DESCRIPTION AND TRAINING OF NEURAL NETWORK DYNAMICS

Richard Rohwer
Centre for Speech Technology Research
Edinburgh University
80, South Bridge
Edinburgh EH1 1HN
SCOTLAND

rr@uk.ac.ed.cstr

Abstract

Attractor properties of a popular discrete-time neural network model are illustrated through numerical simulations. The most complex dynamics is found to occur within particular ranges of parameters controlling the symmetry and magnitude of the weight matrix. A small network model is observed to produce fixed points, limit cycles, mode-locking, the Ruelle-Takens route to chaos, and the period-doubling route to chaos.

Training algorithms for tuning this dynamical behaviour are discussed. Training can be an easy or difficult task, depending whether the problem requires the use of temporal information distributed over long time intervals. Such problems require training algorithms which can handle hidden nodes. The most prominent of these algorithms, back propagation through time, solves the temporal credit assignment problem in a way which can work only if the relevant information is distributed locally in time. The Moving Targets algorithm works for the more general case, but is computationally intensive, and prone to local minima.

1 Introduction

It would be difficult to dispute that thought processes involve time in an essential manner. Therefore neural network models must be understood as dynamical systems if they are to serve as plausible models or practical devices for the physical underpinnings of mental processes in general.

This paper treats some dynamical aspects of the commonly-used discrete-time model

$$y_{it} = f(\sum_j w_{ij} y_{j,t-1}) \tag{1}$$

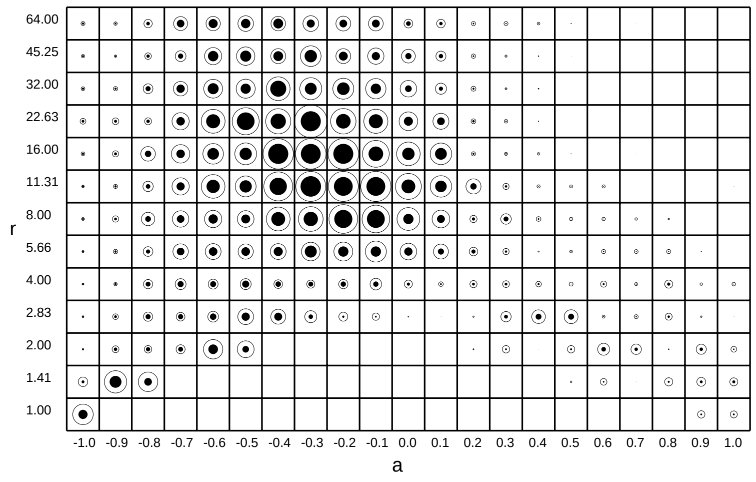
where y_{it} is a real-valued *output* of node i at time t, w_{ij} is a real-valued *weight* from node j to node i, and f is the logistic function

$$f(x) = 1/(1+e^{-x}). \tag{2}$$

We begin by reviewing some results of Renals and Rohwer [18] on some properties of the attractors of these systems. At the very least, these results show that the weight matrix of this system parameterizes a rich variety of temporal patterns. This encourages us to speculate that some of these patterns encode useful calculations. If so, then it is an engineering problem to find training algorithms which match a suitable weight matrix to a given problem.

Actually there is no doubt that some of the temporal patterns generated by the systems (1) represent useful calculations, because it is straightforward to engineer a weight matrix which produces a neural network model functionally equivalent to a conventional computer. One has only to sketch a simple neural circuit for a NAND gate and one for a Flip-Flop to be convinced of this. Pollack [13] has carried this to a further extreme by showing that an infinite Turing tape can be embedded in the infinite precision of the real variable y_{it}. We shall see that (1) produces complex and varied temporal behaviour over broad regions of weight space, even in very small networks, so we are encouraged that there may often be more efficient ways to produce many calculations than the straightforward neural implementation of a Turing machine. And of course, network models lend themselves to learning from examples by continuous parameter tuning, whereas Turing machines require explicit programming.

Following the description of the attractors produced by (1), we turn to a discussion of the *Moving Targets* training algorithm [20, 21, 22], including an update on the status of research in this area. This algorithm is not biologically plausible and has serious practical disadvantages, but it offers a clear notational framework for an ensuing discussion of training issues in discrete-time systems. This discussion will point out that some types of temporal training problems are qualitatively more difficult than others, and that the



20 node network, Dt = 1

Mean and SD of Entropy

max mean power = -33.887dB max sd of power = -36.318dB

max mean entropy = 3.774 max sd of entropy = 2.311

Figure 1: Influence of symmetry and magnitude of weights on complexity of motion. 20-node network. r is magnitude of weight matrix; a specifies amount of symmetry. Totally antisymmetric weight matrix for $a = -1$; totally symmetric for $a = 1$. Linear network when $r \to 0$; hard thresholds when $r \to \infty$.

most difficult class is also the most interesting. We shall also briefly note that the important issues concerning generalization which arise in feedforward networks used for classification problems become even more serious in dynamical problems.

