A Study of Learning Algorithms for Continuous-Time Recurrent Neural Networks

連続時間神経回路網における
学習アルゴリズムの研究

Kenji Doya
銅谷　賢治

1991. 5. 31

A Study of Learning Algorithms for Continuous-Time Recurrent Neural Networks

Kenji Doya

A dissertation submitted to
Faculty of Engineering,
University of Tokyo

May 31, 1991

Contents

List of Figures

Chapter 1

Introduction

The study of information processing in the brain is one of the most active and progressing fields of modern science. Many disciplines of science are being applied to explore the principles of information processing in the brain. For example, neurophysiologists are analyzing the microstructures of nervous systems and explaining their function at the cellular and molecular levels. Cognitive scientists are investigating various aspects of human information processing and describing them by macroscpic models. In order to get a complete understanding of the functions of neural systems there must be a theory that links together our knowledge of the molecular processes in a neuron, the function of neural networks in a region of the cortex, and the total behaviors of animals and humans. This is a major goal of computational neuroscience. In recent years more and more researchers have become aware of the necessity of computational or mathematical approaches to the study of neural information processing.

The objective of this thesis is to develop a general scheme for constructing computational models of dynamical information processing in the brain. Supervised learning in multi-layered networks has been one of the most successful strategies for constructing computational models for given information processing tasks. However, the conventional supervised learning algorithms—perceptron learning [34] and back-propagation learning [35]—can be applied only for feed-forward discrete-time networks.

Feed-forward networks can represent arbitrary mappings between vectors, however, the role of neural systems is not limited to learning static input-output relationships. Even in the simplest animals neural outputs are not only a function of the present external inputs but are also dependent on its internal state and autonomous dynamics. For example, the rhythmic patterns of locomotion of insects, fish, and mammals are generated by autonomous oscillations in their central nervous systems. In higher animals, especially in humans, not only the motor system but also the sensory information processing is dependent on the internal state. In order to model such autonomous functions of neural systems a network must have both internal states and recurrent connections.

Discrete-time operation has been assumed in most of the computational models of neural systems. Temporal behaviors of such models are very much determined by the fact that they operate in discrete-time. In biological neural systems, however, each neuron operates continuously and asynchronously in time. This difference is not so important in modeling the static aspects of neural information processing, because the significance

of those models lies only in the stationary states of the networks. On the other hand, in modeling of dynamical behaviors of neural systems the temporal evolution of the system is essential.

It has been shown by physiological experiments that the mechanisms of both motor control [18] and perception [17] are based on the synchronization or entrainment of oscillatory modules of neurons. In view of this property we note that the behavior of a continuous-time neural network model is robust to perturbations of the incoming signals, since each unit has the characteristics of integration and decay over time. On the other hand, it is rather difficult for a discrete-time model, whose state can change by large amounts at every time step, to adapt its dynamics to the temporal fluctuations of incoming signals. Thus, in order to model the flexible temporal behaviors of neural systems it is inevitable that we will need to employ continuous-time models.

In the following chapters we formulate general supervised learning algorithms for continuous-time neural networks with recurrent connections. The basic strategy is to consider the network dynamics as a mapping between functions in time, of such things as network variables, parameters and output errors, and to derive a gradient descent algorithm using the derivatives of this mapping. Conventional learning algorithms are derived from the differential relationships between connection weights and static input and output vectors, but by employing the derivatives of mappings between functions of time then gradient descent learning algorithms can be generalized to continuous-time networks. This approach also provides a unified formulation of learning algorithms for recurrent networks.

In Chapter 2 a simple network model which can learn oscillatory patterns is described and a learning algorithm is derived. In Chapter 3 the learning algorithm is generalized to a larger class of network models, which include both discrete-time and continuous-time operations and arbitrary network configurations. In Chapter 4 mechanisms for memorizing and regenerating multiple temporal patterns are discussed and several examples are demonstrated by simulations. Chapter 5 describes two models of motor learning from the viewpoint of coordination of neural and physical dynamical systems. The current and future applications of these learning schemes in both neuroscience and engineering are discussed in the concluding chapter.

Chapter 2

Continuous-Time Back-Propagation Learning Algorithm for Adaptive Neural Oscillator Networks

In this chapter we construct a simple network model of rhythmical temporal pattern memory, which is called an Adaptive Neural Oscillator (ANO) [8]. As a learning algorithm for an ANO network we extend the back-propagation learning algorithm [35] to continuous-time network models.

2.1 Memory of motor patterns

When we learn a new motor skill—such as, using a typewriter, playing a guitar or playing tennis—we have to be conscious of the movement of our fingers, arms, legs or other parts of our bodies. However, after repetitions of these new movements, we can execute the tasks without being conscious of the detailed movements of our bodies. This suggests that there exist some neural networks which undertake the regeneration of the motor patterns which were repeatedly given by the higher centers. This idea is illustrated in Figure 2.1. With the help of these motor pattern memory networks, the higher functions of the brain are exempted from lower level control tasks and can concentrate on higher level tasks, such as selection and coordination of simple motor patterns. But how are the various patterns of motion stored in our brains?

Physiological studies in animals such as leech [26], lobster [38], lamprey [18] and cat [39], have revealed that the rhythmical patterns of locomotion are generated by neural oscillator circuits in their spinal cords or ganglia, which are called central pattern generators (CPG) [19]. The structures of CPGs in these animals are determined genetically and therefore the oscillation patterns are hereditary and fixed. On the other hand higher animals can learn new patterns of motion which are not inherited from their parents. How are the acquired motor patterns of the higher animals stored and regenerated in their nervous systems? Physiological experiments with monkeys suggest that the premotor area and supplementary motor area of the cerebral cortex are responsible for the control of sequential motions [41]. However, the neural mechanisms in those areas are

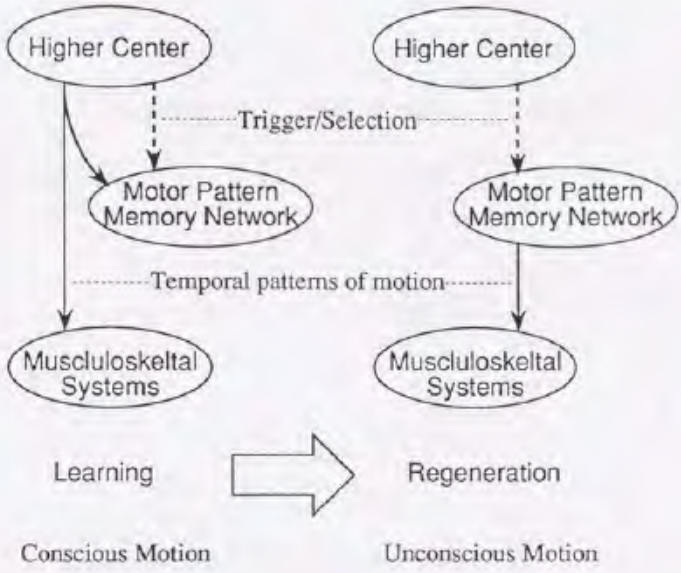


Figure 2.1: The function of motor pattern memory networks.

not yet elucidated.

Motor patterns of industrial robots are usually stored in arrays of the joint angles at each time-step and are replayed by scanning them. Since such time tables have never been found in animal motor nervous systems it would be reasonable to assume that the acquired motor patterns are stored in the connection patterns of some neural networks and regenerated as the waveforms of their network dynamics. In this case, the brain must solve the inverse problem of determining the synaptic weights by which the network generates the desired temporal patterns.

In the following sections we will give one solution to the inverse problem of finding a connection pattern which makes the network oscillate autonomously with a given waveform.

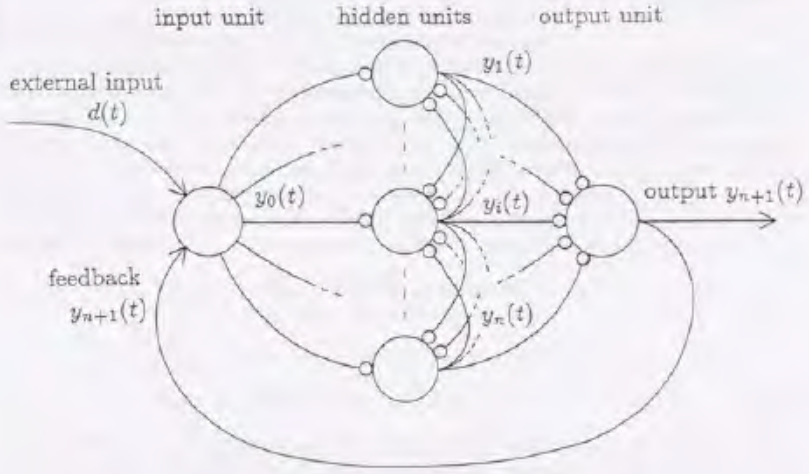


Figure 2.2: The structure of an adaptive neural oscillator network.

2.2 The dynamics of an adaptive neural oscillator network

In this section we construct a simple neural network model of learning of rhythmical temporal patterns, which we call adaptive neural oscillator (ANO). We use a continuous-time neuron model because its temporal behavior is more similar to a biological neuron than a discrete-time neuron model. It is well known that a recurrent neural network with asymmetric connection weights can be an autonomous oscillator [2]. The purpose of an ANO network is to memorize a given waveform as the output waveform of its autonomous oscillation.

The structure of an ANO network is shown in Figure 2.2. The network has $n+1$ continuous-time neuron models, each of which is a cascade of a weighted summation, a linear filter of first order time lag, and a sigmoid output function. One of them, indexed by $n+1$, is the output unit for which a desired output waveform is given. The other n units, indexed by 1 through n, are hidden units whose output is not explicitly specified.

The output unit sums the output waveforms of the hidden units so that its output waveform becomes close to that of the desired output. It is clear that the output unit cannot produce any rhythmic output if all of the hidden units are quiescent. It is desirable

that the hidden units are oscillating with the same period as the desired signal but with different waveforms and phases. The easiest way to realize this condition is to give a forcing input to the hidden units using the desired signal. In this case, if the learning is completed and the output of the network is the same as the the desired waveform, the forcing input can be replaced with the actual output of the network and hence autonomous oscillation with the desired waveform is achieved.

Thus we suppose that the network has two operating modes, the memorizing mode and the regenerating mode. In the memorizing mode each hidden unit is forced to oscillate by the desired signal and learning is executed so as to make the waveform of the output as close as possible to that of the desired signal. In the regenerating mode the output signal is fed back into the input and an oscillation with a waveform similar to the desired signal should arise if the learning has been sufficiently accomplished.

With an additional input unit, which is index by zero and serves only as an input acceptor (see Figure 2.2), we define the dynamics of the network as follows,

$$\begin{aligned} y_0(t) &= \begin{cases} d(t) & \text{(memorizing mode)}, \\ y_{n+1}(t) & \text{(regenerating mode)}, \end{cases} \\ \tau_i \frac{d}{dt} x_i(t) &= -x_i(t) + \sum_{j=0}^{n} w_{ij} y_j(t) + b_i \qquad (i = 1, ..., n+1), \\ y_i(t) &= g(x_i(t)) (i = 1, ..., n+1). \end{aligned} \tag{2.1}$$

The variables $x_i(t)$ and $y_i(t)$ represent the internal state and the output level of the i-th unit respectively, and $d(t)$ is the desired output. The connection weight from the j-th unit to the i-th unit is denoted by w_{ij} and the input bias of the i-th unit is denoted by by b_i. We suppose there are no direct connection from the input unit to the output unit and no self-connections, that is, $w_{n+1,0} = 0$ and $w_{ii} = 0$ for $i = 1, \ldots, n+1$. Note that each unit has a first order delay, or leaky integrating, characteristics with a decay time constant τ_i. We usually use a symmetric sigmoid output function

$$g(x) = \frac{1 - e^{-x}}{1 + e^{-x}} = \frac{2}{1 + e^{-x}} - 1, \tag{2.2}$$

that has $g(0) = 0$ and $g(\pm\infty) = \pm 1$.

If we assume that learning has completed ideally in the memorizing mode, that is, the waveform of the output unit $y_{n+1}(t)$ has become exactly equal to that of the external input $d(t)$, then switching of the input from $d(t)$ to $y_{n+1}(t)$ makes no difference in the waveforms of the hidden units and consequently to that of the output unit. This means that the waveform of the external signal $d(t)$ has become a periodic solution of the closed loop dynamics in the regenerating mode.

In general, however, when the learning has converged, the output $y_{n+1}(t)$ may not be exactly equal to the input $d(t)$. In this case, $d(t)$ is not the periodic solution of the closed loop network. Moreover, even if the learning is complete and there exists a periodic solution, it may not be stable as a solution of the autonomous system and the oscillation with the desired waveform may not be sustained.

To examine the existence and stability of the periodic oscillation in this network we should investigate the functional mapping F from $y_0(t)$ to $y_{n+1}(t)$. The fixed points of the

mapping F correspond to the periodic solutions of the closed loop network. However, since the mapping F is very complicated, it seems to be almost impossible to fully elucidate its properties analytically.

However, if $y_{n+1}(t)$ is sufficiently close to $d(t)$ in the memorizing mode and F is locally a contraction mapping, there is a stable fixed point of F near $d(t)$. Since each unit has a saturating and smoothing output property, it is plausible that F can be locally a contraction mapping, although this will dependent on the waveform $d(t)$ and the connection weights. Thus, at least under some favorite conditions, the network may have a stable periodic solution that is not far from the desired signal $d(t)$. In the later sections in this chapter the existence and stability of the autonomous oscillation will be assured by computer simulations.

2.3 A continuous-time back-propagation learning algorithm

In a primitive model of an adaptive neural oscillator [6], the weights of the hidden units were fixed at random values and the adaptive filter learning algorithm [44] was applied only at the output unit. These networks could memorize and regenerate various waveforms, but their abilities were greatly limited by the fixed random connection weights.

To make the connections of the hidden units also adaptive we modified the back-propagation learning algorithm [35] and applied it to continuous-time network models.

In a continuous-time neural network the present output of a unit is affected not only by the present values of the inputs but also by the preceding input waveforms. Therefore, we must deal with the relationship between the state histories of units rather than just the relationship between the states of the units at each time step. That is, we must consider the state as a function of time rather than the state at a given time. In particular, where in the conventional back-propagation algorithm we worked with derivatives, we must now take function of Fréchet derivatives.

In general, when a mapping $F : x(t) \mapsto y(t)$ is given, the Fréchet derivative of F at $x(t)$ is defined as a linear operator $DF(x(t))$ such that, for any small function $\epsilon(t)$,

$$F(x(t) + \epsilon(t)) = F(x(t)) + DF(x(t))\epsilon(t) + o(\epsilon(t)), \tag{2.3}$$

where $o(\epsilon)$ is a smooth function satisfying

$$\lim_{\epsilon \to 0} \left\| \frac{o(\epsilon)}{\epsilon} \right\| = 0. \tag{2.4}$$

For the composition $G \circ F$ of

$$y(t) = F(x(t)) \quad \text{and} \quad z(t) = G(y(t)), \tag{2.5}$$

we have the chain rule

$$D(G \circ F)(x(t)) = DG(y(t))DF(x(t)). \tag{2.6}$$

In analogy to the formulation and notation of the conventional back-propagation algorithm we will denote this chain rule by

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}. \tag{2.7}$$

We now rewrite the dynamical equations (2.1) by regarding them as a serial composition of mappings between functions of time.

$$\begin{aligned} u_i(t) &= \sum_{j=0}^{n} w_{ij} y_j(t) + b_i, \\ x_i(t) &= (1 + \tau_i \frac{d}{dt})^{-1} u_i(t), \qquad (i = 1, \ldots, n+1), \\ y_i(t) &= g(x_i(t)), \end{aligned} \tag{2.8}$$

where $u_i(t)$ is the total input to the i-th unit and $(1+\tau\frac{d}{dt})^{-1}$ is an operator of first order time lag, that is, a function $x(t)=(1+\tau\frac{d}{dt})^{-1}u(t)$ is given by the solution of the equation

$$\tau\frac{d}{dt}x(t)=-x(t)+u(t). \tag{2.9}$$

The Fréhet derivative of this operator is $D(1+\tau\frac{d}{dt})^{-1}=(1+\tau\frac{d}{dt})^{-1}$, since it is a linear operator.

