A Survey of Current Techniques for Reinforcement Learning

Magnus Borga Tomas Carlsson

4 June 1992

Abstract

This survey considers response generating systems that improve their behaviour using *reinforcement learning*. The difference between *unsupervised learning, supervised learning,* and *reinforcement learning* is described. Two general problems concerning learning systems are presented; the *credit assignment problem* and the problem of *perceptual aliasing*. Notations and some general issues concerning reinforcement learning systems are presented. Reinforcement learning systems are further divided into two main classes; *memory mapping* and *projective mapping* systems. Each of these classes is described and some examples are presented. Some other approaches are mentioned that do not fit into the two main classes. Finally some issues not covered by the surveyed articles are discussed, and some comments on the subject are made.

Contents

1 Introduction **2**
- 1.1 The Credit-Assignment Problem 4
  - 1.1.1 Temporal Difference Methods 5
  - 1.1.2 Dynamic Programming in Reinforcement Learning . . . 6
- 1.2 Perceptual Aliasing 7

2 Reinforcement Learning **10**
- 2.1 Framework 11
- 2.2 Design 13

3 Memory Mapping **16**
- 3.1 The Pole-balancing Problem 17
- 3.2 The Cerebellar Model Articulation Controller (CMAC) 19

4 Projective Mapping **21**
- 4.1 Linear Systems 21
- 4.2 Nonlinear Projective Mapping 22
- 4.3 Back-propagation in RLS 23

5 Other Approaches **26**
- 5.1 The AI Approach 26
- 5.2 Stochastic Automata 26

6 Conclusions **28**

Chapter 1

Introduction

There is a wide class of systems that is called *response generating systems* [6]. It could be any type of system that generates responses to certain stimuli, from a door bell that responds with a signal when a button is pressed, to a sophisticated robot or an animal that performs a complicated interaction with its environment. However complicated the system may be, it needs rules for how to produce proper responses to stimuli relevant to the task the system is to perform.

Principally there are two ways to instruct a system how to produce responses. It can be done by *programming* or by *learning*. Programming means that the system is equipped with rules predefining responses to all meaningful stimuli. A learning system, on the other hand, has no or little predefined behaviour. Instead it governs knowledge about what to do in different situations by trial and error. When dealing with complex problems the programmer's burden will become unbearable, since foreseeing every possible situation becomes impossible. The learning system will of course also face a complex task, but building such a system need not be impossible. In the future, systems may be placed in environments that humans have no experience from. Then it is obvious that learning systems are called for.

Learning is a broad concept and can span from the adaptive updating of filter coefficients to the use of fragmented past experience to compile action strategies for new situations. This spectrum of possible interpretations is difficult to cover in one definition of learning. It can be illustrated by two definitions made by two mathematical learning theorists and two researchers in the field of learning automata. The mathematicians Bush and Mosteller [4]

consider any systematic change in behaviour as learning whether or not the change is adaptive, desirable for certain purposes or in accordance with any other such criterion. Narendra and Thathachar [15], two learning automata theorists, make the following definition of learning: "Learning is defined as any relatively permanent change in behaviour resulting from past experience, and a learning system is characterized by its ability to improve its behaviour with time, in some sense towards an ultimate goal".

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

- unsupervised learning
- supervised learning
- reinforcement learning

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. This is clearly a limitation of the generality of the system. Most systems of this type are only used to learn efficient representations of signals. The learning methods are for example Hebbian learning [10] that performs feature mapping or principal component analysis, and competitive learning (winner-take-all) that perform clustering or pattern classification. Unsupervised learning systems can be very fast, since each component changes its behaviour simultaneously independently of the other components in the network. Some of these methods show similarities to some mechanisms in biological systems.

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 [16]. 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. Another problem with this method is that it is very slow in multi-layer networks, since each layer has to wait for the error to be propagated from the

last layer. Although the mechanisms used for learning in biological systems are mostly unknown, this type of learning systems seem to show very few similarities to biological systems.

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 correct outputs for each input) but where it is quite easy to decide when the system has succeeded or failed in a task (e.g. the pole balancing problem described in section 3.1).

There are two important problems all learning systems have in common; the *credit-assignment* problem and the problem of *perceptual aliasing*.

1.1 The Credit-Assignment Problem

In a complex system that in some way is supposed to improve its performance, there is always a problem in deciding what part of the system that deserves credit or blame for the performance of the whole system. This is called the *credit-assignment problem* [13] or, to be more specific, the *structural* credit-assignment problem. In supervised learning the desired response is known and this problem can be solved, for instance with the back propagation algorithm in feed-forward neural nets [16].

The 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. E.g. consider the loosing team in a football game, that scores one point during the last seconds of the game. It would not be clever to blame the last scored point. This kind of problem is called credit assignment over time or *temporal* credit-assignment problem, investigated by Sutton in [19]. One solution to the temporal credit assignment problem in RLS is to supply the system with an internal reinforcement signal and let the system learn to improve its internal critic, the so called *adaptive critic method* [3].

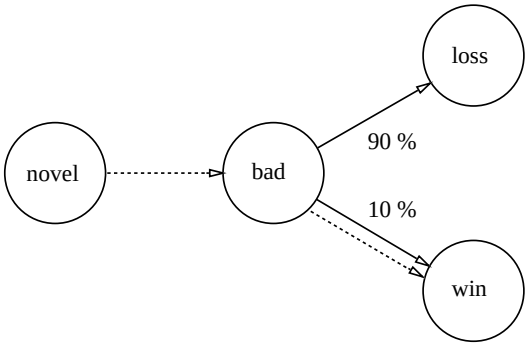

Figure 1.1: An example to illustrate the advantage with TD-methods. See the text on page 5.

