Recurrent Networks: Learning Algorithms *

Kenji Doya
ATR Human Information Science Laboratories; CREST, JST
2-2-2 Hikaridai, Seika, Soraku, Kyoto 619-0288, Japan
Phone: +81-774-95-1251
Fax:+81-774-95-1259
E-mail: doya@atr.co.jp

February 21, 2002

RUNNING HEAD: Recurrent Networks

Correspondence:

Kenji Doya

ATR Human Information Science Laboratories

2-2-2 Hikaridai, Seika, Soraku, Kyoto 619-0288, Japan

Phone: +81-774-95-1251

Fax:+81-774-95-1259

E-mail: doya@atr.co.jp

*Michael Arbib ed. *Handbook of Brain Theory and Neural Networks, 2nd edition*, MIT Press.

INTRODUCTION

The backpropagation algorithm for feed-forward networks (Figure 1a) has been successfully applied to a wide range of problems from neuroscience to consumer electronics (see BACK-PROPAGATION and APPLICATIONS OF NEURAL NETWORKS). However, what can be implemented by a feed-forward network is just a static mapping of the input vectors. It is needless to say that our brain is not a stateless input-output system but a high dimensional nonlinear dynamical system. In order to model dynamical functions of the brain, or to design a machine that performs as well as a brain does, it is essential to utilize a system that is capable of storing internal states and implementing complex dynamics.

This is why learning algorithms for *recurrent neural networks* (Figure 1b), which have feedback connections and time delays, have been studied with enthusiasm. In a recurrent network, the state of the system can be encoded in the activity pattern of the units and a wide variety of dynamical behaviors can be programmed by the connection weights.

A popular sub-class of recurrent networks is those with symmetric connection weights. In this case, the network dynamics is guaranteed to converge to a minimum of "energy" function (see ENERGY FUNCTIONS FOR NEURAL NETWORKS and COMPUTING WITH ATTRACTORS). Typical examples are associative memory networks (see ASSOCIATIVE NETWORKS), optimization networks (see NEURAL OPTIMIZATION), and winner-take-all networks (see WINNER-TAKE-ALL MECHANISMS).

However, steady state solutions are only a limited portion of the capabilities of recurrent networks. A recurrent network can serve as a sequence recognition system (see LANGUAGE PROCESSING) or as a sequential pattern generator (see MOTOR PATTERN GENERATION and SEQUENCE LEARNING). More generally, it is capable of transforming an input sequence into some other output sequence (see TEMPORAL PATTERN PROCESSING). It can be uses as as a non-linear filter (see ADAPTIVE SIGNAL PROCESSING), a non-linear controller (see ADAPTIVE CONTROL: NEURAL NETWORK APPLICATIONS), or a finite state machine (see LANGUAGE PROCESSING and Giles et al. 1992).

This article reviews the learning algorithms for training recurrent networks. There are three major frameworks of learning: *supervised learning* based on the output error signal, *reinforcement learning* based on the scalar reward signal, and *unsupervised learning* based on the statistical feature of the input signal. Our main focus will be on supervised learning algorithms for recurrent networks. We will also give a brief overview of reinforcement and unsupervised learning algorithm.

SUPERVISED LEARNING ALGORITHMS

The problem set-up of supervised learning in recurrent networks is similar to the case of feed-forward networks; a network is given a desired output for an input. An error function is defined and its gradient with respect to the weights are derived. However, the major difference is that the input and output are not static vectors but *time sequences.*

For example, in a recurrent network shown in Figure 1b, units 1 and 2 are output units, 3, 4, and 5 are hidden units, and 6 and 7 are input units. A small change of a connection weight, say w_{43} (shown as black dot) affects the output units, say unit 1, not only through the direct connection from unit 4 to 1 (thick black line), but also through indirect connections, through units 2, 3, and 5 (thick gray lines), and through infinitely many multi-step paths with multiple delays. It makes exact calculation of output error gradient rather complex.

One simple strategy is to neglect all the indirect paths. In this case, although the network state evolves according to the recurrent network as in Figure 1b, a simple back-propagation algorithm is applied by regarding it as a feed-forward network as in Figure 1a. Such coarse approximation methods turned out to be effective in a series of work on language acquisition using *simple recurrent networks* that has recurrent connection between hidden units (see LANGUAGE PROCESSING).

