Hierarchical Reinforcement Learning

Magnus Borga
Computer Vision Laboratory
Linköping University
S-581 83 Linköping Sweden
tel. +46 13 28 27 83
fax. +46 13 13 85 26
email: magnus@isy.liu.se

Abstract

A *hierarchical representation* of the input-output transition function in a learning system is suggested. The choice of either representing the knowledge in a learning system as a discrete set of input-output pairs or as a continuous input-output transition function is discussed. The conclusion that both representations could be efficient, but at different levels is made. The difference between *strategies* and *actions* is defined. An algorithm for using *adaptive critic methods* in a two-level *reinforcement learning system* is presented. Two problems that are faced, the *hierarchical credit assignment problem* and the *equalized state problem* are described. Simulations of a one dimensional hierarchical reinforcement learning system is presented.

1 Introduction

1.1 Reinforcement Learning

Learning systems can be classified according to how the system is trained. Often the two classes unsupervised and supervised learning are suggested. Sometimes reinforcement learning is considered as a separate class to emphasize the difference between reinforcement learning and supervised learning in general.

In *unsupervised learning* there is no external unit or teacher to tell the system what is correct. The knowledge of how to behave is built into the system. Most systems of this type are only used to learn efficient representations of signals by clustering or principal component analysis.

The opposite to unsupervised learning is *supervised learning* algorithms, where an external teacher must show the system the correct response to each input. The most used algorithm is back-propagation [8]. The problem with this method is that the correct answers to the different inputs have to be known, i.e. the problem has to be solved from the beginning, at least for some representative cases from which the system can generalize by interpolation.

In *reinforcement learning*, however, the teacher tells the system how good or bad it performed but nothing about the desired responses. There are many problems where it is difficult or even impossible to tell the system which output is the desired for a given input (there could even be several equally acceptable outputs for each input) but where it is quite easy to decide when the system has succeeded or failed in a task. This makes reinforcement learning more general than supervised learning since it can be used in problems where the solutions are unknown.

There is, however, a problem with the limited feedback. In supervised learning the difference between the actual output and the desired output can be used to find the direction in which to search for a better solution. This type of gradient information does not exist in reinforcement learning. The most common way of dealing with this problem in reinforcement learning is to introduce a noise in the algorithm and thereby search for a better solution in a stochastic way.

More details and examples of reinforcement learning can be found in [1, 2, 5, 6, 7, 9, 10, 11, 12, 13, 14].

1.2 TD-methods and Adaptive Critic

When a system acts in a dynamic environment it can be difficult do decide which of the actions in a sequence that are most responsible for the result. It is not certain that it is the last action that deserves credit or blame for the system's performance. The problem is called the *temporal credit assignment problem*. This problem gets more complicated in reinforcement learning, where the information in the feedback to the system is limited, occurs infrequently or, as in many cases, a long time after the responsible actions have been taken.

In [11] Sutton describes the methods of *temporal differences* (TD). These methods enable a system to learn to predict the future behaviour of an incompletely known environment using past experience. TD methods can be used in systems where the input is a dynamic process. In these cases the TD methods take into account the sequential structure of the input, which the classical supervised learning methods do not.

Suppose that for each state s_k there is a value p_k that is an estimate of the expected future result (e.g. the total accumulated reinforcement). In TD methods the value of p_k depends on the value of p_{k+1} and not only on the final result. This makes it possible for TD methods to improve their predictions during a process without having to wait for the final result.