1.1.1 Temporal Difference Methods

In [20] 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.

Let us look at an example. Consider a game, where a certain position has resulted in a loss in 90% of the cases and a win in 10% of the cases, see figure 1.1. This position is classified as a *bad* position. Now suppose that a player reaches a *novel* state (i.e. a state that has not been visited before) that inevitably leads to the *bad* state, and finally happens to lead to a win. If the player waits until the end of the game and only looks at the result, he would label the novel state as a *good* state, since it lead to a win. Most supervised learning methods would make such a conclusion. A TD method however, would classify the novel state as a *bad* state, since it leads to a bad state and the result *probably* will be a loss. It can make this classification without having to wait until the end of the game.

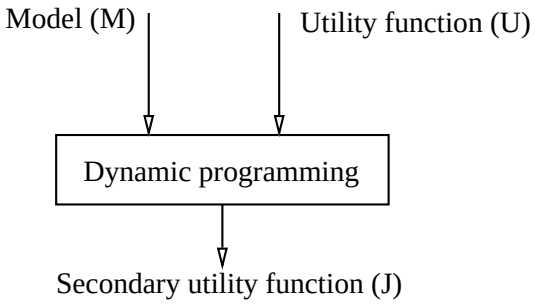

Figure 1.2: Dynamic programming.

Sutton has proved a convergence theorem for one TD method[1] that states that the predictions for each state asymptotically converge to the maximum-likelihood predictions of the final outcome for states generated in a Markov process, when presented to new data sequences. He has also proved that the predictions in this method converge to the maximum-likelihood estimates for such sequences under repeated training on the same set of data.

A TD method was first used by A. L. Samuel in a checkers-playing programme in 1959 [17]. Most reinforcement learning methods could use TD methods, but TD methods could also be used in supervised learning systems.

1.1.2 Dynamic Programming in Reinforcement Learning

There is a relationship between reinforcement learning and dynamic programming, as discussed by Whitehead in [26]. In [24] Werbos loosely defines adaptive-critic methods as methods that try to approximate dynamic programming. Adaptive-critic methods are used to provide the RLS with an internal reinforcement signal. These systems have the capability to improve their performance during a trial (i.e. between the reinforcement signals).

Dynamic programming is the process of generating a *secondary utility function (J)* given a *model* (M) of the environment and a *utility function*

[1] In this TD method the prediction p_k only depends on the following prediction p_{k+1} and not of later predictions. Other TD methods can take into account later predictions with a function that decreases exponentially with time.

(U) in such a way that when J is optimized in the short term, U becomes optimized in the long term,[2] see figure 1.2. In adaptive-critic methods the reinforcement signal is the utility function (U) and the internal reinforcement signal is the secondary utility function (J).

This relationship with dynamic programming has made more mathematical treatment possible (e.g. an optimality theorem concerning one class of reinforcement learning algorithms [22]).

1.2 Perceptual Aliasing

A learning system perceives the external world through a sensory subsystem and represents the set of external states S_E with an internal state representation set S_I. This set can, however, rarely be identical to the real external world state set S_E. Assuming a representation that completely describes the external world in terms of objects, their features and relationships will be unrealistic even for relatively simple problem settings. Also, the internal state is inevitably limited by the sensor system, which leads to the fact that there is a many-to-many mapping between the internal and external states. That is, a state $\bar{s}_e \in S_E$ in the external world can map into several internal states and, what is worse, an internal state $\bar{s}_i \in S_I$ could represent multiple external world states. This phenomenon, illustrated in figure 1.3, is called *perceptual aliasing* [25].

In the case when the learning system is an RLS, perceptual aliasing can cause the system to confound different external states that have the same internal state representation. This can cause the system to make wrong decisions. E.g. let the internal state $\bar{s}_i$ represent the external states $\bar{s}_e^a$ and $\bar{s}_e^b$ and let the system take an action $\bar{a}$. The expected reward for the decision to take the action $\bar{a}$ given state $\bar{s}$, denoted $(\bar{s}_i, \bar{a})$, is now estimated by averaging the rewards for that decision accumulated over time. If $\bar{s}_e^a$ and $\bar{s}_e^b$ occur approximately equally often and the actual accumulated reward for $(\bar{s}_e^a, \bar{a})$ is greater than the accumulated reward for $(\bar{s}_e^b, \bar{a})$ then the expected reward will be underestimated for $(\bar{s}_e^a, \bar{a})$ and overestimated for $(\bar{s}_e^b, \bar{a})$, leading to a nonoptimal decision policy.

[2] Sometimes the whole optimization process is referred to as dynamic programming, i.e. the optimization of J is included in the dynamic programming method.

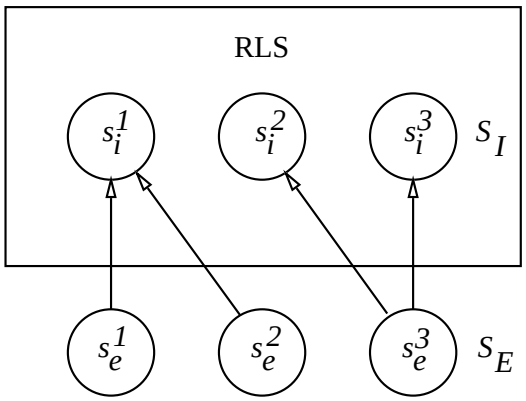

Figure 1.3: Perceptual aliasing.

There are cases when this is no problem and the phenomenon will be a feature instead. This happens if all decisions made by the RLS are *consistent*. The reward for the decision $(\bar{s}_i, \bar{a})$ then equals the reward for all corresponding actual decisions $(\bar{s}_e^k, \bar{a})$, where k is an index for this set of decisions. If the mapping between the external and internal worlds is such that all decisions are consistent, then it is possible to collapse a large actual state space onto a small one where situations that are invariant to the task at hand are mapped onto one single situation in the representation space. In general this will seldom be the case and perceptual aliasing will be a problem.