There are two basic ways to calculate the exact gradient of the output with respect to the weights: the forward methods and backward methods. The forward method estimates the effects of small change in a weight on the network state trajectory in a form of linear

dynamic equation system. This can be calculated concurrently with the network dynamic, thus useful for on-line learning. One drawback is the amount of computation for updating a set of dynamic equations for each weights.

The backward method estimates the causes for the output error backward in time. In discrete time case, it is realized by 'unrolling' the multi-step evolution of the network state as a multi-layer feed-forward network, as in Figure 1c, and applying the standard back-propagation algorithm (Rumelhart et al., 1986). In continuous-time case, it is done by running a set of 'adjoint system' backward in time. Although it requires asynchronous operation, the evolution and storage of the state trajectory first and error gradient calculation afterwards, the amount of computation is much less than in the forward methods (see Pearlmutter, 1995 for a comprehensive review). In the following sections, we will formulate these algorithms for both discrete-time and continuous-time models and then discuss technical problems in using them.

DISCRETE-TIME MODEL

First, we start with a discrete-time recurrent network with n units and m inputs. We denote the state of the i-th unit by y_i and the connection weights from j-th to i-th units by w_{ij}. Both external inputs u_j and recurrent inputs y_j are represented as z_j for convenience.

$$y_i(t+1) = f\left(\sum_{j=1}^{n+m} w_{ij} z_j(t)\right) \qquad (i = 1, ..., n). \tag{1}$$

$$z_j(t) = \begin{cases} y_j(t) & j \le n \\ u_{j-n} & j > n \end{cases},$$

The output nonlinearity $f(\,)$ is usually a squashing function such as $f(x) = 1/(1+e^{-x})$ and $f(x) = \tanh x$, whose derivatives are conveniently given by $f'(x) = f(x)(1 - f(x))$ and $f'(x) = 1 - f(x)^2$, respectively. We can introduce a bias parameter by assuming that one of the inputs u_j is constant.

The goal of learning is to set the parameters w_{ij} so that the output trajectory $(y_1(t), ..., y_n(t))$ follows a desired trajectory $(d_1(t), ..., d_n(t))$ $(t = 1, ..., T)$ with a given initial state $(y_1(0), ..., y_n(0))$

and an input sequence $(u_1(t), ..., u_m(t))$ $(t = 0, ..., T-1)$. We define the error function

$$E = \sum_{t=1}^{T} \sum_{i=1}^{n} \mu_i(t) \frac{1}{2} (y_i(t) - d_i(t))^2 \tag{2}$$

and perform gradient descent on E with respect to the weights w_{ij}. The masking function $\mu_i(t)$ specifies which components of the trajectory are to be supervised at what time. In a typical case, $\mu_i(t) \equiv 1$ for output units and $\mu_i(t) \equiv 0$ for hidden units. When only the end point of the trajectory is specified, $\mu_i(T) = 1$ and $\mu_i(t) = 0$ for $t < T$.

Real-Time Recurrent Learning

The effect of weight change on the network dynamics can be seen by simply differentiating the network dynamics equation (1) by a weight w_{kl} (Williams and Zipser, 1989).

$$\frac{\partial y_i(t+1)}{\partial w_{kl}} = f'(x_i(t)) \left[\sum_{j=1}^{n} w_{ij} \frac{\partial y_j(t)}{\partial w_{kl}} + \delta_{ik} z_l(t) \right], \qquad (i = 1, ..., n), \tag{3}$$

where $x_i(t) = \sum_{j=1}^{n+m} w_{ij} z_j(t)$ is the net input to the unit and δ_{ik} is Kronecker's delta ($\delta_{ik} = 1$ if $i = k$ and otherwise 0). The term $\delta_{ik} z_l(t)$ represents an *explicit* effect of the weight w_{kl} onto the unit k and the term $\sum_{j=1}^{n} w_{ij} \frac{\partial y_j(t)}{\partial w_{kl}}$ represents an *implicit* effect onto all the units due to network dynamics.

