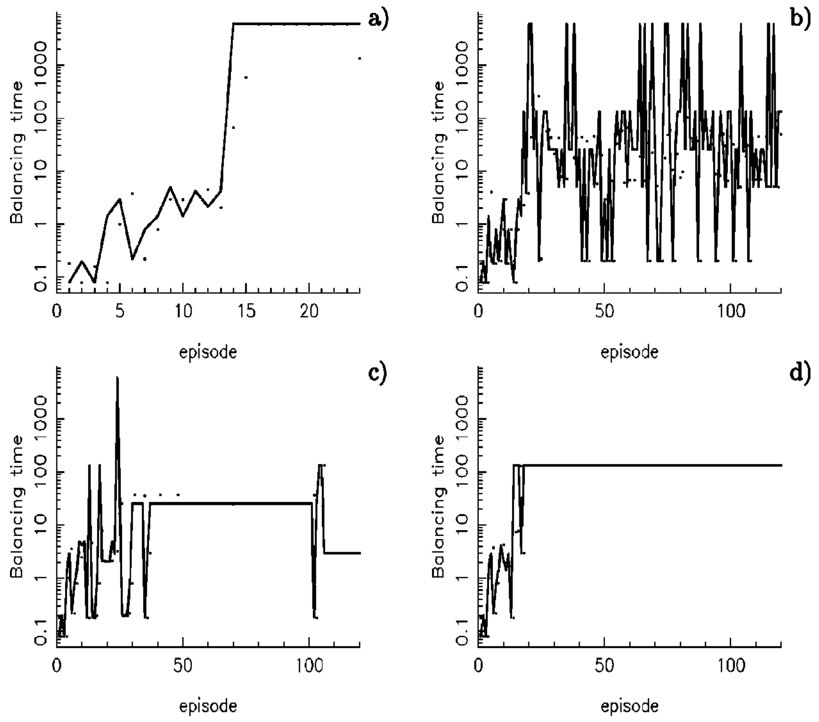

FIG. 6. The problem of cart-pole control, 12-cell boxes partition of phase space. Several examples of the sarsa scheme performance, $\theta_0=-8°$. (a) $\alpha=0.1$, $\beta=0.01$, $\gamma=0.5$, $\lambda=0.5$, $\epsilon=0.001$; (b) $\alpha=0.01$, $\beta=1.0$, $\gamma=0.95$, $\lambda=0.9$, $\epsilon=0.01$; (c) $\alpha=0.1$, $\beta=0$, $\gamma=0.5$, $\lambda=0.5$, $\epsilon=0.0001$; (d) $\alpha=0.1$, $\beta=0$, $\gamma=0.5$, $\lambda=0.5$, $\epsilon=0$.

4% for Q learning and 62% ($T_A = 531$), 37%, and 1% for the sarsa scheme. Therefore, exploration increases the chances to find the optimal policy.

As it is noted in [1], oscillations in Figs. 4–6 may be the result of non-Markovian states. We tested the partition with the help of the entropy estimates. For the optimal policy, we generated a long trajectory of 10^7 data points and evaluated the probabilities p_i of the trajectory to fall into each of the partition domains A_i, the pair probabilities p_{ij} ($A_i \rightarrow A_j$) and triple probabilities p_{ijk} ($A_i \rightarrow A_j \rightarrow A_k$).

For a true Markov chain the following relations should be valid,

$$p_{ij} = p_i P_{ij}, \quad p_{ijk} = p_i P_{ij} P_{jk} = p_{ij} P_{jk}, \tag{13}$$

where P_{ij} are the transition probabilities. Since the trajectory is long, most likely the probabilities p_i are close to their limiting values, so $p_j = \Sigma_i p_i P_{ij}$.

First, it is possible to check the conditions (13). From these conditions it follows that, for Markov chain, the identity $\Delta_{ijk} = p_{ijk} p_j / p_{ij} p_{jk} = 1$ should hold. To characterize the deviation from 1, we calculated the mean value $\langle \max\{\Delta_{ijk}, \Delta_{ijk}^{-1}\} \rangle$ for all $p_{ijk} > 0.01$, and it was equal to 1.8 instead of 1.

