Connection Science

ISSN: 0954-0091 (Print) 1360-0494 (Online) Journal homepage: http://www.tandfonline.com/loi/ccos20

A Local Learning Algorithm for Dynamic Feedforward and Recurrent Networks

JURGEN SCHMIDHUBER

To cite this article: JURGEN SCHMIDHUBER (1989) A Local Learning Algorithm for Dynamic Feedforward and Recurrent Networks, Connection Science, 1:4, 403-412, DOI: 10.1080/09540098908915650

To link to this article: http://dx.doi.org/10.1080/09540098908915650

Published online: 24 Oct 2007.

Connection Science, Vol. 1, No. 4, 1989

A Local Learning Algorithm for Dynamic Feedforward and Recurrent Networks

JÜRGEN SCHMIDHUBER

Most known learning algorithms for dynamic neural networks in non-stationary environments need global computations to perform credit assignment. These algorithms either are not local in time or not local in space. Those algorithms which are local in both time and space usually cannot deal sensibly with 'hidden units'. In contrast, as far as we can judge, learning rules in biological systems with many 'hidden units' are local in both space and time. In this paper we propose a parallel on-line learning algorithms which performs local computations only, yet still is designed to deal with hidden units and with units whose past activations are 'hidden in time'. The approach is inspired by Holland's idea of the bucket brigade for classifier systems, which is transformed to run on a neural network with fixed topology. The result is a feedforward or recurrent 'neural' dissipative system which is consuming 'weight-substance' and permanently trying to distribute this substance onto its connections in an appropriate way. Simple experiments demonstrating the feasibility of the algorithm are reported.

1. Introduction

Various algorithms for supervised learning in recurrent non-equilibrium networks with non-stationary inputs and outputs have been proposed (Robinson & Fallside, 1987; Williams & Zipser, 1988; Pearlmutter, 1988; Gherrity, 1989; Rohwer, 1989). Apart from the fact that these algorithms require explicit teaching signals for the output units, there is a second reason which makes them biologically inplausible: they depend on global computations.

What are the differences between local and global computations in the context of neural networks? We would like to make the distinction between two kinds of local computations in systems consisting of a large number of connected units:

Local in space is meant to say that changes of a unit's weight vector should depend solely on activation information from the unit itself and from connected units. The update complexity for a unit's weight vector at a given time should be only proportional to the dimensionality of the weight vector. This implies that for a completely recurrent network the weight update complexity at a given time is $O(n^2)$ where n is the number of units.

Jürgen Schmidhuber, Institut für Informatik, Technische Universität München, Arcisstr. 21, 8000 München 2, FRG. Tel: 089-2105 2406. schmidhu@tumult.informatik.tu-muenchen.de. Research supported by a scholarship of SIEMENS AG.

Local in time is meant to say that weight changes should take place continually, and that changes should depend only on information about units and weights from a fixed recent time interval. This contrasts to weight changes that take place only after externally defined episode boundaries, which require additional *a priori* knowledge and in some cases high peaks of computation time. The expression 'local in time' corresponds to the notion of 'on-line' learning.

As far as we can judge today, biological systems use completely local computations to accomplish complex spatio-temporal credit assignment tasks. However, the local learning rules proposed so far (like Hebb's rule) make sense only if there are no 'hidden units'.

In this paper (which is based on Schmidhuber, 1989b) we want to demonstrate that local credit assignment with 'hidden units' is no contradiction by itself, by giving a constructive example: We propose a method local in both space and time which is designed to deal with 'hidden units' and with units whose past activations are 'hidden in time'.

2. Classifier Systems and the Bucket Brigade

Holland (1985) has proposed the meanwhile well-known bucket brigade algorithm for classifier systems. In this section we shortly review the main idea of this algorithm.