The equation (3) for each unit $i = 1, ..., n$ constitutes an n-dimensional linear dynamical system (with time-varying coefficients), where $(\frac{\partial y_1}{\partial w_{kl}}, ..., \frac{\partial y_n}{\partial w_{kl}})$ is taken as a dynamical variable. Since the initial state $y_i(0)$ of the network is independent of the connection weights, the appropriate initial condition for (3) is

$$\frac{\partial y_i(0)}{\partial w_{kl}} = 0 \qquad (i = 1, ..., n).$$

Thus we can compute $\frac{\partial y_i(t)}{\partial w_{kl}}$ *forward in time* by iterating (3) simultaneously with the network dynamics (1). From this solution, we can calculate the error gradient as follows.

$$\frac{\partial E}{\partial w_{kl}} = \sum_{t=1}^{T} \sum_{i=1}^{n} \mu_i(t) (y_i(t) - d_i(t)) \frac{\partial y_i(t)}{\partial w_{kl}}. \tag{4}$$

A standard *batch* gradient descent algorithm is to accumulate the error gradient by (4) and update each weight w_{kl} by

$$w_{kl} := w_{kl} - \varepsilon \frac{\partial E}{\partial w_{kl}}, \tag{5}$$

where $\varepsilon > 0$ is a learning rate parameter.

An alternative update scheme is the gradient descent of *current* output error $\sum_{i=1} \frac{1}{2}\mu_i(t)(y_i(t) - d_i(t))^2$ at each time step, namely

$$w_{kl}(t+1) = w_{kl}(t) - \varepsilon \sum_{i=1}^{n} \mu_i(t)(y_i(t) - d_i(t)) \frac{\partial y_i(t)}{\partial w_{kl}}. \tag{6}$$

Note that we assumed that w_{kl} is a constant, not a dynamical variable, in deriving (3), so we have to keep the learning rate ε small enough. However, this *on-line* update scheme was shown to be effective in a number of temporal learning tasks (Williams and Zipser, 1989) and often called "Real-Time Recurrent Learning".

A drawback of this error gradient calculation *forward in time* is that we have to solve an n-dimensional system (3) for *each* of the weights w_{kl} $(i = 1, ..., n; t = 1, ...T)$. It requires $O(n^3)$ memories and $O(n^4)$ computations.

Back-Propagation Through Time

Another learning algorithm for discrete-time model can be derived by "unfolding" a recurrent network into a multi-layer network (Rumelhart et al., 1986). In this scheme, T-step iteration of a recurrent network is regarded as one sweep of operation in a T-layered feed-forward network with identical connection weights w_{ij} between successive layers. The error gradient can be derived in a same way as in the standard back-propagation, except that the output errors are not only given in the last layer but added in each layer.

$$\frac{\partial E}{\partial y_i(t)} = \sum_{j=1}^{n} \frac{\partial E}{\partial y_j(t+1)} f'(x_j(t)) w_{ji} + \mu_i(t)(y_i(t) - d_i(t)) \qquad (i = 1, ..., n). \tag{7}$$

Since the error E is independent of the state at $t > T$, the boundary condition for (7) is given at the final time step as

$$\frac{\partial E}{\partial y_i(T+1)} = 0, \qquad (i = 1, ..., n).$$

Thus, the learning equation (7) can be iterated *backward in time* from $t = T$ to 1.

From the solution $\frac{\partial E}{\partial y_i}$, the error gradients are given by

$$\frac{\partial E}{\partial w_{ij}} = \sum_{t=1}^{T} \frac{\partial E}{\partial y_i(t)} f'(x_i(t-1)) z_j(t-1). \tag{8}$$

and the weights are updated in a *batch* using (5).

The advantage of this algorithm is that we have to solve only *one* n-dimensional system (7) for adjusting all the weights. Therefore only $O(n^2)$ computations are required. However, since the learning equation (7) has to be solved *backward* in time, we cannot update the weights *on-line* and have to store the history of the network state $y_i(t)$ $(i = 1, ..., n; t = 1, ...T)$, which requires $O(nT)$ memories.