Another characteristic may be the entropy. For the case of Markov chain and limiting p_i, the following must hold,

$$H_1 = -\sum_i p_i \ln p_i,$$

$$H_2 = -\sum_{ij} p_{ij} \ln p_{ij} = -\sum_i p_i \left(\sum_j P_{ij} \right) \ln p_i - \sum_{ij} p_i P_{ij} \ln P_{ij} = H_1 + \delta H,$$

$$H_3 = -\sum_{ijk} p_{ijk} \ln p_{ijk} = -\sum_i p_i \left(\sum_{jk} P_{ij} P_{jk} \right) \ln p_i - \sum_{ij} p_i P_{ij} \left(\sum_k P_{jk} \right) \ln P_{ij} - \sum_{jk} \left(\sum_i p_i P_{ij} \right) P_{jk} \ln P_{jk} = H_1 + 2\,\delta H,$$

so $H_3 - H_2 = H_2 - H_1$, $H_1 - 2H_2 + H_3 = 0$. Our estimates gave $H_1 \cong 1.68$, $H_2 \cong 2.61$, $H_3 \cong 3.08$, so $\Delta H = H_1 - 2H_2 + H_3 = -0.46$, so the deviation from a Markov chain is obvious. This may explain the oscillatory behavior in Q learning [1].

2. Self-developing network

To make a self-developing network, we used a special metric in phase space for calculating the distance ρ from a point $\mathbf{x} = \{x, \theta, \dot{x}, \dot{\theta}\}$ to a vector $\mathbf{v}_k$, corresponding to the kth neuron: $\rho(\mathbf{x}, \mathbf{v}_k)^2 = \Sigma(c_i x_i - \mathbf{v}_{ik})^2$. The ranges for different components of $\mathbf{x}$ differ too much, and without a proper metric it is necessary to use too many neurons for efficient control. The set of coefficients c_i in the tests described below was the following: $c_1 = c_3 = 1/x_{\max}$, $c_2 = 1/\theta_{\max}$, $c_4 = 1/(5.6\theta_{\max})$.

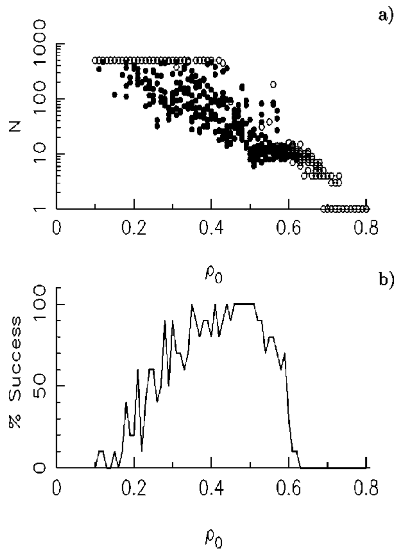

FIG. 7. Dependence of the self-developing network efficiency in the cart-pole problem on the parameter ρ_0. For each ρ_0 value 10 test runs have been done, learning was terminated after finding a policy with $t_B > 1000$ or after 500 episodes. (a) The number of neurons in the network at the end of learning after the good control policy has been found (black circles) or after 500 training episodes (empty circles). (b) The percentage of the runs where good control policy has been found.

The first neuron was placed at the target point $\mathbf{x}_{\text{targ}} = 0$, that is $\mathbf{v}_1 = 0$. When the trajectory moved too far from all existing neurons, a new neuron was added, and the initial values of Q for it were copied from its nearest neighbor. Since, in this problem, we are interested in keeping the trajectory close to 0, we used additional factor to prevent from having too many new neurons far from the target point. So, the final criterion for creating a new neuron was the following: Let the trajectory be at the point $\mathbf{x}$, and $\mathbf{v}_k$ correspond to the nearest of existing N neurons. If $\rho(\mathbf{x}, \mathbf{v}_k) > \rho_0 e^{\kappa \rho(\mathbf{x}, \mathbf{x}_{\text{targ}})}$, then we create a new neuron with $\mathbf{v}_{N+1} = \mathbf{x}$, which corresponds to the state s_{N+1}, and set $Q(s_{N+1}, a) = Q(s_k, a)$. In numerical experiments, we used $\kappa = 0.5$, which proved to be optimal.