The goal of learning is to make the output $y_{n+1}(t)$ as close as possible to the external input $d(t)$. For this purpose we introduce the error function

$$e(t)=\frac{1}{2}(y_{n+1}(t)-d(t))^2. \tag{2.10}$$

The partial Fréchet derivatives of $e(t)$ with respect to the weights of the output unit $w_{n+1,i}\,(i=1,\ldots,n)$ are calculated as follows.

$$\begin{aligned}
\frac{\partial e}{\partial y_{n+1}} &= y_{n+1}(t)-d(t),\\
\frac{\partial e}{\partial w_{n+1,i}} &= \frac{\partial e}{\partial y_{n+1}}\frac{\partial y_{n+1}}{\partial x_{n+1}}\frac{\partial x_{n+1}}{\partial u_{n+1}}\frac{\partial u_{n+1}}{\partial w_{n+1,i}}\\
&= (y_{n+1}(t)-d(t))\frac{1-y_{n+1}(t)^2}{2}(1+\tau_{n+1}\frac{d}{dt})^{-1}y_i(t).
\end{aligned} \tag{2.11}$$

In order to calculate the partial derivatives of $e(t)$ with respect to the weights of the hidden units, we assume that the contribution of $y_i(t)$ to $e(t)$ is made mainly through the direct connection $w_{n+1,i}$. Thus, considering only the direct error propagation from the output unit to the hidden units, we have the following partial derivatives,

$$\begin{aligned}
\frac{\partial e}{\partial y_i} &= \frac{\partial e}{\partial y_{n+1}}\frac{\partial y_{n+1}}{\partial x_{n+1}}\frac{\partial x_{n+1}}{\partial u_{n+1}}\frac{\partial u_{n+1}}{\partial y_i}\\
&= (y_{n+1}(t)-d(t))\frac{1-y_{n+1}(t)^2}{2}(1+\tau_{n+1}\frac{d}{dt})^{-1}w_{n+1,i},\\
\frac{\partial e}{\partial w_{ij}} &= \frac{\partial e}{\partial y_i}\frac{\partial y_i}{\partial x_i}\frac{\partial x_i}{\partial u_i}\frac{\partial u_i}{\partial w_{ij}}\\
&= (y_{n+1}(t)-d(t))\frac{1-y_{n+1}(t)^2}{2}\\
&\quad\cdot(1+\tau_{n+1}\frac{d}{dt})^{-1}w_{n+1,i}\frac{1-y_i(t)^2}{2}(1+\tau_i\frac{d}{dt})^{-1}y_j(t).
\end{aligned} \tag{2.12}$$

Using the equations (2.11) and (2.12) we can derive learning equations as follows. First we define the average error

$$E=\frac{1}{T}\int_0^T e(t)dt. \tag{2.13}$$

The change of the average error ΔE caused by a small change Δw_{ij} in the weight w_{ij} is estimated by

$$\Delta E = \frac{1}{T}\int_0^T \frac{\partial e}{\partial w_{ij}}\Delta w_{ij}dt. \tag{2.14}$$

If we fix the desired output $d(t)$ and the initial state $x_i(0)$, E can be regarded a scalar function of the connection weights $\{w_{ij}\}$, which are fixed during $0 \le t \le T$. Then the partial derivative of E with respect to a weight w_{ij} is

$$\frac{\partial E}{\partial w_{ij}} = \frac{1}{T}\int_0^T \frac{\partial e}{\partial w_{ij}}1\,dt, \tag{2.15}$$

which is the average of a function made by applying the linear operator $\frac{\partial e}{\partial w_{ij}}$ to a unity function. Thus we can use an "epochwise" gradient descent algorithm

$$\Delta w_{ij}(kT) = -\epsilon\frac{1}{T}\int_{(k-1)T}^{kT} \frac{\partial e}{\partial w_{ij}}1\,dt, \tag{2.16}$$

with a positive constant ϵ.

Instead of averaging $\frac{\partial e}{\partial w_{ij}}1$ and updating w_{ij} at the end of each epoch $(k-1)T \le t \le kT$., we can change w_{ij} by the current value of $\frac{\partial e}{\partial w_{ij}}1$ if the change of the weights are sufficiently slow. In this case the learning equation for the weights of the output unit $w_{n+1,i}$ $(i = 1, ..., n)$ is

$$\begin{aligned}\frac{d}{dt}w_{n+1,i}(t) &= -\epsilon_1\frac{\partial e}{\partial w_{n+1,i}}1 \\ &= -\epsilon_1(y_{n+1}(t) - d(t))\frac{1-y_{n+1}(t)^2}{2}(1+\tau_{n+1}\frac{d}{dt})^{-1}y_i(t),\end{aligned} \tag{2.17}$$

and for the weights of the hidden units w_{ij} $(i = 1, ..., n;\ j = 0, 1, ..., n)$ is

$$\begin{aligned}\frac{d}{dt}w_{ij}(t) &= -\epsilon_2\frac{\partial e}{\partial w_{ij}}1 \\ &= -\epsilon_2(y_{n+1}(t) - d(t))\frac{1-y_{n+1}(t)^2}{2} \\ &\quad\cdot(1+\tau_{n+1}\frac{d}{dt})^{-1}w_{n+1,i}(t)\frac{1-y_i(t)^2}{2}(1+\tau_i\frac{d}{dt})^{-1}y_j(t),\end{aligned} \tag{2.18}$$

where ϵ_1 and ϵ_2 are the positive constants which determine the speed of learning. We use these "realtime" learning equations in the following simulations.

The learning equations for the input biases b_i $(i = 0, \ldots, n+1)$ are derived by regarding them as connection weights from a unit whose output is always one. That is,

$$\frac{d}{dt}b_{n+1}(t) = -\epsilon_1(y_{n+1}(t) - d(t))\frac{1-y_{n+1}(t)^2}{2}, \tag{2.19}$$

$$\begin{aligned}\frac{d}{dt}b_i(t) &= -\epsilon_2(y_{n+1}(t) - d(t))\frac{1-y_{n+1}(t)^2}{2} \\ &\quad\cdot(1+\tau_{n+1}\frac{d}{dt})^{-1}w_{n+1,i}\frac{1-y_i(t)^2}{2}, \qquad (i \in H).\end{aligned} \tag{2.20}$$

2.4 Simulations

The abilities of the network were examined by computer simulations. In the following simulations n is the number of the hidden units, T is the period of desired signal $d(t)$, and t_m is the duration of the memorizing mode, or simply the memorizing time. The initial connection weights were chosen randomly with a uniform distribution between $-\delta$ and $+\delta$, and the decay time $\tau_i = 1.0$ for all units.

Since the waveforms that this network can generate are smooth and bounded between -1 and $+1$, we used the desired output of the form

$$d(t) = g(\sum_k (a_k \sin \frac{2\pi kt}{T} + b_k \cos \frac{2\pi kt}{T})). \tag{2.21}$$

2.4.1 Responses of the units in the memorizing mode

In the memorizing mode the response of the output unit to the input signal $d(t)$ changes with the change of the hidden-to-output connection weights. The response of the hidden units also changes gradually with the change of the input-to-hidden and the hidden-to-hidden connection weights by the back-propagation learning.

One example of the changing responses of the hidden and the output units are shown in Figure 2.3. The network has four hidden units and one output unit. We will denote by $y_1, ..., y_4$ the responses of the hidden units and as y_5 the response of the output unit. The input signal $d(t)$, which is also the desired output, is shown by the dotted curve. The waveform of $y_5(t)$ gradually became closer to $d(t)$ with the change of connection weights. If after an insufficient memorizing time ($t_m = 50$) the network was set to regenerating mode then the oscillation could not be sustained (Figure 2.3(b)). With a memorizing time $t_m = 100$, it was found that $y_5(t)$ was sufficiently close to $d(t)$ that the oscillation could be sustained in the regenerating mode (Figure 2.3(c)). After additional memorizing, $t_m = 150$, the period of the autonomous oscillation in the regeneration mode was closer to $d(t)$.

2.4.2 Effectiveness of the learning in the hidden layer

The effectiveness of the continuous-time back-propagation learning in the hidden layer was examined by comparing the regenerated waveforms after learning only with the output unit and those after learning with both the output and hidden units.

In the Figures 2.4, 2.5 and 2.6 the waveforms regenerated after learning only with the output unit are shown on the left side, while those regenerated after learning both with the output and hidden units are shown on the right side. All the learnings were started from the same connection weights, which were uniformly random in $[-2, +2]$. In the case of Figure 2.4 the sinusoidal waveform of period $T = 4$ was successfully regenerated without learning with the hidden units. On the other hand, the desired waveform of period $T = 2$ in Figure 2.5 and the complex waveform in Figure 2.6 were not realized without learning with the hidden units.

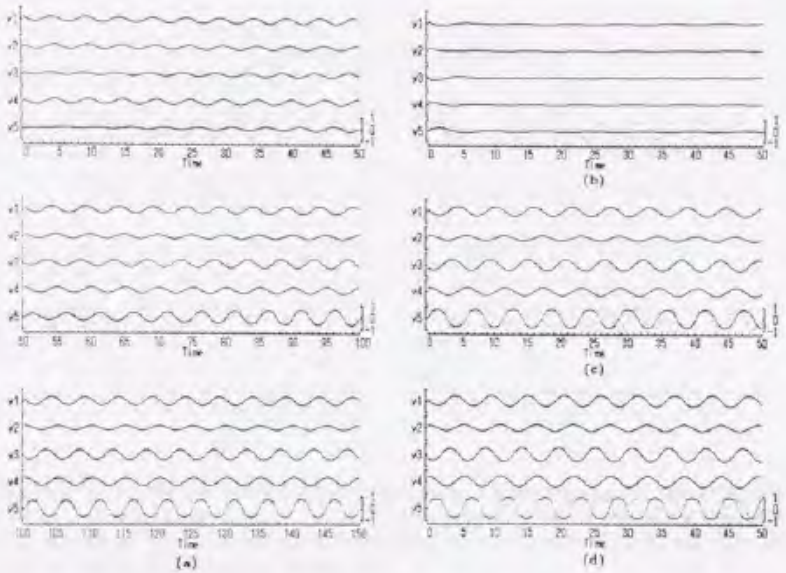


Figure 2.3: Changing responses of the units under learning. (a) memorizing mode, $t_m = 0$ to 150. (b) regenerating mode, $t_m = 50$. (c) regenerating mode, $t_m = 100$. (d) regenerating mode, $t_m = 150$. The number of the hidden units $n = 4$. The solid curve in the frame of y_5 is the response of the output unit and the dotted curve is the forcing input $d(t) = g(2.0 \sin \frac{2\pi}{5.0} t)$, which is also the desired output. The decay time $\tau_i = 1.0$ for all units. The size of random initial connection weights $\delta = 1.0$ and the learning coefficients $\epsilon_1 = \epsilon_2 = 2.0$.

2.4.3 Dependency of the network adaptivity on the number of the hidden units

In Figure 2.7 changes in time of the mean square error are shown in relation to the numbers of hidden units n and the period T of the desired output $d(t)$. Ten simulations were executed for each pair of n and T starting from different random initial connections. From these simulation we note the following: the error decreases faster with an increase in number of hidden units; it takes a longer time to memorize signals with shorter periods ($T = 1.0, 2.0$); in networks with two or four hidden units the errors were not tending to zero for an input period $T = 50$.

2.4.4 Examples of regenerated waveforms

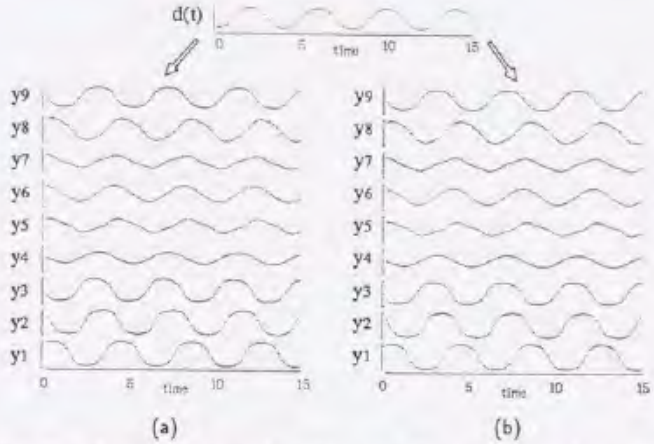


Figure 2.4: The effect of learning in the hidden units. Regenerated waveforms of the output unit y_9 and the hidden units $y_1, \ldots, y_8$ after learning with the desired output waveform $d(t) = g(2.0 \sin \frac{2\pi}{4.0} t)$. The number of the hidden units $n = 8$ and the learning time $t_m = 300$. (a) $\epsilon_1 = 1.0$ and $\epsilon_2 = 0$. (b) $\epsilon_1 = \epsilon_2 = 1.0$.

Examples of regenerated waveforms are shown in Figures 2.8, 2.9, and 2.10. Each of the waveforms shown in the Figures 2.8, 2.9, and 2.10(a) is the regenerated waveforms of the network that gave the least mean square error in the ten simulations of Figure 2.7. The waveforms with short periods ($T = 1.0, 2.0$) tend to be lengthened in regeneration and the opposite holds for long periods ($T = 20, 50$).

In some cases, one network has two or more stable periodic solutions. In the case of Figures 2.10(b) and (c) two symmetric waveforms were regenerated from different initial states $x_i(0)$ (i=1,...,n+1). An interesting observation that can be made is that in this network the positive feedback loop between the units 6 and 8 works like a switching element and the states of the two units ($y_6, y_8 > 0$ or $y_6, y_8 < 0$) determine the mode of oscillation.

In the case of Figure2.10(d) a quasi periodic autonomous oscillation was observed. The output waveform oscillated between the original waveform $d(t)$ and the negative waveform $-d(t)$ with a period of about 56 units of time.

2.4.5 Learning of chaotic oscillation

Although we have so far designed ANO networks for memorizing periodic waveforms, it is also possible to have a neural network learn a chaotic waveform [36]. If we can construct a network which produces the same or similar chaotic output to an unknown system,

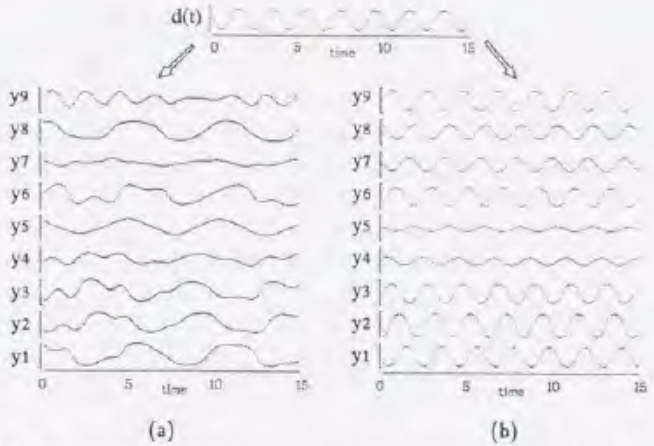


Figure 2.5: The effect of learning in the hidden units. Regenerated waveforms of the output unit y_9 and the hidden units $y_1, \ldots, y_8$ after learning the desired output waveform $d(t) = g(2.0 \sin \frac{2\pi}{2.0} t)$. The number of the hidden units $n = 8$ and the learning time $t_m = 300$. (a) $\epsilon_1 = 1.0$ and $\epsilon_2 = 0$. (b) $\epsilon_1 = \epsilon_2 = 1.0$.

then we have obtained some information about the structure of the system.

As the simplest example of chaotic dynamical system we chose Rössler's system,

$$\begin{aligned} \frac{d}{dt}x &= -y - z, \\ \frac{d}{dt}y &= x + ay, \\ \frac{d}{dt}z &= bx - (c - x)z. \end{aligned} \tag{2.22}$$

The shape of the chaotic attractor with parameters $(a, b, c) = (0.344, 0.4, 5.0)$ is illustrated in Figure 2.11. The trajectory spirals away from the origin in the x-y plane, separates from this plane when x exceeds a threshold determined by c, and then returns to the x-y plane at a point nearer to the origin. The farther the separation point from the origin, the nearer it returns to the origin. This "horseshoe" dynamics leads to the chaotic behavior.