In [25] Whitehead presents a solution to the problem of perceptual aliasing for a restricted class of learning situations. The basic idea is to detect inconsistent decisions by monitoring the estimated reward error, since the error will oscillate for inconsistent decisions, as discussed above. When an inconsistent decision is detected, the system is guided (e.g. by changing its direction of view) to another internal state, uniquely representing the desired external state. In this way all actions will produce consistent decisions, see figure 1.4. The guidance mechanisms are not learned by the system. This is noted by Whitehead who admits that a dilemma is left unresolved:

> In order for the system to learn to solve a task, it must accurately represent the world with respect to the task. However, in order for the system to learn an accurate representation, it must know how to solve the task.

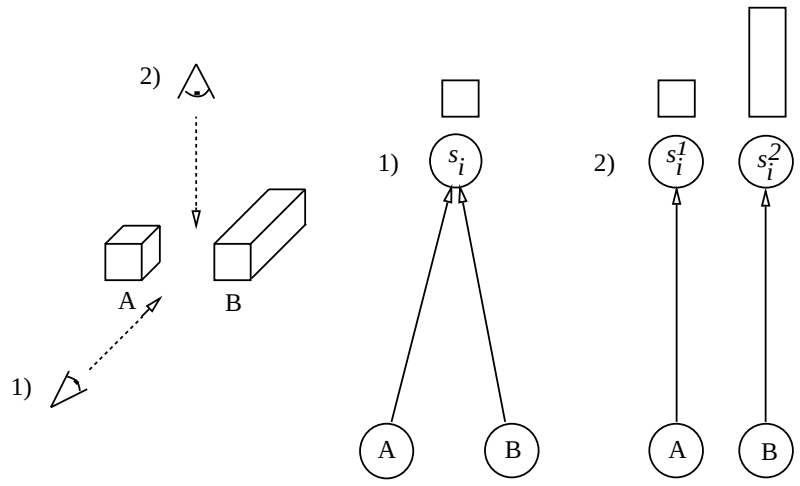

Figure 1.4: Avoiding perceptual aliasing by observing the environment from another direction.

Chapter 2

Reinforcement Learning

Reinforcement learning can be seen as a form of supervised learning, but there is one important difference; while a supervised learning system is supplied with the desired responses under its training, the reinforcement learning system is only supplied with some quality measure of the system's overall performance.

As an example, consider the procedure of training an animal. In general, there is no point in trying to explain to the animal how it should behave. The only way is simply to reward the animal when it does the right thing.

This, of course, makes a reinforcement learning system more general than a supervised learning system, since it can be trained on tasks where the exact desired responses for certain inputs are unknown, but where it is possible to obtain a qualitative measure of the systems performance. On the other hand some new problems are faced.

In [24] Werbos defines an RLS as "any system that through interaction with its environment improves its performance by receiving feedback in the form of a scalar reward (or penalty) that is commensurate with the appropriateness of the response". The goal for an RLS is simply to maximize this "reward", i.e. the accumulated value of the reinforcement signal r, see figure 2.1. In this way, r can be said to define the problem to be solved.

In reinforcement learning the feedback to the system contains no gradient information, i.e. the system does not know in what direction to search for a better solution.[1] Because of this, most RLS are supplied with some stochastic

[1]The problem can be solved by building a model of the environment that can be

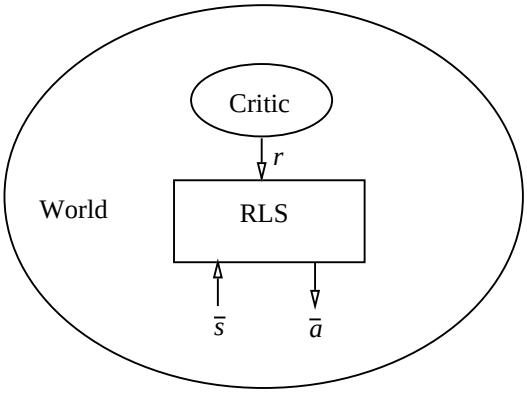

Figure 2.1: The Reinforcement Learning System in the external environment.

behaviour. This can be done by adding noise to the output as in [3]. Another way is by letting the output be generated from some probability distribution and moving the mean value to an optimal output and decreasing the variance as the output gets closer to the optimal value [9]. This stochastic behaviour can also help the system to avoid getting trapped in local minima.

This chapter first presents the notations used in the report and then some issues that concerns reinforcement learning in general.

In this report, reinforcement learning systems (RLS) are divided into two main classes, according to how the mapping from input to output is made. The classes are *memory mapping* and *projective mapping* and are discussed in chapter 3 and in chapter 4 respectively. Some other methods are mentioned in chapter 5.

2.1 Framework

The world, according to Whitehead [25], can be described by the tuple (S_E, A_E, W, R), where S_E and A_E are the sets of world states and physical actions on the world respectively. The state transition function W maps a state and an action into a new state, and the reward function R maps a state into a real valued reward. A natural extension of this function is to let it map both the action and the state in which the action is taken into a

differentiated with respect to the reinforcement signal, see section 2.2.

