

# A Binary Competition Tree for Reinforcement Learning

Magnus Borga    Hans Knutsson

LITH-ISY-R-1623  
1994

# A Binary Competition Tree for Reinforcement Learning

**Magnus Borga**      **Hans Knutsson**  
magnus@isy.liu.se      knutte@iys.liu.se

Computer Vision Laboratory  
Linköping University  
S-581 83 Linköping  
Sweden

August 25, 1994

## **Abstract**

A robust, general and computationally simple reinforcement learning system is presented. It uses channel a representation which is robust and continuous. The accumulated knowledge is represented as a reward prediction function in the outer product space of the input- and output channel vectors. Each computational unit generates an output simply by a vector-matrix multiplication and the response can therefore be calculated fast. The response and a prediction of the reward are calculated simultaneously by the same system, which makes TD-methods easy to implement if needed. Several units can cooperate to solve more complicated problems. A dynamic tree structure of linear units is grown in order to divide the knowledge space into a sufficiently number of regions in which the reward function can be properly described. The tree continuously tests split- and prune criteria in order to adapt its size to the complexity of the problem.

# 1 Introduction

The aim with our research is to develop efficient learning algorithms for autonomous systems (e.g. robots). Such a system is supposed to act in a closed loop with its environment, i.e. the system's output will influence its input. In this case a supervised learning algorithm would not be very useful since it would be impossible for a teacher to foresee all possible situations that can occur in an interaction with a realistic environment. This means that the system must be able to learn from its own experiences and therefore a *reinforcement learning system* is necessary.

An autonomous system must also be able to handle a very large amount of input data in order to use inputs from different sensors (e.g. visual, tactile, etc.). The dimensionality of the output will probably be smaller but the total dimensionality of the decision space (i.e. input *and* output) will be very large and therefore a structure that can handle problems of this size must be used.

Furthermore, different types of learning should be possible, depending on the available information about the problem solution, i.e. if the correct responses are known it should be possible to give them to the system, while the system should look for solutions by itself if they are not supplied by a teacher.

In this paper a new learning system is presented that is an attempt to handle these demands. It can handle supervised learning, reinforcement learning and even infrequent rewards in a similar manner. The large decision space is handled by using competing experts that divide the space into local regions where the dimensionality of the problem can be reduced. The experts are arranged in a binary tree structure that makes competition simple and the search for the best expert efficient. Furthermore, it uses a biologically inspired channel representation that has several advantages compared to ordinary scalar representations.

## 2 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 for each input in a training set. The most used algorithm is back-propagation [?]. 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. For instance, if an autonomous system (for instance a robot) is to

learn how to act in a realistic environment it will be impossible for a teacher to foresee all possible situations and therefore the system must learn from its own experiences.

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 [?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?].

## 2.1 TD-methods and Adaptive Critic

When a system acts in a dynamic environment it can be difficult to decide which of the actions in a sequence that are most responsible for the result since the feedback to the system may occur infrequently or, as in many cases, a long time after the responsible actions have been taken. 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.

In [?] 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. 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 the system describes its environment as a number of discrete states and that for each state  $s_k$  there is a value  $p_k$  that is an estimate of the expected 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 [?] 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 system is divided into two parts; one that learns the output and another that learns the reinforcement predictions.

## 3 Channel Representation

It is a widely accepted fact that the internal representation of information may play a decisive role for system performance in general and for learning systems in particular. The most obvious and probably most commonly used representation is the one that is used to describe physical entities, like e.g. a scalar  $t$  for temperature or a three dimensional vector  $\mathbf{p} = (x \ y \ z)$  for a position in  $\mathbb{R}^3$ . This is, however, not the only way and in some cases not even a very good way to represent information. For example consider an orientation in  $\mathbb{R}^2$  which can be represented by an angle  $\varphi \in [-\pi, \pi]$ . This is not a very suitable representation since there exists a discontinuity at  $\pi$  that causes trouble e.g. when averaging [?].

Another way to represent information is the so called *channel representation* [?]. In this representation a set of channels is used, where each channel is sensitive to some specific feature value in the total signal, for example a certain temperature  $t_i$  or a certain position  $p_i$ . The coding is designed so that the channel has its maximum value (e.g. 1) when the

feature and the channel are exactly tuned to each other, and decreases to zero in a smooth way as the feature and the channel become less similar. This implies that each channel can be seen as a response from a filter that is tuned to some specific feature value. This is similar to the magnitude representation proposed by Granlund [?].

In the example above the orientation in  $\mathbb{R}^2$  could be represented with a set of channels evenly spread out on the unit circle. If three channels are used, with the form

$$c_k = \cos^2 \left( \frac{3}{4}(\varphi - p_k) \right) \quad (1)$$

where  $p_1 = \frac{2\pi}{3}, p_2 = 0$  and  $p_3 = -\frac{2\pi}{3}$ , the orientation can be represented with the channel vector  $\mathbf{c} = (c_1 \ c_2 \ c_3)^T$  which has a constant norm for all orientations and contains no discontinuities.