Messages in form of bitstrings of size n can be placed on a global message list either by the environment or by entities called classifiers. Each classifier consists of a condition part and an action part defining a message it might send to the message list. Both parts are strings out of $\{0, 1, -\}^n$ where the '–' serves as a 'don't care' if it appears in the condition part. (Less important for our purposes, the '–' serves as a 'pass-through' if it appears in the action part.) A non-negative real number is associated with each classifier indicating its 'strength'.

During one cycle all messages on the message list are compared with the condition parts of all classifiers of the system. Each matching classifier computes a 'bid' by multiplying its specificity (the number of non-don't cares in its condition part) with the product of its strength and a small factor. The highest bidding classifiers may place their message on the message list of the next cycle, but they have to pay with their bid which is distributed among the classifiers active during the last time step which set up the triggering conditions (this explains the name bucket brigade).

Certain messages result in an action within the environment (like moving a robot one step). Because some of these actions may be regarded as 'useful' by an external critic who can give payoff by increasing the strengths of the currently active classifiers, learning may take place. The central idea is that classifiers which are not active when the environment gives payoff but which had an important role for setting the stage for directly rewarded classifiers can earn credit by participating in 'bucket brigade chains'. The success of some active classifier recursively depends on the success of classifiers that are active at the following time ticks.

As an additional means for improving performance Holland introduces a genetic algorithm to construct new classifiers from old successful ones. This feature will not be important for our purposes.

3. The Neural Bucket Brigade (NBB)

In this section we propose a combination of principles of the bucket brigade algorithm with principles of neural networks. Competition can be introduced naturally into

neural networks by a mechanism of lateral inhibition. What we still need is a mechanism analogous to the process of bidding and paying in classifier systems. This mechanism must establish recursive dependencies 'through time'. We introduce a local method for shifting 'weight substance' (initially provided by the environment) from weights that are allowed to transport activation information at a certain time to those weights that were 'setting the stage' one time tick earlier.

The basic network structure is an arbitrary (possibly cyclic) directed graph, where the nodes are familiar processing units. Some units are used for input purposes, others serve as outputs and may be coupled with effectors that may change the environment, which in turn may change the current input. Thus we have external and internal feedback.

The set of non-input units is partitioned into predefined 'competitive subsets'. All non-input units synchronously try to get activated by summing their weighted inputs at each time tick. All members of a predefined competitive subset laterally inhibit each other (by some 'winner-take-all' mechanism) thus competing for being active. Unlike with most other approaches to goal directed learning the basic building blocks of the network are not simple units but winner-take-all subsets, each of which should have at least two members.

All weights are randomly initialized with a positive real value, and are modifiable. Initially we will assume that there is instant decay: A unit active at time t manages to send its contributions to connected units that try to get activated at $t+1$, then the sender is switched off instantly.

All units active at time t take away a fraction of the positive weights of their outgoing connections (if there are any) that lead to winners active at time $t+1$, and distribute this 'weight-substance' proportionally to the respective contributions among the incoming connections (if there are any) coming from winners (or input units) active at time $t-1$. Since the weights determine the context-dependent strength of a unit, winners 'get paid' for setting the stage for their successors. Input units do not have any incoming connections that they could strengthen, they get activated by the environment thus representing holes through which the weight-substance of the system is leaking. The environment's influence is completed by sometimes rewarding (or punishing) the connections to currently active units in the case of useful output behaviour. (An external critic decides what kind of behavior is useful.) The sum of all positive weights in the system remains constant, except for the weight-substance that is leaking through the input units and the new substance that is entering the system in the case of payoff. Thus we have a dissipative system which is consuming weight-substance provided by the environment.

More formally, at time t we denote the activation of the jth unit by $x_j(t)$, the weight on the directed connection between units i and j by $w_{ij}(t)$, and the contribution of some connection by $c_{ij}(t)=x_i(t-1)w_{ij}(t-1)$.

The activation rule works as follows: Unit j gets activated at time t if it is an input unit and receives a perception, or if it wins the competition between the units in the competitive subset it belongs to by having the largest positive net input $\text{net}_j(t)=\Sigma_i\, c_{ij}(t)$. We assume the simplest case: $x_j(t)$ equals 1 if unit j is active, and 0 otherwise. (For instance, a conventional boolean unit with two possible activation states may be implemented by a competitive subset with two members.)