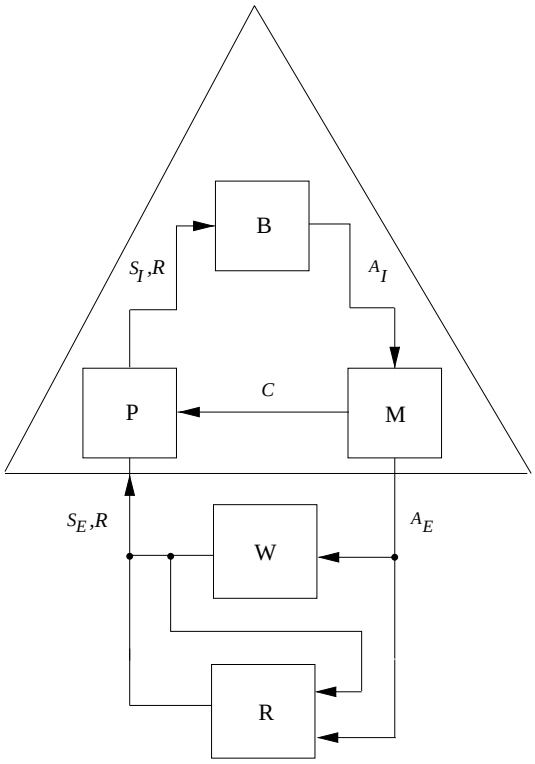

Figure 2.2: The subsystems in an RLS. P and M form the sensory-motor subsystem and B is the decision subsystem.

reward.

Again following Whitehead [25], the RLS can be divided into two subsystems; a sensory-motor subsystem and a decision subsystem, see figure 2.2. The purpose of the sensory-motor subsystem is threefold. It perceives the external world constituting a mapping from the set of external states S_E and the sensory-motor configuration C to the set of internal states S_I. On the motor side the RLS has a set of internal motor commands A_I, that either effect the way the world is perceived by changing the configuration C, or are translated into external actions A_E. The decision subsystem is the controller and it has access to the internal state set S_I which it maps into the set of internal actions A_I.

The objective of the RLS is to learn and implement a decision policy that maximizes some function $f(r)$, that accumulates the received reward. Here r

is the reinforcement, that is some measure of the overall performance of the system. This objective involves not only to implement an optimal mapping from S_I to A_I but also to control the systems sensory-motor subsystem in order to gain knowledge about the external world and to avoid perceptual aliasing.

The reinforcement-signal can either be produced outside the RLS (e.g. by a supervisor that evaluates the performance of the system) or be produced inside the RLS (i.e. the rules of what is "good performance" are built into the system). These two methods could of course be combined. Some fundamental rules could be built into the system, while the task-dependent rules could be used by some external supervisor. In fact, there must always be a built-in rule that tells the RLS to maximize $f(r)$.

2.2 Design

There are essentially three problems encountered when dealing with RLS:

- System architecture design
- Construction of a critic
- Design of rules for improving the behaviour

Many *system architecture designs* have been proposed, but the mainstream one is the feed-forward input-output net. However, in [2], Ballard suggests that it is unreasonable to suppose that peripheral motor and sensory activity is correlated in a meaningful way. Instead, it is likely that abstract sensory and motor representations are built and related to each other. Also, combined sensory and motor information must be represented and used in the generation of new motor activity. This implies a learning hierarchy and that learning occurs on different temporal scales [5, 6, 7, 8]. Representing regularities in the sensory information that are relevant to the behaviour of the system as invariants gives the system more time to solve problems and makes it possible for the system to associate new situations with old experience.

The *critic*, or *reinforcement*, must be capable of evaluating the overall performance of the system and be informative enough to allow learning. The

quality of the reinforcement signal is a critical factor, affecting the performance of any RLS, since this signal is the one and only piece of information available to the system about its performance.

In some cases it is obvious how to choose the reinforcement signal. E.g. in the pole balancing problem described in section 3.1, the reinforcement signal is chosen as a negative value upon failure and zero otherwise [3]. Many times, however, it is not that clear how to measure the performance, and the choice of reinforcement signal could affect the learning capabilities of the system.

The reinforcement signal should contain as much information as possible about the problem. The learning performance of a system could be improved considerable if a "pedagogical" reinforcement is used.

An example of this can be found in [9], where an RLS for learning real-valued functions is described. This system was supplied with two input variables and one output variable. In one case the system was trained on an XOR-task. Each input could be 0.1 or 0.9 and the output could be any real number between 0 and 1. The optimal output values was 0.1 and 0.9 according to the logical XOR-rule. At first the reinforcement signal was calculated as

$$r = 1 - |error|,$$

where *error* is the difference between the output and the optimal output. The system some times converged to wrong results, and in several training runs it did not converge at all. A new reinforcement signal was calculated as

$$r' = \frac{r + r_{task}}{2}.$$

The term r_{task} was set to 0.5 if the latest outputs on similar inputs were less than the latest outputs on dissimilar inputs and to -0.5 otherwise. With this reinforcement signal the system starts with trying to satisfy a weaker definition of the XOR-task, which is that the output should be higher for dissimilar inputs than for similar inputs. The learning performance of the RLS improved in several ways with this new reinforcement signal.

Should the critic in general be built on the action taken, the new state or on the action taken given the state before the action? No unanimous answer to this seemingly simple question is to be found among learning researchers, illustrating the fact that a common theoretical framework is lacking in this area.

Rules for improving the behaviour can be based on essentially two different strategies, either on differentiating a model or on active exploration.

Differentiating a model establishes the gradient of the reinforcement as a function of the model system parameters. The model can be known a priori and built into the system, or it can be learned and refined during the training of the system. Knowing the gradient of the error means knowing in which direction in the parameter space to search for a better performance. How far to go in the direction of the gradient is, however, not known, and the step length must be short enough to prevent the system from oscillating. The update rules in systems incorporating projective mapping and model differentiation often consist of adding a displacement to the weight vector in the direction of the gradient.