CONTINUOUS-TIME MODEL

A continuous-time model is a natural choice for modeling systems that are governed by differential equations. Time constants of continuous-time models are convenient parameter for setting local memory spans for individual units. They can also be adjusted by learning as mentioned below.

Slightly different versions of continuous-time models have been studied. Here, we focus on the following model (Pearlmutter, 1989),

$$\tau_i \dot{y}_i(t) = -y_i(t) + f\left(\sum_{j=1}^{n+m} w_{ij} z_j(t)\right), \qquad (i = 1, ..., n), \tag{9}$$

$$z_j(t) = \begin{cases} y_j(t) & j \leq n \\ u_{j-n} & j > n \end{cases} .$$

However, similar derivations apply to other models as well (Doya and Yoshizawa, 1989).

We define an error integral

$$E = \int_0^T \sum_{i=1}^{n} \mu_i(t) \frac{1}{2} (y_i(t) - d_i(t))^2 dt \tag{10}$$

and derive a gradient descent algorithm for minimizing E for a desired trajectory $(d_1(t), ..., d_n(t))$ $(0 \leq t \leq T)$ with a given initial state $(y_1(0), ..., y_n(0))$ and an input sequence $(u_1(t), ..., u_m(t))$.

Variation Method

The effect of a change in a weight w_{kl} on the state $y_i(t)$ can be estimated by differentiating the network dynamics equation (9) as follows.

$$\tau_i \frac{d}{dt}\left(\frac{\partial y_i}{\partial w_{kl}}\right) = -\frac{\partial y_i}{\partial w_{kl}} + f'(x_i(t))\left[\sum_{j=1}^{n} w_{ij}\frac{\partial y_j}{\partial w_{kl}} + \delta_{ik} z_l(t)\right], \qquad (i = 1, ..., n). \tag{11}$$

This forms an n-dimensional linear differential equation system with the state variable $(\frac{\partial y_1}{\partial w_{kl}}, ..., \frac{\partial y_n}{\partial w_{kl}})$ and is called a *variation system* of the network dynamics (9). The initial condition for this system is given by

$$\frac{\partial y_i(0)}{\partial w_{kl}} = 0 \qquad (i = 1, ..., n),$$

because the initial state of the network is independent of the weights. We can numerically integrate (11) *forward in time* concurrently with the network dynamics (9).

From the solution $\frac{\partial y_i(t)}{\partial w_{kl}}$ $(0 \leq t \leq T)$, the error gradient is given by

$$\frac{\partial E}{\partial w_{ij}} = \int_0^T \sum_{i=1}^{n} \mu_i(t)(y_i(t) - d_i(t))\frac{\partial y_i(t)}{\partial w_{kl}} dt. \tag{12}$$

We can use either the *batch* update scheme (5) at the end of a sequence, or the *on-line* update scheme

$$\dot{w}_{kl} = -\varepsilon \sum_{i=1}^{n} \mu_i(t)(y_i(t) - d_i(t))\frac{\partial y_i(t)}{\partial w_{kl}} \tag{13}$$

with sufficiently small learning rate $\varepsilon > 0$.

The error gradient for a time constant τ_k is given by the following variation equation.

$$\tau_i \frac{d}{dt}\left(\frac{\partial y_i}{\partial \tau_k}\right) = -\frac{\partial y_i}{\partial \tau_k} + f'(x_i(t))\left[\sum_{j=1}^{n} w_{ij}\frac{\partial y_j}{\partial \tau_k} - \delta_{ik}\dot{y}_k(t)\right], \qquad (i = 1, ..., n). \tag{14}$$

Adjoint Method

The *backward* algorithm for a continuous-time model can be derived in several ways, for example, by finite difference approximation (Pearlmutter, 1989). Here we derive the algorithm as an "adjoint" system of the *forward* learning equation (11).