If non-input unit j is active then its weights change according to

$$\Delta w_{ij}(t) = -\lambda c_{ij}(t) + \frac{c_{ij}(t-1)}{\Sigma_i\, c_{ij}(t-1)} \sum_{k \text{ wins}} \lambda c_{jk}(t) + \text{Ext}_{ij}(t)$$

where $0<\lambda<1$ determines how much of its weight some particular connection has to pay to those connections that were responsible for setting the stage at the previous time step. $\text{Ext}_{ij}(t)$ is the 'external payoff' that the environment gives to w_{ij} at time t, and may be computed like this: If the external critic does not know at time t whether useful behavior took place then $\text{Ext}_{ij}(t)=0$. Else, if the critic notices a useful action, and if unit j was active at time t, then $\text{Ext}_{ij}(t)=\eta c_{ij}(t)$ with η being a proportionality factor. As it will be demonstrated in the section describing the experiments, there is much room for more or less supervised strategies to determine Ext_{ij}: Every unit might get instructed at every step, or just a few units at certain isolated time steps, etc.

The *weights* of the system (as opposed to the activations in Hopfield-networks or feedback-BP) have reached a stable state when every connection at any time is giving back as much weight-substance as it is receiving during the next time step. This means that (parallel) chains of units and connections cooperating in time have evolved.

It is important to see the local character of this method. No book-keeping of past activations is required, not even the accumulative computation of, say, a weighted sum of past activations. Each weight and each unit in principle performs the same operation at each time tick. No such things as 'epoch boundaries' are required during training.

3.1. The NBB and Temporal Difference Methods

It seems to be unlikely that the NBB performs gradient descent in some sensible global error measure. However, Sutton's temporal difference (TD) methods (Sutton, 1988) (a generalization of both gradient descent methods and an old principle proposed by Samuel, 1959) might offer a framework for analyzing the NBB's convergence properties.

Following Sutton's discussion of relations between the bucket brigade for classifier systems and TD-methods, at a given time the strength of a connection w_{ij} leading to an active unit j (or the fraction of its contribution, λc_{ij}) may be interpreted as a prediction of the weight substance it will receive. This prediction recursively depends on the predictions of weights that will be active at later time ticks. Thus w_{ij} also predicts the ultimate environmental payoff, which terminates the recursion. A dynamic equilibrium of weight flow means that predictions meet reality.

Unfortunately, the competitive element introduced by the winner-take-all-networks makes an analysis of the NBB anything but straight-forward. The same holds in the case of classifier systems: Nobody so far has proven a theorem that demonstrates that the bucket brigade mechanism *must* work as desired.

3.2. A Possible Extension to Continuous Time

The method introduced above still has elements of global control: There is the clock for synchronous updates, for instance. However, the bucket brigade credit assignment concept is also potentially relevant for continuous time models of neural processing. To come closer to asynchronous models from biology we now give up the assumption of predefined competitive subsets and of instant decay. To save the concept of winning units we explicitly introduce fixed inhibitory connections (e.g. a variant of the on-center-off-surround structure (see Kohonen, 1988; Grossberg, 1976)).

We assume that the output $x_j \in [0,1]$ of unit j and the transmission properties of the excitatory connections are governed by differential equations that say that x_j does not change significantly during the time needed to transport activation information from

one unit to its successors. Then we write down a continuous time version of the weight changes caused by the neural bucket brigade in case of net_j being greater than zero:

$$\frac{\partial w_{ij}}{\partial t} = -\lambda x_i w_{ij} x_j + \frac{x_i w_{ij}}{\Sigma_i\, x_i w_{ij}} \sum_k \lambda x_j w_{jk} x_k + \text{Ext}_{ij}$$

Only positive weights appear in this formula, the inhibitory connections have to remain fixed. Tentatively denoting $\Delta_k\ w_{jk} x_k$ by back_j we find (by letting $\partial w_{ij}/\partial t = 0$) that the weight-flow through a positive weight w_{ij} that does not receive external payoff has reached a dynamic equilibrium if net_j equals back_j all the time.