2 Attractor properties

A broad overview of the temporal patterns generated by systems (1) is indicated by figure 1. This shows an entropy-like measure of the complexity of the power spectra of the attractors generated at 273(=21x13) points of a

2-parameter subspace of the otherwise random 225-parameter weight space of a 20-node network. The two parameters are r and a in

$$w_{ij} = r\sigma(a)S_{ij} + (1 - \sigma(a))A_{ij}, \tag{3}$$

where S is symmetric, A is antisymmetric, and $\sigma(a)$ varies from 0 to 1 as a varies from -1 to 1, changing most rapidly when a is near -1 or 1. Thus w is purely antisymmetric when $a = -1$, purely symmetric when $a = 1$, and random when $a = 0$. The steepness of the slope of (2) at 0 increases with the other parameter, r. Means (represented by dark disks) and standard deviations (represented by surrounding circles) of this complexity measure were taken over 10 randomizations of the weight matrix at each of 10 initial states of the nodes. Full details appear in [18].

We see that the most complex behaviour is centered on slightly antisymmetric matrices in a particular range of r-values. The blank area at low r (nearly linear networks) indicates fixed point behaviour. The blank areas at $a = 1$ (symmetric matrices) correspond to fixed points and period-2 oscillations (which are not detected by the measure used). A theorem of Frumkin and Moses [4] proves that these are the only possible motions for purely symmetric weights when $r \to \infty$. The purely antisymmetric case ($a = -1$) shows fixed points and period-4 limit cycles, consistent with a theorem of Gutfreund, Reger, and Young [7] on the $r \to \infty$ antisymmetric case. Qualitatively similar results were obtained for networks with between 4 and 25 nodes. The overall complexity increases with the number of nodes (very roughly linearly), with the region of maximum complexity remaining the same. (To achieve this constancy, the magnitude of the random weights is varied inversely as the square root of the number of nodes, thus keeping the RMS expected input to each node constant for random states.)

The relationship between the results for the discrete system and the differential system

$$\frac{dy_i(t)}{dt} = -y_i(t) + f(r\sum_j w_{ij}y_j) \tag{4}$$

were investigated by varying the parameter Δt in the mapping

$$y_i(t + \Delta t) = -(1 - \Delta t)y_i(t) + \Delta t f(r\sum_j w_{ij}y_j(t)). \tag{5}$$

This mapping reduces to (1) when $\Delta t = 1$ and to (4) when $\Delta t \to 0$. Renals [16] reports that the maximum complexity gradually increases as Δt is reduced, and that the region of maximum complexity moves toward higher r values and lower (more antisymmetric) a values. The region occupied by fixed points grows, presumably because of the growing influence of the term $-(1-\Delta t)y_i(t)$. This is consistent with the results of Simard, et. al. [26], who report fixed point behaviour at low r in the differential system (4), and those of Kurten and Clark [10] who found chaotic behaviour in a system similar to (4), but with a limited number of connections into each node (limited *fan-in*). These results were confirmed in further simulations by Scheurich [24], who also studied the effects of restricted fan-in in the discrete-time system (1). A further survey in the region of large r and small Δt (< 0.1) would be highly desirable, because smooth variation of the complexity measure with decreasing Δt has not yet been convincingly observed, and therefore we cannot yet be certain that its continuum limit is meaningful.

Renals [18, 17] also studied the structure of the attractors of some of the most complex systems found in the survey by using bifurcation diagrams, attractor sections, dimension calculations, and Lyapanov exponents.

The bifurcation diagram in figure 2 indicates how the attractor changes as r is varied. For each point on the r-axis, a random 8-node network was iterated for 10000 time steps from the same initial state to run out transients. Then the output of a particular node was plotted in the y-direction after each step of a subsequent run of 10000 steps. Qualitatively similar results were obtained whichever node was used. When $r = 0$, there is a fixed point at $y = 0.5$. This can be trivially understood by inspection of equations (1) and (3). The location of this fixed point drifts continuously as r is increased, until more complex motion sets in at $r = 5.2822$. Here the state follows points along a topologically circular path at an irrational frequency of about 0.26 revolutions per iteration of the mapping (1). An enlarged view of the bifurcation diagram in this region is shown in figure 3. A second incomensurate frequency appears at $r = 5.2840$, soon after the bifurcation to oscillatory motion. These two frequencies drift continuously, mode-locking to form limit cycles when their frequencies become comensurate with each other and the period of the mapping. A third frequency appears along with a broad-band spectrum at about $r = 6.9$, the attractor section begins to show backfolding, and more chaotic behaviour. This pattern is consistent with the Ruelle-Takens-Newhouse route to chaos [12]. At $r = 10.69$ there is