Active exploration is necessary when no model to help estimating the gradient is available. The only information is the reinforcement for the state-action pair. To be able to search the parameter space a stochastic behaviour component is needed. At first the system may respond in a stochastic way, a behaviour that is modified as the system happens to do something that is rewarded. The modification can be the updating of an action probability distribution, as is often the case in memory mapping systems.

Chapter 3

Memory Mapping

One approach to reinforcement learning is called *memory mapping*. This means that each input-vector is used as a pointer to a memory location that contains the response to that input, or contains a probability distribution from which the response is selected at random. In memory mapping the external state space must be quantized since the memory address is discrete and there is a finite number of memory locations. The quantization can be seen as a matter of system design, and be fix for a given problem or learned by a system having a fixed number of memory locations to play with [8]. The pure memory mapping approach would of course be impossible in a general system, since the memory size would become extremely large. However, in small specialized systems, e.g. in the pole-balancing problem mentioned below, this is no problem.

Extended stochastic learning automata [21] is a family name for one way of implementing the memory mapping concept. These automata make selections from a probabilistic and countable set of actions. The probabilities are then updated according to the evaluative feedback. An advantage of this method is that it can handle cases when two or more actions are equally good. One example is the control system by Musgrave and Loparo [14], presenting an RLS that learns by updating action probabilities $P(\bar{a}_j)$, where j is an index for the set of possible actions.

Another related system is that of Whitehead and Ballard [25]. The system accomplish learning by updating an action value function $Q(\bar{s}, \bar{a})$ and selecting the action having the largest action value for a given state.

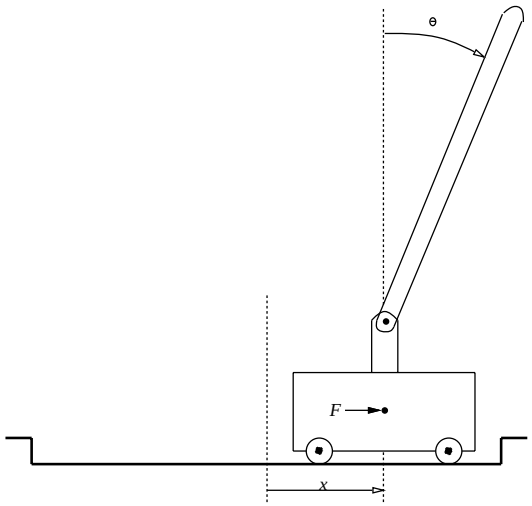

Figure 3.1: The cart-pole system.

3.1 The Pole-balancing Problem

As an example of memory-mapping systems, consider a system designed by Barto, Sutton and Anderson that solves a control problem called pole balancing [3]. The system to be controlled consists of a cart on a track with an inverted pendulum (a pole) attached to it, as illustrated in figure 3.1. The cart can move between two fixed endpoints of the track, and the pole can only move in the vertical plane of the cart and the track. The only way to control the movement of the cart is to apply a unit force F to it. The force must be applied at every time-step, so the controller can only chose the direction of the force (i.e. right or left). This means that the output $\bar{a}$ from the RLS is a scalar with only two possible values, F or $-F$. The state vector $\bar{s}$, that describes the cart-pole system, consists of four variables:

x cart position,

θ angle between the pole and the vertical,

$\dot{x}$ cart velocity, and

$\dot{\theta}$ angle velocity.

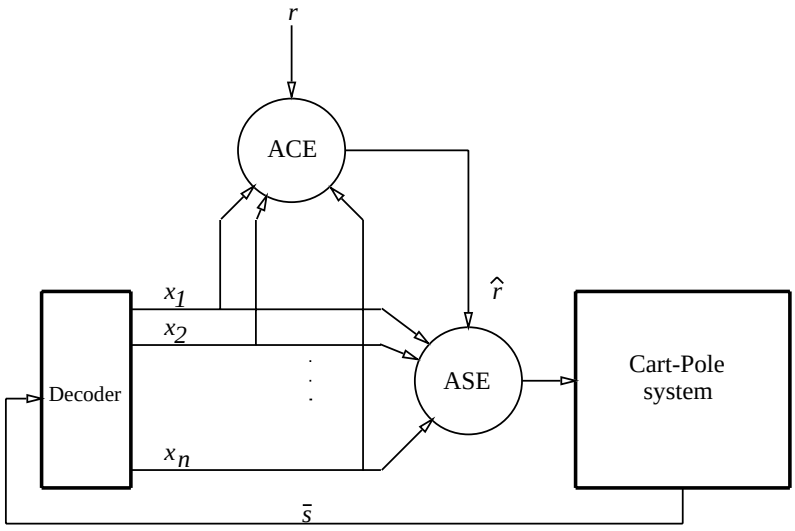

Figure 3.2: ACE and ASE in the pole-balancing problem.

The reinforcement signal r is given to the RLS as a failure signal (i.e. negative reinforcement) only when the pole falls or the cart hits the end of the track, otherwise $r = 0$. The task for the system is to avoid negative reinforcement by choosing the "best" action a for each state $\bar{s}$.

The four-dimensional state space is divided into 162 disjoint states by quantizing the state variables. Each of these states could be seen as a memory position that holds the "best" action for that state. The state vector $\bar{s}$ is mapped by a decoder into an internal state vector $\bar{x}$, in which one component is 1 and the rest are 0. This vector is used as a pointer to the current state.