It should be noted that there is an important difference between a continuous time version based on local on-center-off-surround wiring, and the discrete time version above. While the discrete time version assumes instant activation decay when the input to a competitive subset disappears, there will be no activation decay in case of on-center-off-surround structures. It remains to be seen whether the bucket brigade mechanism can sensibly work in case of such hysteresis effects. The only experiments conducted so far were based on the discrete time version (see below).

4. Simple Experiments

For all experiments reported in this section, the same learning rates and the same initialization conditions were used. Specifically, in the beginning of a training phase all weights were randomly initialized between 0.999 and 1.001. Both η and λ were always set equal to 0.005. No systematic attempts have been made to optimize these parameters for given tasks.

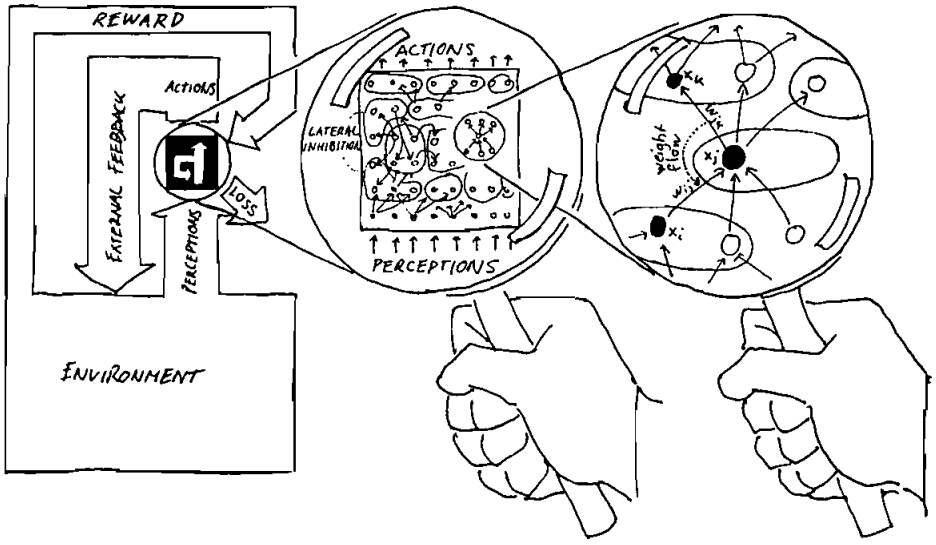

Figure 1. Weight substance given by the environment in case of successful behavior is flowing through an agent living in a changing environment. The direction of weight flow is opposite to the direction of activation flow originating from perceptions. Credit assignment (appropriate weight changes) is done by local computations only (see text for full explanation).

In all cases, the discrete time version of the algorithm described above was employed. 32-bit floating point arithmetic was used for the simulations.

XOR-problems. Any algorithm for learning sequential tasks should also allow the learning of static pattern association, since static learning tasks can be viewed as sequential tasks where inputs and desired outputs do not change over time.

In a preliminary experiment we tested whether the NBB is capable of adjusting a network such that it solves a static non-linearly separable task. The classical example for such a task is the XOR-problem.

The network was of the feed-forward type: A layer of three input units was connected to a predefined competitive subset of three hidden units and a predefined competitive subset of two output units. The subset of hidden units also was connected to the subset of output-units.

At the beginning of each cycle all unit activations were reset to 0, and one of the four binary XOR input patterns was randomly chosen. During the cycle this pattern was presented to the first two input units for a period of 6 time ticks. The activation of the remaining input unit was always set to 1, in order to provide a modifiable bias for every non-input unit in the network.