The *adaptive heuristic critic* algorithm by Sutton [10] is an example of a TD-method where a secondary or internal reinforcement signal is used to avoid the temporal credit assignment problem. In this algorithm a prediction of future reinforcement is calculated at each time step. The prediction made at time τ of the reinforcement at time t is

$$p[\tau, t] = \mathbf{v}[\tau] \bullet \mathbf{x}[t], \tag{1}$$

where $\mathbf{v}$ is a vector of weights called the *reinforcement association vector* and $\mathbf{x}$ is the input vector. If no external reinforcement signal is present at the time ($r[t] = 0$), then the internal reinforcement signal is principally the difference between the prediction from the previous step of the reinforcement at the

current time step and the new calculated prediction. If there actually exists a reinforcement signal it is added to the internal reinforcement. The internal reinforcement at time $t+1$ is calculated as

$$\hat{r}[t+1] = r[t+1] + \gamma p[t, t+1] - p[t, t], \tag{2}$$

where $\gamma \in [0..1]$ is a factor that controls how much the prediction should decrease for each step away from reinforcement. With this rule the system gives itself credit for moving to a state with higher predicted reinforcement and a negative internal reinforcement for moving to a state with less predicted reinforcement. The system must itself learn to predict the reinforcement, and this is done by changing the reinforcement association vector. The update rule suggested by Sutton is

$$\mathbf{v}[t+1] = \mathbf{v}[t] + \beta \hat{r}[t+1]\tilde{\mathbf{x}}[t], \qquad \mathbf{v}[0] = 0, \tag{3}$$

where β is a positive constant and $\tilde{\mathbf{x}}[t]$ is a weighted average of $\mathbf{x}$ which decreases backwards in time.

An interesting example of a reinforcement learning system using adaptive critic can be found in [1].

1.3 Discrete Versus Continuous System State Space

A response generating system can be seen as a mapping from a set of external states (inputs) to a set of actions (outputs). This mapping can be done in two principally very different ways. One method is to divide the state space into a set of discrete states and store the optimal response for each state. I call this type a memory mapping system. The other method is to approximate a continuous function from the input space to the output space. I call this method projective mapping, although the function does not have to be linear. The latter method is the most common one in feed forward neural networks, where an input vector is projected on a "weight vector".

In a learning system there is often a problem in how to decide which part or parts of the system to give credit or blame for the result. It is called the *structural credit assignment problem*. This problem gets especially complicated in reinforcement learning where the feedback to the system is a scalar. The memory mapping method is one way to avoid the structural credit assignment problem in reinforcement learning, since the "box" that contains the selected action is known.

There is, however, a serious problem with memory mapping: such a system acting in a realistic complex environment would need an unreasonable amount of memory. Another problem is that the input-output transition function is often required to be smooth, i.e. if there are two input-output pairs $(\mathbf{x}_a, \mathbf{y}_a)$ and $(\mathbf{x}_b, \mathbf{y}_b)$, and a new third input $\mathbf{x}_c$ that in some sense lies between $\mathbf{x}_a$ and $\mathbf{x}_b$, then it is often desirable that the output $\mathbf{y}_c$ lies between $\mathbf{y}_a$ and $\mathbf{y}_b$. With projective mapping this kind of interpolation is achieved.

In reality the environmental state space and hence the output space is piecewise continuous. Consider the task of moving from one point to another. There is often an infinite number of solutions and if two different paths are selected, it is often possible to interpolate these paths and get a new path that lies somewhere in between. On the other hand, if you are passing a tree