This system consists of two parts: the ASE (associative search element) and the ACE (adaptive critic element), see figure 3.2. The ASE implements the mapping from the internal state vector $\bar{x}$ to the action a,

$$a(t) = f[\sum_{i=1}^{n} w_i(t)x_i(t) + noise(t)]$$

where f is the signum function and $noise(t)$ is the mean zero Gaussian distribution. This mapping is changed by altering the weights w_i in such a way that the internal reinforcement signal $\hat{r}$ is maximized. The ACE produces the internal reinforcement signal $\hat{r}$ as a mapping $\bar{x} \mapsto \hat{r}$. This mapping is chosen

so that the accumulated value of the reinforcement signal is maximized. The ACE will develop a mapping that gives a positive value of $\hat{r}$ when the state changes from a "dangerous" state to a state that is "safer" or vice versa. In this way the system can maximize the reinforcement r in the long term by maximizing the internal reinforcement $\hat{r}$ for each decision. For more details, see [3].

The ASE and ACE use weighted summation of the input vector and could therefore be used in the type of systems we call projective mapping if the decoder was removed.

3.2 The Cerebellar Model Articulation Controller (CMAC)

One problem with the memory-mapping model, where each state of the environment is represented by one memory location, is that the memory would grow to an enormous size as soon as the system get capabilities of any practical use. Another problem is that the system has to be trained for every single state, i.e. there is no way for the system to generalize between states.

One way of handling these problems in memory-mapping is called the *cerebellar model articulation controller* (CMAC). This method was first described by J. S. Albus in 1975 [1]. Instead of letting each state address one unique memory location only, this method allows each state to address a *set* of memory locations. The contents of these locations are then summed together to form the output. This means that the system is capable of a some sort of generalization, i.e. a new stimulus that the system has not been exposed to before, but which is similar to some other stimuli that is known to the system, would generate a response that is similar to the response to the known stimulus. This method also reduces the amount of memory that is needed, since each state is represented by a combination of several memory locations. Consider a system with 100 memory locations where each state is represented by 10 locations. This would yield $100!/90! \approx 6 \times 10^{19}$ states, meaning that a large state space is mapped into a smaller memory space.

The CMAC method has been used by C. Lin and H. Kim [12] in the same pole-balancing problem as described above. In this case the mapping between states and memory locations was done by hash coding. Due to interpolation

between states, the CMAC system had a higher learning speed, and a smaller memory could be used.

Chapter 4

Projective Mapping

The second main approach to reinforcement learning is *projective mapping*. The inner product between the input vector and the weight vector is calculated, which means that this type of systems can handle continuous input vectors. The learning is done by changing the weights in such a way that the desired mapping is obtained. This is the most common method used in neural networks. Each component in the response vector is a function (linear or non-linear) of such an inner product.

These systems do not demand any unreasonable amounts of memory, and the possibility of generalization by interpolation is obvious.

This chapter begins with describing the special case of linear systems. In section 4.2 an example of nonlinear, and the use of back-propagation in reinforcement learning is discussed in section 4.3.

4.1 Linear Systems

Consider a multi-layer feed-forward network with linear units, i.e. the output from each unit is calculated as

$$y_i = \bar{w}_i^T \bar{x}$$

which means that the output vector from the first layer is calculated as

$$\bar{y} = \mathbf{W}\bar{x}.$$

The output $\bar{u}$ from the second layer is calculated in the same way,

$$\bar{u} = \mathbf{V}\bar{y},$$

which can be written like

$$\bar{u} = \mathbf{VW}\bar{x} = \mathbf{Z}\bar{x}.$$

This means that any multi-layer feed-forward net with linear units could be implemented as a single-layer net, i.e. as a multiplication of the input vector with a matrix.

4.2 Nonlinear Projective Mapping

Gullapalli [9] proposes an RLS that computes a real-valued output as a function of a random activation. The system consists of four parts:

- Parameter computers
- Random number generator
- Output function
- Update rules

The *parameter computers* is a number of functions g_j, that compute distribution parameters p_j and take the inner product of the input state vector $\bar{x}$ and a weight vector $\bar{w}_j$ as their inputs:

$$p_j(t) = g_j(\bar{w}_j(t)^T \bar{x}(t))$$

These parameters are used by a *random number generator*, the distribution $d(p_1, .., p_n)$, to produce random activation values $a(t)$.

Finally the output is formed by feeding these activation values to an *output function* f. This function is often a nonlinear *squashing function* such as e.g the *logistic function*:

$$f(x) = \frac{1}{1 + e^{-x}}$$

Modification of the weight vectors $\bar{w}_j$ for the case where the distribution is the normal distribution $d(p_1, p_2) = N(\mu, \sigma)$, is accomplished by following the *update rules*:

$$\bar{w}_j(t+1) = \bar{w}_j(t) + \alpha \Delta_w(t) \bar{x}(t)$$

where α is a learning rate parameter and

$$\Delta_w(t) = (r(t) - \hat{r}(t)) \frac{a(t) - \mu(t)}{\sigma(t)}$$

denoting the reinforcement $r(t)$ and the expected reinforcement

$$\hat{r}(t) = \bar{v}(t)^T \bar{x}(t) + thresh(t).$$

Here the vector $\bar{v}(t)$ has a function corresponding to that of $\bar{w}$ in the calculation of $p_1(t) = \mu(t)$. Finally $p_2 = \sigma$ is a function of the expected reinforcement so that a large expected reinforcement shows up as a small deviation σ and vice versa. Gullapalli motivates the use of the fraction in the calculation of $\Delta_w(t)$ by reasoning as follows. If the system has received more reinforcement than expected, then it is desirable to move the mean closer to the current activation value by updating the mean in the direction of the fraction. On the other hand, if the reinforcement was lower than expected, the system should adjust its mean in the direction opposite to that of the fraction. Gullapalli claims that the above equations lead to such a behaviour.