The task for the network was to switch on the first output unit if the XOR of the input pattern was 1, and to switch on the second output unit otherwise. The task was formulated as a reinforcement learning task: At each time tick the environment gave a payoff (playing a role similar to the role of a reinforcement signal) $\text{Ext}_{ij}(t) = \eta c_{ij}(t)$ to w_{ij} if unit j was an output unit and if it was switched on correctly at time t. In all other cases $\text{Ext}_{ij}(t)$ was set to equal to 0. (Recall that payoff can be considered as a bit of weight substance which has to be distributed in an appropriate way by the NBB algorithm.)

The network was said to correctly classify a single pattern if it switched on the corresponding output unit during the last three time ticks of a cycle, without the weight changing mechanism being employed. The network was said to have solved the problem if it correctly classified the four input patterns. To test whether the network had already solved the problem, after each training cycle the weight changing mechanism became disengaged, and the network's classification capabilities were tested on each of the four input patterns.

During 20 test runs the network needed an average of 619 pattern presentations to find a solution. So each of the four patterns had to be presented for about 155 times, which corresponds to the notion of 155 'epochs'.

Most of the 20 solutions were brittle in the sense that further training did not necessarily stabilize them. For instance, after a solution had been found, another 5 training cycles could lead to worse performance again. So we measured the number of cycles needed to achieve a stable solution.

A solution was considered to be stable if 100 additional pattern presentations did not disturb the performance of the network. The precise testing procedure was as follows: After each random pattern presentation the weight changing mechanism became disengaged. In a second phase a new pattern was randomly selected, and it was tested whether the network could classify it correctly. Then the bucket brigade mechanism was switched on again. This procedure was reiterated until the network produced 100 correct classifications (in the second phase) in a row. During 10 test runs it was found that each pattern had to be presented for about 674 times in order to reach this criterion.

Using two instead of three hidden units, an average of 160 presentations per

pattern was required to find a solution for the problem. However, it was not possible to obtain stable solutions, according to the criterion above.

The XOR problem was also tested with a different network architecture: Instead of using straight-through connections from the three input units to the output units, the input units were connected only to two hidden competitive subsets, each containing two units. Both hidden subsets also received input from a unit which was always on. In 10 test runs the network failed twice to find a solution within 4000 random pattern presentations. During the remaining 8 test runs an average of 263 presentations per pattern was required to solve the problem. Similarly, for 8 out of 10 test runs an average of 911 presentations per pattern was required to find a stable solution, according to the criterion above.

Encoding-problems. Another task that had to be solved was an 'encoding problem'. Eight 8-dimensional binary patterns, each having unit length, had to be associated with themselves. A bottleneck of hidden units made the task non-trivial: 8 input units were connected to three hidden competitive subsets containing two units each. These were connected to a competitive subset of 8 output units. (An additional unit that was always on was connected to all non-input units.) Note that since each of the hidden subsets could have only two different states, an extreme solution was required: It was necessary to fully exhaust the representation capacity of the bottleneck.

The learning mechanism employed for this problem illustrates how the bucket brigade mechanism can be employed in a more supervised manner: Unlike with the XOR-problems above, the connections leading to the output unit which *should* have been activated in response to a given input pattern received external payoff, even if the output unit erroneously had not been activated. Besides this modification, the learning and testing procedures were the same as with the XOR-problems.

During 10 test runs, an average of 1364 presentations per pattern was necessary to obtain a solution. However, it was not possible to obtain stable complete solutions.

Widening the bottleneck to 4 hidden subsets allowed the algorithm to find stable solutions. About 2460 presentations per pattern sufficed to satisfy the criterion for stability (10 test runs were conducted).

We also conducted experiments where the input patterns for successive cycles were not chosen randomly but in periodical sequential order. Here we found that the average time to find a solution increased, and that the solutions tended to be more brittle in the sense that it took much longer to achieve stable solutions. This suggests that the random element in the process of pattern selection introduces a stabilizing effect. One might suppose that similar stabilizing effects could be achieved by using stochastic activation rules. However, this has not been tested.