A pair of n-dimensional linear systems

$$\dot{\boldsymbol{p}} = A(t)\boldsymbol{p} + \boldsymbol{b}(t) \qquad \text{and} \qquad \dot{\boldsymbol{q}} = -A^*(t)\boldsymbol{q} - \boldsymbol{c}(t)$$

are called "adjoint" to each other, where A^* denotes the transpose of matrix A. A useful property of adjoint systems is that their solutions satisfy the following "Green's equality."

$$\int_0^T \boldsymbol{q}(t) \cdot \boldsymbol{b}(t) dt - \int_0^T \boldsymbol{c}(t) \cdot \boldsymbol{p}(t) dt = \boldsymbol{q}(T) \cdot \boldsymbol{p}(T) - \boldsymbol{q}(0) \cdot \boldsymbol{p}(0).$$

We can actually compose an adjoint system of the variation equation (11)

$$\dot{q}_i = \frac{q_i(t)}{\tau_i} - \sum_{j=1}^{n} \frac{f'(x_j(t))}{\tau_j} w_{ji} q_j(t) - \mu_i(t)(y_i(t) - d_i(t)), \tag{15}$$

where we put $p_i = \frac{\partial y_i}{\partial w_{kl}}$, $A_{ij}(t) = \frac{f'(x_i(t))}{\tau_i} w_{ij} - \frac{\delta_{ij}}{\tau_i}$, $b_i(t) = \frac{f'(x_i(t))}{\tau_i} \delta_{ik} y_l(t)$, and $c_i(t) = \mu_i(t)(y_i(t) - d_i(t))$. With the boundary conditions $p_i(0) = \frac{\partial y_i(0)}{\partial w_{kl}} = 0$ and $q_i(T) = 0$, the Green's equality becomes

$$\int_0^T \sum_{i=1}^{n} q_i(t) \frac{f'(x_i(t))}{\tau_i} \delta_{ik} y_l(t) dt = \int_0^T \sum_{i=1}^{n} \mu_i(t)(y_i(t) - d_i(t)) \frac{\partial y_i}{\partial w_{kl}} dt. \tag{16}$$

Note that the right hand side is identical to the error gradient (12). Thus, we have an alternative form of the error gradient

$$\frac{\partial E}{\partial w_{kl}} = \int_0^T q_k(t) \frac{f'(x_k(t))}{\tau_k} z_l(t) dt. \tag{17}$$

Similarly, the error gradient for a time constant is given by

$$\frac{\partial E}{\partial \tau_k} = \int_0^T q_k(t) \frac{f'(x_k(t))}{\tau_k} (-\dot{y}_k(t)) dt. \tag{18}$$

As in the discrete-time case, we first run the network dynamics (9) *forward in time* and then run the adjoint system (15) *backward in time* with the terminal condition $q_i(T) = 0$. The weights are updated in batch by (5).

TECHNICAL REMARKS

Forward or Backward

The forward algorithms require $O(n^4)$ computations. Therefore it is not suitable for a fully connected network with tens or hundreds of units. However, for a small sized network or a network with only local connections, on-line weight update can be an advantage.

In order to allow on-line weight update with the efficiency of the backward algorithm, a truncated version of back-propagation through time algorithm has been proposed (Schmidhuber, 1992)

Teacher Forcing

So called "teacher forcing" technique has been shown to be helpful, especially in training a network into an autonomous dynamical system (Doya and Yoshizawa, 1989, Williams and Zipser, 1989). In this scheme, the desired output $d_i(t)$ is used to drive the network dynamics in place of the feedback of its actual output $y_i(t)$.

The reasons for the need of teacher forcing are:

- The state of the network is assigned to desired one of the many attractor domains.
- In learning oscillatory patterns, unless the phase of the network output is synchronized to the teacher signal, there will be an apparently large error (Doya and Yoshizawa, 1989).
- It will avoid a local minimum solution of static output at the mean value of the dynamic teacher signal (Williams and Zipser, 1989).
- The linearized equation for an limit cycle trajectory is not asymptotically stable if the system is running autonomously (Doya, 1992).

One problem with this technique is that the trajectory learned with teacher forcing may not be stable when the network is run autonomously after learning. Several heuristics have

been proposed for enhancing the stability of the non-forced trajectory.

Noisy forcing: add some noise to the forcing input.

Partial forcing: use a mixed input $z_i(t) = y_i(t) + \alpha(d_i(t) - y_i(t))$ with $0 < \alpha < 1$ and decrease the forcing rate α with the progress of learning.