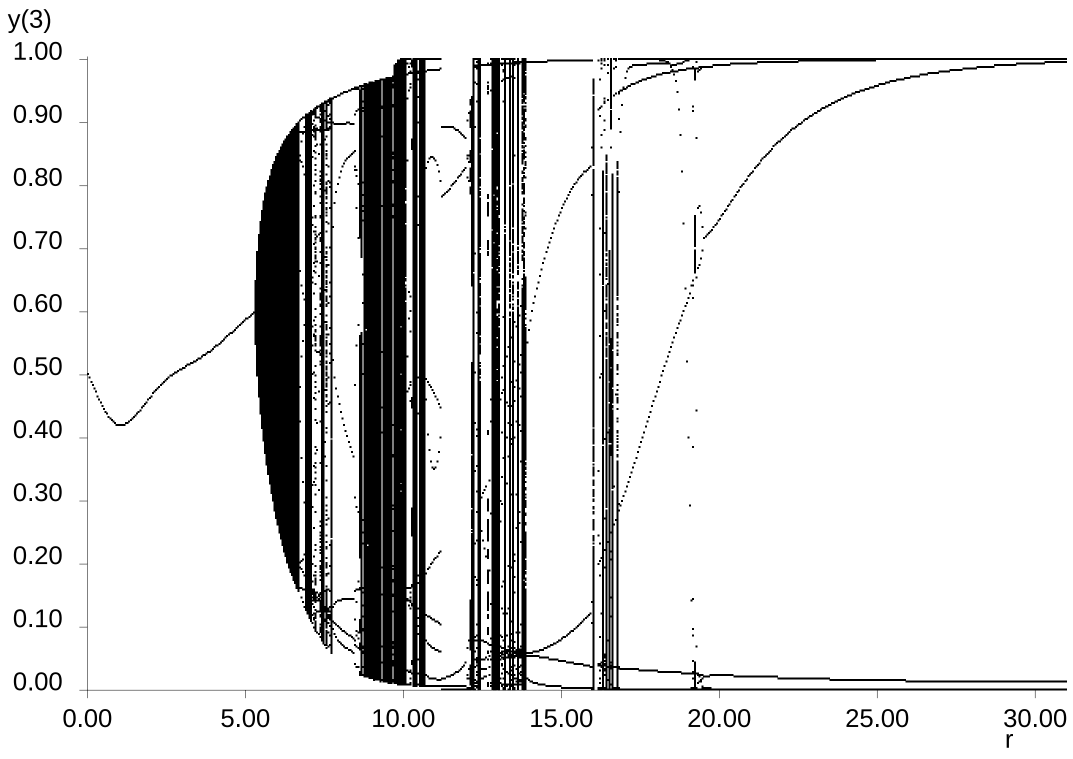
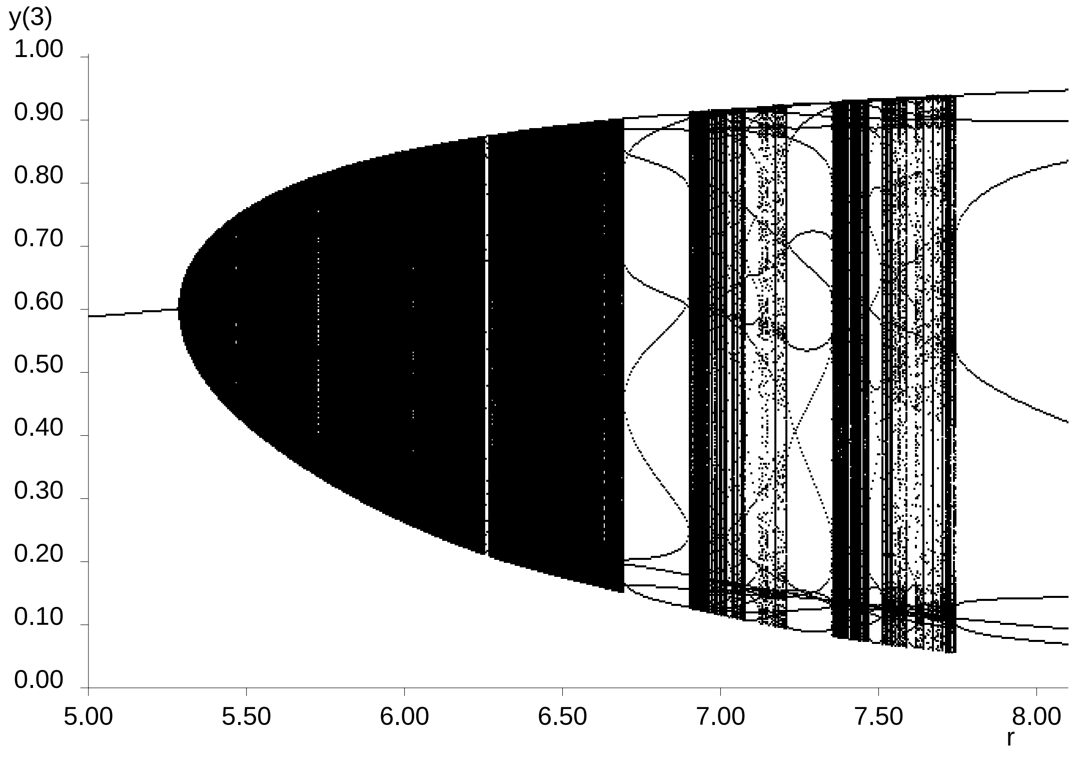
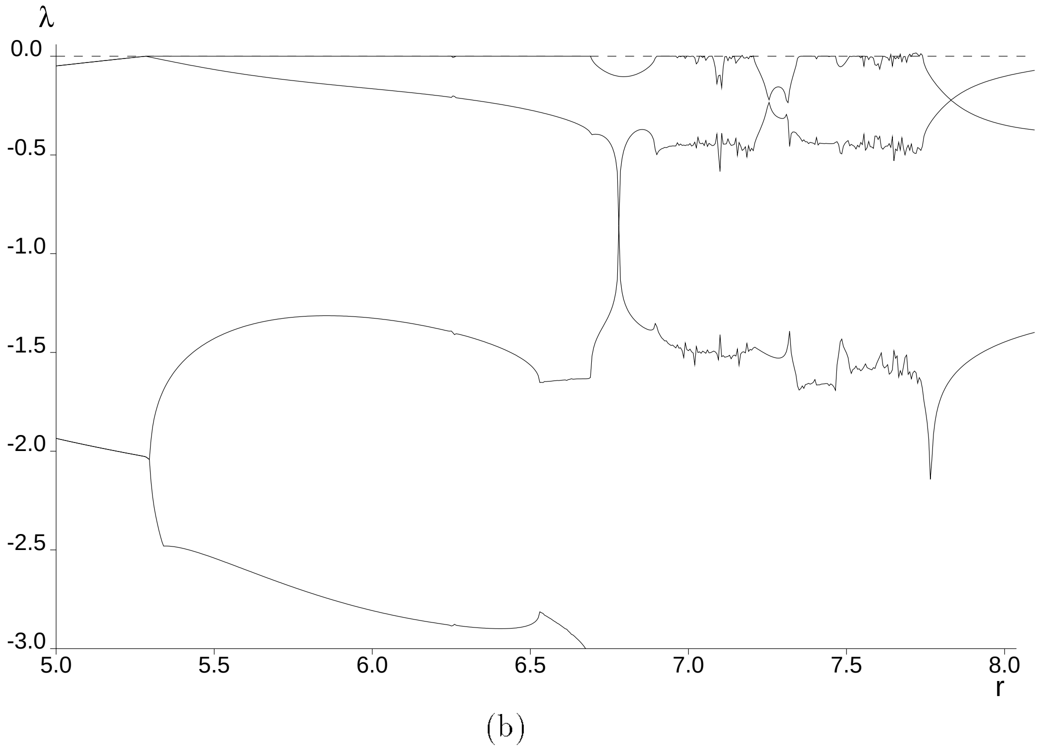