The channel representation is in fact quite a natural way to handle information. It is inspired by biological systems where the nerve cells are sensitive to some specific feature value for which they responds strongly [?, ?, ?]. The channel representation is also a robust way to handle information. If for example a one-dimensional physical entity is represented with a set of sufficiently overlapping channels it can be reconstructed at least approximately even if one or a few channels fail to operate. This is not possible if a scalar representation is used.

Another appealing feature of the channel representation is that it seem possible to use rather simple operations on a set of channels to implement a function that would need much more complicated operations if ordinary scalars were used. To see this, consider any continuous non-linear function  $y = f(x)$ . If  $x$  is represented by a sufficiently large number of channels  $c_k$  of a suitable form, then the output  $y$  is simply a weighted sum of the input channels,  $y = \sum w_k c_k$ .

It is not obvious how for instance a scalar value  $x$  is to be coded into channels. According to the description above of a channel it should be positive and have its maximum for one specific value of  $x$  and it should decrease smoothly to zero from this maximum. In addition, to be able to represent all values of  $x$  in an interval there must be overlapping channels in this interval. It is also convenient if the norm of the channel vector is constant. One channel form that fulfills these demands is

$$c_k = \begin{cases} \cos^2 \left( \frac{\pi}{3}(x - k) \right) & |x - k| < \frac{3}{2} \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

This set of channels have not only a constant norm, but also a constant squared sum of its first derivatives. This means that a change  $\Delta x$  in  $x$  always gives the same change  $\Delta \mathbf{c}$  in  $\mathbf{c}$  for any  $x$ . Of course, not only scalars can be coded into channel vectors. Any vector  $\mathbf{v}$  in a vector space  $V$  can be mapped into the direction of a channel vector  $\mathbf{q}$  in a larger vector space  $Q$  that is spanned by the channels. For more details about this type of channel representation see Nordberg et al [?].

## 4 A New Reinforcement Learning Method

As mentioned earlier (section 3) the channel representation makes it possible to realize a rather complicated function as a linear function of the input channels. If supervised learning is used one would simply train a weight vector  $\mathbf{w}_i$  for each output channel  $q_i$ . This could e.g. be done in the same way as in ordinary feed forward one layer neural networks, i.e. to minimize some error function

$$\epsilon = \sum_i |q_i - \tilde{q}_i|^2, \quad (3)$$

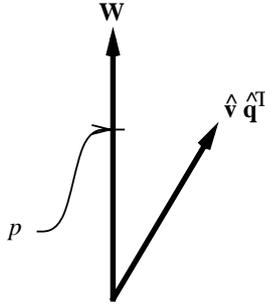


Figure 1: *The reward prediction  $p$  for a certain stimulus-response pair  $(\mathbf{v}, \mathbf{q})$  viewed as a projection onto  $\mathbf{W}$  in  $V \otimes Q$ .*

where  $\tilde{q}_i$  is the correct output for each output channel supplied by the teacher. This means for the whole system that a matrix  $\mathbf{W}$  is trained so that a correct output vector is generated as

$$\mathbf{q} = \mathbf{W}\mathbf{v}. \quad (4)$$

In reinforcement learning, however, the correct output is not known and the only feedback is a scalar  $r$  that is a measure of the performance of the system. But the reward signal is a function of the stimuli vector  $\mathbf{v}$  and the response vector  $\mathbf{q}$ , at least for an environment that is not completely stochastic. If this function can be represented in the system, then the best response for each stimulus can be found.

#### 4.1 Learning the Reward Function

If the reward function is continuous we can approximate it in some interval with a linear function of the outer product between the normalized input- and output vectors. This approximation will be used as a prediction  $p$  of the reward and is calculated as

$$p = \langle \mathbf{W} \mid \hat{\mathbf{q}}\hat{\mathbf{v}}^T \rangle, \quad (5)$$

where  $\langle \cdot \mid \cdot \rangle$  denotes the scalar product, see figure 1. The space  $V \otimes Q$  will be called the *decision space*. The normalization implies that the algorithm does not differ between input vectors with the same orientation but of different length. This is however not a problem if the channel representation described in section 3 is used. In fact any vector  $\mathbf{v}$  in an  $(n-1)$ -dimensional vector space can be mapped into the orientation of a unit-length vector in an  $n$ -dimensional space [?].