Only one of the three components of (2.22) was given to an ANO network as the desired output

$$d(t) = \frac{1}{12}x(t), \tag{2.23}$$

so that it did not exceed $(-1, +1)$. Although it is theoretically possible to simulate the dynamics of (2.22) with only three units, four units (three hidden and one output) were

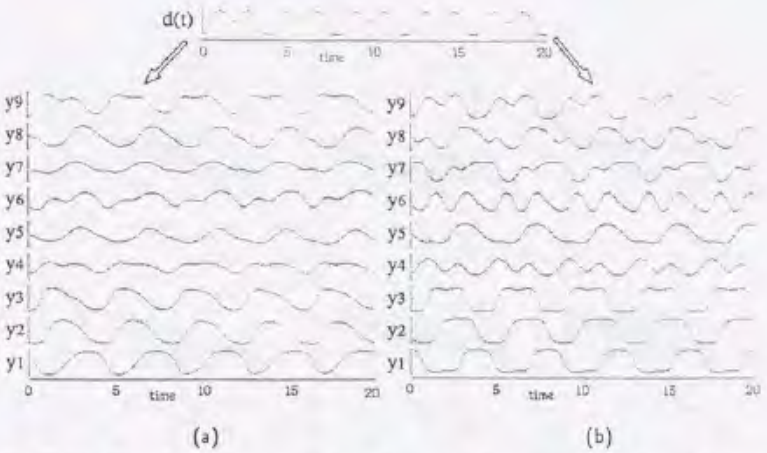


Figure 2.6: The effect of learning in the hidden units. Regenerated waveforms of the output unit y_9 and the hidden units $y_1, \ldots, y_8$ after learning the desired output waveform $d(t) = g(2.0 \sin \frac{2\pi}{4.0} t + 2.0 \cos \frac{2\pi}{2.0} t)$. The number of the hidden units $n = 8$ and the learning time $t_m = 300$. (a) $\epsilon_1 = 1.0$ and $\epsilon_2 = 0$. (b) $\epsilon_1 = \epsilon_2 = 1.0$.

required. Figure 2.12(a) shows the waveforms of autonomous oscillation of the network after memorizing time $t_m = 50,000$. Figure 2.12(b) shows the network trajectory in four-dimension state space projected to a two-dimension plane. Although the shape of the attractor looks different from the original one shown in Figure 2.11, we can still find a scroll, jumps and a horseshoe mechanism.

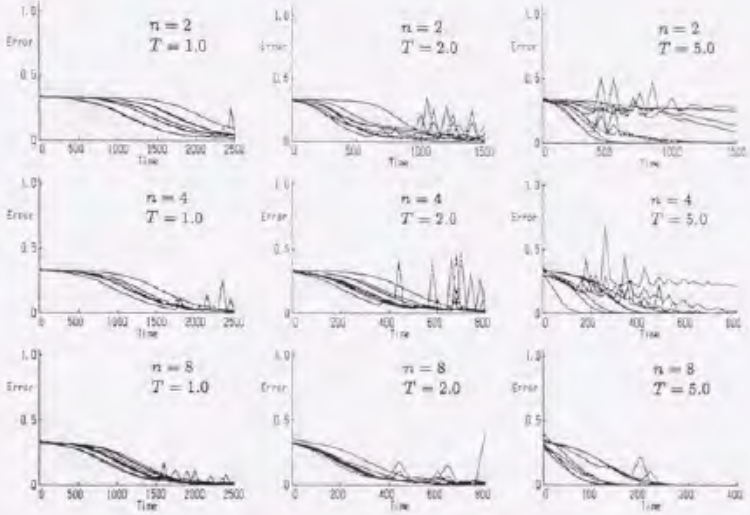


Figure 2.7: Changes in time of the mean square error with different numbers of hidden units and desired oscillating periods. For each setting of the number n of the hidden units and the period T of the external input $d(t) = g(2.0 \sin \frac{2\pi}{T} t)$, 10 different learning phases were executed starting from random initial connection weights w_{ij}, that were uniform in [-1.0, +1.0] ($\delta = 1.0$). The decay time of each unit $\tau_i = 1.0$ and the learning coefficients were $\epsilon_1 = \epsilon_2 = 1.0$.

2.5 Discussions

2.5.1 Recurrent connections in the hidden layer

In deriving the learning rule for the connection weights to the hidden units we have neglected the effect of the indirect connections from a hidden unit to the output units via other hidden units. Accordingly, the partial derivatives given by equation (2.18) are not a good approximation for a hidden unit which is weakly connected to the output unit but strongly connected to other hidden units. In other words, the learning algorithm used is not strictly a gradient descent, but it can successfully decrease the error function as is shown by the computer simulations.

A more general learning algorithm which takes into account error propagation between the hidden units is derived in Chapter 3.

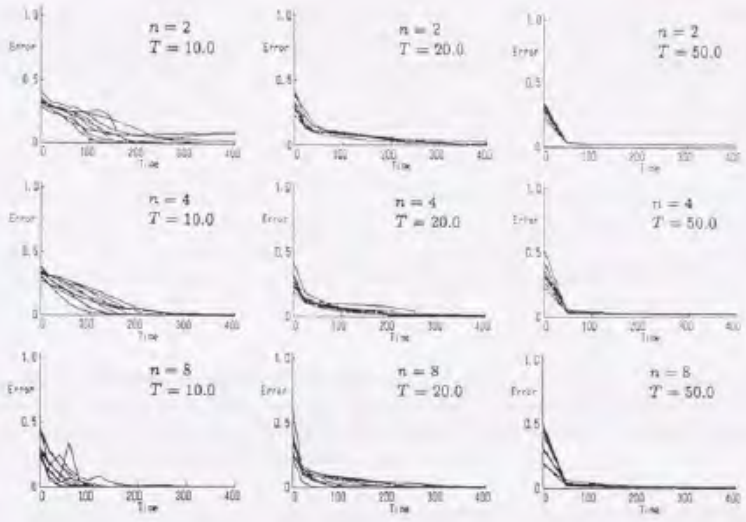


Figure 2.7 (continued)

2.5.2 Initial condition dependence

It is well known that the back-propagation learning algorithm can often be trapped in a local minimum, especially when the desired input-output relationship is complicated. Whether it falls into a global minimum or not depends on the initial connection weights of the network, which are usually set at some random values.

The continuous-time back-propagation learning algorithm inherits the same problem of initial condition dependence. When the desired waveform $d(t)$ is simple (e.g. a sinusoidal waveform), the error goes to zero for almost all initial connection weights, but in the case of complex waveforms (e.g. Figure 2.10.(b)), it was often trapped in some local minima and the error did not go to zero. The frequency of successful learning is dependent on the waveform $d(t)$, the number of hidden units, and the parameters of learning ϵ_1, ϵ_2 and δ.

2.5.3 Stability of the periodic solutions

In the construction of the adaptive neural oscillator network we have made a hypothesis: if the output $y_{n+1}(t)$ is close enough to the input $d(t)$ in the memorizing mode, then the autonomous dynamical system in the regenerating mode has a stable periodic solution with a waveform similar to $d(t)$. This hypothesis was found to be true in almost all

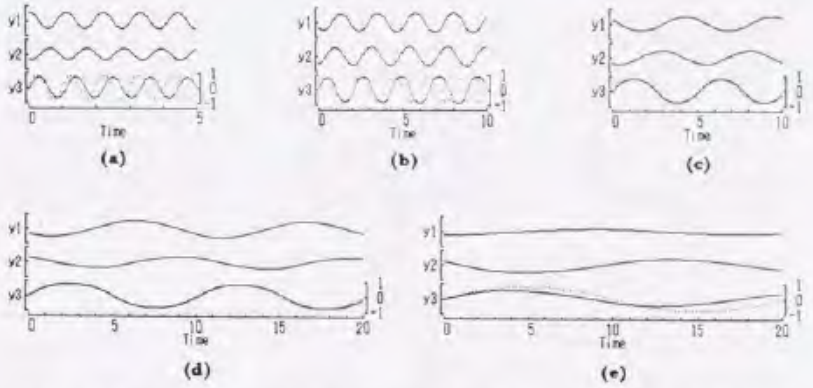


Figure 2.8: Waveforms regenerated with two hidden units. (a) $T = 1.0, t_m = 2500$. (b) $T = 2.0, t_m = 1500$. (c) $T = 5.0, t_m = 1500$. (d) $T = 10.0, t_m = 400$. (e) $T = 20.0, t_m = 400$. The external input $d(t) = g(2.0 \sin \frac{2\pi}{T} t)$. The size of random initial connection weights was $\delta = 1.0$ and the learning coefficients were $\epsilon_1 = \epsilon_2 = 1.0$.

the simulations with simple waveforms, such as sinusoidal and square waveforms. But in rare cases with complex waveforms, as shown in Figure 2.6, even when $y_{n+1}(t)$ was very close to $d(t)$ in the memorizing mode, the waveforms were unstable in the regenerating mode.

In order to avoid such an unstable realization of the output $y_{n+1}(t)$, two methods can be considered. One is to use a mixed input

$$y_0(t) = \alpha d(t) + (1 - \alpha) y_{n+1}(t) \tag{2.24}$$

with $0 < \alpha < 1$ in order to decrease the ratio of forcing input. If we decrease α with the progress of learning, the dynamics of the network is continuously changed into the regeneration mode. Another method is to add some noise to input $y_0(t)$. In both cases, if the mapping $F : y_0(t) \mapsto y_{n+1}(t)$ is not contractive, then the output error become significantly large so that the connection weights are changed.

In general, the stability of a periodic solution of an autonomous dynamical system

$$\frac{d}{dt}\boldsymbol{x}(t) = \boldsymbol{f}(\boldsymbol{x}(t)) \tag{2.25}$$

can be determined by examining the linearized system along the solution [32]

$$\frac{d}{dt}\boldsymbol{y}(t) = A(t)\boldsymbol{y}(t), \tag{2.26}$$

where

$$A(t) = \frac{\partial f_i(\boldsymbol{x}(t))}{\partial x_j(t)}. \tag{2.27}$$

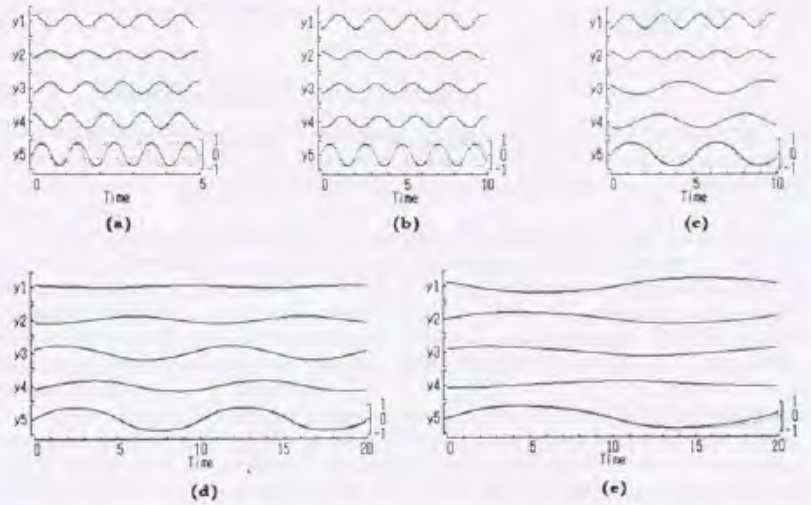


Figure 2.9: Waveforms regenerated with four hidden units. (a) $T = 1.0, t_m = 2500$. (b) $T = 2.0, t_m = 800$. (c) $T = 5.0, t_m = 800$. (d) $T = 10.0, t_m = 400$. (e) $T = 20.0, t_m = 400$. The external input $d(t) = g(2.0 \sin \frac{2\pi}{T} t)$. The size of random initial connection weights was $\delta = 1.0$ and the learning coefficients were $\epsilon_1 = \epsilon_2 = 1.0$.

If the period of the solution $x(t)$ is T, then $A(t)$ is a T-periodic matrix. In the case of ANO network (2.1),

$$A(t) = H(WG(t) - I), \tag{2.28}$$

where $H = \mathrm{diag}(1/\tau_1, \ldots, 1/\tau_{n+1})$, $W = \{w_{ij}\}$ and $G(t) = \mathrm{diag}(g'(x_1(t)), \ldots, g'(x_{n+1}(t)))$.

Let $\Psi(t)$ be a fundamental matrix of the system (2.26), that is, a matrix that satisfies

$$\frac{d}{dt}\Psi = A(t)\Psi, \quad \text{and} \quad \Psi(0) = I. \tag{2.29}$$

Then the solution $\boldsymbol{x}(t)$ of the autonomous system 2.25 is asymptotically stable if only one of the eigen values of $\Psi(T)$, namely λ_1, is equal to one and the others satisfy $|\lambda_i| < 1$ $(i \neq 1)$. We can make an estimation about the product of the eigenvalues from Liouville's law

$$\prod_i \lambda_i = \det \Psi(T) = Det\Psi(0) \exp\left(\int_0^T \mathrm{trace} A(t) dt\right). \tag{2.30}$$

In our case, this implies

$$\prod_{i=2}^{n+1} \lambda_i = \exp\left(-T \sum_{i=1}^{n+1} \frac{1}{\tau_i}\right) = \prod_{i=1}^{n+1} e^{-\frac{T}{\tau_i}} < 1, \tag{2.31}$$

because we assumed that $w_{ii} = 0$. Thus it can be seen that the product of the eigen values is always smaller than one, however, some of them can be greater than one if $n \leq 2$.

2.5.4 Neurobiological interpretation

In this chapter we have employed the bipolar output neuron model ($-1 < y(t) < +1$). If we regard the output $y(t)$ as the impulse frequency of the neuron it can not be less than zero. So we must regard $y(t)$ as the deviation of the impulse frequency from the average of each neuron.

On the basis of the experiments in monkeys [41] and the observations of the behaviors of patients with brain damages [27], it has been suggested that the premotor area and the supplementary motor area of the cerebral cortex participate in the control of skilled motions. It has also been proposed that the pathway from the association cortex to the lateral cerebellum is involved in the control of preprogrammed motions [1]. It is possible that some networks similar to the adaptive neural oscillators are working in those areas, or some other regions in the brain.

It is improbable that back-propagation learning is performed in biological neural networks; we should find some other learning principles without backward error propagation. It has been shown that perceptron-like learning is actually performed in the cerebellum of the rabbit [22]. In the present model, if it is supposed that there are sufficiently many hidden units with random lateral connections, then a learning at the output unit may be sufficient to memorize a variety of waveforms [6].

The structure of the central pattern generators of the lobster [38] and the leech [26] have been identified through exhaustive experiments of intracellular recording with multiple electrodes. This technique requires a great deal of time and effort. Moreover, it is difficult to apply such a method to mammals, since their neurons are very small and the networks are much more complicated. But even when we can only know the output waveform of the central pattern generator, there is a possibility of predicting the structure of the network from the output waveform through simulations of adaptive neural oscillator learning.

