Appeared in:
IEEE International Conference on Systems, Man and Cybernetics (IEEE SMC'98)-
pages 1491-1495, San Diego, California, October 11-14,1998.

Integrating Symbolic Knowledge in Reinforcement Learning

G. Hailu, G. Sommer

Christian Albrechts University
Department of Cognitive Systems
Preusserstrasse 1-9, D-24105 Kiel, Germany

Abstract

A *tabula rasa* learning technique has worked well in a well defined grid like problems [1]. Nevertheless, it has severe limitations when applied in complex domains. In order to build a true learning system in complex domains, we have to begin to integrate a considerable amount of bias with the learner that has the ability to adapt *a priori* knowledge. This bias can assume a variety of forms. In this paper, in addition to reflex rules [8], symbolic knowledge about the environment is embedded into the learning system [5]. The incorporation of such knowledge aids the learner to identify and split *key* states rapidly. The learner is tested on a B21 robot for a goal reaching task. Experimental results show that after few trials the robot has indeed learned to unfold its path and to consistently follow the shortest path to the goal.

1 Introduction

For more than a decade reinforcement learning (RL) has been studied extensively and its properties are well understood. One of its nice property is that it allows agents to be programmed by reward and punishment, without the need to specify how the task is achieved. Unfortunately, it has an inherent problem - its learning time increases exponentially with the size of the state space. Consequently, RL has remained difficult to implement in realistic domains that are characterized by large state and action spaces - typically robot domain. Yet, despite this inherent problem,there is still a surge of interest in putting RL onto a real robot.

In the past, researchers have tried to overcome the in ability of RL to scale well to learning tasks with large state and action spaces. Mahadevan *et. al.* [6] have decomposed the task into sets of simple sub-tasks each with its own prewired applicability predicate. Matarić [7] has minimized the state space by transforming state-action pairs to condition-behavior pairs and maximized learning by designing reward rich heterogeneous reinforcement. Recently, Millán [8] has tremendously accelerated RL by integrating it with reflex rules that focus exploration where it is mostly needed. The common characteristic of the above examples is that the basic RL algorithm has been endowed with some *built-in knowledge*. In each case, however, the built-in knowledge has different forms and is used for different purposes: in [6] to break down and to arbiter tasks, in [7] to design rich reward and, in [8] to focus exploration.

This paper is concerned with the use of symbolic knowledge to pre-structure the state space. There is a conflict between the required number of states and the actual states constructed by the *adaptive state space construction* algorithm [8]. The constructed states are much higher than what the problem demands. As aresult, the number of learning instances, interactions with the environment, that the agent must have in order to correctly categorize all the states is prohibitively high. Therefore in order to make learning possible, symbolic knowledge about the environment is embedded into the controller. The incorporation of such knowledge enables the learner to distinguish key states instantly and there by accelerates the learning process considerably.

2 The robot and its task

The B21 robot from RWI, Fig. 1, has been used as our experimental platform. The robot is a four-wheeled cylindrical synchro-drive with two parts: a base and an enclosure. The base carries 32 infra-red (IR) and 32 tactile sensors. Whereas the enclosure carries 24 tactile, 24 IR and 24 sonar sensors.

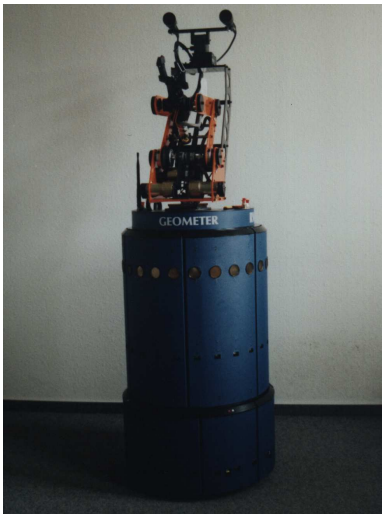

Figure 1: The B21 robot

The task of the robot is to reach a specified goal ($\boldsymbol{p}_g$), through a (sub)-optimal path. Optimality is defined on a certain payoff function. For every action the robot has chosen, it receives an immediate reinforcement r_t that has two components. The first component penalizes the robot when it either collides with or approaches an obstacle. Whereas the second component penalizes the robot in proportion to the angle between the robot heading and the vector connecting the current robot and goal locations. The immediate reinforcement value is the sum of these two components and the payoff function is defined as the sum of immediate reinforcements the robot receives until it reaches the goal, i.e., $R = \sum_t r_t$.

