FORSCHUNGSBERICHTE KÜNSTLICHE INTELLIGENZ

Towards compositional learning with dynamic neural networks

J. H. Schmidhuber

Report FKI-129-90

April 1990

T U M

TECHNISCHE UNIVERSITÄT MÜNCHEN
Institut für Informatik, Arcisstr. 21, 8000 München 2, West Germany

Towards Compositional Learning in Dynamic Networks

Jürgen Schmidhuber*
Institut für Informatik
Technische Universität München
Arcisstr. 21, 8000 München 2, Germany
schmidhu@tumult.informatik.tu-muenchen.de

Abstract

None of the existing learning algorithms for neural networks with internal and/or external feedback addresses the problem of learning by composing subprograms, of learning 'to divide and conquer'. In this work it is argued that algorithms based on pure gradient descent or on temporal difference methods are not suitable for large scale dynamic control problems, and that there is a need for algorithms that perform 'compositional learning'. Some problems associated with compositional learning are identified, and a system is described which attacks at least one of them. The system learns to generate sub-goals that help to achieve its main goals. This is done with the help of 'time-bridging' adaptive models that predict the effects of the system's sub-programs. An experiment is reported which demonstrates the feasibility of the method.

Introduction

Terminology

External feedback. Consider a neural network receiving inputs from a non-stationary environment and being able to produce actions that may have an influence on the environmental state. Since the new state may cause new inputs for the network we speak of *external feedback.*

Internal feedback. If the network topology is cyclic, then input activations from a given time may alter the way that inputs from later times are processed. In this case there is a potential for the 'representation of state', or 'short term memory', and we speak of *internal feedback.*

Dynamic Learning Algorithms and Networks. A problem that requires credit assignment to past activation states is called a *dynamic problem.* Learning algorithms for handling dynamic problems are called *dynamic learning algorithms.* Learning algorithms that are no dynamic algorithms are called *static algorithms.* For instance, all algorithms that require settling into equilibria while the inputs have to remain stationary are considered to be static algorithms, although the settling process is a dynamic one based on internal feedback.

If a given network type can be employed for dynamic problems, and if there exists a corresponding learning algorithm, then we sometimes speak of a *dynamic network.*

The credit assignment problem. If a neural network is supposed to learn externally posed tasks then it faces Minsky's *fundamental credit assignment problem*: If performance is not sufficient, then which component of the network at which time did in which way contribute to the failure? How should critical components change behavior to increase future performance?

*This work was supported by a scholarship from SIEMENS AG

Supervised Learning. A learning task is a *supervised* learning task if there are externally defined desired outputs at certain times, but the network never needs to discover output actions on its own. Supervised learners have to consider only the internal feedback for performing credit assignment.

Reinforcement Learning. A learning task is a *reinforcement* learning task if the teacher only indicates once a while whether the system is in a desirable state or not, without giving information about how to reach desirable states. Usually an evaluative (non-instructive) teaching mechanism sometimes provides a scalar signal, the reinforcement, whose value indicates success or failure. During training the network is supposed to discover on its own outputs that eventually lead to desirable states. In contrast to supervised learning, there can be something like *undesired inputs* caused by former output actions. In general the external *unknown* dynamics have to be taken into consideration to perform credit assignment.

Reinforcement learning is strongly related to *control tasks.* With many control tasks more information is available about goal states than just a simple reinforcement signal. However, just as with reinforcement learning, the (sequential) outputs necessary to achieve the goal states in general are not known.

Existing Algorithms for Dynamic Networks and some Major Weaknesses

Various algorithms for attacking the fundamental credit assignment problem with dynamic learning algorithms in more or less complex non-stationary environments have been proposed. These algorithms can be classified according to various criteria. Some algorithms are suited for 'pure' reinforcement learning with internal feedback [25] [14], some are suited only for supervised learning [10] [9] [26] [12] [3]. Some are based on pure gradient descent [25] [9] [10] [26], some are inspired by temporal difference methods (Sutton's TD(λ) with small λ [21]) [2] [1] (only external, but no internal feedback), and [15] [17] [19] [13] (internal feedback possible). Some algorithms involve the construction of a model of certain aspects of the environment, e.g. by using TD-methods [2] [1] [15] or Werbos' heuristic dynamic programming [23] [6] or by using pure gradient descent methods [11] [4] [8] [18] [16] (using the approach of *system identification*).