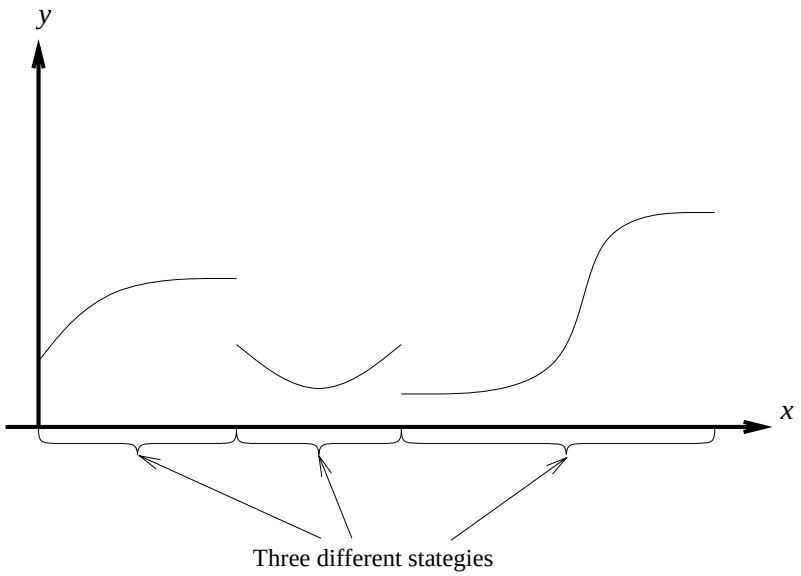

Figure 1: *The input-output transition function for a one dimensional input vector and a one dimensional output.*

for instance, you can choose to walk at the right side or the left side of the tree but there is no success in trying to interpolate these two choices. There *are* at the same time an infinite number of ways to move on either side of the tree.

This implies that there are two different kinds of responses, one that can be interpolated and another that can not.

2 Learning at Multiple Levels

The discrete set of alternative responses where there is no meaning in interpolation I will call *strategies*, and the continuous possibilities of outputs within each strategy I will call *actions*. In the previous example the choice of which side of the tree to pass on is the choice between the two strategies, left and right, and the decision of which specific path to walk is a decision of a sequence of actions.

The decisions of strategies and the decisions of actions are decisions at different levels and I will refer the actions to the first level and the strategies to the second level. It is, however, not obvious in which order the decisions are made; to generate an action the strategy must be known, but the choice of a certain strategy probably depends on how good the different possible actions in that strategy are in the present situation.

The input space can be seen as divided into a finite number of subspaces, where for each subspace there is one best choice of strategy and the input-output transition function varies in a smooth way. The total input-output transition function of the system can be seen as a collection of continuous input-output transition functions, one for each strategy, see figure 1.

Now the problem of learning the total input-output transition function can be divided into several learning problems at different levels. At the first level, there are a number of smooth and continuous functions to be learned, one

for each strategy. For a given strategy an "ordinary" reinforcement learning problem is faced where projective mapping could be used.

At the second level, the best choice among the set of strategies is the learning task. The number of strategies are, however, finite, so at this level a memory mapping method could be efficient.

Hierarchical methods has successfully been used in information processing a long time, see e.g. Granlund and Knutsson [4].

2.1 The Hierarchical Credit Assignment Problem

A system that is learning to solve a problem at different levels faces a new kind of problem. It can, for instance, upon failure be difficult to know if the local input-output transition function that was used is bad or if the wrong strategy was chosen. I call this problem the *hierarchical credit assignment problem.* This problem can of course be seen as a special case of the structural credit assignment problem, but to separate two kinds of problems I will call the credit assignment problem within a level *structural* and the credit assignment problem between different levels *hierarchical.*

2.2 Representation of Two-level Knowledge

Consider a learning problem where the solution contains n different strategies. The knowledge of the solution to this problem could be represented with n strategy vectors $\{\mathbf{s}_1(\mathbf{x}), ..., \mathbf{s}_n(\mathbf{x})\}$, each vector representing one strategy. The norm and the direction of each vector are functions of the input vector $\mathbf{x}$. The direction of each action vector represents the action as a function of the input, and the norm of each vector is a measure of the expected performance of this input-output pair.

For a given input input vector $\mathbf{x}$ there is now a set of proposed actions, and for each action there is an estimate of its performance. The strategy to choose is the one that is represented by the strategy vector with the largest norm, and the action to output is represented by the direction of that vector.

This type of representation for hierarchical processing was suggested by Granlund 1978 [3]. It has been shown to be very useful as a representation of features at different levels of abstraction. It should also be useful in a hierarchical learning system, since it makes it possible to compare alternative strategies at the same level, and also enables a smooth change between strategies.

2.3 Using Adaptive Critic in Two-level Learning

The adaptive critic methods described in section 1.2 could be used in a two-level learning hierarchy. In these methods predictions of future reinforcement are made, and these predictions can be used to select strategy.