Any mobile robot controller that uses the robots' absolute position to generate a control signal suffers from the inaccuracy of determining the robot position. This is even severe in the presented learning system, where the robot position $\boldsymbol{p}_r(t)$ has been used twice; first, to decode the relative distance between the robot and the goal (section 3) and second to provide a part of the reinforce function from which the robot learns (previous paragraph). From the two, the latter one is more sensitive to the inaccuracy of robot position. Because it leads to inconsistent reinforce function that makes learning difficult or even impossible*. Therefore, for learning

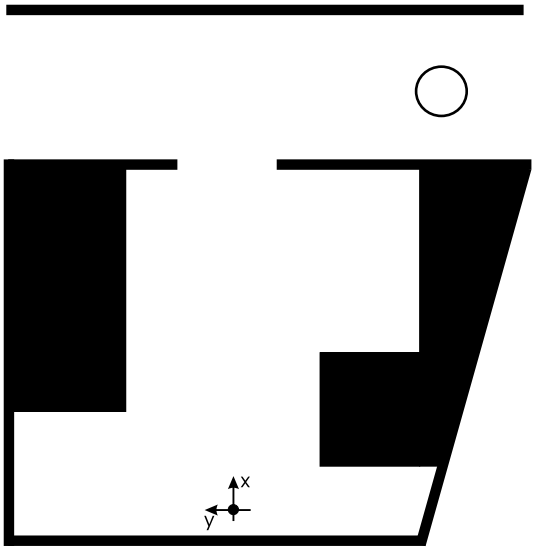

Figure 2: The experimental environment

to occur, the sensed position of the robot must not differ from the actual value considerably. In order to guarantee this, we have exploited the crucial property of the robots' dead reckoning system. Dead reckoning system performs satisfactory provided that the robot does not move for an extended periods of time without reaching the goal. This characteristic puts directly a limit on how far and how hidden the goal should be placed away from the initial robot position, Fig. 2.

3 Embedding

The input $\boldsymbol{x} = [\boldsymbol{s} \mid \boldsymbol{d}]$ to the controller is a vector of 32 elements, each between $[0, 1]$. The first 24 elements are normalized depth readings of the sonar sensors†, while the remaining eight inputs are codified distance between the robot and the goal. In the work of [8], where sensor values are made independent of the robot heading, the input to the controller turns out to be a function of the robot position (if sensors noise is neglected), i.e., $\boldsymbol{x} = [\boldsymbol{s}(\boldsymbol{p}_r(t))|\boldsymbol{d}(\boldsymbol{p}_r(t), \boldsymbol{p}_g)]$. In this case, the adaptive state space construction algorithm automatically splits key states that require different actions.

For most platforms, however, the sensors can not be aligned independently of the base. Consequently, the perceived sensory data would be different every time the robot visits a given location at different headings, i.e., $\boldsymbol{x} = [\boldsymbol{s}(\boldsymbol{p}_r(t), \theta_r(t))|\boldsymbol{d}(\boldsymbol{p}_r(t), \boldsymbol{p}_g)]$. This results in huge states which the adaptive state space construction algorithm could not cope with identifying and splitting key states quickly. To

*Noting this, Millán [9] has eliminated the dependence of the reinforcement value on the odometry reading by building other types of sensor that are capable of detecting the goal *directly*.

†Since IRs are short range ($\approx 0.3m$) proximity sensors, they are used herein emergency routine only.

overcome this problem, we have *embedded* symbolic knowledge into the learning architecture[‡] [5] so that key states can be categorized instantly.

Since there is no general method of embedding knowledge, *ad hoc* method of constructing environment knowledge is followed. We have constructed four *symbolic knowledge* by partitioning the environment into regions that are considered to be the same for the purpose of learning and generating actions. The symbols are: a `concave` region that misleads and fold the robot path, a `door` region through which the robot has to carefully pass, a `corridor` where the goal is located, and a vast space inside the `room` from where the robot starts off. Each symbol is associated with a metric data that roughly define the start and end coordinates of a region for which the symbol stands for. These symbols together with their corresponding metric data are supplied to the controller as built-in knowledge. From the metric data and the robot position disjointed rules have been written to single out a particular symbol where the robot is in. This early splitting of the state space based on prior environment knowledge can be viewed as one way of giving leverage to the adaptive state space construction algorithm so that during the course of learning it can construct appropriate states for each partition.