Figure 2: Bifurcation diagram for entire range of r studied.



(a)



(b)

Figure 3: (a) Expanded bifurcation diagram for range of r showing mode locking. (b) Corresponding Lyapunov exponents.

a chaotic attractor of correlation dimension of approximately 2.2 [15]. The largest Lyapunov exponent is seen to be negative for fixed points and limit cycles, rising to zero when there is no modelocking.

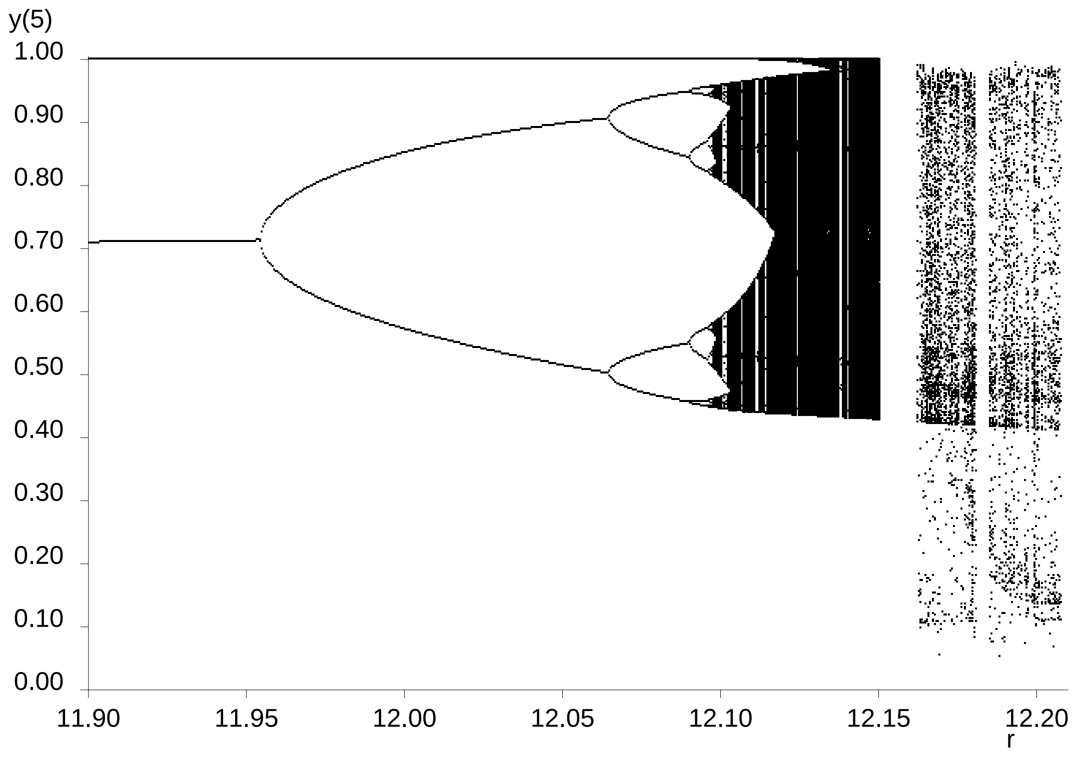
The period doubling route to chaos [3] can be clearly seen in the bifurcation diagram around $r = 12$ figure 4. The correct universal Feigenbaum constants were computed from this diagram.