4.3 Back-propagation in RLS

In [9] Gullapalli used $\Delta\bar{w}$, the update vector of the output layer, as an error estimate for back-propagation through a hidden layer. The system then becomes a hybrid between a reinforcement learning and supervised learning system, since the output layer faces a reinforcement learning task and the hidden layer faces a supervised learning task. He claims that $\Delta\bar{w}$ under certain conditions can be shown to be an unbiased estimate of the gradient of the reinforcement signal with respect to the activation in the hidden layer.

Williams [28] uses back-propagation to update the system parameters in all layers. He assumes that the RLS output is computed from a probability distribution $g_i(z, \bar{w}_i, \bar{x}_i) = P\{y_i = z | \bar{w}_i, \bar{x}_i\}$, where $\bar{w}_i$ is a weight vector, $\bar{x}_i$ is the input vector to the i:th unit and y_i is the output from the i:th unit. All the

weight vectors of the network constitute a matrix denoted W. If $\frac{\partial \ln g_i}{\partial w_{ij}}$ exists for all j, and the goal function to maximize is denoted $J(W) = E\{r|W\}$, then a criterion for a learning algorithm to improve the value of J, at least on average, is that the inner product $E\{\Delta W|W\} \cdot \nabla J(W) \geq 0$ with equality only when $\nabla J(W) = 0$. Williams denotes this the *hillclimbing property.*

As mentioned earlier, it is necessary to build an internal model of the environment if back-propagation is to be used for establishing the gradient of the reinforcement signal. Two different kinds of information is then back-propagated, i.e. model prediction errors to train the internal model and predicted reinforcement to train the original network.

In [27] Williams introduced a class of RLS that update their weights on the form:

$$\Delta w_{ij} = \alpha_{ij} \left(r - b_{ij}\right) e_{ij}$$

In back-propagation the direction to move in weight space is known but not how big the step in that direction should be. The *learning rate factor* α_{ij} can be considered as the step length and is supposed to be nonnegative and to satisfy some other conditions e.g to depend on the input x_i and to be small enough to avoid overshots in the estimation process. The *reinforcement baseline* b_{ij} is to be conditionally independent of y_i, given W and x_i. The adaptive critic element by Barto and Sutton [3] can be seen as adaptively updating this baseline or expected reinforcement. Lastly the *characteristic eligibility*, e_{ij}, is computed as $e_{ij} = \frac{\partial \ln g_i}{\partial w_{ij}}$. The eligibility can be looked upon as a measure of how receptive a weight is to credit or blame. A reinforcement algorithm on this form is shown to satisfy the previously mentioned hillclimbing property. This class of algorithms calls for back-propagation through the deterministic parts of the net to compute the eligibilities e_{ij}.

Furthermore, if the probability distributions g_i are continuous and on the form

$$g = \frac{1}{\sigma} h(\frac{y - \mu}{\sigma})$$

where μ and σ are arbitrary translation and scaling parameters respectively, then $\frac{\partial J}{\partial y}$ and $\frac{y-\mu}{\sigma} \cdot \frac{\partial J}{\partial y}$ can be shown to be unbiased estimates of $\frac{\partial J}{\partial \mu}$ and $\frac{\partial J}{\partial \sigma}$ respectively. This result implies that it is possible to back-propagate through all parts of the net, deterministic as well as stochastic, compensating with a factor $\frac{y-\mu}{\sigma}$ when computing $\frac{\partial J}{\partial \sigma}$.

Williams also make some comments on how to make algorithms for random number generators having separate control over their parameters. Using the update rule mentioned before with the reinforcement baseline representing the reinforcement in the past is said to lead to a reasonable learning behaviour.

The problem of learning temporal behaviour is also treated by Williams in [27] where the system is supposed to work for k time steps and then receive a reinforcement r. The update rule is extended also to incorporate time:

$$\Delta w_{ij} = \alpha_{ij} \left(r - b_{ij} \right) \sum_{t=1}^{k} e_{ij}(t)$$

Now the eligibilities e_{ij} not only depend on the input x_i but also on the time t. This extended learning algorithm is shown to possess the hillclimbing property mentioned earlier.

In the same article Williams points out the flexibility of the reinforcement baseline b_{ij}. This could be different for each unit in a network and be used for individual tailoring of credit assignment. He suggests that informational connections could be added to a network to receive signals not affecting the output but calculating the baseline of a unit.

Werbos [23] makes the noteworthy comment that these strategies leave out the crucial problem of maximizing some function $f(r)$, of the accumulated reinforcement. The established theorems talk about the expected next reinforcement $E\{r|W\}$ instead of the expected accumulated reinforcement $E\{f(r)|W\}$, which is the more interesting quantity.

Chapter 5

Other Approaches

Here some RLS designs are presented, that do not belong to the two previous described methods.

5.1 The AI Approach

Smith and Goldberg [18] have described an RLS that uses rules of type IF *condition* THEN *action* to produce the output from the system. The rulebase is modified by a genetic algorithm.

Lee and Berenji [11] applied the AI approach with IF-THEN rules on the pole-balancing problem described earlier. This system was much faster in learning the pole-balancing task than Barto's method. It could also adapt to changes in length and mass of the pole without any failure.

5.2 Stochastic Automata

In their survey [15], Narendra and Thathachar describe a stochastic automaton. The automaton is equipped with a set of states, a set of actions and a corresponding set of action probabilities. An action is chosen at random using the action probabilities, and the reinforcement from the environment is observed. Now the probabilities are updated based on the received reinforcement and a new action is taken. Note that the actions taken are independent of the state of the environment. No sensory information but the reinforce-

ment signal is considered, which is the main difference between stochastic automata and the extended stochastic automata mentioned earlier.