Changes in the value of ρ_0 lead to a sort of phase transition in the algorithm behavior. For small ρ_0 values the number of neurons did not stabilize until the upper limit for N has been reached. The performance of RL algorithms in this situation was not good. For too large ρ_0, the partition was too coarse. There was an interval of ρ_0 values, in which the performance of the methods was the best, see Fig. 7.

The growing network helped obtain even simpler partitioning, including only eight neurons (instead of 12 in boxes) for symmetric control (that is, with the same values of f in both directions) with most probably infinite t_B. Another interesting effect was that, for self-developed partitions, usually the value of ΔH was smaller than that for 12-cell boxes. Once we obtained a partition with 13 neurons for which $\Delta H = -0.02$. But, when we set this partition manually and

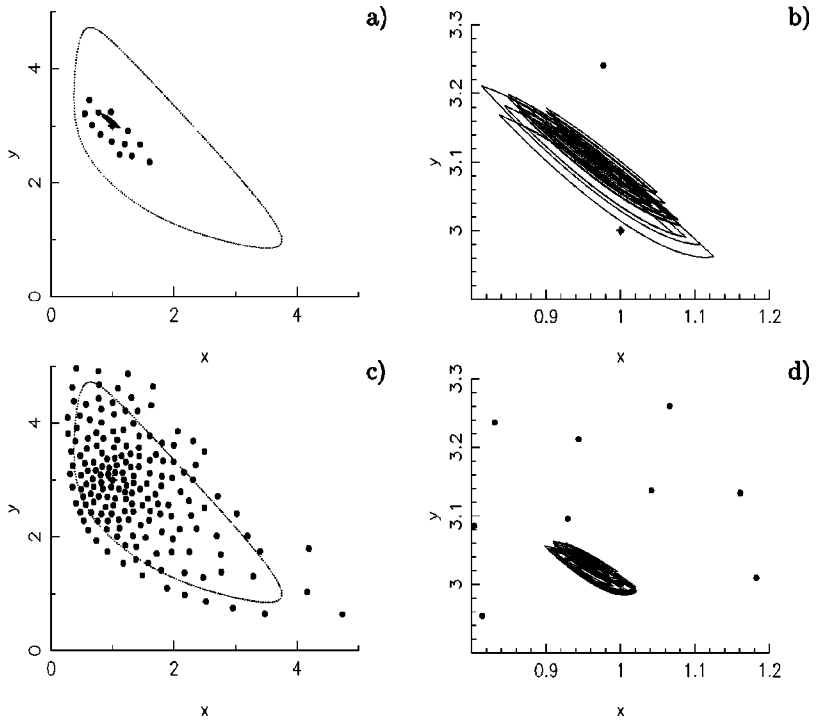

FIG. 8. Control of the Brusselator model. Here, in contrast with the cart-pole, there are more gradations of control impact: c and b can take 5 values each (0.8, 0.9, 1.0, 1.1, 1.2 of their ''main'' values $c_0=1$ and $b_0=3$), that is there may be 25 different control actions. Dotted line shows the stable limiting cycle in the model, and the cross — the unstable focus $x=1$, $y=3$, in the vicinity of which trajectory should be kept by the control. Black circles show the location of the VQ network neurons, which correspond to discrete states s_i, used in the control. Solid line shows the trajectory under deterministic control, learned by RL with self-developing VQ network. Two different kinds of problem were considered. (a, b) Like in the cart-pole problem, initial data were chosen in the vicinity of the focus, and when the trajectory moved too far away from it, the controller received a failure signal, after which new episode started again from near the focus. Neurons appear only near to the focus, and a few of them are enough. Panel (b) shows the controlled trajectory in larger scale. (c, d) Learning started from the limiting cycle and was not splitted into episodes, and the reward signal was negative with the value proportional to the distance from the focus. Here neurons cover the whole neighborhood of the cycle, and their number is much larger.

tested the performance of Q learning, the convergence was not very good. Therefore, even the closeness of *optimal* controlled behavior to Markov chain does not guarantee good convergence of Q learning. This may happen, for example, if the properties of transitional probabilities $P(a,s,s')$ are policy dependent. It also proves that sarsa can find optimal policy faster than Q learning, but may also ''lose'' it more easily. The problem with convergence still remains.