Intermittent chaos was observed by Scheurich [24] in some regions of a similar system.

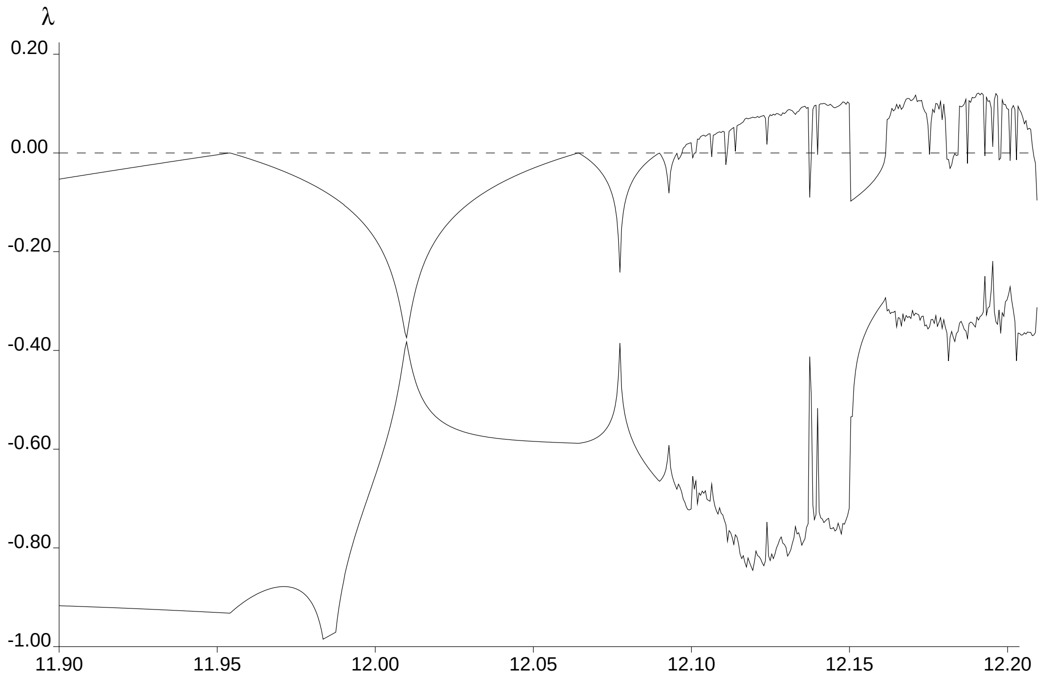
3 Dynamics and Cognition

A science of neurodynamics should one day explain the cognitive or computational significance (if any) of the temporal patterns produced by neural network models. Fixed point attractors serve as models of memories in the associative memory models typified by Hopfield's model [8]. However biologists such as Nelson and Bower [30], Baird [1], or Gray, et. al. [6], are more disposed to using oscillatory or chaotic attractors for this purpose. Skarda and Freeman [27] suggest that chaos, especially of the intermittent variety, might model an "idle" state in which the mind is prepared to trigger any of a wide variety of associative memories, which would correspond to the quasiperiodic parts of an intermittently chaotic attractor. Mode locking is required by Shastri's dynamical model of variable binding [25] in order to temporally correlate events on an ensemble of nodes.

The techniques of nonlinear dynamics concentrate primarily on attractors, but the cognitive role of attractors may be overshadowed by the role of transients. A conventional computer program halts at the end of a computation (if all goes well). Viewed as a dynamical system, all the computation occurs on the transient, although the final result is encoded in the fixed point attractor. If we associate "mental states" or "memories" with attractors in a neural network model, then an important part of the cognitive theory must concern how the network state moves from one attractor to another. If this can happen without external influence, then technically these "attractors" may not *really* be attractors after all. A point is on an attractor, loosely speaking, if and only if a small neighborhood of the point is revisited an infinite number of times as the state evolves forever under the dynamical



(a)



(b)

Figure 4: (a) Expanded bifurcation diagram for range of r showing period doubling. (b) Corresponding Lyapunov exponents.

law. Therefore the technical issue is whether a memory which has been visited once is eventually revisited according to a given model. In a theory such as this, the structure of the entire attractor would provide information about switching between mental states, and the states themselves would be described by a theory of "sub-attractors". Alternatively it may be better to model memories as proper attractors in networks or subnetworks and invoke external perturbations (from the environment or other subnetworks) to move the state between attractors. A formalism which takes a unified view of these two scenarios might be quite useful. The method of Crutchfield [2] might be a step in this direction.

4 Training

Neural network models might be useful as practical engineering tools for applications such as pattern recognition, qualitative reasoning, and motor control. Abstractly, one would like a network to produce temporal patterns, perhaps in response to external temporal patterns, which have specified properties. It can be a difficult matter to specify the desired properties formally. Connectionist systems offer the possibility of avoiding this issue by using training methods based on learning from examples. An example consists of a partially specified temporal pattern.