If, for each strategy, a prediction of future reinforcement and an action are calculated then the strategy with the highest prediction is selected and corresponding action is used as output. The update of the elements in the reinforcement association vectors $\mathbf{v}^i$ and the action association vectors $\mathbf{w}^i$ are made only for the chosen strategy and hence, the internal reinforcement

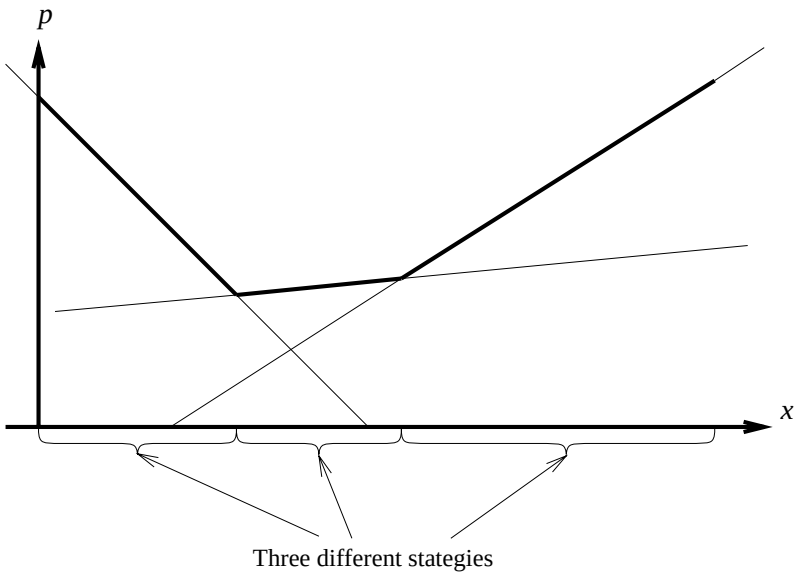

Figure 2: *The total (thick line) and local (thin lines) reinforcement prediction functions.*

signal $\hat{r}$ depends on which strategy was chosen. In this way the structural credit assignment problem is reduced considerably.

The system will now contain a number of reinforcement prediction functions. The total reinforcement prediction function $\mathbf{x} \rightarrow p$ is the maximum of the local reinforcement prediction functions. In figure 2 an example of the reinforcement prediction functions for three strategies and a one-dimensional input is showed.

3 The Badminton Player

As an experiment, a system that is playing "badminton" with itself was designed. For simplicity the problem was made one-dimensional and possible to solve with only two strategies. The position of the shuttlecock is represented by a variable x. The system can choose between two strategies, and both strategies contain actions that can change the value of x by adding the action value a, positive or negative, and a small noise, to x. The noise is added to prevent the system from becoming static, i.e. always generating the same actions. Now the actions must become dependent of x, since x is not completely deterministic.

The two strategies must alternate, i.e. the shuttlecock must pass the net between each action. It is, however, not a priori defined which strategy that shall act on a certain side.

The reinforcement signal to the system is zero except upon failure, when $r = -1$. Failure is the case when x does not change sign, or when $|x| > 0.5$.

3.1 The Algorithm

Both the predictions p of future reinforcement and the actions a are linear functions of the input variable x. There is one pair of reinforcement association

vectors $\mathbf{v}^i$ and one pair of action association vectors $\mathbf{w}^i, i = \{1, 2\}$. For each strategy i the predicted reinforcement is calculated as

$$p^i = N(\mu_p^i, \sigma_p), \tag{4}$$

where

$$\mu_p^i = v_1^i x + v_2^i, \tag{5}$$

and the action as

$$a^i = N(\mu_a^i, \sigma_a), \tag{6}$$

where

$$\mu_a^i = w_1^i x + w_2^i. \tag{7}$$

$N(\mu, \sigma)$ is the normal distribution with mean μ and variance σ^2. σ_p and σ_a are variable variance parameters that will be discussed later.

The system chooses the strategy c such that

$$p^c = \max_i \{p^i\}, \quad i \in \{1, 2\} \tag{8}$$

and outputs the corresponding action a^c.