Part-time forcing: turn on forcing to synchronize the network to the teacher and then turn off forcing to train the autonomous trajectory.

Bifurcation Boundaries

In many learning tasks, the goal is not only to replicate particular sample trajectories but to reconstruct some "attractors" in the state space, such as fixed points, limit cycles, and chaotic attractors.

For example, when a network is trained as a finite state machine, it must have distinct attractors in order to represent discrete states. For another example, when a network is trained as a periodic oscillator, it must have a limit cycle attractor. When we gradually change network parameters, we expect that the shape and location of attractors change continuously. However, that is not always true. At some points in the parameter space, attractors can emerge, disappear, or change their stability. Such a phenomenon is known as "bifurcation" in nonlinear systems theory (see DYNAMICS AND BIFURCATION IN NEURAL NETS and CANONICAL NEURAL MODELS).

With some kinds of bifurcation, e.g. saddle-node bifurcation, the state of the network changes drastically. Even if the equilibrium or the trajectory persists, the linearized equations that are used for gradient computation can lose asymptotic stability. Accordingly, when the network goes through a bifurcation point, the solution of the learning equation can grow rapidly and gradient descent algorithm can be unstable (Doya, 1992).

Although this might sound a rare, pathetic situation, bifurcation is actually an inevitable step in many learning tasks (Doya, 1992). If the connection weights w_{ij} are initialized with

small random values, the network dynamics has a single global attractor point. In order to have multiple attractor domains or a limit cycle, the network must go through some bifurcation boundary. Conversely, until the network goes through an appropriate bifurcation, even a simple memory task can be very difficult due to exponential decay of the error gradient.

Incremental Training

It has been reported that gradual increase of the complexity of training examples is critical for successfully training a network as a finite state machine (see LANGUAGE ACQUISITION). A possible reason for this is that a network can acquire memory mechanisms only gradually, by going through bifurcation boundaries. If we impose examples that require many internal states with long time delay from the beginning, we might simply screw up the network. This problem of "developmental" capability of recurrent networks needs further examinations.

DISCUSSION

A fully-connected recurrent neural network can potentially be a very powerful system for temporal information processing. Based on the universal approximation theorem for three layered networks (UNIVERSAL APPROXIMATORS and KOLMOGOROV'S THEOREM), it has been shown that a recurrent network can, with enough units, approximate any dynamical system (Funahashi and Nakamura, 1993). It has also been shown that a recurrent neural network, with its analog valued computation, can have super-Turing computational power (see NEURAL AUTOMATA AND COMPUTATIONAL COMPLEXITY). However, these theories do not guarantee that such a network can be readily achieved by error gradient descent learning.

As mentioned above, the error gradient can decay or expand exponentially in time, which makes gradient descent more difficult than in the case of feed-forward networks. Convergence of learning depends critically on the choice of network topology, initial weights, and the choice of training samples. These are part of the reasons why networks with specialized

architectures have been crafted for specific problems, for example, networks with tapped delay-lines or local recurrent loops (see ADAPTIVE SIGNAL PROCESSING; ADAPTIVE CONTROL: NEURAL NETWORK APPLICATIONS).

Nevertheless, in the studies of grammar learning (see LANGUAGE PROCESSING), successful cases were reported in which recurrent neural networks could learn context-free and context-sensitive languages (Rodriguez, 2001). In these examples, fractal structures in the network state space were utilized to approximate multiple "counters," which are necessary for processing complex grammatical structures like palindromes. Interesting findings in such studies were that recurrent networks can generalize in terms of the length of the strings.

Bayesian approaches are recently applied to learning of dynamics in recurrent networks (see BAYESIAN METHODS AND NEURAL NETWORKS, BAYESIAN NETWORKS, and GRAPHICAL MODELS). It has been pointed out that a recurrent network can be trained by the method of extended Kalman filtering, which has a similar properties with the RTRL algorithm with teacher forcing (Williams, 1992). EM methods for estimating the states of the hidden units and the weight parameters have been formulated (Ghahramani and Hinton, 2000). This seems to be a theoretically more sound way of nonlinear dynamical system estimation. However, since EM is essentially a local optimization process, whether this new wave of modeling methods can escape from the issue of bifurcation remains to be seen. Many recent approaches to temporal sequence processing are reviewed in (Sun, 2001), and other articles in the same book.