In all cases, the learned control regime was chaotic with the values of the largest Lyapunov exponent between 5.62 and 5.72.

C. Control of a periodical chemical reaction causes hyperchaos

In this case, the unstable manifold is two-dimensional, and it is also hard to propose any intuitive control strategy (though there exists a very simple control strategy that has been found by RL). This problem proved to be simpler than the cart-pole one. We obtained good results with boxes, randomly distributed neurons, and self-developing network. We shall present results only for the self-developing network (Fig. 8). An interesting feature of the Brusselator control is that the convergence of RL algorithms was essentially better than that for the cart-pole. Also, there is no need to introduce any special metric for the use of a self-developing network.

We considered two kinds of control problems. The first one, as in the case of the cart-pole, was episodic. The trajectory always started in the vicinity of the unstable focus $\mathbf{x}_0$ and the episode terminated when $|\mathbf{x}-\mathbf{x}_0|>0.5$. The reinforcement signal was $r=e-e^{2|\mathbf{x}-\mathbf{x}_0|}$. An example of learned behavior and locations of the neurons are shown in Figs. 8(a,b). The parameter values of the Q learning were $\alpha=0.1$, $\beta=0$, $\gamma=\lambda=0.5$, $\epsilon=0.05$, $\tau=0.2$. The period of the limit cycle is $T\approx 7.2$.

The second task was continuous learning without a terminal state. The trajectory started at some point, remote from the focus (in our experiments from $x=y=1$), and the controller received the reward signal $r=e-e^{2|\mathbf{x}-\mathbf{x}_0|}$. To come close to the focus, the controller must explore the domain near it, place a number of neurons there, and learn the corresponding Q values. To solve this problem with the growing network, it was necessary to wait for about 2000 cycle periods. An example of the results is shown in Figs. 8(c,d). In this problem, sarsa was more efficient. The parameters are the same as in panels a,b.

The control of Brusselator motion, near the focus, always occurred in hyperchaotic regimes. There were two positive