Apart from environment knowledge, two fuzzy behaviors [10] Fig. 3, *obstacle avoidance* and *goal following* are used as reflex to enable the learner to act initially in some reasonable way. The fuzzy reflex works as follows. First, the possible robot headings have been fuzzified into three fuzzy sets: `left, straight` and, `right`. The obstacle avoidance behavior receives the range data of the sonars and outputs avector $\boldsymbol{\alpha}_a$ - whose elements indicate the activation levels of the above fuzzy sets. Likewise, the goal following behavior inputs the acute angle θ between the robot heading and the vector connecting the current robot and goal locations and outputs a similar vector $\boldsymbol{\alpha}_g$. A simple behavior blender with constant *desirability functions* d_a and d_g is used to combine the output of the two behaviors, i.e., $\boldsymbol{\alpha}_f = d_g\boldsymbol{\alpha}_g + d_a\boldsymbol{\alpha}_a$. Subsequently, a defuzzifier decodes the fused vector $\boldsymbol{\alpha}_f$ to a crisp value α using centroid technique.

4 Learner

The architecture of the learner is an actor-critic type proposed by [8]. In most actor critic systems, two networks are adapted over time - an action network and a critic network [12]. In the proposed

[‡]Admittedly, embedding makes the system less autonomous. But it is also clear that without it, RL would not achieve a worth while performance within a bounded time.

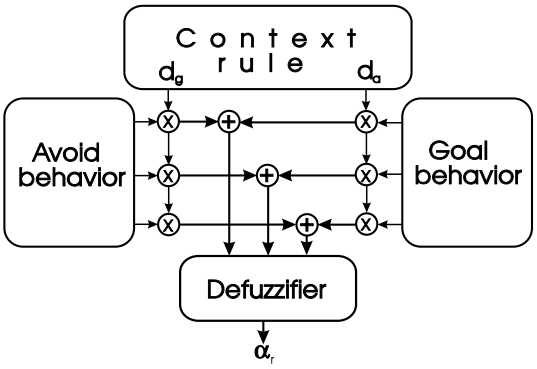

Figure 3: Two fuzzy behaviors as reflex

architecture, however, these networks are lumped into one network. Besides, unlike the former ones where the training rules adjusts some parameters or weights of the action or the critic network, the latter one adapts directly the critic and action values.

The learner, Fig. 4, consists of a gradually growing RBF neurons [2] in the input layer and a stochastic neuron in the output layer. Whenever a situation is perceived, the learning system first determine the symbol associated with the current situation. Within the symbol existing neurons (if any neuron was dedicated to the symbol before) compete to win the situation. If a winning neuron exists, it will be connected to the output layer to generate action. Action is generated by exploring a restricted area around a prototypical action. To enforce exploration a *Gaussian stochastic unit* with parameters (μ, σ) is introduced at the output layer [3]. The extent of the exploration is determined by the critic value of the winning unit and the temperature factor $T(n)$. At the end of every trial n the temperature is cooled down so that the stochastic unit produces a progressively deterministic output [1].

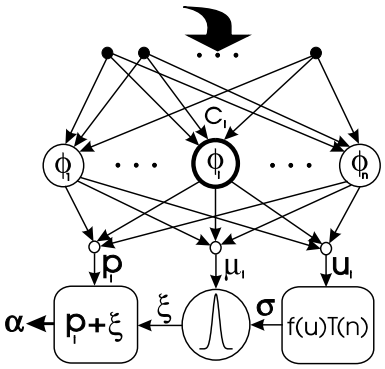

Figure 4: The learning architecture

A new neuron is introduced into the network either when existing neurons can not generalize the current situation or if a selected neuron has performed poorly for the previous situation. When a

new neuron is created four learning parameters [8]: prototypical action p_j, expected discounted sum of reinforcement u_j, weight w_j, and center location $\boldsymbol{c}_j$ are attached to it. While the prototypical action and the center position are initialized by the action suggested by the reflex and by the currently perceived situation respectively, the weight and the expected discounted sum of reinforcement are initialized to zero.