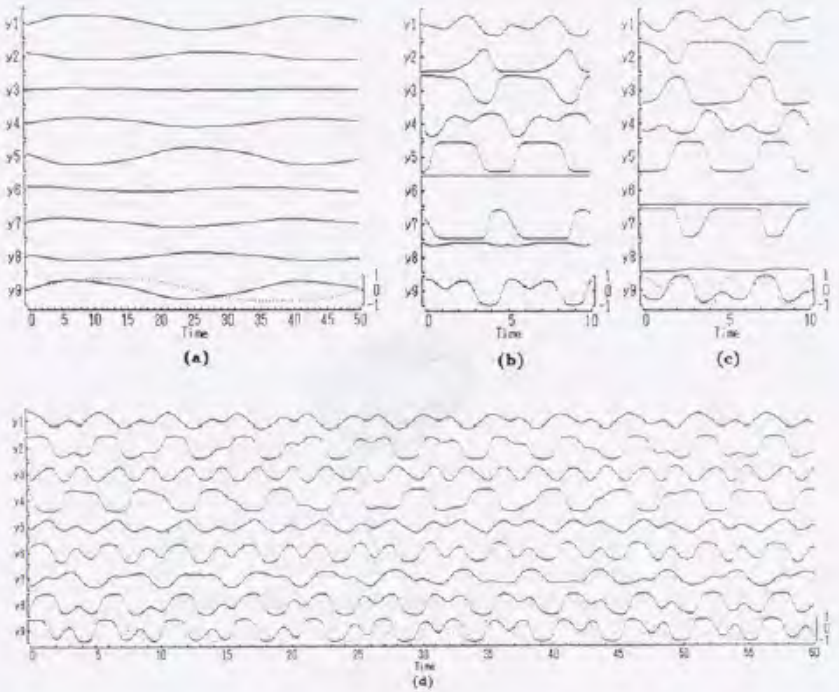


Figure 2.10: Waveforms regenerated with eight hidden units. (a) $d(t) = g(2.0 \sin \frac{2\pi}{T} t), T = 50.0, \delta = 1.0, \epsilon_1 = \epsilon_2 = 1.0, t_m = 400$. (b) and (c) $d(t) = g(2.0 \sin \frac{2\pi}{5.0} t + 2.0 \cos \frac{4\pi}{2.5} t), \delta = 4.0, \epsilon_1 = \epsilon_2 = 0.5, t_m = 2000$. (d) $d(t) = g(2.0 \sin \frac{2\pi}{5.0} t + 2.0 \sin \frac{2\pi}{2.5} t), \delta = 4.0, \epsilon_1 = \epsilon_2 = 0.5, t_m = 2000$.

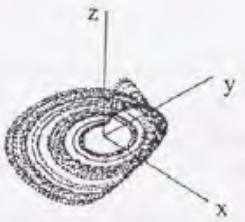
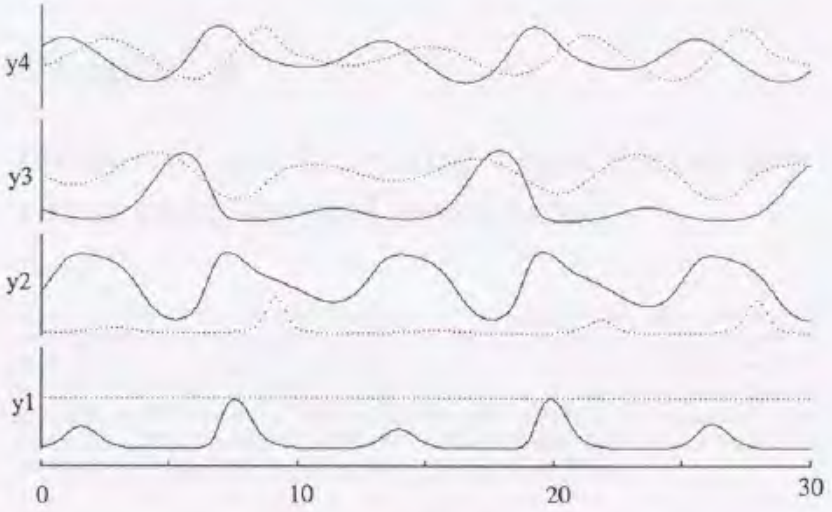
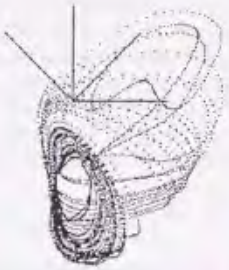


Figure 2.11: The chaotic attractor of Rössler equation.



(a) The waveforms of chaotic oscillation.



(b) The projection of the chaotic attractor.

Figure 2.12: Chaotic oscillation of an ANO network after learning of the output $x(t)$ of Rössler equation; $n = 3$ and $t_m = 50,000$.

Chapter 3

Generalized learning algorithms for recurrent neural networks

In the previous chapter we derived a supervised learning algorithm for memorizing periodic patterns in a continuous-time network with one output unit. In this chapter we generalize the learning algorithms to generating both transient and oscillatory patterns in either discrete-time or continuous-time networks. In the derivation of the learning algorithm in the previous chapter we neglected the indirect error propagation in the hidden layer. We now derive a more strict gradient descent algorithm and compare its performance to that of the simplified one by computer simulations.

3.1 Temporal pattern generator networks

In this chapter we deal with the neural network models as shown in Figure 3.1, which we call temporal pattern generator (TPG) networks. The network consists of three types of units: input, hidden, and output units. The hidden and output units are connected recurrently. The states of the input units determine the temporal patterns of the output. To generate transient patterns some of the input patterns are used as trigger inputs. The desired waveforms are specified for the output units. In the learning of oscillatory patterns the desired signals are substituted for the feedback signals from the output units.

We define the dynamics of TPG networks by the following equation:

$$y_i(t) = g_i(\, h_i \sum_{j \in I \cup H \cup O} w_{ij} y_j(t)), \quad (i \in H \cup O), \tag{3.1}$$

where $y_i(t)$ is the output of the i-th unit at time t; $g_i(\ )$ is a nonlinear output function; h_i is a linear operator of time lag of the i-th unit; w_{ij} is the connection weight from the j-th unit to the i-th unit; and I, H and O are the indices of the input, hidden and output units. Bias inputs are omitted since they can be regarded as connection weights from an input unit whose output is always $y_j(t) \equiv 1$.

If we employ an operator of time delay

$$h_i : \ h_i y(t) = y(t-1), \tag{3.2}$$

then equation (3.1) represents the standard discrete-time model

$$y_i(t) = g_i(\sum_j w_{ij} y_j(t-1)). \tag{3.3}$$

If we choose the operator of first order time-lag

$$h_i : \ u(t) \mapsto x(t) \quad \text{s.t.} \quad \tau_i \frac{d}{dt} x(t) = -x(t) + u(t), \tag{3.4}$$

then (3.1) represents the dynamics of a continuous-time model

$$\begin{aligned} \tau_i \frac{d}{dt} x_i(t) &= -x_i(t) + \sum_j w_{ij} y_j(t), \\ y_i(t) &= g_i(x_i(t)). \end{aligned} \tag{3.5}$$

In the following sections we derive two versions of learning algorithms for the general network model defined by equation (3.1). One is the generalized version of the continuous-time back-propagation learning algorithm derived in the previous chapter, which take into account only local dependency between the states of the units. Another is a more strict gradient learning algorithm in which the global interdependency of the states of the units is represented in a simultaneous dynamical equation system. In the last section, learning algorithms for decay times of the units are derived.

3.2 Direct back-propagation algorithm

First we derive a learning algorithm which take into account only local relationships between the states of the units. As described in the previous chapter we use Fréchet derivatives of the mappings between the functions of time and notationally express them in the same way as the derivatives of vector functions.

If we differentiate equation (3.1) for i-th unit by a weight w_{ij} of the same unit, we wave

$$\frac{\partial y_i}{\partial w_{ij}} = g_i'(t)\, h_i\, y_j(t) \qquad (i \in H \cup O; j \in I \cup H \cup O), \tag{3.6}$$

where $g_i'(t)$ denotes the gradient $\dfrac{dg(x)}{dx}$ at $x_i(t) = g^{-1}(y_i(t))$. We neglected the recurrent effects of a change in w_{ij} on the inputs $y_j(t)$ to the i-th unit. A gradient learning algorithm for the weights of output units are derived from (3.6).

In a recurrent network, the state of a hidden unit gives effects on the state of an output unit through many different pathways as shown in Figure 3.2. However, by taking into account only the direct connection from a hidden unit to an output unit, we can derive a simple learning algorithm which we call "direct back-propagation" learning algorithm. If we differentiate equation (3.1) for k-th output unit by the output y_i of a hidden unit, we wave

$$\frac{\partial y_k}{\partial y_i} = g_k'(t)\, h_k\, w_{ki} \qquad (k \in O; i \in H). \tag{3.7}$$

Applying the chain rule for (3.6) and (3.7), we have the differential relationship between k-th output unit and a weight of i-th hidden unit

$$\begin{aligned} \frac{\partial y_k}{\partial w_{ij}} &= \frac{\partial y_k}{\partial y_i}\frac{\partial y_i}{\partial w_{ij}} \\ &= g_k'(t)\, h_k\, w_{ki} g_i'(t) h_i y_j(t), \qquad (k \in O; i \in H; j \in I \cup H \cup O). \end{aligned} \tag{3.8}$$

A gradient learning algorithm for the weights of hidden units are derived from (3.8).

We define an error function of the i-th unit ($i \in O$)

$$e_i(t) = \frac{1}{2}(y_i(t) - d_i(t))^2 \tag{3.9}$$

and the total error

$$e(t) = \sum_{i \in O} e_i(t). \tag{3.10}$$

In general, the output of one output unit may give effects on other output units. However, if we want the total error $e(t)$ to be zero, then the error at each output unit $e_i(t)$ must be zero. Thus we assume that the learning in the i-th output unit should be made only to decrease $e_i(t)$ and approximate the differential relationship between a weight w_{ij} of an output unit and the total error $e(t)$ by

$$\begin{aligned} \frac{\partial e}{\partial w_{ij}} &= \frac{\partial e_i}{\partial w_{ij}} = \frac{\partial e_i}{\partial y_i}\frac{\partial y_i}{\partial w_{ij}} \qquad (i \in O, j \in I \cup H \cup O) \\ &= (y_i(t) - d_i(t)) g_i'(t) h_i y_j(t). \end{aligned} \tag{3.11}$$

Similarly, the derivative of $e(t)$ with respect to a weight w_{ij} to i-th hidden unit is approximated by

$$\begin{aligned}\frac{\partial e}{\partial w_{ij}} &= \sum_{k\in O}\frac{\partial e_k}{\partial y_k}\frac{\partial y_k}{\partial y_i}\frac{\partial y_i}{\partial w_{ij}}(i\in H; j\in I\cup H\cup O)\\ &= \sum_{k\in O}(y_k(t)-d_k(t))g_k'(t)\,h_k\,w_{ki}g_i'(t)\,h_i\,y_j(t).\end{aligned} \tag{3.12}$$

In order to decrease the average error

$$E=\frac{1}{T}\int_0^T e(t)dt, \tag{3.13}$$

we can use "epochwise" gradient descent algorithm

$$\begin{aligned}\Delta w_{ij} &= -\epsilon\frac{\partial E}{\partial w_{ij}}\\ &= -\epsilon\frac{1}{T}\int_0^T\frac{\partial e}{\partial w_{ij}}1dt.\end{aligned} \tag{3.14}$$

Instead of averaging $\frac{\partial e}{\partial w_{ij}}1$ and updating the weights at the end of each period $(k-1)T\le t\le kT$, we achieve almost equivalent learning by continuously updating weights by $epsilon P(e,w_{ij})1$ provided ϵ is sufficiently small. This "real-time" learning algorithm for an output unit of a continuous-time model is given by

$$\begin{aligned}\frac{d}{dt}w_{ij} &= -\epsilon\frac{\partial E}{\partial w_{ij}}1\\ &= \begin{cases}-\epsilon(y_i(t)-d_i(t))g_i'(t)h_iy_j(t) & (i\in O),\\ -\epsilon\sum_{k\in O}(y_k(t)-d_k(t))g_k'(t)\,h_k\,w_{ki}g_i'(t)\,h_i\,y_j(t) & (i\in H).\end{cases}\end{aligned} \tag{3.15}$$

It has been shown by computer simulations that this "direct back-propagation" algorithm, which takes into account only the direct effect of a hidden unit on the output units, was sufficient in the learning of oscillatory patterns [8] and the prediction of sequences produced by a simple automaton [14].

3.3 Simultaneous back-propagation algorithm

If we want to estimate the recurrent influence of a change in a weight of a unit on all of the hidden and output units in the network, it is necessary to express the interdependence in a set of simultaneous equations [8, 11, 45]. Differentiating (3.1) for the k-th unit by a weight w_{ij} ($i \in H \cup O$, $j \in I \cup H \cup O$) we have the following simultaneous equations for ($k \in H \cup O$)

$$\begin{aligned} \frac{\partial y_k}{\partial w_{ij}} &= g_k'(t)\, h_k\Big(\sum_{l \in I \cup H \cup O} \frac{\partial}{\partial w_{ij}} (w_{kl} y_l(t)) \Big) \\ &= g_k'(t)\, h_k\Big(\sum_{l \in H \cup O} w_{kl} \frac{\partial y_l}{\partial w_{ij}} + \delta_{ki} y_j(t) \Big), \end{aligned} \tag{3.16}$$

where δ_{ki} is Kronecker's delta function. Note that if we restrict the range of summation $\sum_l w_{kl} \frac{\partial y_l}{\partial w_{ij}}$ to $k \in O$ and $l \in H$, (3.16) is equivalent to (3.6) and (3.7) of direct algorithm.

The operator equation system (3.16) implies that functions $p_k^{ij} = \frac{\partial y_k}{\partial w_{ij}}1$, the change in y_k caused by a small change in the weight w_{ij}, obeys the simultaneous equation

$$p_k^{ij}(t) = g_k'(t)\, h_k\Big(\sum_{l \in H \cup O} w_{kl} p_l^{ij}(t) + \delta_{ki} y_j(t) \Big), \quad (k \in H \cup O). \tag{3.17}$$

This is the linearized system along the trajectory of the network dynamics (3.1) with a forcing input $y_j(t)$ in the i-th component. If the trajectory of the network dynamics is a stable one, such as a limit cycle, then this linearized dynamical system has a stable solution that we can compute from the initial condition

$$p_k^{ij}(0) = 0, \tag{3.18}$$

which means that the initial state of the network does not depend on the connection weights.

We can derive gradient algorithms using the solution of (3.17) and (3.18). A discrete-time version is of this algorithm was derived by Williams & Zipser [45] and a continuous-time version by Doya & Yoshizawa [8]. The real-time algorithm for continuous-time network is expressed as,

$$\begin{aligned} \frac{d}{dt} w_{ij} &= -\epsilon \frac{\partial e}{\partial w_{ij}} \\ &= -\epsilon \sum_{k \in O} \frac{\partial e_k}{\partial y_k} \frac{\partial y_k}{\partial w_{ij}} \\ &= -\epsilon \sum_{k \in O} (y_k(t) - d_k(t)) p_k^{ij}(t). \end{aligned} \tag{3.19}$$

When we implement this "simultaneous back-propagation" algorithm special attentions must be paid to the stability of the linear system (3.17). With the change of the weights by learning the network trajectory can transiently become unstable. In such

a case the linearized system will become unstable and some of the components p_i^{ij} can increase exponentially in time, which causes a destructively rapid change of the weights. One way to avoid this is to limit the speed of weight change, for example, by a sigmoid function.

3.4 Learning of decay times

In the simulations of the previous chapter it was seen that a continuous-time network with uniform decay time τ for each unit can learn oscillatory signals with periods in the range τ to 20τ. However, it is desirable that the decay time of each unit is adjustable so that the network can efficiently learn the given temporal pattern. In this section we will derive two learning equations for the decay times, which correspond to the preceding two algorithms for connection weights.