All these algorithms have at least one thing in common: They show significant drawbacks when the credit assignment process has to bridge long time gaps between past actions and later consequences. Consider the following example (which is just meant to illustrate some fundamental problems):

A robot controlled by a dynamic neural network arrives at home and detects that it can not open the door because it has not got the key with it. It left the key on a table at the university (where it teaches an undergraduate course on neural networks). A wise action (to be detected by the credit assignment process) would have been to grasp the key from the table before leaving the university.

What would happen if the robot used a gradient descent method [11] [8] [16] [18] for credit-assigning its past behavior?

Essentially all past activations of all units in the system would contribute to the computation of an error gradient for the network weights. (The error may be given by negative reinforcement, or by the difference between the desired input (the view of an open door, say) and the actual input to the robot.) Traces of every past action of the robot, of every step that it made between its work place and its home would be taken into consideration for credit assignment. However, most of these steps are totally irrelevant in the context of the present task (which is to modify the network such that something like the current undesireable state will not occur again). In many cases there will have been only *a few* events in the past that might have contributed to the current failure, including the state corresponding to the decision to leave the university without the key.

What would happen if the robot used a TD-based method in the style of [2] [1] [14] [15] for credit-assigning its past behavior?

Only the most recent state(s) that the robot went through would be associated with a modified prediction of the undesireable event. The robot would have to repeat the same mistake again and again to allow credit assignment to relevant decisions that have been taken somewhere in the beginning of each of the many unsuccessful trials.

Obviously both approaches show awkward performance in the case where the robot already has learned a lot of action sequences in the past (like 'walking home' or 'grasping the key'). Both approaches

tend to modify 'sub-programs', instead of modifying the trigger conditions for sub-programs. They do not have an explicit concept of something like a sub-program. Pure gradient descent methods *always* consider *all* past states for credit assignment, no matter how much has been learned previously. TD(λ) methods with small λ's *always* consider only the most recent states, no matter how much has been learned previously. In general both tend to consider the wrong states.

There is a need for the learning of *dynamic temporal attention* to those past events that probably are *relevant* for credit assignment. (First approaches to the learning of dynamic *spatial* attention have been described in [20] and [24].)

Compositional Learning

If there is no prior knowledge about typical consequences of certain action sequences, then the robot cannot be expected to sensibly reduce the number of past states that are 'critical' for credit assignment. However, if it is assumed that the robot has already learned to perform well on certain sub-tasks, then an intelligent credit assignment process should make use of available sub-programs to ease the learning of new tasks.

In fact, the robot incrementally should use information about the starting conditions and the effects of sub-programs to compose more complicated sub-programs.

Compositional learning means to find solutions for new tasks by sequentially combining solutions to older tasks. It means learning to 'divide and conquer'. It means to divide the task of finding a sequence of actions leading from a current state to a goal state by decomposing the problem into sub-tasks, such that already existing sub-programs for the sub-tasks can be combined. It means to ignore irrelevant details of past sub-programs. It means to focus on *the cuts between sub-programs.* Compositions of sub-programs may serve as sub-programs for even more complicated tasks.

Note that there is a symmetry between credit assignment to past action sequences and planning new ones. In both cases sub-programs have to be combined to build a bridge from start states to goal states. The same mechanism that allows to explain a current state by composing representations of past sub-processes can be used to reach a future goal by composing sub-processes. Since credit assignment makes sense only in cases where tasks that should have been solved in the past are similar to tasks that will have to be solved in the future, planning and learning are essentially equal. From now on they will be considered as one and the same.

In this paper it is assumed that the 'divide and conquer problem' can be divided and partly conquered by decomposing it into two problems, namely, the *'dividing-problem'*, and the *'conquering-problem'*.

The dividing-problem is the problem of structuring all kinds of sequences of events and/or actions into 'parts that belong together'. It is the problem of deciding *what* a good sub-program is, which its initial conditions are, and when it ends. The dividing-problem has to do with unsupervised learning.

The conquering-problem is to select among many available sub-programs and to combine them in a way that allows to reach a given goal state.

In the sequel first the problem of conquering will be isolated and studied under the assumption that the dividing-problem is already solved. (The dividing-problem will be addressed later.) So we focus on the problem of generating appropriate sub-goals for non-trivial tasks, given that there already are a number of working sub-programs available for simpler tasks. A learning procedure for a connectionist sub-goal generator is proposed. The latter makes use of an adaptive model network that predicts the effects of the system's own sub-programs.