Where can instabilities arise from? The brittleness of the first solutions was attributed to the empirically observed fact that competing units within a competitive subset often had very similar net inputs. This in turn was attributed to a property of the NBB algorithm: Weights of connections leading to units that loose a competition remain the same. Consider a unit j that does not participate in a bucket brigade chain causing a correct classification of pattern A. This means that j's weights do not change. If the weights of j are slightly increased during the presentation of another correctly classified pattern, this modification may also lead to a winning situation for j during the next presentation of A. This may be the case if the net input of the competitor of j who usually won during A's presentation was only slightly larger than j's net input. This may cause an incorrect classification of A. The interplay of these effects may lead to instabilities.

Sequence generation. Important *raisons d'être* for the NBB are given by time-varying inputs or outputs. The task described next required oscillatory behavior of certain ouputs in response to a stationary input.

Two input units were connected to a competitive subset of three output units, which were fully interconnected. The task was to switch on the first and the second output unit in an alternating manner as long as the first input unit was switched on. The second input unit served to provide stop-signals: Its activation had to be answered by a stationary output of the third output unit.

The learning procedure for this problem demonstrates how 'teacher forcing' (applied by Williams & Zipser, 1988, to a similar problem) can be incorporated into the NBB in a straight-forward manner: Instead of using the actual activations of the output units at time t for computing the outputs at time $t+1$, the desired activations at time t were used.

While the network was continually running, the input units were activated randomly: the probability that the first input was switched on at a given time tick was 75%, the probability that the second input unit was switched on was 25%. Payoff was given whenever the correct output unit was switched on at a given time tick. Within less than 30 time ticks the system found stable solutions for this task. However, similar to the quite different algorithm emplyed by Williams & Zipser, without teacher forcing the task could not be reliably learned.

Sequence recognition. One of the simplest tasks involving non-stationary environments may be to recognize different kinds of motion. We conducted a simple experiment with time-varying perceptions. A one dimensional 'retina' consisting of 5 input units (plus one additional unit which was always turned on) was fully connected to a competitive subset of two output units. This subset of output units was completely connected to itself, in order to allow recurrency. The task for the network was to switch on the first output unit after an illumination point has wandered across the retina from the left to the right (within 5 time ticks) and to switch on the first output unit after the illumination point has wandered from right to the left.

During one cycle one of the two sequences (which had been chosen randomly) was presented to the network twice. Payoff was given as described for the stationary XOR experiments. In 1 out of 10 test runs the network did not find a stable solution within 3000 cycles (according to a criterion analogue to the one used for the stationary experiment). In the remaining 9 test runs an average of 223 cycles per sequence was needed to achieve a stable solution.

The experiments described above share a rather simple nature. It remains to be seen how well the NBB can deal with more difficult problems, like the learning of motor control for autonomous agents in a changing environment.

5. Conclusion

There is an analogy between the NBB and competitive learning (Grossberg, 1976; Kohonen, 1988; Rumelhart & Zipser, 1986). Competitive learning also can be interpreted as a shifting of weight substance. However, here it is the weakly contributing incoming connections to a unit that have to pay to the strongly contributing incoming connections. In contrast, the NBB causes weight shifts from outgoing to incoming connections. This is the key feature used for relating present system states to past states. (Recently we have proposed another local learning scheme for recurrent

networks where a relation between past and present states is established by a second adaptive network; Schmidhuber, 1989a, 1990.)

Due to the local nature of all computations, the discrete time version of the NBB can easily be implemented such that the time complexity of one update cycle (activation changes and weight changes) is $O(n)$ where n is the number of weights in the system. For some particular connection all information needed at a given time is its current weight, its contribution during the current time step and its contribution during the last time step. For some particular unit all information needed at a given time is its current activation, the summed contributions it receives during the current time step, and the summed contributions it received during the last time step.