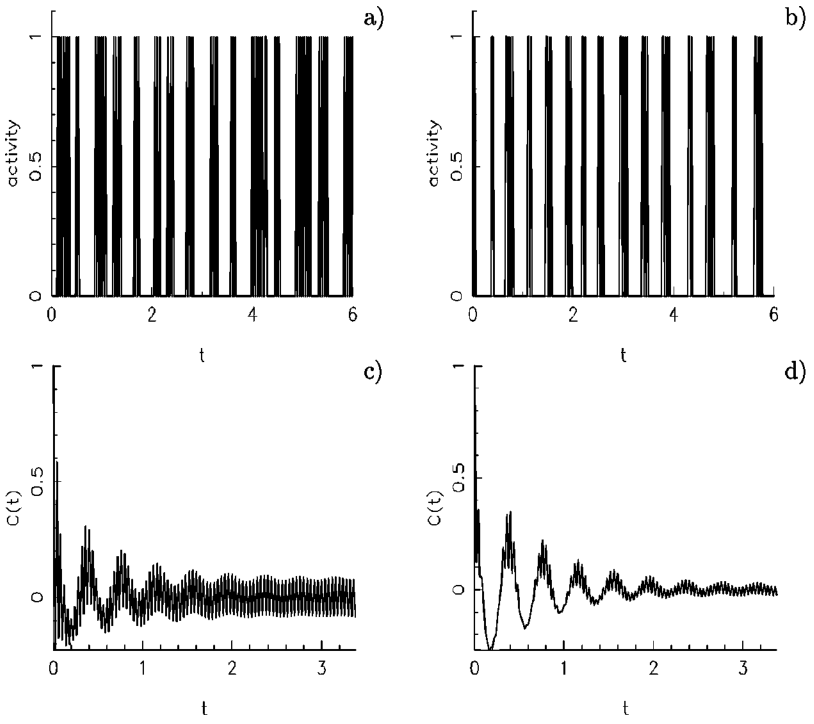

FIG. 9. ‘‘Encephalogram’’ for the cart-pole-control. The self-developing network was used, when the optimal control regime was found it contained 26 neurons, but on the resulting chaotic attractor only seven of them were used. Panels (a) and (b) show the temporal activity for two of the neurons (1 when control was performed by this neuron and 0 otherwise). Panels (c) and (d) show corresponding autocorrelation functions. Plots show both determinism and randomness.

Lyapunov exponents $\lambda_{1,2}>0$ with close values, $\lambda_1 \cong \lambda_2$. Since the values of λ depend on *c* and *b*, which were varied during control, the observed values cannot be related to a single linear problem. Typical values were between 0.4 and 0.65.

VI. ‘‘ENCEPHALOGRAM’’ FROM A NEURAL CONTROLLER.

An important way of brain studies involves recording its electrical activities and subsequent analysis of these recordings. We also made such recordings for the controlling network. We chose one of the neurons and when it was active we set the activity signal to 1, and 0 otherwise. The examples of such activity record for two neurons from the network of 26 and corresponding autocorrelation functions are shown in Fig. 9.

The recordings show the presence of both randomness (autocorrelation function decays) and some regularity (strong high-frequency periodic component). It is hard to make any direct comparisons with the activities of a real nervous system, but perhaps this observation may be pursued further in connection with brain chaoticity detected by electroencephalogram (EEG).

VII. CONCLUSIONS

Discrete control is used in some machine-learning schemes. When the control is used to keep a dynamical system in the vicinity of an asymptotically unstable equilibrium, a typical outcome of the control problem is the creation of a chaotic attractor. Thus, we have an effect that we may call ‘‘chaos with control.’’ It will be good to know if such chaos with control appears in the activities of the brain.

There are methods in machine learning and artificial neural network theories that can find policies to perform such a control. Some examples of application of these techniques are presented in the paper. We have also discussed possible problems that can arise during such applications. This coding technique in reinforcement learning with growing networks appears to be efficient in controlling unstable equilibria. We considered only one of the simplest neural network implementations of reinforcement learning. There are many others, with different types of neural networks, as described, e.g., in [2,1,19,15].

We would like to point out that in reinforcement learning, chaos in principle may be a part of the learning process. According to Freeman, chaos may be important for the brain’s ability ‘‘to generate insight and the trials of trial and error problem solving’’ [6]. In RL such ‘‘insight’’ is provided by exploration with partially random policies (cf. Secs. IV and V). Similarly, one can generate a *chaotic policy* by using a chaotic dynamical system instead of a random signal. Note that some investigators have already attempted to use chaotic signals for solving optimization problems during learning of neural networks, e.g., [26]. As our preliminary results show, RL may work well with such chaotic exploration. If one builds a chaotic system into neural controller, then purely deterministic system will be able to produce randomlike policies. We hope to present a more detailed study of the subject in our next paper.

ACKNOWLEDGMENTS

This work has been supported through a grant to M.K.A. from the Defense Research Establishment Suffield under Contract No. W7702-8-R745/001/EDM. The authors also would like to thank Dr. Simon Barton for his continuing support and helpful discussions. The work was carried out using computing facilities provided by Multimedia Advanced Computational Infrastructure (MACI).

APPENDIX A: EQUATIONS OF MOTION FOR CART-POLE

We use the following notation:

x, coordinate of cart center (m)

θ, angle between the vertical and pole direction, for clockwise motion $\theta > 0$

$g = 9.8$, gravity acceleration (m/s^2)

$l = 0.5$, pole length (m)

$m_p = 0.1$, mass of the pole (kg)

$m_c = 1.0$, mass of the cart (kg)

$\varphi_1 = -\mu_p \dot{\theta}$, friction in the axis of pole, $\mu_p = 5\times 10^{-6}$

$\varphi_2 = -\mu_c \mathrm{sgn}(\dot{x})$, friction for the cart, $\mu_c = 5\times 10^{-4}$

f, external force, which is applied to cart and has to balance the pole. $|f| = f_{\max}$, and its sign depends on the present state of the system, $f_{\max} = 10$ n.