Biologically Inspired Learning Methods

It has been suggested that the network architectures of the cerebellum, the basal ganglia, and the cerebral cortex are specialized for different frameworks of learning, namely, the cerebellum for supervised learning, the basal ganglia for reinforcement learning, and cerebral cortex for unsupervised learning (Doya, 1999). The circuits of the cerebellum (Figure 2a) and the basal ganglia (Figure 2b) have roughly feed-forward structures. While the learning

in the cerebellum is characterized by the specific error signals carried by the climbing fibers to the Purkinje cells, the learning of the basal ganglia is characterized by the reward signal broadcasted by the dopaminergic input to the striatum (see CEREBELLUM AND MOTOR CONTRO and BASAL GANGLIA). They both form long recurrent loops starting from and ending in the cerebral cortex. The circuit of the cerebral cortex is characterized by the massive recurrent connections, within and between functional columns, and between cortical areas (Figure 2c). Learning in the cerebral cortex is characterized by Hebbian learning (see CORTICAL HEBBIAN MODULES). Since the cerebral cortex embodies the most successful application of recurrent networks, both within the cortex and in the cortico-cerebellar and cortico-basal ganglia recurrent loops, it is natural to try to draw insights from the cortical network architecture.

The combination of recurrent excitation and lateral inhibition can implement a winner-take-all mechanism (see WINNER-TAKE-ALL NETWORKS). In combination with Hebbian plasticity and certain regulatory mechanisms, self-organization of feature defectors can be achieved (see SELF-ORGANIZATION AND THE BRAIN, COMPETITIVE LEARNING, NEURONAL REGULATION AND HEBBIAN LEARNING). This basic framework is shared by recent studies of receptive field formation and independent component analysis, which combine bottom-up Hebbian plasticity with lateral or top-down anti-Hebbian plasticity (see PATTERN FORMATION AND NEURAL MAPS, INDEPENDENT COMPONENT ANALYSIS, and UNSUPERVISED LEARNING WITH GLOBAL OBJECTIVE FUNCTIONS).

The BOLTZMANN MACHINES (q.v.), with its wake and sleep modes, is another basic model of cortical processing. Its extension to layered recurrent networks, the Helmholtz machine, is capable of extracting the hidden structure of the sensory data and reproducing the data by top-down processing (see HELMHOLTZ MACHINES AND SLEEP-WAKE LEARNING).

One of the main open issues in REINFORCEMENT LEARNING (q.v.) is how to learn

a good behavior when the environmental states are not perfectly observable (see IDENTIFICATION AND CONTROL). In such a case, the agent should store a certain 'belief state' and update it according to the model of the environment. Actions are chosen according to the predicted future reward based on the belief state. Such complex operations could be implemented in the cortico-cerebellar and cortico-basal ganglia loops, with the cerebral cortex representing the belief state, the cerebellum implementing the internal model of the environment, and the basal ganglia prediction the future reward. Better understanding of the cortico-cerebellar-basal ganglia system may give some clue for designing an adaptive agent under uncertainty.

REFERENCES

Doya, K., 1992, Bifurcations in the learning of recurrent neural networks, Proceedings of 1992 IEEE International Symposium on Circuits and Systems, New York:IEEE, vol. 6, pp. 2777–2780.

Doya, K. and Yoshizawa, S., 1989, Adaptive neural oscillator using continuous-time back-propagation learning, Neural Networks, 2:375–386.

Doya, K., 1999, What are the computations in the cerebellum, the basal ganglia, and the cerebral cortex, Neural Networks, 12, 961–974.

Elman, J. L., 1990, Finding structure in time, Cognitive Science, 14:179–211.

Funahashi, K., and Nakamura, Y., 1993, Approximation of dynamical systems by continuous time recurrent neural networks, Neural Networks, 6:801-806.

Ghahramani, Z. and Hinton, G. E., 2000, Variational learning for switching state-space models, Neural Computation, 12:831–864.