4.1 Learning of Dynamics from examples– notation

Let us formalize the partial specification of a set of temporal patterns. A set of sequences of fully-specified network states can be given by numbers Y_{itp} for node i at time t in sequence p. (A weight matrix may or may not exist which produces a given pattern.) Let us call the combination of indices (itp) an *event*. A sequence of events might be only partially specified because of the absence of data for Y_{itp} at particular events. Let us call such events *hidden*, in analogy with the hidden layers of feedforward networks for which there is no training data. At non-hidden (let us say, *visible*) events, Y_{itp} might represent a desired output of the dynamical system to be trained, or else an input to be imposed by the environment. Let us refer to the former case as a *target* event and the latter as an *input* event. When environmental inputs are imposed on the network, the dynamical law (1) is replaced by

$$\left.\begin{array}{rcll} y_{itp} & = & f(\sum_j w_{ij} y_{j,t-1,p}) & i \notin I \\ y_{itp} & = & Y_{itp} & i \in I \end{array}\right\} \quad (6)$$

where I is the set of input events.

In general, a node can participate in different types of events at different times. Therefore a phrase such as "the set of target nodes" is meaningful only with reference to a given time and sequence, unless (as if often true) this set happens to be the same for all times and sequences.

4.2 The importance of hidden events

Training problems which do not involve hidden events are relatively easy to solve, provided a solution exists. Because the training data completely specifies the network state at every time step, the network needs only to learn how to associate each state with its successor, ignoring the overall ordering of the states. This is precisely the form of problem normally presented to a feedforward network with one layer of weights. Therefore the problem reduces to that of training a feedforward network. Lapedes and Farber [11] exploit this fact in their work on emulating chaotic time-series using a neural network model.

A training problem can be presented in such a way that hidden events are avoided entirely. It may be that a problem does not specify input or target data for every time step, but such a problem can be extended by supplying target data *ad hoc* for such events. If the extended problem is solved, the original problem is solved as well.

Hidden nodes are therefore necessary only if it is impossible to train the feedforward network problem for state transitions. There are two possible cases of this necessity: The feedforward problem may be well-posed or ill-posed. In a well-posed problem, there are no two transitions for which the same initial state is mapped into different final states. If the feedforward problem cannot be solved with a network having one layer of weights, a solution can be found by adding one or more layers of hidden nodes to the feedforward network [5]. If n layers of hidden nodes are added to the feedforward network, then the the hidden nodes of the corresponding recurrent network are updated at $(n+1)$-times the rate of the others.

The difficult (and more interesting) case is the ill-posed feedforward problem. The overall problem may be well-posed, in the sense that there are no two identical sequences of states which precede transitions into two different states; disambiguation requires information from further in history than the previous time step. This information can only be carried by hidden events because the input and target events do not uniquely specify a successor state. If the overall problem is ill-posed, it can be converted to a well-posed problem by choosing different initial hidden events for the ambiguous sequences.

If information from the past few time steps suffices to uniquely specify a transition, then the problem can be reduced to the feedforward case by extending the network to include time-delay lines:

$$y_{it} = f(\sum_{\tau=0}^{T} \sum_{j} w_{\tau ij} y_{j,t-1}) \tag{7}$$

Equivalently, $T-1$ sets of nodes are added, each of which is dedicated to maintaining a copy of past node activations up to the maximum delay time T. However, if the information required for disambiguation lies in the distant past, this method requires a correspondingly large T and therefore involves a very large weight matrix.

In the most difficult problems, the training data from nearby time steps does not carry enough information to uniquely specify the target data for subsequent time steps. It is necessary for the network to learn to detect and remember information which is of no use in the present, but may be useful in the distant future. Hidden nodes are absolutely necessary in this case, because no other nodes can express the required memories. With no training data to apply directly, the training algorithm must decide what should happen at the hidden events. This is the *temporal credit assignment problem*, [28].

Even if transitions can be unambiguously specified by current or recent training data, it may be a good idea to use hidden nodes, and to encourage the use of information from the distant past in order to achieve good generalization performance. The information required for generalization may distributed over a long time interval even though random noise serves to make each state in the training data unique, given its predecessor.

5 Two gradient descent algorithms for solving the credit assignment problem

In this section we compare two training algorithms which are closely related mathematically, but solve the temporal credit assignment problem in very different ways. Both algorithms use a gradient method to minimize a sum of positive definite measures of error. For simplicity, let us consider a quadratic error measure:

$$E = \tfrac{1}{2} \sum_{(itp) \in T} \{y_{itp} - Y_{itp}\}^2. \tag{8}$$

Here Y_{itp} is the target data for event $(itp) \in T$, and y_{itp} is the corresponding output produced by the network.