Learning to Generate Sub-Goals

Figure 1 shows the basic components of a system consisting of a number of interacting neural networks. The heart of the system is a neural network with internal and external feedback, called the control network. The control network serves as a program executer. It receives input in form of a start state, a desired goal state, and time-varying inputs from the environment. The start and goal states serve as

'program names'. It is assumed that the control network already has learned to solve a number of tasks. This means that there already are various working programs that actually lead from the start states to the goal states by which the programs are indexed. These programs may have been learned by one of the algorithms for dynamic networks mentioned above, *or by a recursive application of the principle outlined below.*

A second important module is a static network which receives input in form of a start state and a goal state, and produces an output that indicates whether there is a program that leads from the start state to or 'close' to the goal state. An output of 1 means that there *is* an appropriate sub-program, an output of 0 means that there is no appropriate sub-program. An output between 0 and 1 means that there is a sub-program that leads from the start state to a state that comes close to the goal, in a certain sense. This measure of closeness has to be given by some evaluative process that may be adaptive or not, and which will not be specified in detail in this section. The second module represents the system's current model of its own capabilities and is called the evaluator network. It is assumed that the evaluator network is able to correctly predict that each of the already existing sub-programs works. It is also assumed that the evaluator network is able to correctly predict failures for start and goal states that do not have a corresponding working sub-program, and to predict the 'closeness' to goal states. The evaluator network can be trained in an exploratory phase that applies the available sub-programs to various combinations of start and goal states. (In more interesting situations the evaluator network will be trained on-line and in parallel with the other system components.)

Finally, the system contains a static network which serves as a sub-goal generator. The sub-goal generator receives as input the external start-input to the control network, and the desired input (the goal) for the control network at the end of the task.

The output of the sub-goal generator is a sub-goal, of course. Like the goal, the sub-goal is an activation pattern describing the desired external input at the end of some sub-program, which also is the start input for another sub-program. We concentrate on the most simple case, namely, the case where solutions for given tasks can be found by generating only one sub-goal. (However, the generalization to more than one sub-goal is straight-forward.) The sub-goal generator should output a sub-goal such that there exists a sub-program leading from the start state to the sub-goal, and that at the same time there exists a sub-program leading from the sub-goal to the goal state.

How does the sub-goal generator, which initially is a *tabula rasa*, learn to generate appropriate sub-goals? We take two copies of the evaluator network, and connect them to the sub-goal generator as shown in figure 2. The desired output of each of the copies is 1. Whenever one of the outputs of the copies is below 1, an error gradient is propagated through the evaluator network (copies) down into the sub-goal generator. The evaluator network (as well as its copies, of course) remain unchanged during this procedure. Only the weights of the sub-goal generator change. For a given problem the procedure is iterated until the complete error is zero (corresponding to a solution obtained by combining the two sub-programs), or until a local minimum is reached (no solution found). *The gradient descent procedure is used for a search in sub-goal space.*

In a certain sense the mechanism is similar to approaches using model networks for control tasks and for reinforcement learning [7] [22] [11] [4] [8] [16] [18]. In contrast to conventional supervised learning algorithms which 'make outputs differentiable with respect to *programs'* (the weights of a network with fixed topology are considered as its program), these approaches also 'make *inputs* differentiable with respect to *programs'* [16]. The approach described in this paper even goes one step further: It makes inputs differentiable with respect to *program names* (recall that program names are given by combinations of start and goal states). This is done by using 'time-bridging' evaluator networks for modelling the effects of programs.

A simple illustrative experiment

A simple experiment was conducted in order to demonstrate sub-goal learning. The programming was done by Rudolf Huber, who currently is a student of computer science at TUM.