The internal reinforcement signal at time $t + 1$ is calculated as

$$\hat{r}[t + 1] = r[t + 1] + \gamma \mu_p^{max}[t, t + 1] - p^c[t, t]. \tag{9}$$

This is in principle the same equation as equation 2, except that now there are two predictions at each time, and that there is a difference between the calculated prediction μ_p and the actual prediction p. $\mu_p^{max}[t, t+1]$ is the maximum predicted reinforcement calculated with the reinforcement association vector from time t and the input from time $t+1$. If $r = -1$ then $\mu_p^{max}[t, t+1]$ is set to zero. $p^c[t, t]$ is the prediction that caused the choice of the selected strategy.

Learning is accomplished by changing the weights in the reinforcement association vectors and the action association vectors. Only the vectors associated with the chosen strategy are altered.

The association vectors are updated according to the following rule:

$$\mathbf{w}^c[t + 1] = \mathbf{w}^c[t] + \alpha \hat{r}(a^c - \mu_a^c)\mathbf{x}, \tag{10}$$

and

$$\mathbf{v}^c[t + 1] = \mathbf{v}^c[t] + \beta \hat{r}\mathbf{x}, \tag{11}$$

where c denotes the strategy choice, α is a positive learning rate constant, and $\mathbf{x} = \begin{pmatrix} x \\ 1 \end{pmatrix}$.

3.1.1 Variance Parameters

As mentioned in the first section, stochastic search methods are often used in reinforcement learning. Here, this is accomplished by generating the predictions and actions at random from a normal distribution as shown in equations 4 and 6. The amount of search behaviour is controlled by the parameters σ_p and σ_a.

In many reinforcement algorithms found in the literature, the noise is set to a small but fix constant. However, at the beginning, when the knowledge of

the system is poor, a wide search behaviour would be desirable, while later on, when the system can produce better responses, the noise should become lower so that the precision increases. The modulation of the variance parameters is crucial for the learning performance of the algorithm.

In [5] Gullapalli suggested a variance that depends on the predicted reinforcement. That is logical, because the prediction can be seen as a measure of how sure the system is that the calculated output is good, and when the prediction of reinforcement is low, a large deviation from the intended output would probably not decrease the performance of the system very much anyway.

In the algorithm presented here there are two predictions of future reinforcement, one for each strategy, so the prediction associated to the chosen strategy should be used. In the calculation of σ_p, however, the choice of strategy is not yet made, and hence the largest prediction is used, since the strategy associated to this prediction *probably* will be chosen.

The reinforcement to the system is 0 or -1, so any reasonable predictions would lie between 0 and -1. At the time of learning, however, the predictions could of course occasionally be outside that interval.

I have, after some experiments, chosen the following equations for the variance parameters:

$$\sigma_p = \max\{0,\ -0.1 * \max(\mu_p^i)\}, \tag{12}$$

and

$$\sigma_a = \max\{0,\ -0.1 * p^c\}. \tag{13}$$

The first "max" in the two equations is to make sure that the variances do not become negative. The negative signs are there because of the (relevant) predictions being negative. In this way, the higher the prediction of reinforcement, the more precision in the output.

3.1.2 The Equalized State Problem

One problem that can occur with this algorithm is when both strategies prefer the same part of the input space. This means that the two reinforcement prediction functions predict the same reinforcement for the same inputs, and both strategies generate the same actions. I call this the *equalized state problem*.

Consider e.g. the two figures 3 and 4. Figure 3 describes a successful run of the system. The top graph displays the number of failures in the 100 latest steps, and it shows that that the system improves it behaviour from almost 100 % failure at the start to almost perfect behaviour after less than 2000 steps. The bottom graph displays the breakpoint between the two strategies, i.e the point where the predictions of future reinforcement for the two strategies are equal. As the graph displays, the the system has found the correct breakpoint, zero, after 500 steps.