First let us consider the cart-pole system moving without friction. The kinetic energy of the system is

$$T = \frac{1}{2} m_c \dot{x}^2 + \frac{1}{2} m_p \dot{x}^2 + \frac{1}{2}\frac{m_p l^2 \dot{\theta}^2}{3} + \frac{m_p \dot{x}\dot{\theta} l \cos\theta}{2}. \tag{A1}$$

The gravitational potential energy is

$$U = \frac{m_p g l \cos\theta}{2}. \tag{A2}$$

The Euler-Lagrange equations (with friction and control terms added) are as follows.

(1) Equation for the pole

$$\frac{m_p l^2 \ddot{\theta}}{3} = \frac{m_p g l \sin\theta}{2} - \frac{\ddot{x} m_p l \cos\theta}{2} + f + \varphi_1 \tag{A3}$$

or

$$\frac{2l}{3}\ddot{\theta} = g \sin\theta - \ddot{x}\cos\theta + \frac{2\varphi_1}{m_p l}. \tag{A4}$$

(2) Equation for the center of mass,

$$m_c \ddot{x} + m_p \frac{d}{dt}\left(\dot{x} + \frac{l\dot{\theta}\cos\theta}{2}\right) = \varphi_2 \tag{A5}$$

or

$$(m_c + m_p)\ddot{x} + \frac{m_p l}{2}(\ddot{\theta}\cos\theta - \dot{\theta}^2 \sin\theta) = f + \varphi_2. \tag{A6}$$

Equations (A4) and (A6) can be rewritten as the following system:

$$\cos\theta \cdot \ddot{x} + \frac{2l}{3}\ddot{\theta} = g\sin\theta - \frac{2\varphi_1}{m_p l},$$

$$\ddot{x} + \frac{m_p l \cos\theta}{2(m_c+m_p)}\ddot{\theta} = \frac{f + \varphi_2}{(m_c+m_p)} + \frac{m_p l \sin\theta}{2(m_c+m_p)}\dot{\theta}^2.$$

To simplify notation let us denote (taking into account the relations for φ_i)

$$f_1 = g\sin\theta - \frac{2\mu_p}{m_p l}\dot{\theta},$$

$$f_2 = \frac{f - \mu_c \mathrm{sgn}(\dot{x})}{(m_c+m_p)} + \frac{m_p l}{2(m_c+m_p)}\dot{\theta}^2 \sin\theta,$$

$$c_1 = \frac{2l}{3},$$

$$c_2 = \frac{m_p l}{2(m_c+m_p)}\cos\theta,$$

then we get

$$\ddot{\theta} = \frac{f_1 - f_2\cos\theta}{c_1 - c_2\cos\theta}, \quad \ddot{x} = \frac{c_1 f_2 - c_2 f_1}{c_1 - c_2 \cos\theta}. \tag{A7}$$

The initial conditions used in this paper were $x(0) = \dot{x}(0) = \dot{\theta}(0) = 0$, $\theta(0) = \theta_0 \neq 0$.

APPENDIX B: PROOF OF THE EXISTENCE OF A POSITIVE LYAPUNOV EXPONENT FOR PERTURBED MAPPING

Lemma. Let the linear dynamical system $u_{k+1} = Bu_k$, B = const have at least one positive Lyapunov exponent $\lambda_1 > 0$. Then the system

$$v_{k+1} = (B + \epsilon C_k)v_k \tag{B1}$$

also has at least one positive Lyapunov exponent provided the perturbation C_k is bounded, $|C_k| < c$ and ϵ is small enough.