5.1 Backpropagation through time

The two algorithms differ in their definitions of the functional form of y_{itp}, and the range of summation in (8). They produce the same values for y_{itp} if $E = 0$. The first algorithm is back-propagation through time as defined by Rumelhart, Hinton and Williams [23]. In this case y_{itp} is defined for $(itp) \in T$ simply by (6); ie., y_{itp} is whatever will be produced by running the network with the given training data for inputs. These are functions only of the weight matrix w, so E is a function only of w. The derivatives dE/dw_{ij} are computed and used in a gradient-based minimization algorithm such as gradient descent or the conjugate gradient method [14].

These derivatives take the form given by the familiar delta rule:

$$\frac{dE}{dw_{ij}} = \sum_{pt} \delta_{itp} y_{jtp} \tag{9}$$

where

$$\delta_{itp} = \left\{ \begin{array}{ll} 0 & (itp) \in I \\ (y_{itp} - Y_{itp}) y_{itp}(1 - y_{itp}) & (itp) \in T \\ \sum_j \delta_{j,t+1,p} w_{ji} y_{itp}(1 - y_{itp}) & (itp) \in H \end{array} \right\}. \tag{10}$$

There is no error measure defined directly on the hidden events, but the δ's defined on these events govern the size of their contribution to the derivatives with respect to the weights leading into them. Therefore the

δ's provide at least an approximate measure of how credit is assigned to the various hidden events. Note that the δ's diminish *multiplicitively* with each time step, going backwards in time. Thus the magnitude of the delta's tends to diminish exponentially with time. Therefore this credit assignment method is unlikely to work in problems which can be solved only by using distant contextual information.

The same derivatives dE/dw_{ij} can be computed from a feedforward iteration rule given by Robinson [19], Kuhn [9] and Williams and Zipser [29]. Being the same derivative, it suffers from the same defect. The feedforward iteration rule has a similar multiplicative property.

5.2 Moving Targets

The *Moving Targets* algorithm was introduced by Rohwer in [20] and discussed further in [21] and [22]. In this algorithm errors are assigned directly to hidden events by assigning to each hidden event a real variable which is treated as though it were target training data. These variables are the moving targets. (Perhaps "variable" would be a better word than "moving".) targets. Thus, the total error measure is like (8), except that the sum is extended to include hidden events,

$$E = \tfrac{1}{2} \sum_{(itp) \in T \cup H} \{y_{itp} - Y_{itp}\}^2. \tag{11}$$

and the definition of y_{itp} is changed to

$$\left.\begin{array}{rcll} y_{itp} & = & f(\sum_j w_{ij} Y_{j,t-1,p}) & i \notin I \\ y_{itp} & = & Y_{itp} & i \in I \end{array}\right\} \tag{12}$$

instead of (6). The only difference between (6) and (12) is that the $y_{j,t-1,p}$ in the sum has been changed to $Y_{j,t-1,p}$. When $(j, t-1, p)$ is a target event, this designates the (constant) target training datum for this event. When it is a hidden event, this designates a (variable) moving target.

The training data specifies desired results for the target events only. Therefore we are free to adjust the moving targets to any values which happen to be helpful. The same is true of the weights. Therefore training proceeds by initializing the weights and moving targets arbitrarily and minimizing the error with respect to both sets of variables using a gradient-based algorithm.

The derivatives of (11) with respect to the weights and moving targets are much simpler than the derivatives with respect to the weights in back propagation through time, because the moving targets $Y_{j,t-1,p}$ are independent variables, and therefore not functions of the weights. Therefore the chain rule does not iterate over more than two time steps. The derivatives are:

$$\frac{dE}{dw_{ij}} = \sum_{(tp) \text{ such that } (itp)\in T\cup H} (y_{itp} - Y_{itp})y_{itp}(1 - y_{itp})Y_{j,t-1,p} \tag{13}$$

and for $(itp) \in H$,

$$\frac{dE}{dY_{itp}} = \sum_{j \text{ such that } (j,t+1,p)\in T\cup H} (y_{j,t+1,p} - Y_{j,t+1,p})y_{j,t+1,p}(1 - y_{j,t+1,p})w_{ji} - (y_{itp} - Y_{itp}) \tag{14}$$

with y_{itp} given by (12), not (1).

In order to prevent Y_{itp} from varying outside the interval $(0, 1)$ achievable by y_{itp} because of the restricted range of (2), the moving target variables can be written in terms of new independent variables X by defining:

$$Y_{itp} = f(X_{itp}). \tag{15}$$

This modifies (13) and (14) slightly. This adjustment was used in most of the numerical simulations done with the moving targets algorithm, but it is not known how helpful or unhelpful it is.

The credit assignment mechanism used by the Moving Targets algorithm is explicit: An error is assigned directly to each hidden event. Distant contextual information can be used by this algorithm when necessary, because errors at events at distant times compete additively in the error measure (11), rather than diminish exponentially with time as with Back Propagation. An example is presented in [21] and [22] in which a network trained by the Moving Targets algorithm learns to use information from 100 time steps in the past to solve a simple problem. However, computational experience with this algorithm shows it to be impractical because it requires long computation times and is prone to local minima.

6 Conclusions

We have seen that simple neural network models are capable of highly complex dynamical behaviour. Several textbook attractor types have been observed. Training networks to use these capabilities productively can be an easy or difficult task, depending whether the problem requires the use of temporal information distributed over long time intervals. Such problems require training algorithms which can handle hidden nodes. The most prominent of these algorithms, back propagation through time, solves the temporal credit assignment problem in a way which can work only if the relevant information is distributed locally in time. The Moving Targets algorithm works for the more general case, but is computationally intensive, and prone to local minima.