Figure 4 shows a run that failed because the two reinforcement association vectors converged to the same values. Both strategies could generate perfect actions on the same side of zero, but neither strategy could handle an input on the other side. The performance remains at a level of about 70 % failure. When the reinforcement prediction functions has converged to become almost equal, a very small change in either function would cause a dramatic change

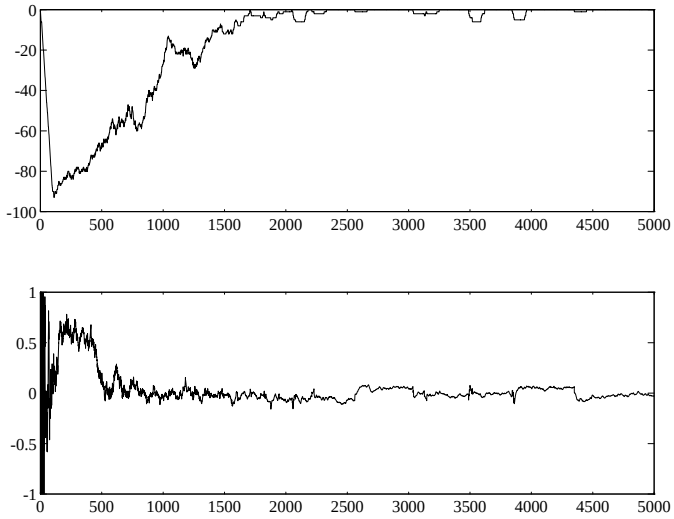

Figure 3: *A successful run of the system. The top graph displays the accumulated reinforcement over the 100 latest steps and the bottom graph displays the breakpoint between the two strategies.*

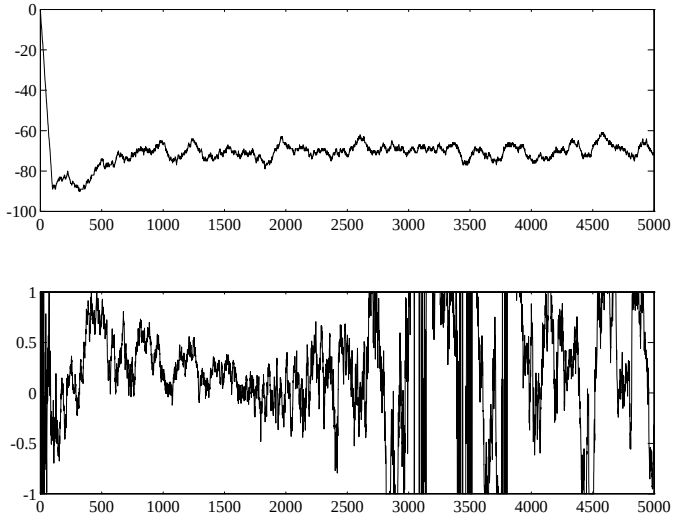

Figure 4: *A run that failed because the two reinforcement association vectors converged to the same values.*

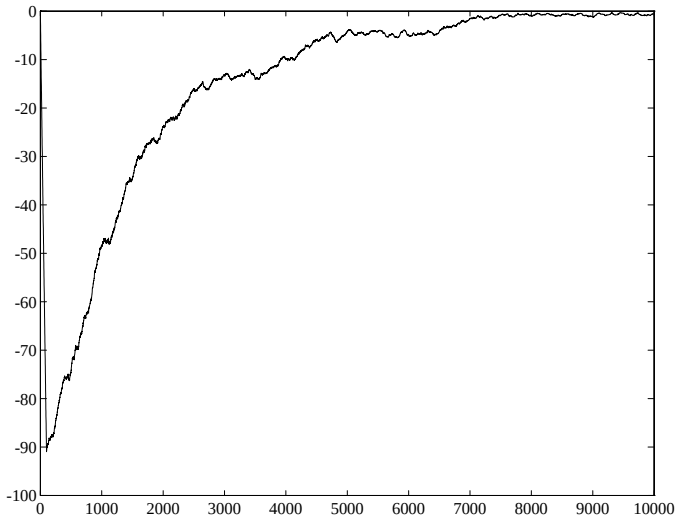

Figure 5: *Mean performance over 20 runs.*

in the breakpoint. This can be seen in th bottom graph where the breakpoint begins to oscillate after about 3000 steps.