Now  $\mathbf{W}$  can be trained in the same manner as in supervised learning but with the aim to minimize the error function

$$\epsilon = |r - p|^2. \quad (6)$$

Let each triple  $(\mathbf{v}, \mathbf{q}, r)$  of stimulus, response, and reward denote an *experience*. Consider a system that has learned a number of experiences. How should a proper response be chosen by the system? The prediction  $p$  in equation 5 can be rewritten as

$$p = \hat{\mathbf{q}}^T \mathbf{W} \hat{\mathbf{v}} = \langle \hat{\mathbf{q}} \mid \mathbf{W} \hat{\mathbf{v}} \rangle. \quad (7)$$

The choice of  $\hat{\mathbf{q}}$  that gives the highest predicted reward is obviously the  $\hat{\mathbf{q}}$  that lies in the same direction as  $\mathbf{W}\mathbf{v}$ . Now, if  $p$  is a good prediction of the reward  $r$  for a certain stimulus  $\mathbf{v}$  that choice of  $\hat{\mathbf{q}}$  would be the one that gives the best reward. An obvious choice of the response  $q$  is then

$$\mathbf{q} = \mathbf{W}\mathbf{v}. \quad (8)$$

This equation together with equation 7 gives the prediction as a function of the stimulus vector, i.e.

$$p = |\mathbf{W}\hat{\mathbf{v}}|. \quad (\text{See appendix ??}) \quad (9)$$

## 4.2 The Learning Algorithm

The training of the matrix  $\mathbf{W}$  is a very simple algorithm. For a certain experience  $(\mathbf{v}, \mathbf{q}, r)$  the prediction  $p$  should in the optimal case equal  $r$ . This means that the aim is to minimize the error in equation 6. The desired weight matrix  $\mathbf{W}'$  would yield a prediction

$$p' = r = \langle \mathbf{W}' | \hat{\mathbf{q}}\hat{\mathbf{v}}^T \rangle. \quad (10)$$

The direction to search for  $\mathbf{W}'$  can be found by calculating the derivative of  $\epsilon$  with respect to  $\mathbf{W}$ . From equations 6 and 7 we get the error

$$\epsilon = |r - \langle \mathbf{W} | \hat{\mathbf{q}}\hat{\mathbf{v}}^T \rangle|^2 \quad (11)$$

and the derivative becomes

$$\frac{d\epsilon}{d\mathbf{W}} = 2(r - p)\hat{\mathbf{q}}\hat{\mathbf{v}}^T. \quad (12)$$

This means that  $\mathbf{W}'$  can be obtained by changing  $\mathbf{W}$  a certain amount  $a$  in the direction  $\hat{\mathbf{q}}\hat{\mathbf{v}}^T$ , i.e.

$$\mathbf{W}' = \mathbf{W} + a\hat{\mathbf{q}}\hat{\mathbf{v}}^T. \quad (13)$$

Equation 10 now gives that

$$r = p + a(\hat{\mathbf{v}}^T\hat{\mathbf{v}}) \quad (\text{See appendix ??}) \quad (14)$$

which gives

$$a = r - p. \quad (15)$$

To make the learning procedure stable, it is common to take only a small step in the gradient direction. The update rule therefore becomes

$$\mathbf{W}' = \mathbf{W} + \alpha(r - p)\hat{\mathbf{q}}\hat{\mathbf{v}}^T \quad (16)$$

where  $\alpha$  is a “learning speed factor” ( $0 < \alpha \leq 1$ ).

## 4.3 Bootstrapping

In the beginning, the system suffers from a more or less total lack of knowledge of what responses give high rewards. This is simply due to the fact that the system may never have tried a good response for the present stimulus. The only thing for the system to do in these cases is to generate random output and use the following reward to increase its knowledge about the problem. This is a kind of bootstrapping mechanism and a standard procedure in reinforcement learning.

The simplest way to produce this behaviour is to add a noise to the output signal. The noise should be large in the beginning when the system has poor knowledge about the problem and should decrease as the knowledge increases. The trivial way to do this would be to let the noise level be a monotonically decreasing function of time. A more sophisticated method is to use the predicted reward to decide the noise level. This seems like a more natural way for the system to work. If the system predicts a high reward, it should mean that the system is rather sure of how to generate a good response and, opposite, if it predicts a low reward this should mean that the response is not very reliable.

Consequently, a high predicted reward should give a low noise level and a low prediction should give a high noise level.

There are at least two obvious advantages with this approach compared to the time dependent noise. The first is that the noise decreasing rate does not have to be predetermined, but will be dependent on the problem. The second is that the noise level can be different for different stimuli, since the system at a given instance probably have different amounts of knowledge about different parts of the problem space.