Short term memory can be identified in activations wandering around feedback loops. Such loops may even become stable: A competitive subset of units that is permanently referencing itself can lead to a local dynamic equilibrium of weight flow (and of activation flow running in the opposite direction). Such equilibria may get perturbed by new inputs from the environment or from other competitive subsets that do not participate in the loop.

One difference to Holland's bucket brigade algorithm is that there is no analogue to the creation of new classifiers at run time: The number of connections in an NBB system remains fixed. The justification for this is given by the fact that weights are modifiable, while the 'specificity' of a classifier is not. (Compiani *et al.*, 1989, consider more relationships between classifier systems and neural networks.)

We certainly do not want to suggest that the brain uses a weight shifting mechanism for, e.g., physically transporting transmitter substance from synapses of outgoing connections to synapses of incoming connections. However, we do not want to exclude the possibility that some kind of local feedback mechanism exists whose effects on the synapses are similar to the effects caused by the NBB.

A major property of the brain seems to be that the motoric actions which it causes depend on local computations only. The major contribution of this paper is to propose at least one possibility for how completely local computations within a neural network-like system may lead to goal directed parallel/sequential behavior.

The NBB represents a general credit assignment scheme for neural network-like structures. 'General' often seems to imply 'weak'. How 'weak' is the NBB? It remains to be seen whether the NBB can be successfully applied to difficult control tasks.

References

Compiani, M., Montanari, D., Serra, R. & Valastro, G. (1989) Classifier systems and neural networks. In E. R. Caianello (Ed.) *First Workshop on Parallel Architectures and Neural Nets.*

Gherrity, M. (1989) A learning algorithm for analog fully recurrent neural networks. *IJCNN International Joint Conference on Neural Networks,* **1.**

Grossberg, S. (1976) Adaptive pattern classification and universal recoding. 1. Parallel development and coding of neural feature detectors. *Biological Cybernetics,* 23.

Holland, J.H. (1985) Properties of the bucket brigade. *Proceedings of an International Conference on Genetic Algorithms.* Hillsdale, NJ.

Kohonen, T. (1988) *Self-organization and Association Memory,* 2nd edn. New York, Springer.

Pearlmutter, B.A. (1988) *Learning state space trajectories in recurrent neural networks,* technical report. Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

Robinson, A.J. & Fallside, F. (1987) Static and dynamic error propagation networks with application to speech coding. *Proceedings of Neural Information Processing Systems, American Institute of Physics.*

Rohwer, R. (1989) The 'moving targets' training method. In J. Kindermann & A. Linden (Eds) *Proceedings of 'Distributed Adaptive Neural Information Processing',* St Augustin, 24–25 May.

Rumelhart, D.E. & Zipser, D. (1986) Featuring discovery by competitive learning. In D. E. Rumelhart & J. L. McClelland (Eds) *Parallel Distributed Processing.* Vol. 1. Cambridge, MA: MIT Press.

Samuel, A.L. (1959) Some studies in machine learning using the game of checkers. *IBM Journal on Research and Development,* 3.

Schmidhuber, J.H. (1989a) Networks adjusting networks. In J. Kindermann & A. Linden (Eds) *Proceedings of 'Distributed Adaptive Neural Information Processing',* St Augustin, 24–25 May.

Schmidhuber, J.H. (1989b) The neural bucket brigade. In R. Pfeifer, Z. Schreter, Z. Fogelman & L. Steels (Eds) *Connectionism in Perspective,* pp. 439–446. Amsterdam: Elsevier.

Schmidhuber, J.H. (1990) Recurrent networks adjusted by adaptive critics. *IJCNN International Joint Conference on Neural Networks.*

Sutton, R.S. (1988) Learning to predict by the methods of temporal differences. *Machine Learning,* 3.

Williams, R.J. & Zipser, D. (1988) *A learning algorithm for continually running fully recurrent networks,* Technical ICS Report 8805. La Jolla, CA: University of California.