4.1 Adaptation

Each of the above parameters are adapted by different adaptation algorithms. However, the basic error source used for adaptation is the TD error [11] between the actual and expected discounted sum of the reinforcement value of the winning unit j, i.e.,

$$\delta(t+1) = r_{t+1} + \gamma u_i(t+1) - u_j(t) \qquad (1)$$

where i is the next state and r_{t+1} is the immediate reinforcement received in going from state j to state i. Based on the TD error, the utility value of the winning neuron $u_j(t)$ is updated by,

$$u_j(t+1) = u_j(t) + \Delta u_j(t+1) \qquad (2)$$

$$\Delta u_j(t+1) = \begin{cases} \eta_r\ \delta(t+1) & \delta(t+1) > 0 \\ \eta_p\ \delta(t+1) & \delta(t+1) < 0 \end{cases} \qquad (3)$$

where η_r and η_p with $\eta_p < \eta_r$ are two learning rates. Millan [8] has suggested to adapt the utility value at lower rate when the TD error is negative than when it is positive. A negative TD error may not necessary mean that the expected discounted sum of reinforcement value is lower than the actual one. It could be caused by a bad exploration in which case the utility value should not be altered. Williams' REINFORCE algorithm [13], that performs gradient descent on the expected total reinforcement, is employed to adapt the weight w_j.

$$w_j(t+1) = w_j(t) + \Delta w_j(t+1) \qquad (4)$$

$$\Delta w_j(t+1) = \begin{cases} \beta_r \delta(t+1) e_j & \delta(t+1) > 0 \\ \beta_p \delta(t+1) e_j & \delta(t+1) < 0 \end{cases} \qquad (5)$$

Where e_j is the eligibility factor that determine how influential w_j was in determining the stochastic action [13]. For the same reason like the utility update, two learning rates β_r and β_p with $\beta_p < \beta_r$ are used to adapt the weight too. Depending on the performance of the winning neuron (as measured by its TD error) the center position $\boldsymbol{c}_j$ is either shifted toward the previous sensation (Eqn. 6) or is left untouched.

$$\boldsymbol{c}_j = \boldsymbol{c}_j + \Delta \boldsymbol{c}_j \quad \text{where} \quad \Delta \boldsymbol{c}_j = \epsilon(\boldsymbol{x} - \boldsymbol{c}_j) \qquad (6)$$

Finally, when the robot reaches the goal through a trajectory whose total payoff is greater than the maximum payoff so far obtained, then a) prototypical actions p_j of all neurons that lie along the current trajectory are overridden by a more accurate learned action, b) utility values u_j of all neurons that were active during the current trial are backed up [6],

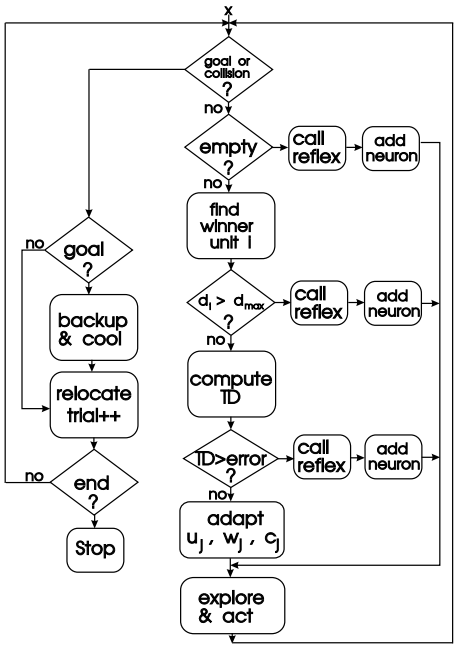

Figure 5: The control flow

$$p_j(t) \leftarrow a(t) \quad t = T \ldots 0 \qquad (7)$$

$$u_j(t-1) \leftarrow r_t + \gamma u_i(t) \quad t = T \ldots 1 \qquad (8)$$

Where T is the total number of steps the robot has required to reach the goal. Figure 5 shows the control flow diagram of the learning process.

5 Experimental results

Figures 6 and 7 depict the trajectories of the robot during the first and the last trials and figures 8, 9 and 10 show the learning curves against the number of trials. Ten sets of experiments, each consisting of 20 trials, were carried out. While the vertical error bars on the learning curves indicate the variations of the performance of the network in the set of ten experiments, table 1 shows the performance of the final network obtained from these experiments.