Pearlmutter, B. A., 1989, Learning state space trajectories in recurrent neural networks,

Neural Computation, 1:263–269.

*Pearlmutter, B. A., 1995, Gradient calculations for dynamic recurrent neural networks: A survey, IEEE Transactions on Neural Networks, 6:1212–1228.

Rodriguez, P., 2001, Simple recurrent networks learn context-free and context-sensitive languages by counting, Neural Computation, 13:2093–2118.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J., 1986, Learning representations by back-propagating errors, Nature, 323:533–536.

Schmidhuber, J., 1992, A fixed size storage $O(n^3)$ time complexity learning algorithm for fully recurrent continually running networks, Neural Computation, 4:243–248.

*Sun, R., 2001, Introduction to sequence learning, in Sequence Learning: Paradigms, Algorithms, and Applications, (R. Sun and C. L. Giles, Eds.), New York:Springer-Verlag, pp. 1–10

Williams R. J. and Zipser D., 1989, A learning algorithm for continually running fully recurrent neural networks, Neural Computation, 1:270–280.

Williams, R. J., 1992, Training recurrent networks using the extended Kalman filter, Proceedings of International Joint Conference on Neural Networks, Baltimore, MD, pp. 241-250.

FIGURE CAPTIONS

Figure 1

Examples of a feed-forward network (a) and a recurrent network (b), where units 1 and 2 are output units, 3, 4, and 5 are hidden units, and 6 and 7 are input units. The multi-step operation of a recurrent network (b) can be unrolled as a multi-layer feed-forward network (c).

Figure 2

Network architectures of the cerebellum (a), the basal ganglia (b) and the cerebral cortex (c). See BASAL GANGLIA (q.v.) for abbreviations.

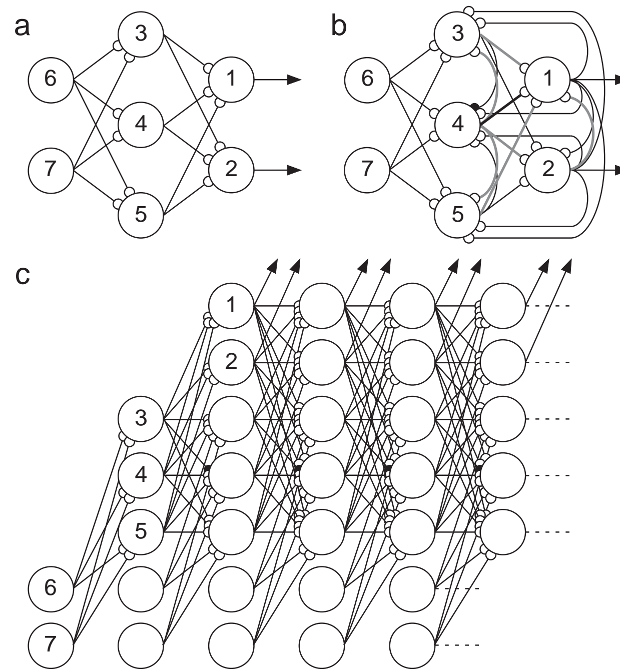

Figure 1

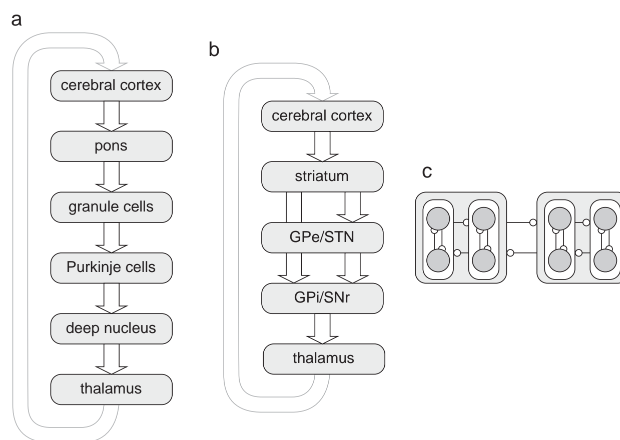

Figure 2