Proof. For the sake of simplicity let us suppose that the eigenvalues of B are real and nondegenerate, and there is basis of its eigenvectors $e_1, \cdots, e_n$ with eigenvalues $l_1, \cdots, l_n$, $|l_i| = e^{\lambda_i}$. Let $v_k = a_k e_1 + b_k g_k$, $g_k \in \mathrm{span}\{e_2, \ldots, e_n\}$ and $|g_k| = 1$. Then

$$v_{k+1} = Bv_k + \epsilon C_k v_k = l_1 a_k e_1 + b_k B g_k + \epsilon C_k v_k. \tag{B2}$$

On the other hand, we have $v_{k+1} = a_{k+1}e_1 + b_{k+1}g_{k+1}$, $|g_{k+1}| = 1$.

Now we need the estimates of $|a_{k+1}|$ and $|b_{k+1}|$. To obtain these estimates, we decompose the perturbation term $\epsilon C_k v_k$ by using the linearly independent but in general nonorthogonal vectors e_1 and g_{k+1} . Due to nonorthogonality, absolute values of components may be greater than the length of vector itself. Let θ be the angle between e_1 and g_{k+1}. If $\epsilon C_k v_k = \alpha e_1 + \beta g_{k+1}$, then $|\epsilon C_k v_k|^2 = \alpha^2 + \beta^2 + 2\alpha\beta\cos\theta$ and $|\alpha|,|\beta| \leqslant |\epsilon C_k v_k|/\sqrt{1 - |\cos\theta|}$. Let

$$r = \min_{g\in \mathrm{span}\{e_2, \ldots, e_n\}, |g|=1} \sqrt{1 - |(e_1\cdot g)|},$$

then $|\alpha|,|\beta| \leqslant |\epsilon C_k v_k|/r$ for any g_k.

From Eq. (B2) we immediately obtain

$$|a_{k+1}| \geqslant |l_1 a_k| - \epsilon r^{-1} |C_k v_k| \geqslant |l_1 a_k| - \epsilon r^{-1} c(|a_k| + |b_k|),$$

$$|b_{k+1}| \leqslant |l_2 b_k| + \epsilon r^{-1} |C_k v_k| \leqslant |l_2 b_k| + \epsilon r^{-1} c(|a_k| + |b_k|).$$

Let us choose the initial data v_0 such that $|a_0| > |b_0|$, and $\kappa_k = b_k / a_k$ or $b_k = \kappa_k a_k$. Suppose that $\kappa_k < q$, where $0 < q < 1$ — a fixed number. Then

$$|\kappa_{k+1}| = \frac{|b_{k+1}|}{|a_{k+1}|} \leqslant \frac{|l_2 \kappa_k| + \epsilon r^{-1} c(1 + |\kappa_k|)}{|l_1| - \epsilon r^{-1} c(1 + |\kappa_k|)} < q$$

provided ϵ is sufficiently small,

$$\epsilon < \frac{qr(|l_1| - |l_2|)}{2c} = \frac{qr}{2c}(e^{\lambda_1} - e^{\lambda_2}). \tag{B3}$$

Therefore, for this ϵ the inequality $|\kappa_k| < q$ holds for any k.

Since $|v_k| \leqslant |a_k| + |b_k| \leqslant (1+q)|a_k| < 2|a_k|$, we obtain $|a_{k+1}| \geqslant |l_1 a_k| - \epsilon c 2 |a_k| = (e^{\lambda_1} - 2\epsilon c)|a_k|$. If $e^{\lambda_1} - 2\epsilon c > 1$ or

$$\epsilon < \frac{e^{\lambda_1} - 1}{2c},$$

then $|a_k|$ will grow exponentially.