Despite the use of a temperature in the described reinforcement learning architecture, an asymptotically deterministic optimal policy has not been achieved. This is because noise in the: a) sensor readings triggers either an in correct neuron or even creates a new state from which a policy different from the current optimum one is followed, b) motor action makes the robot to land in an in correct state from which again a different policy is followed. Therefore, for a situated agent noise makes

certain parts of the policy to fluctuate and waiting for asymptotic convergence is not feasible.

During the first few trials, the robot has taken many steps (Fig. 9) to reach the goal and the trajectories followed were not better than what the basic reflex could have produced [4]. During these early trials, the learning system has incurred a high payoff (Fig. 10), and the number of neurons added to the network has grown sharply (Fig. 8). As trials went on, however, the robot has started to unfold its path gradually and neurons have began to be added at a much reduced slope than earlier trials. On the sixth trial and afterwards the robot has straighten its path, except at the eighth trial where the robot has left the optimum path in search for a better one. In subsequent trials, however, the robot has returned to its previous performance. A similar phenomena is also observed in the work of [8].

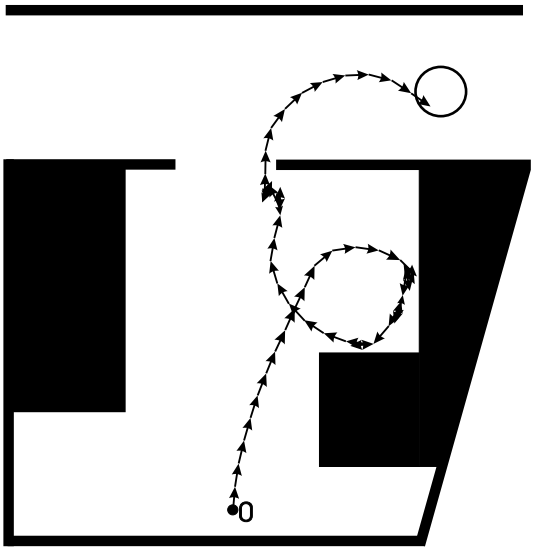

Figure 6: First trial

Comparing our results with that of [8] the following observations can be made. In our system, the dependence of the sensor reading on the orientation of the robot is combated by integrating symbolic knowledge into the learner and letting input neurons cover a wide width. This strategy, however, don't eliminate the dependence completely. The net effect of this is that in the last few trials the network has not been abated to grow (Fig. 8) like that of [8]. The second observation, which is the consequence of the first one, is the variances of final network performance (table 1) are larger than the one reported in [8].