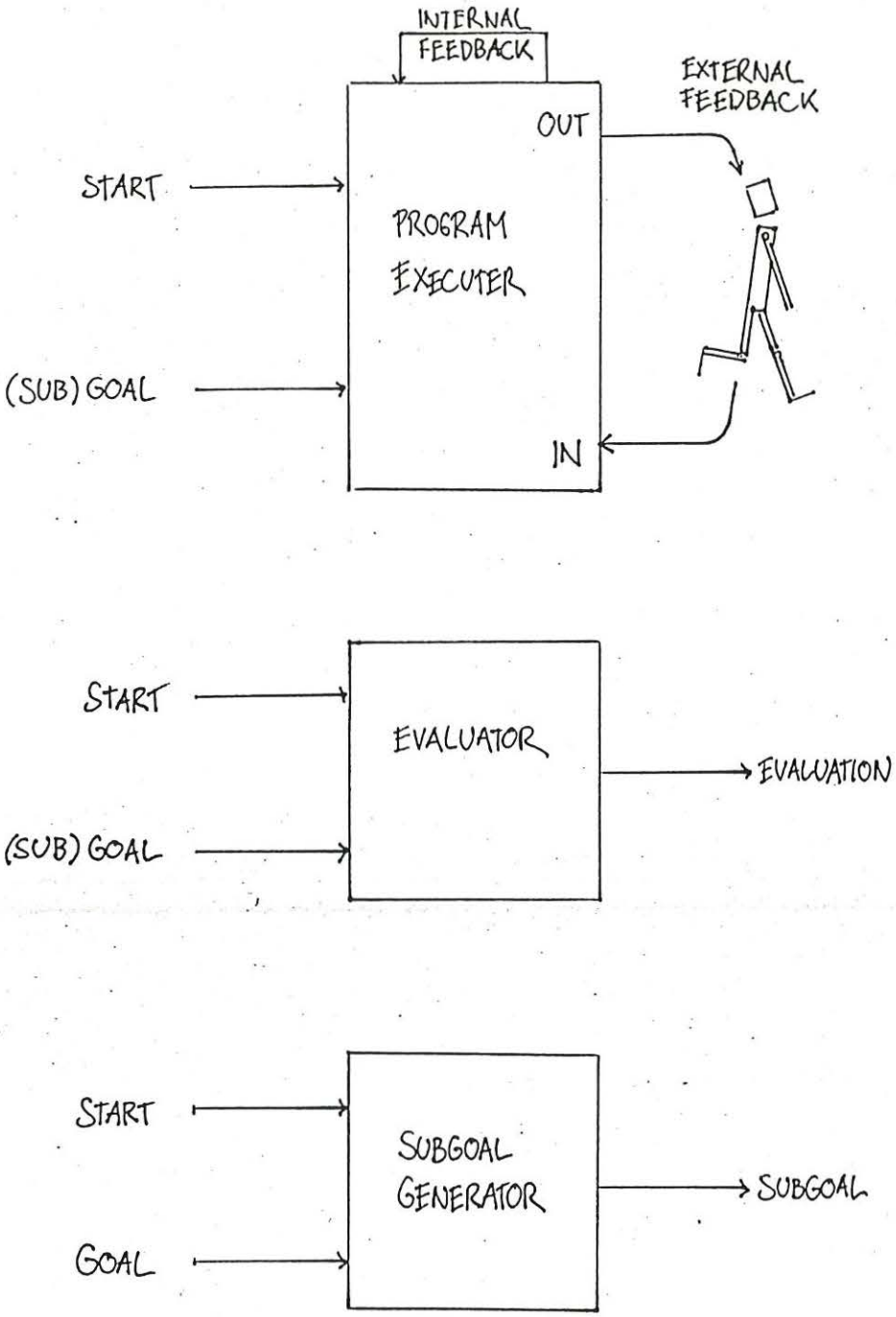

Figure 1: A program executer, an evaluator, and a sub-goal generator are shown. (See text for full explanation.)