Finally, since $|v_k| \geqslant |a_k| - |b_k| \geqslant (1-q)|a_k|$, it should also grow exponentially, which gives

$$\lim_{k\to\infty} \frac{1}{k} \ln |v_k| \geqslant \ln(e^{\lambda_1} - 2\epsilon c) > 0, \tag{B4}$$

and, according to Lyapunov theorems, the system (B1) should have at least one positive Lyapunov exponent, provided

$$\epsilon < \min\left\{ \frac{e^{\lambda_1} - 1}{2c}, \frac{qr}{2c}(e^{\lambda_1} - e^{\lambda_2}) \right\}.$$

If the multiplicity of the Lyapunov exponent λ_1 for B is greater then 1, this proof can be repeated with replacing e_1 to an evolving vector f_k belonging to invariant subspace for λ_1, while g_k belongs to its complement. f_0 always can be chosen such that the relations $|Bf_k| \geqslant e^{\lambda_1}$ and $|Bg| \leqslant e^{\mu}$ hold, where μ is the largest Lyapunov exponent different from λ_1. [Inequality for λ_1 may arise, e.g., from matrices like $\left(\begin{smallmatrix} l & \gamma \\ 0 & l \end{smallmatrix}\right)$), for which norm grows as $k|l|^k$.] If all Lyapunov exponents of the unperturbed system $\lambda_i = \lambda_1$, like in the Brusselator problem, then Eq. (B4) follows directly from Eq. (B2) with weaker condition for ϵ. This completes the proof.

[1] L. P. Kaebling, M. L. Littman, A. W. Moore, J. Artif. Intell. Res. **4**, 237 (1996).

[2] R. S. Sutton and A. G. Barto, *Reinforcement Learning* (Bradford Book and The MIT Press, Cambridge, MA, 1998).

[3] A. G. Barto, R. S. Sutton, and C. W. Anderson, IEEE Trans. Syst. Man Cybern. **SMC-13**, 834 (1983).

[4] W. J. Freeman, Int. J. Bifurcation Chaos Appl. Sci. Eng. **2**, 451 (1992).

[5] M. R. Guevara, L. Glass, M. C. Mackey, and A. Shrier, IEEE Trans. Syst. Man Cybern. **SMC-13**, 790 (1983).

[6] W. J. Freeman, Sci. Am. **264(2)**, 78 (1991).

[7] Y. Yao, W. J. Freeman, B. Burke, and Q. Yang, Neural Networks **4**, 103 (1991).

[8] D. N. Burghes and A. Graham, *Introduction to Control Theory, Including Optimal Control* (Ellis Horwood, Chichester, 1980).

[9] P. R. Kumar, SIAM J. Control Optim. **23**, 329 (1985).

[10] D. Michie and R. A. Chambers, in *Boxes: An Experiment in Adaptive Control*, edited by E. Dale, D. Michie, Machine Learning 2 (American Elsevier, New York, 1968), pp. 137–152.

[11] *Neural Networks for Control* edited by W. T. Miller, R. S. Sutton, and P. J. Werbos (MIT Press, Cambridge, 1990).

[12] *Handbook of Intelligent Control. Neural, Fuzzy and Adaptive Approaches*, edited by D. A. White and D. A. Sofge (Van Nostrand Reinhold, New York, 1992).

[13] R. Hecht-Nielsen, *Neurocomputing* (Addison-Wesley, Reading, MA, 1990).

[14] G. Nikolis and I. Prigogine, *Self-Organization in Non-Equilibrium Systems* (Wiley, New York, 1977).

[15] C. Touzet, Rob. Auton. Syst. **22**, 251 (1987).

[16] S. Gadaleta and G. Dangelmayr, Chaos **9**, 775 (1999).

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

[18] R. S. Sutton and A. G. Barto, Psychol. Rev. **88**, 135 (1981).

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

[20] *Dynamic, Genetic and Chaotic Programming*, edited by B. Souĉek (Wiley, New York, 1992).

[21] J.-P. Eckmann and D. Ruelle, Rev. Mod. Phys. **57**, 617 (1985).

[22] T. Kohonen, *Self-Organizing Maps* (Springer, Berlin, 1997).

[23] S. Haykin, *Neural Networks: A Comprehensive Foundation* (Macmillan, New York, 1994).

[24] V. Cherkassky and F. Mulier, *Learning From Data* (Wiley, New York , 1998).

[25] B. Fritzke, Neural Proc. Lett. **2**, 9 (1995).

[26] Y. Hayakawa, A. Marumoto, and Y. Sawada, Phys. Rev. E **51**, R2 693 (1995).