If we differentiate equation (3.6) with respect to τ_i, we have

$$\frac{d}{dt}x_i + \tau_i \frac{d}{dt}\frac{\partial x_i}{\partial \tau_i} = -\frac{\partial x_i}{\partial \tau_i} + \frac{\partial \sum_j w_{ij} y_j}{\partial \tau_i}. \tag{3.20}$$

If we assume that the inputs $y_j(t)$ do not depend on the decay time τ_i we have

$$\tau_i \frac{d}{dt}\frac{\partial x_i}{\partial \tau_i} = -\frac{\partial x_i}{\partial \tau_i} - \dot{x}_i, \tag{3.21}$$

Thus the differential relationship between the decay time and the state of the unit is expressed as

$$\frac{\partial x_i}{\partial \tau_i} = -h_i \dot{x}_i. \tag{3.22}$$

By the same approximations used in deriving direct back-propagation algorithm, the following learning rules for the decay time of the output and hidden units are derived;

$$\begin{aligned}
\frac{d}{dt}\tau_i &= -\epsilon \frac{\partial e}{\partial y_i}\frac{\partial y_i}{\partial x_i}\frac{\partial x_i}{\partial \tau_i} 1 \\
&= -\epsilon (y_i(t) - d_i(t)) g_i'(t) h_i \dot{x}_i(t), \qquad (i \in O), \qquad (3.23) \\
\frac{d}{dt}\tau_i &= -\epsilon \sum_{k \in O} \frac{\partial e}{\partial y_k}\frac{\partial y_k}{\partial y_i}\frac{\partial y_i}{\partial x_i}\frac{\partial x_i}{\partial \tau_i} 1 \\
&= -\epsilon \sum_{k \in O} (y_k(t) - d_k(t)) g_k'(t) h_k w_{ki} g_i'(t) h_i \dot{x}_i(t), \quad (i \in H). \qquad (3.24)
\end{aligned}$$

In order to take into account the global interdependency of the states of the units, we differentiate the equation (3.6) for k-th unit by τ_i, giving

$$\delta_{ki}\frac{d}{dt}x_k(t) + \tau_k \frac{d}{dt}\frac{\partial x_k}{\partial \tau_i} = -\frac{\partial x_k}{\partial \tau_i} + \sum_{l \in H \cup O} w_{kl} g_l'(t) \frac{\partial x_l}{\partial \tau_i}. \tag{3.25}$$

Thus we have the following differential relationship between the decay time and the states of the units

$$\frac{\partial x_k}{\partial \tau_i} = -h_k \Big(\sum_{l \in H \cup O} w_{kl} \frac{\partial x_l}{\partial \tau_i} - \delta_{ki} \dot{x}_k(t) \Big). \tag{3.26}$$

Hence, the simultaneous back-propagation algorithm for a decay time is

$$\frac{d}{dt}\tau_i = -\epsilon \sum_{k \in O} \frac{\partial c}{\partial y_k} \frac{\partial y_k}{\partial x_k} \frac{\partial x_k}{\partial \tau_i} 1. \tag{3.27}$$

When implementing these algorithms it is necessary to impose an additional constraint so that the decay time do not become negative.

3.5 Simulations

Results of computer simulations that compare the performances of the two algorithms derived in the preceding sections were compared in three learning tasks, each from five sets of random initial connection weights. Figure 3.3 shows the error curves of the learning of two sinusoidal waveforms with different periods. In this case direct algorithm had better performance compared to simultaneous algorithm. Figure 3.4 is the learning of a complex waveform. In this case simultaneous algorithm is working better. Figure 3.5 is the learning of the chaotic oscillation of Roessler equation. Also in this case, simultaneous algorithm achieved lesser errors.

The simultaneous algorithm requires $O((n_O + n_H)^3(n_O + n_H + n_I))$ computations at each time step, whereas the direct algorithm requires $O(n_O n_H(n_O + n_H + n_I))$ computations where n_O, n_H, n_I denotes the numbers of output, hidden and input units respectively. If we provide many hidden units—which is required when the target temporal pattern has a complex waveform—the difference is very large.

Therefore it can be concluded that the simultaneous back-propagation algorithm is advantageous only when we want a network with small numbers of units to learn a complex waveform.

3.6 Discussions

3.6.1 Back-propagation through time

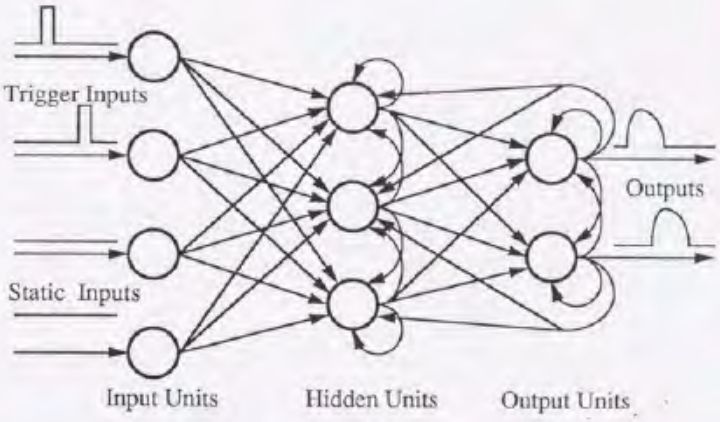
In discrete-time recurrent networks it is possible to apply the back-propagation algorithm of feed-forward networks by unrolling the network as shown in Figure 3.6 [35]. Pearlmutter extended this idea to continuous-time networks using a discrete-time approximation at infinitesimal time steps [29]. These schemes are called "back-propagation through time" [46].

However, these algorithms are not suitable for continuously running networks because we must compute the network dynamics forward in time, save the history of the network state, and then calculate the error propagation backward in time.

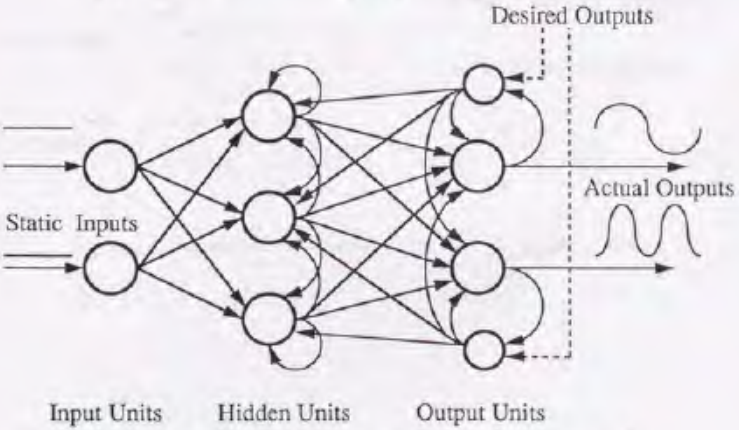
3.6.2 Bifurcation of the network dynamics

To apply gradient descent learning algorithms to recurrent networks we have assumed that the error function is smooth in the space of network parameters. It is true in feed-forward networks with smooth output functions. However, in recurrent networks the output changes discontinuously with a change of parameters at some of the bifurcation points—such as at saddle-node bifurcations and subcritical Hopf bifurcations. At such points gradient descent learning is not valid since it assumes continuity of the error function. In computer simulations it is sometimes observed that error functions abruptly increase, see Figures 2.7, 3.3, 3.4 and 3.5.

Since bifurcation points are usually measure zero in the space of parameters, we may expect that the gradient decent algorithms are almost always valid. However, in most of the significant learning tasks, such as learning of multistable dynamics and limit cycles, the network must pass through bifurcation points. In the worst case, the learning process will never converge. This is a fundamental problem of supervised learning in recurrent neural networks yet to be investigated.



(a) Model of a transient temporal pattern generator network.



(b) Model of an oscillatory temporal pattern generator network.

Figure 3.1: Structure of temporal pattern generator network.

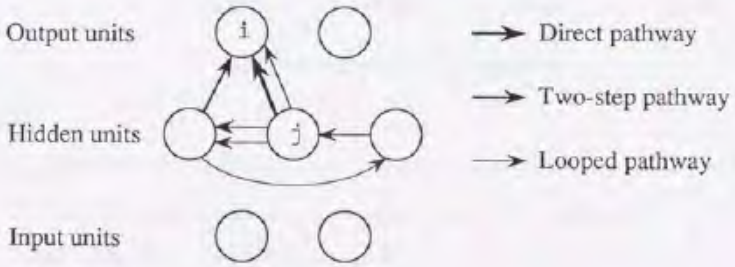
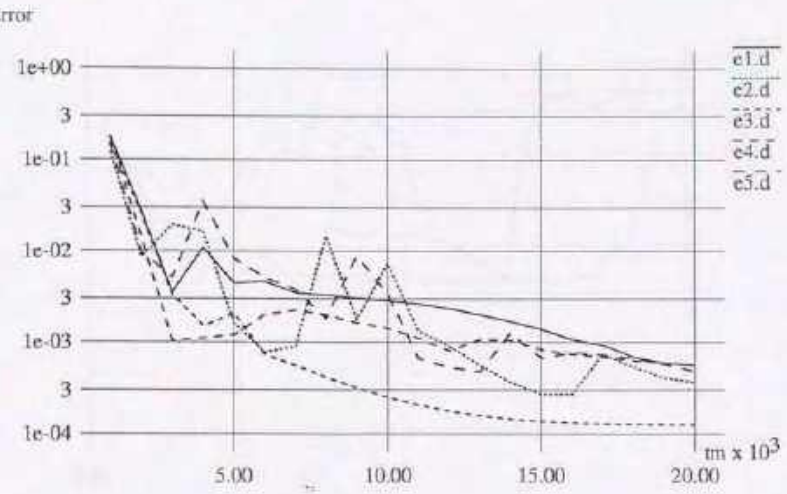
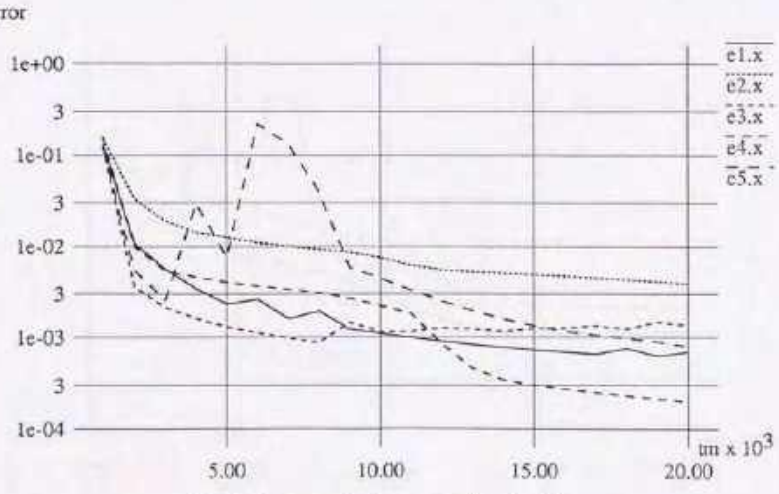


Figure 3.2: Signal pathways from a hidden unit to an output unit.

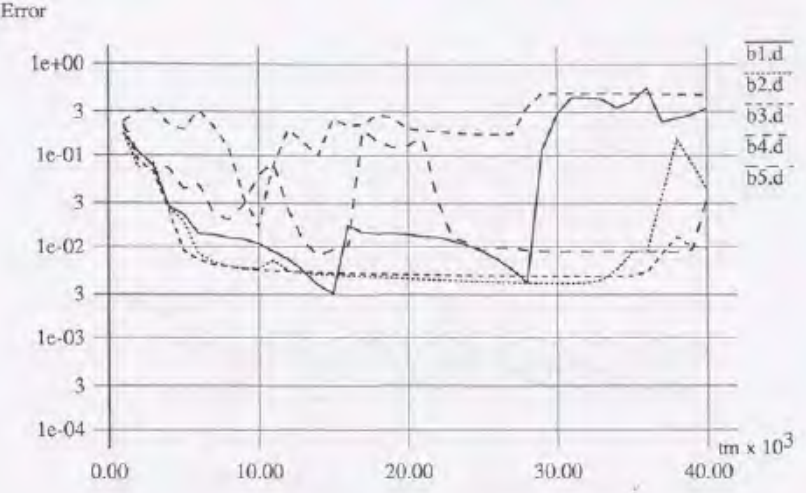


(a) Direct back-propagation algorithm.

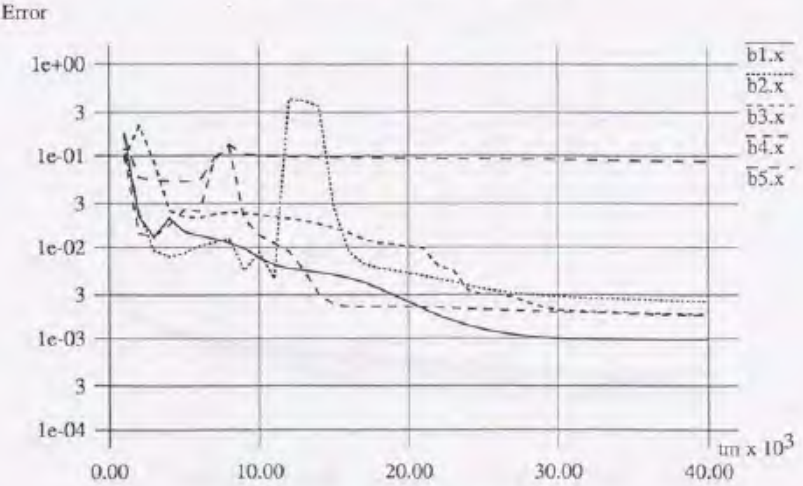


(b) Simultaneous back-propagation algorithm.

Figure 3.3: Comparison of the two algorithms in learning sinusoidal waveforms $d_1(t) = g(2.0\sin(\frac{2\pi}{5.0}t))$ and $d_2(t) = g(2.0\sin(\frac{2\pi}{10.0}t))$; $n_O = 2$ and $n_H = 4$.

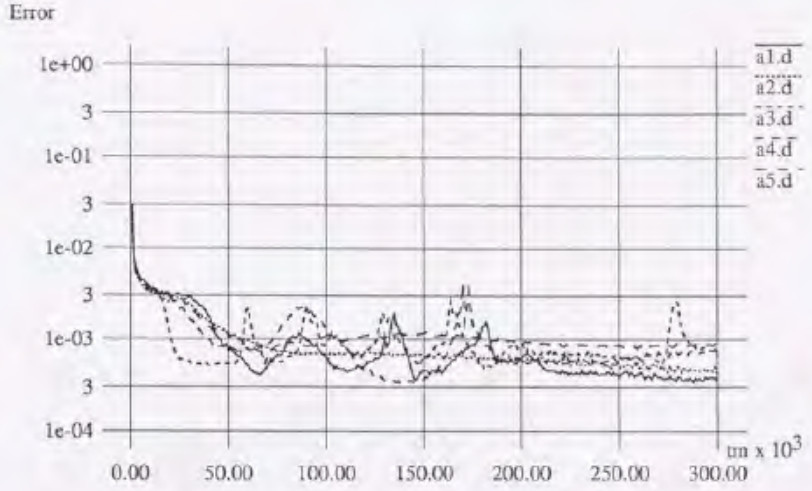


(a) Direct back-propagation algorithm.

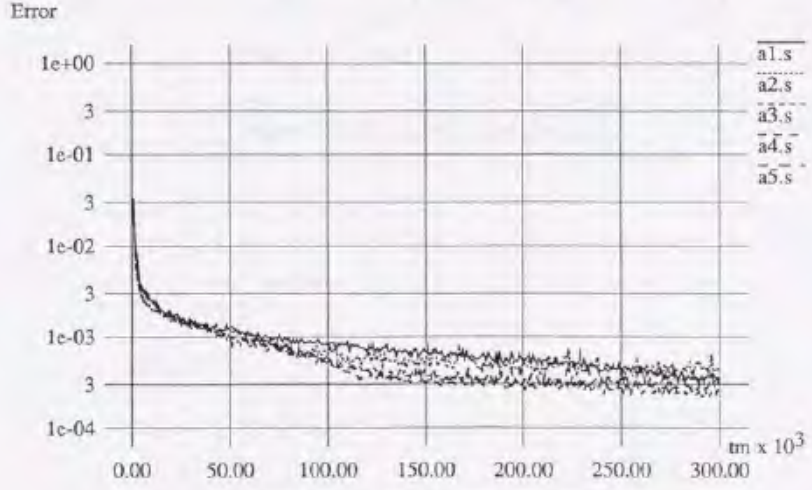


(b) Simultaneous back-propagation algorithm.

Figure 3.4: Comparison of the two algorithms in learning a complex waveform $d(t) = g(2.0\cos(\frac{2\pi}{5.0}t) + 2.0\sin(\frac{2\pi}{10.0}t))$; $n_O = 1$ and $n_H = 6$.



(a) Direct back-propagation algorithm.



(b) Simultaneous back-propagation algorithm.

Figure 3.5: Comparison of the two algorithms in learning the chaotic waveforms of Roessler equation (2.22); $n_O = 2$ and $n_H = 4$.