Chapter 6

Conclusions

In most cases, the adaptive critic has been used to solve the temporal credit-assignment problem by learning to predict the expected reinforcement for the different states that the system's environment may enter. Then the assumption is made that some external states are better than others. For instance, in Barto's pole-balancing problem it is obvious which states are good and which are not; the "safe" states, where the cart is in the middle of the track and the pole stands straight up are of course better than more "dangerous" states like those at the end of the track. In the general case, however, things are not quite that easy. The expected reinforcement almost certainly depends not only on the state of the external world, but also on the action taken by the system. The internal reinforcement signal should therefore be a function of both the state vector and the action vector, i.e.

$$\hat{r} = f(\bar{s}, \bar{a}).$$

The generation of reinforcement signal is a subject that in most cases has not been discussed. Whether the reinforcement signal is generated inside the system or by some external unit it, is still an interesting issue how it should be determined. As we have seen, the way of generating reinforcement can be crucial to the systems learning capabilities. The reinforcement should give some hint of whether or not the system is "on the right track" in solving the problem, even if the best solution is far away. Perhaps the reinforcement signal could be adaptive, so that it initially gives a great deal of credit for a relative moderate improvement, but get harder in its critic as the system becomes better.

The capabilities of generalization and association seem to be two important features in learning. Very few systems, however, have been investigated that use such features, and in most articles these capabilities are not even mentioned. Some times interpolation and extrapolation are described as generalization, but useful definitions of these two concepts are still wanted. The fact that a small distance between two state vectors only produces a small difference between the generated action vectors is often used as an argument for seeing projective mapping systems as associating systems. This is, however, not the only form of association. To be able to associate two parts of the curve outlined by the changing state vector may be even more important, since it reflects an ability to associate new situations with older ones. Once an association is made, it may be fruitful to test if it holds over a larger area by generalizing the correspondence to new areas and evaluate the hypothesis.

A problem to be encountered before any associations or generalizations of successful actions and states are possible, is having the system generate good responses. In the case of memory mapping without gradient information, it is necessary for the system to respond in a random manner, waiting for an action to be rewarded. This may take a long time if the parameter space is large and has many dimensions. No solution to this problem is to find among the references. For the class of projective mapping systems mentioned in [28], Williams has proved that they possess the hillclimbing property, meaning that the weight vectors are updated in a direction improving the expected reward. The measure is, however, a local one, and the problem of being trapped in a local extrema is not solved.

In this survey no practical applications of RLS have been found. No one seems to have been able to use these ideas in larger systems, e.g. in large multilayer neural networks. Unanimous design rules are lacking. Altogether, very little research has been done in this area compared to i.e. supervised learning. However, reinforcement learning seems to be a more general and natural way of learning than supervised learning and therefore better suited for autonomous learning systems.

Bibliography

[1] J. S. Albus. A new approach to manipulator control: The cerebellar model articulation controller (CMAC). *Trans. of the ASME Journal of Dynamic Systems, Measurements, and Control*, pages 220–227, 1975.

[2] D. H. Ballard. *Computational Neuroscience*, chapter Modular Learning in Hierarchical Neural Networks. MIT Press, 1990. E. L. Schwartz, Ed.

[3] 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.

[4] R. R. Bush and F. Mosteller. *Stochastic models for learning.* New York: Wiley, 1955.

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

[6] G. H. Granlund. Integrated analysis-response structures for robotics systems. Report LiTH–ISY–I–0932, Computer Vision Laboratory, Linköping University, Sweden, 1988.

[7] G. H. Granlund and H. Knutsson. Hierarchical processing of structural information in artificial intelligence. In *Proceedings of 1982 IEEE Conference on Acoustics, speech and signal processing*, Paris, May 1982. IEEE. Invited Paper.

[8] G. H. Granlund and H. Knutsson. Compact associative representation of visual information. In *Proceedings of The 10th International Conference on Pattern Recognition*, 1990. Report LiTH–ISY–I–1091, Linköping University, Sweden, 1990, also available as a reprint.

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

[10] D. O. Hebb. *The Organization of Behavior.* Wiley, New York, 1949.

[11] C.-C. Lee and H. R. Berenji. An intelligent controller based on approximate reasoning and reinforcement learning. *Proccedings on the IEEE Int. Symposium on Intelligent Control*, pages 200–205, 1989.

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

[13] M. L. Minsky. *Computers and thought*, chapter Steps towards artificial intelligence, pages 406–450. McGraw–Hill, 1963. E. A. Feigenbaum and J. Feldman, Eds.

[14] 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.

[15] K. S. Narendra and M. A. L. Thathachar. Learning automata - a survey. *IEEE Trans. on Systems, Man, and Cybernetics*, 4(4):323–334, 1974.

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

[17] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM J. Res. Develop.*, 3(3):210–229, 1959.

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

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

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

[21] R. S. Sutton and P. J. Werbos. *Neural networks for control*, chapter General Principles, pages 38–42. Cambrdige Mass, MIT Press, 1990. W. T. Miller, Ed.

[22] C. Watkins. *Learning from delayed rewards*. PhD thesis, Cambridge University, 1989.

[23] P. J. Werbos. Backpropagation: Past and future. In *IEEE Int. Conf. on Neural Networks*, pages 343–353, July 1988.

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

[25] S. D. Whitehead and D. H. Ballard. Learning to perceive and act. Technical report, Computer Science Department, University of Rochester, 1990.

[26] 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.

[27] R. J. Williams. A class of gradient-estimating algorithms for reinforcement learning in neural networks. In *IEEE First Int. Conf. on Neural Networks*, pages 601–608, June 1987.

[28] R. J. Williams. On the use of backpropagation in associative reinforcement learning. In *IEEE Int. Conf. on Neural Networks*, pages 263–270, July 1988.