This problem can of course be solved if the supervisor that generates the external reinforcement signal knows approximately where the breakpoint should be, and which strategy that should act on which side. The supervisor could then "punish" the system for selecting the wrong strategy by giving negative reinforcement. In general, however, this is not known, and the equalized state problem is a fact.

While there is no obvious general solution to the problem, it is at least possible to avoid it, since the equalized state is possible for the system to detect. If the change of the selected reinforcement association vector $\mathbf{v}^c$ is denoted $\delta\mathbf{v}$, the vectors $\mathbf{v}^i$ can be considered as converged when $\delta\mathbf{v}$ is sufficiently small for a number of steps. When this has happened the difference between $\mathbf{v}^1$ and $\mathbf{v}^2$ can be calculated. If this difference is small then the reinforcement prediction functions are (almost) the same, and the equalized state is reached. The algorithm can now stop and take proper measures.

In my simulations I simply let the algorithm start from the beginning, i.e. the association vectors $\mathbf{v}^i$ and $\mathbf{w}^i$ are reset, when the equalized state problem is detected.

3.2 Results

The algorithm was run 20 times, and each run lasted 10,000 time steps. The algorithm was restarted each time an equalized state was detected, but the time counter continued, so that the total length of the run was always the same.

Figure 5 shows the mean performance, i.e. the number of failures during 100 steps, over the 20 runs, and figure 6 shows the mean breakpoint between the two strategies. In figure 7 the performance for each run is plotted. Here one can see that in most runs the algorithm converged at about 2000 steps.

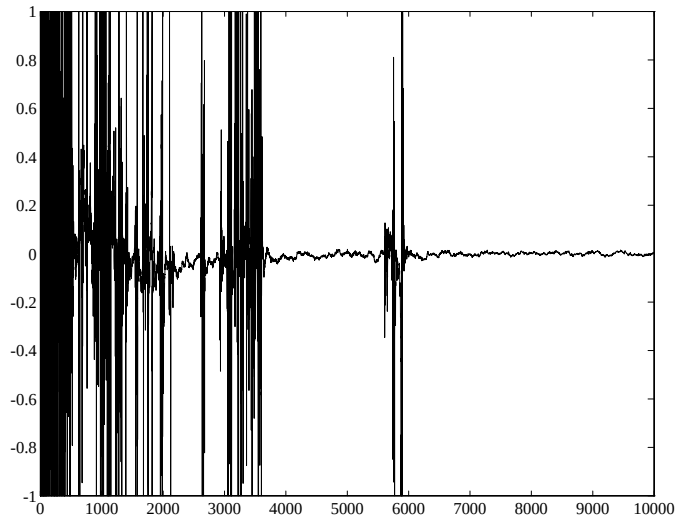

Figure 6: *Mean breakpoint over 20 runs.*

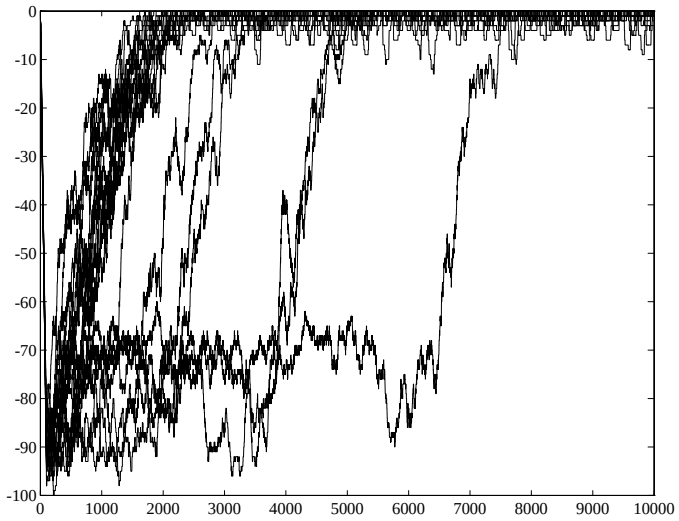

Figure 7: *The 20 individual runs.*

The worst run did not converge until after about 7500 steps but it was reset due to the equalized state problem several times, and the last reset was made after more than 5000 steps.