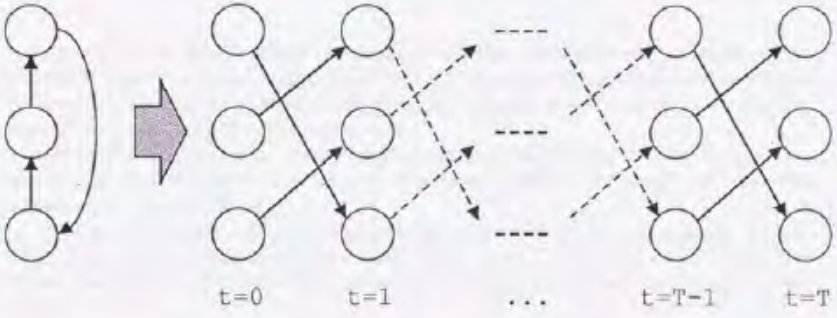


Figure 3.6: An unrolled recurrent network.

Chapter 4

Control of regenerated temporal patterns

It is known that a central pattern generator network can have multiple oscillating modes and one of them is selected by the input from the higher centers [26, 38]. In this chapter we investigate schemes for memorizing multiple temporal patterns in one network and controlling temporal patterns of regeneration.

One approach is to include control inputs to the network, as mentioned in the previous chapter, so that the network is trained to generate different temporal patterns with different static control inputs.

Another scheme is to select one of the multiple attractors in the state space of the network by an initial state configuration. This scheme is similar to the Hopfield associative memory except that the network converges to a the limit cycle instead of an equilibrium point.

A third possibility is regeneration of a sequence after the presentation of a part of it. Sequential memory of humans, such as episodes of experiences, sentences and songs, have such characteristics.

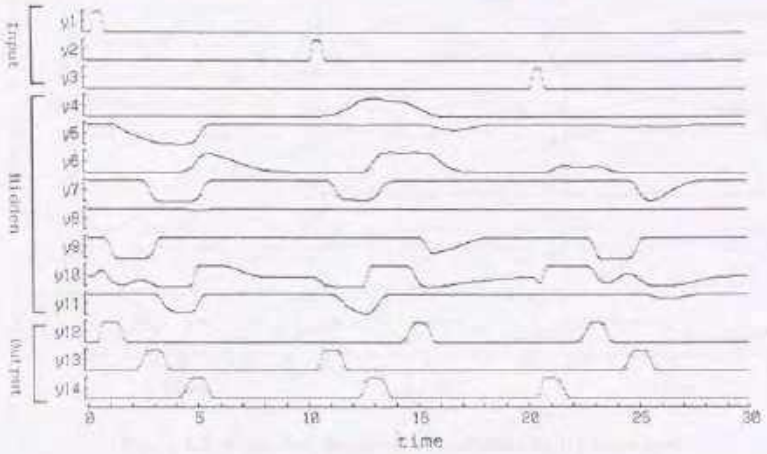


Figure 4.1: Three transient temporal patterns generated by different trigger inputs.

4.1 Selection and modulation of temporal patterns by static inputs

In the network models shown in Figure 3.1 the input patterns modify activation levels of the hidden and output units and modulate the output waveforms. Figure 4.1 shows an example of a simulation where transient patterns are generated. One of the three different transient sequences are selected by a trigger input given to one of the input nodes.

Figure 4.2 shows an example where the period of the oscillation is controlled by the the input level. The network has one input unit, four hidden units and one output unit. It was trained to generate sinusoidal waveforms with period 2.0, 3.0 and 4.0 at the input levels -1, 0, and $+1$, respectively. Figure 4.3 shows the relationship between the input level and the period of oscillation. The period is continuously controlled by the input level.

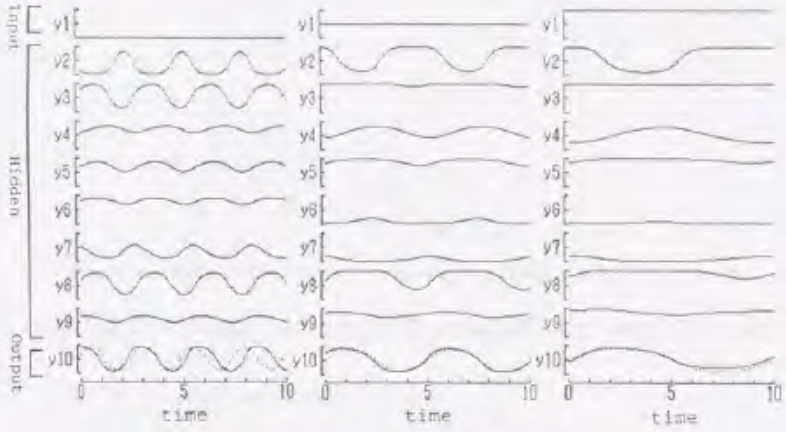


Figure 4.2: Control of the period of oscillation by the input level.

4.2 Selection of limit cycles by initial states

We can consider the ANO learning process as a Hopf bifurcation. At the beginning of the learning the connection weights are given by small random values and the origin is a unique global equilibrium point of the network because each unit has a decay term. After some learning the origin becomes a stable focus. With additional learning the origin becomes unstable and a limit cycle attractor emerges around it.

If there are distinct stable equilibria in separate regions of the phase space of the network then by inducing a Hopf bifurcation at each equilibrium there can be produced multiple limit cycle attractors. In order to prepare multiple equilibrium attractors the principle of an associative memory using the autocorrelation matrix [28, 20] can be employed.

The following algorithm was used in simulations.

1. Choose m pairs of key patterns $\mathbf{s}^k = (s_1^k, ..., s_n^k)^t$ $(|s_i^k| < 1)$ one for each of the required periodic waveforms $d^k(t)$ $(k = 1, ..., m)$.

2. Let the initial connection weights be

$$w_{ij} = \sum_{k=1}^{m} s_i^k s_j^k + \mu_{ij} \quad (i, j = 1, ...n), \tag{4.1}$$

where μ_{ij} are small random numbers. If the number of key patterns m is small enough with respect to the number of the hidden units n, there are stable equilibria in a neighborhood of each of the key patterns.

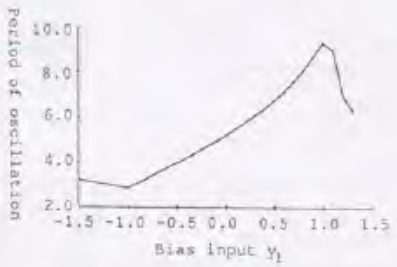


Figure 4.3: Relationship between the period of oscillation and the input level.

3. Repeat ANO learning until the error becomes sufficiently small for all patterns $k = 1, ..., m$. Namely,

   (a) set the initial state of the network as

   $$y_i(0) = s_i^k, \quad x_i(0) = g^{-1}(s_i^k) \quad (i = 1, ..., n). \tag{4.2}$$

   (b) then execute the continuous-time back-propagation learning with the input waveform $d^k(t)$, which is the desired output waveform, for a given time t_1.

After the completion of this algorithm if the network is set to regenerating mode with an initial state near $\mathbf{s}^k$ then a limit cycle attractor with some output waveform similar to $d^k(t)$ will be generated. This can be seen as a process of associative memory between the static patterns and the dynamical patterns.

The result of a computer simulation is shown in Figure 4.4. The number of key patterns $m = 2$ and the number of hidden units $n = 20$. Two key patterns were made of uniform random numbers in the interval $[-1.0, +1.0]$. The memorized waveforms were $d^1(t) = g(2.0 \sin \frac{2\pi t}{10.0})$ and $d^2(t) = g(2.0 \sin \frac{2\pi t}{5.0})$. Figure 4.4(a) is a two-dimensional projection of the 21-dimensional phase space. The 21 arrows are unit vectors for y_i $(i = 1, ..., 21)$. There are two distinct limit cycles. The output waveforms corresponding to these limit cycles are shown in Figure 4.4 (b) and (c).

Figure 4.5 shows the state space of a network with 50 hidden units which has memorized three different waveforms. It is well known that an associative memory network with n units can at most memorize $0.15n$ stable patterns but it is not yet clear how many limit cycles can be stored in a network.

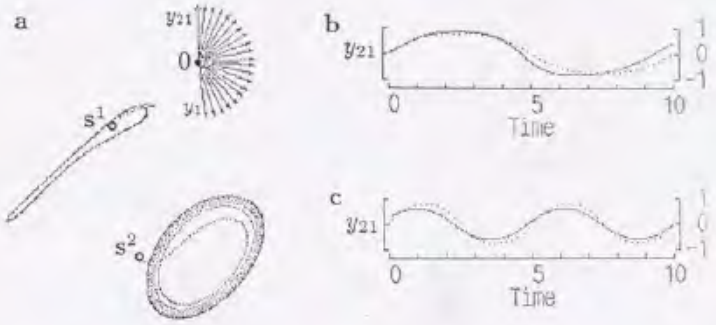


Figure 4.4: Two limit cycles memorized in a network of 21 units. (a) The phase space of the 21-neuron network. (b) and (c) Regenerated waveforms starting from the initial states s^1 and s^2.

4.3 Recalling a sequence from its part

It is a common ability of people to be able to recall long phrases or melodies just hearing a part of them. Such a recalling mechanism can be modeled by the learning in ANO network.

A simulation was performed with a network with six hidden units and three output units. The network was alternately trained to generate two different periodic sequences

1 2 3 1 2 3 ...

and

1 3 2 1 3 2 ...,

where 1, 2, and 3 represent a pulse-like output from one of three output units. After 20000 units of time, the network acquired two different limit cycle attractors.

Figure 4.6 shows the regeneration of the memorized sequences. In the first four units of time the state of the output units were replaced with one of the memorized sequences. After that the network was set into its autonomous dynamics. In Figure 4.6(a) from the partial sequence "2 1", one of the memorized sequence "2 1 3 2 1 3 2 ..." was recalled and in Figure 4.6(b) the sequence "3 1 2 3 1 2 3 ..." was recalled from the partial sequence "3 1".

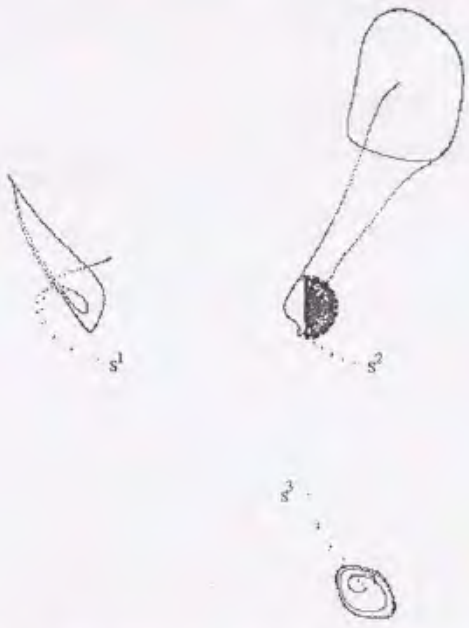


Figure 4.5: Three limit cycles memorized in a network of 51 units.

4.4 Generation of complex patterns

It is relatively difficult to train a network to oscillate with complex waveforms. However, by connecting together several ANO networks each of which generating relatively simple waveforms we can construct a large scale network which can produce various and complex waveforms.

Figure 4.7 is an example with two ANO networks. The higher ANO outputs the patterns $(y_6, y_7) = (1,0)$ and $(0,1)$ alternately. Its period is controlled by the level of the the input y_1. The lower ANO has three output units. Its output sequences are determined by the input patterns given from the higher ANO as follows:

input pattern		output sequence
$(y_6, y_7) = (1,0)$	$\rightarrow$	1 2 3 1 2 3 ...
$(0,1)$	$\rightarrow$	1 3 2 1 3 2 ...

Figure 4.8 shows examples of the waveforms generated by this network. The period of oscillation of the higher ANO was about four times that of the lower ANO and the output sequence was

1 2 3 1 2 3 1 3 2 1 3 2 1 2 3 1 2 3 ...

When the level of the input to the higher ANO is changed different sequences were generated in the lower ANO.

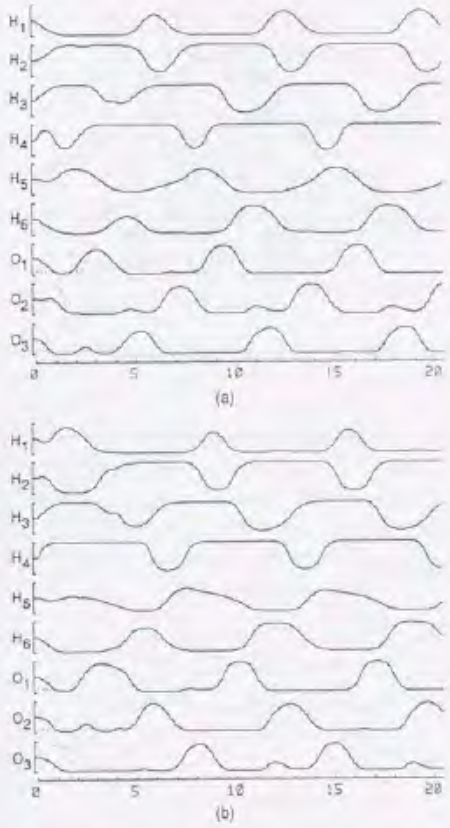


Figure 4.6: Recalling sequences from the parts of them.

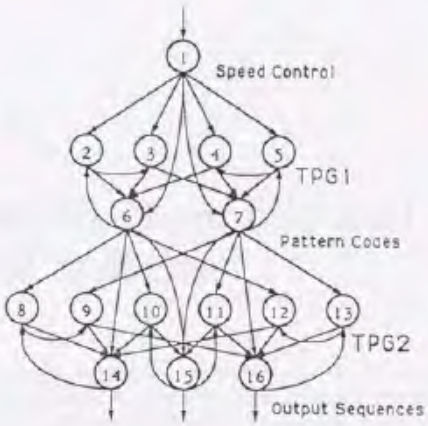


Figure 4.7: Hierarchical connection of two ANO networks.

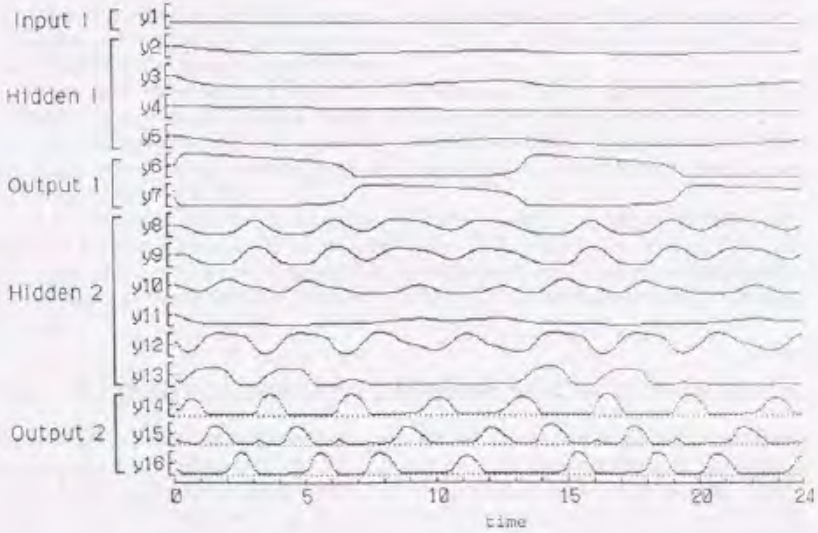


Figure 4.8: Waveforms regenerated by the hierarchical ANO network.

Chapter 5

Coordination of neural and physical dynamical systems

The skilled control of human and animal motion is the result of the cooperative interaction of neural networks and physical systems—muscles, skeltons, physical environments and sensory organs. In this chapter we investigate the process of coordination of neural and physical dynamical systems.

It has been shown from physiological studies that animal motion is realized by a distributed neural control system as shown in Figure 5.1. Each part of the body is controlled by a local CPG network which can autonomously generate a rhythmical pattern. However, it is essential that the temporal patterns generated by CPGs are matched to the characteristics of the physical systems and that the oscillation of different CPGs are synchronized with the appropriate phases.