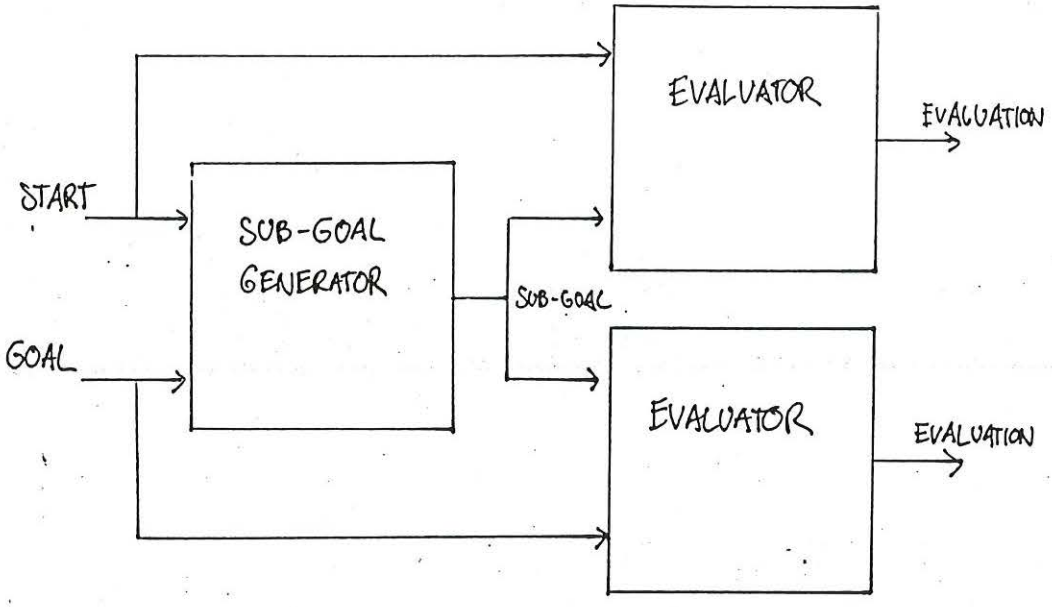

Figure 2: Two (or more) copies of the evaluator serve to compute a gradient for the sub-goal generator. (See text for full explanation.)

A conventional algorithm was used to train an artificial 'animal' to march from a certain point in a one-dimensional world to another one. Both start and goal states were indicated by coordinates of corresponding points. None of the programs taught to the animal took more than ten time steps. The maximal stepsize of the animal was limited in such a way that there were combinations of start and goal states where the animal could not know a corresponding program after training.

The evaluator was trained in a second phase to predict for given combinations of start and goal states whether there was a corresponding program or not. One reason for chosing a simple environment was to isolate the sub-goal generation process from effects that could be introduced by an *adaptive on-line* evaluation function (an adaptive critic, say). For our simple environment it was easy to define a *prewired* evaluation function: The 'goodness' of some program indexed by a start and a goal state was given by the difference between the position of the animal after program execution and the desired goal state. (Of course, future research will focus on parallel on-line learning of all components of the system, however, as always, it is preferable to proceed incrementally from small problems to bigger ones.)

In the final phase the sub-goal generator was trained: Combinations of start and goals states that did not have a working program associated with them were given to the sub-goal generation process described in the last section.

The sub-goal generator actually learned to generate appropriate sub-goals for the animal.

Currently some ongoing experiments are being conducted with two-dimensional environments (where there are obstacles).

The 'dividing problem' and 'causality detectors'

What *is* a good sub-program that is worth being memorized via a corresponding start/end combination? Similarily: Which sub-sequences that are visible in the environment 'belong together'? These questions certainly have to do with *unsupervised* temporal regularity detection.

To represent the *causal structure* of the environment in an efficient way, the following basic idea is proposed.

A sub-sequence that 'belongs together' is a sequence during which it is easy for an adaptive predictor to predict the input at a given time step from previous time steps. The adaptive predictor can be trained by any learning algorithm for dynamic recurrent networks. Whenever there is a mismatch between expectation and reality, there is a reason to memorize the situation in a separate associative device which only memorizes the *unexpected* events. There is also a reason for generating a new *name* for a sub-sequence (one can simply take the start and the end state of the sub-sequence) that bridges the time *between* the last unexpected events. This is very natural and also efficient in a certain sense, since the *expected* events *do not have to be memorized*, they can be *deduced* by considering what the system already knows.

What we get is some sort of *causality detection*: Causality detection aims at the reduction of the representation of the external dynamics such that a 'minimal' description of the causal temporal structure of the environment is obtained. Of course, 'minimality' here is relative to the current knowledge of the system: An environment may have some deep causal structure, but the learning system may be far away from being able to detect it.

Note that this kind of causality detection is reminiscent of what you yourself do: You like to remember especially the unexpected events. Those events that always repeat themselves do not call your attention.

Clearly, for goal-directed learners there are other interesting candidates for situations worth to be memorized. Regularity detection should be heavily influenced by selective attention [20][24] and current system goals. Currently research is being done on integrating 'causality detectors' and sub-goal generators into a coherent whole.

Future research

In the near future it is intended to apply sub-goal generators to problems of attentive vision as described in [20]. There the goal is to generate (without a teacher) fovea-trajectories involving translations and rotations which lead a non-stationary 'fovea' to focus on a part of the plane corresponding to a certain target. For the most interesting applications this requires to consider the interplay of effects introduced by parallel on-line learning of all system components.

Sub-goal generation is an essential but in general also very complex process. With many problems there is a need for a hierarchy of goals and sub-goals, and there also is a need for mechanisms stepping through that hierarchy in order to find appropriate sub-goals. I am currently working on an architecture where there are *internal actions* that allow the learning system itself to influence the way it builds associations, shifts attention, and triggers sub-goal generation. Of course, the goal is to make the internal actions themselves adaptive. It is intended to use causality detectors as described in the last section for credit assignment for both external *and* internal actions.