References

[1] Bill Baird. Associative memory in a simple model of oscillating cortex. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, page 68, San Mateo CA, 1990. Morgan Kaufmann.

[2] James P. Crutchfield and Karl Young. Inferring statistical complexity. *Physical Review Letters*, 63:105–108, 1989.

[3] Mitchell J. Feigenbaum. Universal behaviour in nonlinear systems. *Los Alamos Science*, 1:4–27, 1980.

[4] A. Frumkin and E. Moses. The physicality of the Little model. *Physical Review A*, 34:714–716, 1986.

[5] K. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2:183, 1989.

[6] Charles M. Gray, Peter König, Andreas K. Engel, and Wolf Singer. Oscillatory responses in cat visual cortex exhibit inter-columnar synchronization which reflects global stimulus properties. *Nature*, 338:334–337, 1989.

[7] H. Gutfreund, J. D. Reger, and A. P. Young. The nature of attractors in an asymmetric spin glass with deterministic dynamics. *Journal of Physics A: Math. Gen.*, 21:2775–2797, 1988.

[8] J. J. Hopfield. Neural networks and physical systems with emergent computational abilities. *Proceedings of the National Acadamy of Sciences USA*, 79:2554, 1982.

[9] G. Kuhn, R. Watrous, and B. Ladendorf. Connected recognition with a recurrent network. *Speech Communication*, 9:41–48, 1990.

[10] K. E. Kürten and J. W. Clark. Chaos in neural systems. *Physics Letters*, 114A:413–418, 1986.

[11] Alan Lapedes and Robert Farber. How neural nets work. In Dana Z. Anderson, editor, *Neural Information Processing Systems, Denver CO 1987*, pages 442–457. American Institute of Physics, New York, 1988.

[12] S. Newhouse, D. Ruelle, and F. Takens. Occurence of strange Axiom-A attractors near quasiperiodic flow on T^m, $m \leq 3$. *Communications of Mathematical Physics*, 64:35, 1978.

[13] J. B. Pollack. Implementations of recursive distributed representations. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 1, pages 527–536, San Mateo CA, 1989. Morgan Kaufmann.

[14] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vettering. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, Cambridge, UK, 1988.

[15] Stephen J. Renals. *Speech and Neural Network Dynamics.* PhD thesis, Edinburgh University, 1990.

[16] Steve Renals. Chaos in neural networks. In L. B. Almeida and C. J. Wellekens, editors, *Neural Networks*, volume 412 of *Lecture Notes in Computer Science*, pages 90–99. Springer-Verlag, Berlin, 1990.

[17] Steve Renals. PhD thesis, Dept. of Physics, Edinburgh University, to appear, Sept. 1990.

[18] Steve Renals and Richard Rohwer. A study of network dynamics. *Journal of Statistical Physics*, 58:825–847, 1990.

[19] A. J. Robinson, M. Niranjan, and F. Fallside. Generalising the nodes of the error propagation network. Technical Report CUED/F-INFENG/TR.25, Cambridge University Engineering Department, 1988.

[20] R. Rohwer. The 'moving targets' training algorithm. In J. Kindermann and A. Linden, editors, *Distributed Adaptive Information Processing (DANIP)*, pages 175–196, Munich, 1990. R. Oldenbourg Verlag.

[21] R. Rohwer. The 'moving targets' training algorithm. In L. B. Almeida and C. J. Wellekens, editors, *LN412*, pages 100–109. Springer-Verlag, Berlin, 1990.

[22] R. Rohwer. The 'moving targets' training algorithm. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 558–565, San Mateo CA, 1990. Morgan Kaufmann.

[23] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Procressing*, volume 1, pages 318–362. MIT Press, Cambridge MA, 1986.

[24] J. Scheurich. Phase effects in random networks. Undergraduate dissertation, Edinburgh University, 1990.

[25] Lokendra Shastri and Venkat Ajjanagadde. From simple associations to systematic reasoning: a connectionist representation of rules, variables and dynamic bindings. Technical Report MS-CIS-90-05, Dept. of Computer and Information Science, University of Pennsylvania, 1990.

[26] P. Y. Simard, M. B. Ottaway, and D. H. Ballard. Fixed point analysis for recurrent neural networks. In David S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 1, pages 149–158. Morgan Kaufmann, San Mateo CA, 1989.

[27] Christine A. Skarda and Walter J. Freeman. How brains make chaos in order to make sense of the world. *Behavioural and Brain Sciences*, 10:161–195, 1987.

[28] R. Williams. Towards a theory of reinforcement-learning connectionist systems. Technical Report NU-CCS-88-3, College of Computer Science, Northeastern University, Boston, MA, 1988.

[29] R. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Networks*, 1:270–280, 1989.

[30] M. Wilson and J. Bower. Computer simulation of oscillatory behaviour in cerebral cortical networks. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, page 84, San Mateo CA, 1990. Morgan Kaufmann.