It has been experimentally shown that the temporal patterns generated by a CPG is dependent on sensory feedback inputs, the lateral inputs from other CPGs, and the command inputs from the brain, since the waveform and period of oscillation of a CPG are disordered when the sensory feedback fibers or lateral connections between other CPGs are removed [19, 30].

In the following sections we investigate the synchronizing mechanism of neural and physical dynamical systems. First we investigate synchronization of network dynamics and physical systems, which is essential in locomotion control. Then we investigate the synchronization of multiple CPG networks, which is important in multipedal locomotion control.

5.1 Synchronization of physical and neural systems

Many neural network model have been proposed for the control of motion trajectories, such as the motion of an arm [24, 23], that calculate the motor commands by compensating for the unfavorable characteristics of the physical systems—such as nonlinearity, excess degree of freedom and time-lag. In the control of locomotion, however, trajectories of limbs need not be given explicitly but they should be determined so that the animal can move efficiently and stably. In order to achieve stable and efficient locomotion it is

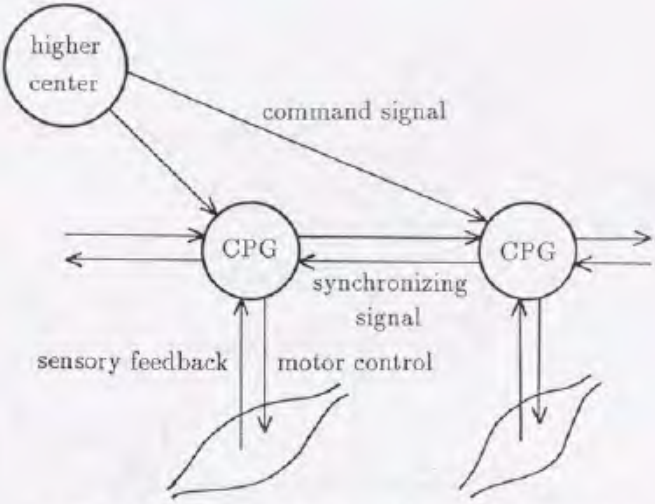


Figure 5.1: Interaction between CPGs and the physical environment.

much more important to utilize the properties of the physical environment than to defeat them.

In this section we construct a model of synchronization of neural dynamics to a physical system. As the simplest example of a locomotion system with oscillatory dynamics we consider a rolling robot as shown in Figure 5.2. This round robot rolls across a surface by changing its balance, that is, by shifting its weight relative to its center of rotation. In order to move the robot in a given direction at a constant speed the movement of its weight must be synchronized to the rotation angle θ of the body. If the reciprocating motion of its weight is too fast or slow compared to the current speed of rolling, then the rotation of the body cannot be entrained and the motion becomes unstable.

Consider a control network for this robot consisting of four input units $y_1, \ldots, y_4$, four hidden units $y_1, \ldots, y_4$, and one output unit y_9. The input y_1 controls the speed of oscillation of the network. Other three inputs are feedback signals from the rolling robot; $y_2(t) = \sin\theta(t)$, $y_3(t) = \cos\theta(t)$, and $y_4(t) = 0.2\omega(t)$. The output of the network $y_9(t)$ controls the displacement of its weight from the center of the robot; $u(t) = 0.8\,R\,y_9(t)$, where R is the radius of the robot.

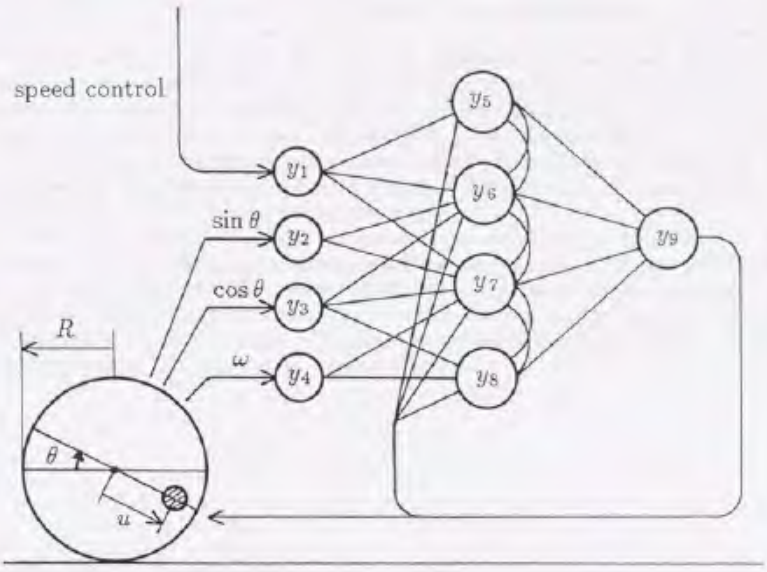


Figure 5.2: Control of a rolling robot by a CPG network.

The dynamics of the network is defined by the following equation,

$$y_i(t) = g_i(\, h_i\{\sum_{k\in N} w_{ik}y_k(t) + \sum_{l\in P} w_{il}y_l(t)\}), \quad (i = 5,\ldots,9), \tag{5.1}$$

where $P = \{2, 3, 4\}$ denotes the indices of the units that represent the state of the physical system, and N denotes the other indices.

In order to synchronize the phase of oscillation of the network to the physical state it is required that the sum of the physical input $\sum_{l\in P} w_{il}y_l(t)$ are synchronized to the sum of the networks inputs $\sum_{k\in N} w_{ik}y_k(t)$. Thus we define an error function of the i-th unit

$$f_i(t) = \frac{1}{2}\{\alpha \sum_{k\in N} w_{ik}y_k(t) - \sum_{l\in P} w_{il}y_l(t)\}^2, \tag{5.2}$$

and derive a gradient learning algorithm as follows,

$$\tau_p \frac{d}{dt}w_{ij} + w_{ij} = -s(t)\frac{\partial f_i}{\partial w_{ij}} = s(t)\{\alpha \sum_{k\in N} w_{ik}y_k - \sum_{l\in P} w_{il}y_l\}\, y_j(t), \quad (j \in P), \tag{5.3}$$

where $s(t)$ is a parameter which controls the speed of learning. If we set $s(t)$ positive only when the robot is rolling in desired direction then the phase relationship of the neural and physical systems at that time will be stabilized.

Firstly, the network was trained to oscillate with sinusoidal waveform of periods 4.0, 3.0, and 2.0 with the corresponding inputs $y_1 = -1$, 0, and $+1$. The connection weights from physical input units were kept zero. In this learning phase, each of the three waveforms was trained for 100 units of time. Figures 5.3(a),5.4(a) and 5.5(a) show the oscillation of the network and the robot after 10 such training phases. The periods of autonomous oscillation were 4.40, 3.05, and 2.75 at the input levels $y_1 = -1$, 0, and $+1$, respectively. Since the state of the robot is not yet fed back to the network, the oscillation of the network and the robot is not synchronized and the rolling motion of the robot was not entrained to the network output.

In the next phase, the feedback connections from the robot were established by the learning rule (5.3). The parameters of learning were $\alpha = 0.2$, $\tau_p = 10.0$, and $s(t) = \omega(t)$. Figures 5.3(b),5.4(b) and 5.5(b) show the coupled oscillation of the neural and physical systems after 8 sets of physical training. It is seen that the rolling motion of the robot is entrained to the neural output. The period of oscillation were 3.60, 2.90, and 2.65 for the input level $y_1 = -1$, 0, and $+1$, respectively.

5.2 Synchronization of multiple CPGs

It has been shown in the studies of multipedal locomotion that the movement of each leg of an animal is controlled by a distinct oscillator network and that the neural interconnection between those CPGs are used to synchronize the oscillations in different CPGs [30]. It has been shown both mathematically and computationally that two neural oscillator networks with mutually positive connections will oscillate in phase, and those with negative connections in opposite phase. The problem here is how to synchronize the oscillations of different CPGs with a specific phase required for efficient locomotion in a given physical environment.

We consider a two-legged robot as shown in Figure 5.6. The joint angles of the fore leg and the hind leg were controlled by different CPGs, each of which consists of three continuous-time neuron models. First CPG1 and CPG2 were trained to oscillate with periods 2.5 and 2.0 respectively. Without interconnection of the two CPGs, the relative phase of the CPGs, and accordingly the phase of the swing of the legs, changed gradually with time. The task of the learning is to fix the relative phase of two CPGs when the robot is walking effectively in a given direction.

We used the correlation learning scheme which we derived in the previous section. The connection weights between CPG1 to CPG2 were changed by the following equation,

$$\tau_p \frac{d}{dt} w_{ij} + w_{ij} = s(t)\{\alpha_1 \sum_{k \in N_1} w_{ik} y_k - \sum_{l \in N_2} w_{il} y_l\} y_j(t), \quad (i \in N_1, j \in N_2), \quad (5.4)$$

$$\tau_p \frac{d}{dt} w_{ij} + w_{ij} = s(t)\{\alpha_2 \sum_{k \in N_2} w_{ik} y_k - \sum_{l \in N_1} w_{il} y_l\} y_j(t), \quad (i \in N_2, j \in N_1), \quad (5.5)$$

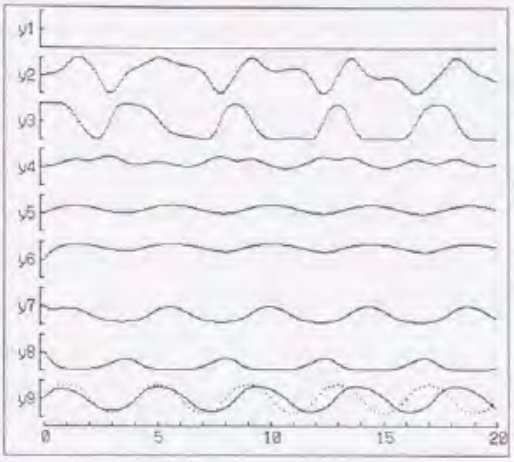
where N_1 and N_2 are the indices of the units in CPG1 and CPG2.

Figure 5.7 shows an example of the process of learned synchronization. The oscillation of two CPGs were initially independent, and began to entrain after 10 seconds. After 40 seconds of correlation learning, the two CPGs settled into stable synchronization.

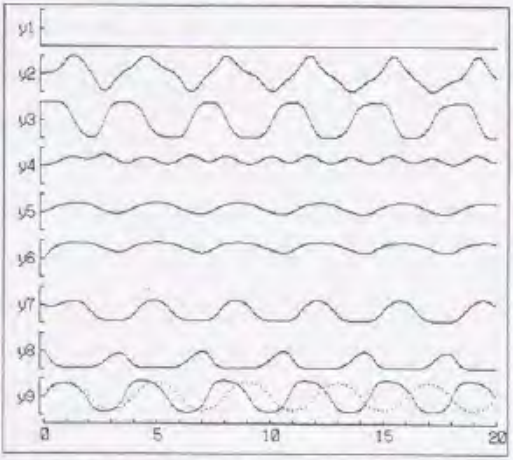
5.3 Discussion

It was shown that the learning scheme (5.3) gives a solution to the synchronization problem of neural and physical systems. The synaptic modification rule is based on the correlation between formerly established inputs and new incoming signals. This is consistent with empirical data of physiological experiments, such as long term potentiation of synapses in hippocampus [4].

However, this algorithm depends very much on an optimistic assumption that the system eventually finds a fairly good motion pattern. It is not assured that a motion pattern acquired by this learning scheme is an optimal one. Therefore we should seek a more reliable learning mechanism that satisfies the biological constraints.

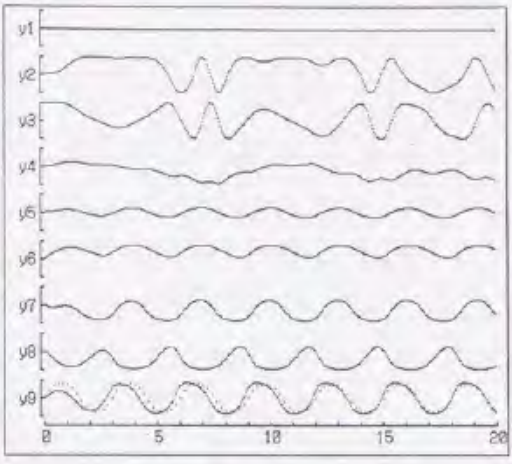


(a) Without sensory feedback inputs.

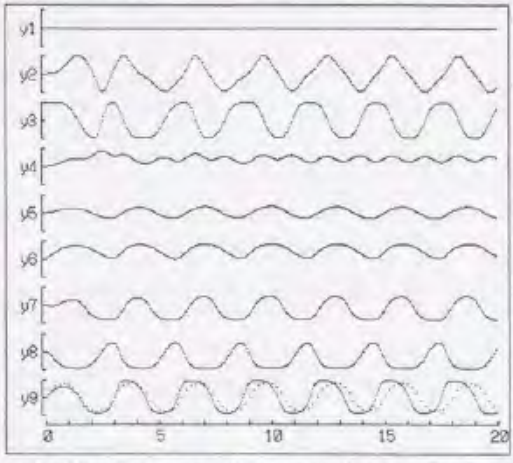


(b) After the learning of sensory feedback connections.

Figure 5.3: Coupled dynamics of an ANO and a rolling robot: $y_1 = -1$.

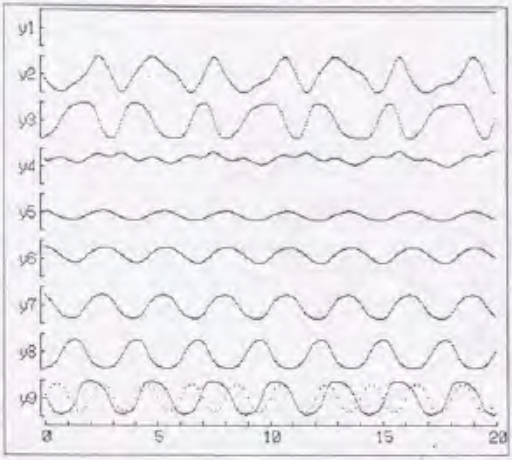


(a) Without sensory feedback inputs.

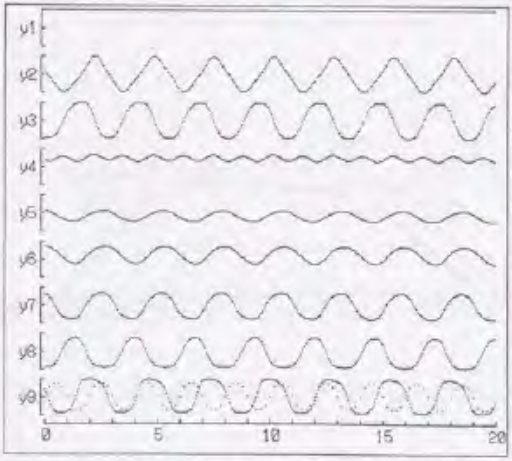


(b) After the learning of sensory feedback connections.

Figure 5.4: Coupled dynamics of an ANO and a rolling robot: $y_1 = 0$.



(a) Without sensory feedback inputs.



(b) After the learning of sensory feedback connections.

Figure 5.5: Coupled dynamics of an ANO and a rolling robot: $y_1 = +1$.