I suggest that by considering systems that can dynamically manipulate the way *how* they learn (to a certain well-defined degree), we will enter a *very* promising and exciting field which will help to bridge the gap between so-called 'sub-symbolic' and so-called 'symbolic' computation. This will open the door to *self-introspective* neural systems and to all the benefits that usually are associated with introspection and *meta-learning.*

References

[1] C. W. Anderson. *Learning and Problem Solving with Multilayer Connectionist Systems.* PhD thesis, University of Massachusetts, Dept. of Comp. and Inf. Sci., 1986.

[2] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13, 1983.

[3] M. Gherrity. A learning algorithm for analog fully recurrent neural networks. In *IJCNN International Joint Conference on Neural Networks, Vol 1*, 1989.

[4] M. I. Jordan. Supervised learning and systems with excess degrees of freedom. Technical Report COINS TR 88-27, Massachusetts Institute of Technology, 1988.

[5] J. Kindermann and A. Linden, editors. *Proceedings of 'Distributed Adaptive Neural Information Processing', St.Augustin, 24.-25.5,*. Oldenburg, 1990.

[6] G. Lukes, B. Thompson, and P. Werbos. Expectation driven learning with an associative memory. In *Proc. IEEE International Joint Conference on Neural Networks, Washington, D. C.*, 1990.

[7] P. W. Munro. A dual back-propagation scheme for scalar reinforcement learning. *Proceedings of the Ninth Annual Conference of the Cognitive Science Society, Seattle, WA*, 1987.

[8] Nguyen and B. Widrow. The truck backer-upper: An example of self learning in neural networks. In *IJCNN International Joint Conference on Neural Networks, Vol 2*, 1989.

[9] B. A. Pearlmutter. Learning state space trajectories in recurrent neural networks. Technical report, Dept. of Comp. Sci., Carnegie Mellon Univ., Pittsburgh, 1988.

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

[11] T. Robinson and F. Fallside. Dynamic reinforcement driven error propagation networks with application to game playing. In *Proceedings of the 11th Conference of the Cognitive Science Society, Ann Arbor*, 1989.

[12] R. Rohwer. The 'moving targets' training method. In Kindermann and Linden [5].

[13] J. H. Schmidhuber. Applying temporal difference methods to fully recurrent reinforcement learning networks. *In preparation,* 1990.

[14] J. H. Schmidhuber. A local learning algorithm for dynamic feedforward and recurrent networks. *Connection Science,* 1(4):403–412, 1990.

[15] J. H. Schmidhuber. Networks adjusting networks. In Kindermann and Linden [5].

[16] J. H. Schmidhuber. An on-line algorithm for dynamic reinforcement learning and planning in reactive environments. In *Proc. IEEE/INNS International Joint Conference on Neural Networks, San Diego,* 1990.

[17] J. H. Schmidhuber. Recurrent networks adjusted by adaptive critics. In *Proc. IEEE/INNS International Joint Conference on Neural Networks, Washington, D. C.,* volume 1, pages 719–722, 1990.

[18] J. H. Schmidhuber. Reinforcement learning with interacting continually running fully recurrent networks. In *Proc. INNC International Neural Network Conference, Paris,* 1990.

[19] J. H. Schmidhuber. Temporal-difference-driven learning in recurrent networks. In R. Eckmiller, G. Hartmann, and G. Hauske, editors, *Parallel Processing in Neural Systems and Computers,* pages 209–212. North-Holland, 1990.

[20] J. H. Schmidhuber and R. Huber. Learning to generate focus trajectories for attentive vision. Technical Report FKI-128-90, Institut für Informatik, Technische Universität München, 1990.

[21] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning,* 3:9–44, 1988.

[22] P. J. Werbos. Building and understanding adaptive systems: A statistical/numerical approach to factory automation and brain research. *IEEE Transactions on Systems, Man, and Cybernetics,* 17, 1987.

[23] P. J. Werbos. Consistency of HDP applied to a simple reinforcement learning problem. *Neural Networks,* 2:179–189, 1990.

[24] S.D. Whitehead and D. H. Ballard. Active perception and reinforcement learning. Technical Report 331, University of Rochester, Dept. of Comp. Sci., 1990.

[25] R. J. Williams. Reinforcement-learning in connectionist networks: A mathematical analysis. Technical Report 8605, Institute for Cognitive Science, University of California, San Diego, 1986.

[26] R. J. Williams and D. Zipser. Experimental analysis of the real-time recurrent learning algorithm. *Connection Science,* 1(1):87–111, 1989.