The algorithm has not been optimized with respect to convergence time. The convergence speed depends e.g. on the setting of the learning rate constants α and β and the modulation of the variance parameters σ^p and σ^a. These parameters has only been tuned to values that works reasonably well.

4 Conclusions and Discussion

In this paper I have suggested that a rather complicated, piecewise continuous input-output transition function can be seen as a discrete finite set of *strategies*, each containing a smooth continuous input-output transition function. The new problem of selecting strategy exists on a different level of abstraction, and the learning problem becomes hierarchical.

In reinforcement learning, adaptive critic methods could be used to handle the two-level learning problem. A new problem, the *hierarchical credit assignment problem*, is faced. Another problem, the *equalized state problem*, can occur if the approximate limits between the strategies not are known by the supervisor. This problem can, however, easily be detected by the system itself.

An example of an algorithm that solves a two level learning task for a one dimensional dynamic problem has been shown. The algorithm cold be used for problems of higher dimensions, at the expense of lower learning rate.

The algorithm presented used only two strategies. More than two strategies could of course be used in the same manner if needed, but there is a practical limit of the number of strategies. Since the reinforcement prediction functions are linear and global, the "outermost" prediction functions would be very steep, and the break points instable, if too many strategies were used. This problem could be solved with other reinforcement prediction functions of higher order, e.g. Gaussian functions.

The equalized state problem is a serious problem and more work will be done in finding methods to prevent the reinforcement prediction functions from converging to the same values.

References

[1] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Trans. on Systems, Man, and Cybernetics*, SMC-13(8):834–846, 1983.

[2] M. Borga and T. Carlsson. A Survey of Current Techniques for Reinforcement Learning. Report LiTH-ISY-I-1391, Computer Vision Laboratory, S–581 83 Linköping, Sweden, 1992.

[3] G. H. Granlund. In search of a general picture processing operator. *Computer Graphics and Image Processing*, 8(2):155–178, 1978.

[4] G. H. Granlund and H. Knutsson. Hierarchical processing of structural information in artificial intelligence. In *Proceedings of 1982 IEEE Confer-*

ence on Acoustics, speech and signal processing, Paris, May 1982. IEEE. Invited Paper.

[5] V. Gullapalli. A stochastic reinforcement learning algorithm for learning real-valued functions. *Neural Networks*, 3:671–692, 1990.

[6] C-S. Lin and H. Kim. CMAC-based adaptive critic self-learning control. *IEEE Trans. on Neural Networks*, 2(5):530–533, 1991.

[7] J. L. Musgrave and K. A. Loparo. Entropy and outcome classification in reinforcement learning control. In *IEEE Int. Symp. on Intelligent Control*, pages 108–114, 1989.

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

[9] Robert E. Smith and David E. Goldberg. Reinforcement learning with classifier systems. *Proceedings. AI, Simulation and Planning in High Autonomy Systems*, 6:284–192, 1990.

[10] R. S. Sutton. *Temporal credit assignment in reinforcement learning.* PhD thesis, University of Massachusetts, Amherst, MA., 1984.

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

[12] Chris Watkins. *Learning from delayed rewards.* PhD thesis, Cambridge University, 1989.

[13] P. J. Werbos. Consistency of HDP applied to a simple reinforcement learning problem. *Neural Networks*, 3:179–189, 1990.

[14] S. D. Whitehead, R. S. Sutton, and D. H. Ballard. Advances in reinforcement learning and their implications for intelligent control. *Proceedings of the 5th IEEE Int. Symposium on Intelligent Control*, 2:1289–1297, 1990.