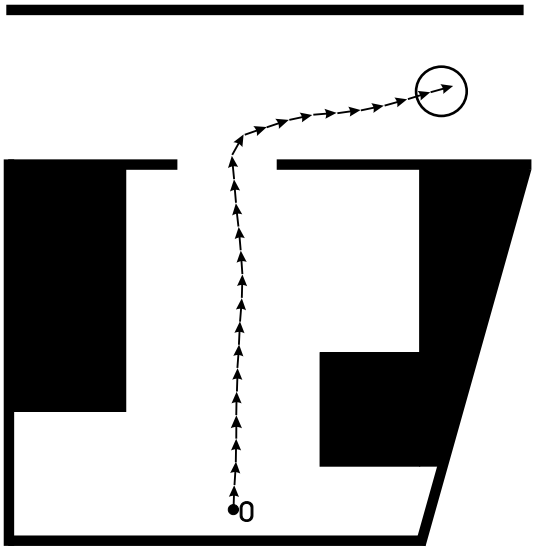

Figure 7: Last trial

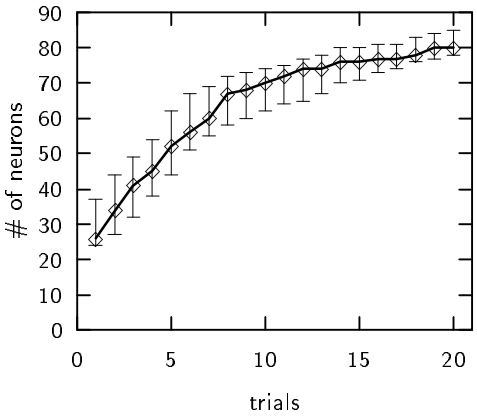

Figure 8: Number of neurons vs. trials

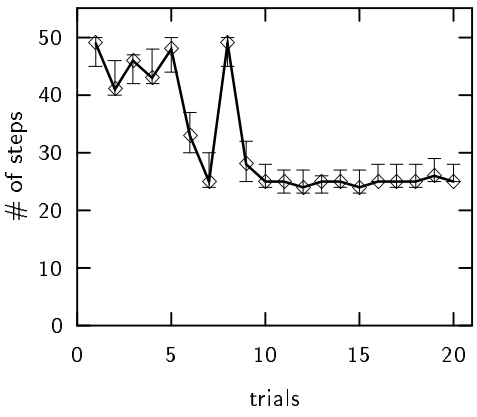

Figure 9: Number of steps vs. trials

6 Conclusion

Two kinds of built-in knowledge have been used to support RL on B21 robot. The first one is *a priori* knowledge of the environment to pre-structure the state space rapidly. Whereas the second one is two fuzzy behaviors combined with fixed desirability values to focus exploration. Ten set of experiments each lasting for twenty trials have been conducted. In all the experiments, the results have

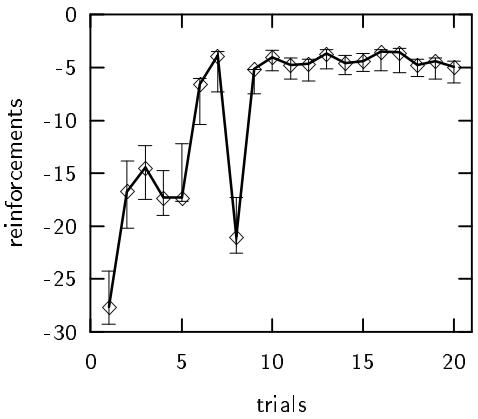

Figure 10: Total reinforcementsvs. trials

Quantities	Mean	Variance
Number of neurons	82.5	4.7286
Number of steps	27.7	1.9000
Total reinforcement	-6.24	0.8752

Table 1: Final Network Performance

shown that the robot has indeed learned to unfold its path and to 6 consistently follow a trajectory that has a minimum payoff value.

7 Acknowledgment

We would like to thank to José. R. Millán for a useful discussion on TESEO's architecture that helped us greatly in this work. The support given to the first author by DAAD under grant code 413/ETH-4-BOA is also acknowledged.

References

[1] Andrew G. Barto, Steven J. Bradtke, and Satinder P. Singh. Learning to act using real time dynamic programming. *Artificial Intelligence*, 72(1):81–138, 1995.

[2] Jörg Bruske and Gerald Sommer. Dynamic cell structure learns perfectly topology preserving map. *Neural Computation*, 7(4):834–846, 1995.

[3] Vijaykumar Gullapalli. A stochastic reinforcement learning algorithm for learning real valued function. *Neural Networks*, 3:671–692, 1990.

[4] Getachew Hailu and Gerald Sommer. Learning by biasing. In *IEEE Proceedings of Robotics and Automation*, pages 2168–2173, Belgium, Leuven, 1998.

[5] Leslie P. Kaelbling. *Learning in Embedded Systems*. The MIT Press, Cambridge, 1993.

[6] Sridhar Mahadevan and Jonathan Connell. Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence*, 55:311–365, 1992.

[7] Maja J. Matáric. Reward functions for accelerated learning. In *Proceedings of the Eleventh International Conference on Machine Learining*, 1994.

[8] José R. Millán. Rapid, safe and incremental learning of navigation stratagies. *IEEE Transactions on Systems, Man, and Cybernetics*, 26(3):408–420, 1996.

[9] José R. Millán. Incremental acquisition of local networks for the control of autonomous robots. In *7th International Conference on Artificial Neural Networks*, pages 739–744, Lausanne, Switzerland, 1997.

[10] D. W. Payton, J. K. Rosenblatt, and D. M. Keirsey. Plan guided reaction. *IEEE Transaction on Systems, Man, and Cybernetics*, 20(6):1370–1382, 1990.

[11] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

[12] Paul J. Werbos. A menu of design for reinforcement learning over time. In *Neural Networks for Control*. Bradford Book, 1992.

[13] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.