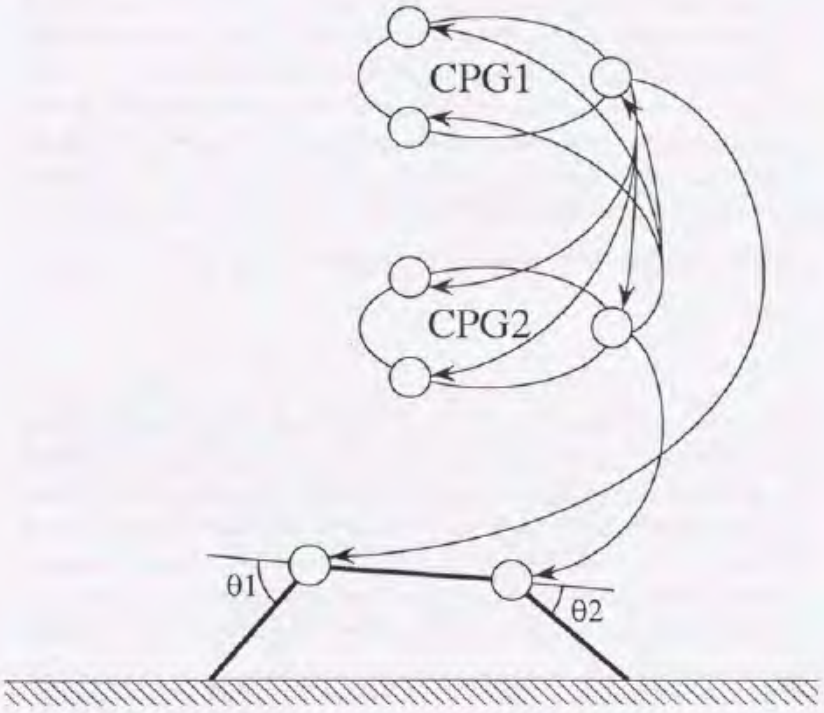


Figure 5.6: Control of a two-legged robot by two CPGs.

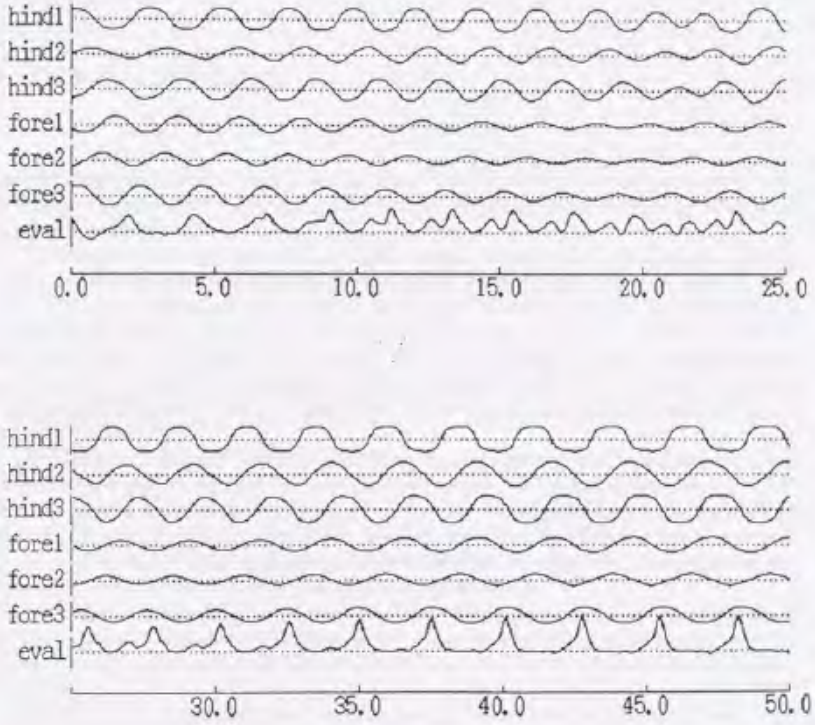


Figure 5.7: The relative phase of two CPGs was locked so that the robot walked stably in a given direction.

Chapter 6

Conclusion

In this dissertation supervised learning algorithms for recurrent continuous-time continuously running neural networks were derived and their abilities were examined by computer simulations.

In Chapter 2 a continuous-time neural network model called an Adaptive Neural Oscillator was proposed and the back-propagation learning algorithm was applied to it using the idea of derivatives of the mapping between functions of time. It was seen from computer simulations of a network with uniform decay time τ that an ANO network can learn to oscillate with a period in the range of τ to 20τ. Simple sinusoidal waveforms were easily learned. It was also possible to learn waveforms consisting of multiple frequency components. However, it is not clear what sort of waveforms can be memorized in the network by learning.

In Chapter 3 general learning algorithms for recurrent neural networks were formulated that can be applied to both continuous-time and discrete-time models. The discrete-time versions of these algorithms correspond to the simple recurrent network by Elman [14] and the real-time recurrent learning algorithm by Williams and Zipser [45]. From simulations of learning of oscillatory waveforms it was seen that the elaborate and simplified learning algorithm had comparable performance.

In Chapter 4 three schemes were proposed for memorizing multiple temporal patterns in one network. Oscillation of a network can either be controlled continuously by static input patterns, selected by the initial state of the network, or evoked from a part of a sequence.

In Chapter 5 a correlation learning scheme is applied for synchronization of neural and physical dynamical systems. Connection weight from external inputs were established so that the sum of external inputs simulates the inputs from within the network. This learning scheme was tested by two simulations with locomotory robots for constructing sensory feedback connections from physical systems and lateral coupling between multiple oscillator networks.

6.1 Applications of learning in recurrent networks

In the last few years many researchers have become aware of the potential of the recurrent neural networks. Elman [14] showed that some structures in temporal sequences are represented in the hidden layer of a recurrent network after learning to predict the strings generated by a simple grammar. Williams and Zipser [45] demonstrated that recurrent networks can be trained to be finite state automata. These works activated the application of recurrent networks to recognition and generation of grammatical sequences [5, 40].

Recurrent networks are being used to model the functions of biological neural systems. Tsung and Selverston [42] modeled the CPG of lobster by a recurrent discrete-time network and applied the learning algorithm proposed by Williams and Zipser. Recently, Rowat and Selverston [33] have applied a continuous-time back-propagation algorithm to a more biologically plausible models of CPG.

Application of supervised learning of recurrent network has not been limited to the models of motor control systems. Qian and Sejnowski applied back-propagation through-time to determine the lateral connections of a network model of binocular stereo vision. Zipser [47] trained a recurrent network to solve delayed matching tasks and gave an interpretation of temporal patterns of activities in neurons which are involved with working memory.

6.2 Explorations of recurrent networks

Supervised learning in recurrent neural networks has a large potential still to be studied. For example, in the modeling of biological neural systems and in applications to temporal information processing, such as, voice recognition, language understanding and generation, processing of music, and robotic control.

However, in formulating learning algorithms careful attentions must be given to decreasing the amount of computation and to increase the probability of successful learning by avoiding local minima and learning instabilities. Restriction of the topology of the network is a practical means to this end.

It is not as easy to elucidate the abilities of asymmetric recurrent neural networks as to investigate those of feed-forward or symmetric recurrent neural networks. However, the networks in our brains are neither feed-forward nor symmetric. In order to reveal the secrets of brains it is indispensable to determine the abilities of information processing by recurrent neural networks. The learning schemes derived in this study should be an effective tool in finding our way through the labyrinth of recurrent networks.

Acknowledgments

It was during my master course at University of Tokyo when I was first introduced to the world of nonlinear oscillations and neural networks. Prof. Shuji Yoshizawa has been my supervisor for seven years. I wish to thank him for his continual supervision and encouragement.

I am grateful to Prof. Jin-ichi Nagumo, Prof. Shun-ichi Amari, Prof. Ryoji Suzuki, Prof. Kaoru Nakano and Prof. Masatoshi Ishikawa for their kind supports and suggestions. I am thankful to Dr. Chiaki Nishimura and Dr. Kenjiro Maginu for teaching me much about the theory of differential equations and nonlinear dynamics.

I wish to give my best thank to Dr. Kevin Judd for brushing up my awkward English expressions. This thesis could not be completed without his help; he kindly corrected almost every line of my manuscript, both linguistic and mathematical mistakes. I am thankful to Mr. Hisashi Ito who spared much of his time on our computing environment, and Dr. Hiro-humi Yanai and Mr. Noboru Murata who were best colleagues to have discussions with.

Finally, I thank my wife Minako for her daily helps and cheers for me.

Bibliography

[1] Allen, G. I., & Tsukahara, N.: Cerebro cerebellar communication systems, *Physiological Reviews*, **54**, 957–1006 (1974).

[2] Amari, S.: Characteristics of random nets of analog neuron-like elements, *IEEE Transactions*, **SMC-2**, 643–657 (1972).

[3] Anderson, J. A.: A simple neural network generating interactive memory, *Mathematical Bioscience*, **14**, 197–220 (1972).

[4] Barrionuevo, G. & Brown, T. H.: Associative long-term potentiation of in hippocampal slices, *Proceedings of National Academy of Science USA*, **80**, 7347-7351 (1983).

[5] Cleeremans, A., Servan-Schreiber, D., & McClelland, J. L.: Finite state automata and simple recurrent networks, *Neural Computation*, **1**, 372–381 (1989).

[6] Doya, K. & Yoshizawa, S.: Memory of dynamical patterns in neural networks (in Japanese), *Proceedings of the 2nd Biological and Physiological Engineering Symposium*, 9–12, Toyohashi, Japan (1987).

[7] Doya, K. & Yoshizawa, S.: Motor pattern memory in neural networks (in Japanese), *Technical Report of the Institute of Electronics, Information and Communication Engineers*, **MBE87-141**, 293–300 (1988).

[8] Doya, K. & Yoshizawa, S.: Adaptive neural oscillator using continuous-time back-propagation learning, *Neural Networks*, **2**, 375–386 (1989).

[9] Doya, K. & Yoshizawa, S.: Memorizing oscillatory patterns in the analog neuron network, *Proceedings of IJCNN 1989 in Washington D.C.*, I-27–32 (1989).

[10] Doya, K. & Yoshizawa, S.: Memory of dynamical patterns in neural networks (in Japanese), *Proceedings of the 4th Biological and Physiological Engineering Symposium*, 47–50, Tokyo, Japan, (1989).

[11] Doya, K. & Yoshizawa, S.: Neural network model of temporal pattern memory (in Japanese), *Transactions of the Institute of Electronics, Information and Communication Engineers*, **J73-D-II**, 1150–1157 (1990).

[12] Doya, K. & Yoshizawa, S.: Memorizing hierarchical temporal patterns in analog neuron networks, *Proceedings of IJCNN 1990 in San Diego*, III-299–304 (1990).

[13] Doya, K.: Learning temporal patterns in recurrent neural networks, *Proceedings of 1990 IEEE SMC Conference*, 170–172 (1990).

[14] Elman, J. L.: Finding structure in time, *UCSD Center for Research in Language Technical Report*, 8801, (1988).

[15] Fujita, K., Doya, K., & Yoshizawa, S.: A neural network which learns motor patterns, Proceedings of the 1988 Academic Conference of Society of Instrumentation and Control Engineer, 359–360, Narashino, Japan (1988).

[16] Gherrity, M.: A learning algorithm for analog, fully recurrent neural networks, *Proceedings of IJCNN 1989 in Washington D.C.*, I-643–644 (1989).

[17] Gray, C. M., Koenig, P., Engel, A. K., & Singer, W.: Oscillatory responses in cat visual cortex exhibit inter-columnar synchronization which reflects global stimulus properties, *Nature*, **338**, 334–337 (1989).

[18] Cohen, A. H., Rossignol, S. R., & Grillner, S.: *Neural Control of Rhythmic Movements in Vertebrates*, John Wiley & Sons (1988).

[19] Hisada, M.: What is the pattern generators of motion? (in Japanese), in Ito, M. ed.: Central mechanisms of motion control, Ishiyaku Shuppan (1979).

[20] Hopfield, J. J.: Neural networks and physical systems with emergent collective computational abilities, *Proceedings of National Academy of Science of USA*, 79, 2554–2558 (1982).

[21] Hopfield, J. J.: Neurons with graded response have collective computational properties like those of two-state neurons, *Proceedings of National Academy of Science of USA*, **81**, 3088–3092 (1984).

[22] Ito, M., Sakurai, M. & Tongroach, P.: Climbing fibre induced depression of both mossy fibre responsiveness and glutamate sensitivity of cerebellar Purkinje cells, *Journal of Physiology*, **324**, 113–134 (1982).

[23] Jordan, M. I.: Supervised learning and systems with excess degrees of freedom, *University of Massachusetts COINS Technical Report*, **88-27** (1988).

[24] Kawato, M.: The feedback-error-learning neural network for supervised motor learning; In: Eckmiller, R. (ed): *Neural Network for Sensory and Motor Systems*, Elsevier, Amsterdam (1990).

[25] Kohonen, T.: Correlation matrix memories, *IEEE Transactions*, **C-21**, 353–359 (1972).

[26] Kristan Jr., W. B., Wittenberg, G., Nusbaum, M. P., & Stern-Tomlinson, W.: Multifunctional interneurons in the behavioral circuits of the medicinal leech, *Experientia*, **44**, 383–389 (1988).

[27] Luria, A. R.: *Higher cortical functions in man*, New York: Basic Books Inc. (1980).

[28] Nakano, K.: Associatron – a model of associative memory, *IEEE Transactions*, **SMC-2**, 380–388 (1972).

[29] Pearlmutter, B. A.: Learning state space trajectories in recurrent neural networks, *Proceedings of IJCNN 1989 in Washington D.C.*, II-365–372 (1989).

[30] Pearson, K.: The control of walking, *Scientific American*, **235**, 6, 72-86 (1976).

[31] Pineda, F. J.: Generalization of back-propagation to recurrent neural networks, *Physical Review Letters*, **59**, 2229–2232 (1987).

[32] Pontrjagin, L.: Ordinary differential equations, Addison-Wesley (1961).

[33] Rowat, P. F. & Selverston, A. I.: Learning algorithms for oscillatory networks with gap junctions and membrane currents, *Network*, **2**, 1 (1991).

[34] Rosenblatt, F.: *Principles of Neurodynamics*, Spartan (1961).

[35] Rumelhart, D. E., Hinton, G. E., & Williams, R. J.: Learning representations by back-propagating errors, *Nature*, **323**, 9, 533–536 (1986).

[36] Sato, M., Joe, M., & Hirahara, T.: APOLONN brings us to the real world – Learning nonlinear dynamics and fluctuations in nature, *Proceedings of IJCNN 1990 in San Diego*, I-581–587 (1990).

[37] Quian, N & Sejnowski, T.: Learning to solve random-dot stereograms of dense and transparent surfaces with recurrent backpropagation, *Connectionist Models Summer School*, Morgan Kaufmann (1989).

[38] Selverston, A. I. & Moulins, M.: Oscillatory neural networks, *Annual Review of Physiology*, **47**, 29–48 (1985).

[39] Shik, M. L. & Orlovsky, G. N.: Neurophysiology of locomotor automatism, *Physiological Reviews*, **56**, 465–501 (1976).

[40] Smith, A. W. & Zipser, D.: Encoding sequential structure: Experience with the real-time recurrent learning algorithm, *Proceedings of IJCNN 1989 in Washington D.C.*, I-645–648 (1989).

[41] Tanji, J. et al.: Relation of neurons in the nonprimary motor cortex to bilateral hand movement, *Nature*, **327**, 618–620 (1987).

[42] Tsung, F. S., Cottrell, G. W., & Selverston, A. I.: Some experiments on learning stable network oscillations, *Proceedings of IJCNN 1990 in San Diego*, I-169–174, (1989).

[43] Werbos, P. J.: Generalization of backpropagation with application to a recurrent gas market model, *Neural Networks*, **1**, 339–256 (1988).

[44] Widrow, B. & Stearns, S. D.: *Adaptive signal processing*, Englewood Cliffs, NJ: Prentice Hall (1985).

[45] Williams, R. J. & Zipser, D.: A learning algorithm for continually running fully recurrent neural networks, *Neural Computation*, **1**, 270–280 (1989).

[46] Williams, R. J. & Zipser, D.: Gradient based learning algorithms for recurrent connectionist networks, *Northeastern University, College of Computer Science Technical Report*, NU-CCS-90-9, (1990).

[47] Zipser, D: Short term active memory: A recurrent network model of the neural mechanism, *1990 IEEE NIPS Abstracts*, 2 (1990).

[48] Zipser, D: A subgrouping strategy that reduces complexity and speeds up learning in recurrent networks, *Neural Computation*, **1**, 552–558 (1989).


Kodak Color Control Patches
Blue
Cyan
Green
Yellow
Red
Magenta
White
3/Color
Black
Kodak Gray Scale
C
Y
M
A 1 2 3 4 5 6 M 8 9 10 11 12 13 14 15 B 17 18 19