#### 4.4 Similarities to TD-methods

The description above of the learning algorithm assumed a reward signal as a feedback to the system for each single decision (i.e. stimulus-response pair). This is, however, not necessary. Since the system produces a prediction of the following reward this prediction can be used to produce an internal reward signal in a similar way as in the TD-methods described in section 2.1. This is simply done by using the next prediction as a reward signal when an external reward is lacking. In other words

$$\hat{r} = \begin{cases} r & r > 0 \\ \gamma p[t+1] & r = 0 \end{cases} \quad (17)$$

where  $\hat{r}$  is the internal reward that replaces the external reward  $r$  in the algorithm described above,  $p[t+1]$  is the next predicted reward, and  $\gamma$  is a prediction decay factor ( $0 < \gamma \leq 1$ ) that makes the predicted reward decay as the distance from the actual rewarded state increases. This means that the system can handle dynamic problems with infrequent reward signals.

In fact, this system is more suited to use TD-methods than the earlier methods mentioned in section 2.1 since those systems had to use a separate subsystem to calculate the predicted reward. With the algorithm suggested here, this prediction is calculated by the same system as the response.

#### 4.5 Competitive Reinforcement Learning

In a reinforcement learning system there is a problem with the limited feedback. When the system receives the reward (or critic) from the teacher it can be difficult for the system to decide what part or parts of the system that is responsible for the rewarded action. This is called the *structural credit assignment problem*.

One way to deal with the structural credit assignment problem is to divide the whole system into several subsystems and for each timestep let one of the subsystems decide the response of the whole system. That subsystem is then the one that takes the credit or blame when receiving the feedback from the teacher. This is similar to the competing experts presented by Jacobs, Jordan, Nowland, and Hinton [?] but here used in a reinforcement learning system.

Now, how should the system select which of the subsystems that should generate the response? Since each subsystem calculates not only a potential response but also a prediction of the following reward the subsystem with the highest predicted reward should be chosen.

It is not certain that one of the subsystems will become the best for all possible stimuli. On the contrary, if different subsystems specializes on different parts of the problem they would together be able to solve a more complicated problem than each subsystem itself could solve. This implies that a rather complicated stimuli-response function could be implemented and learned by a sufficiently large number of rather simple

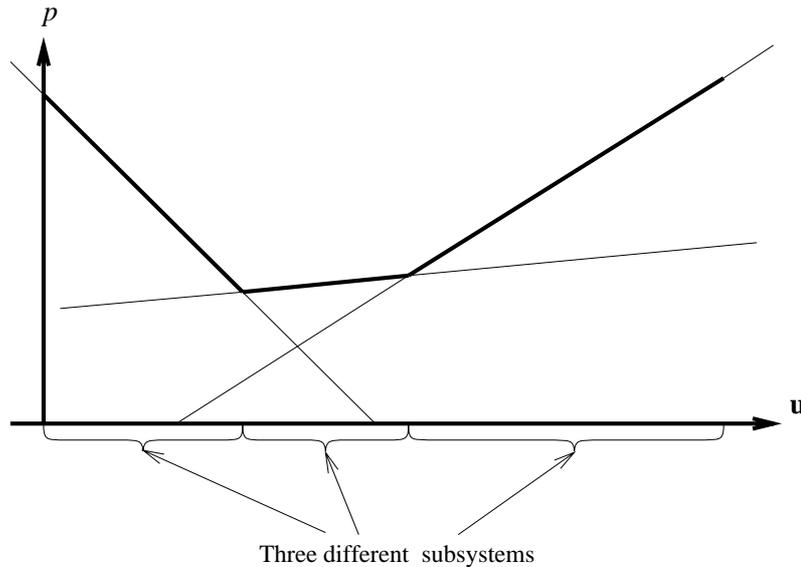


Figure 2: *The global (thick line) and local (thin lines) reward prediction functions.*

computing units if the units can predict their performance for each stimulus. The total (global) reward prediction function will then be the maximum of the subsystems (local) prediction functions as showed in figure 2

If several systems of the kind described in section 4.2 are used as subsystems in a larger system the subsystem with the highest reward prediction is chosen to generate the response. When the reward signal is received it is then the selected subsystem’s weight matrix  $\mathbf{W}_i$  that is modified according to equation 16. If an internal reward signal is used to handle infrequent rewards as described in section 4.4 the next predicted reward ( $p[t+1]$  in equation 17) is the prediction made by the subsystem that is the “winner” at  $t+1$ , i.e. the maximum of the predicted rewards. This may not be the same subsystem that uses the prediction as internal reward to update its weight matrix, but that does not change the theory since it is the global prediction function that is to be learned by the system.

## 5 The Tree

One way to arrange the competing subsystems is to use a binary tree structure (see figure 3). This structure has several advantages. An obvious advantage is that if a large number of subsystems is used the search time to find the winner will be of order  $\mathcal{O}(\log_2 n)$  where  $n$  is the number of experts, while it will be of order  $\mathcal{O}(n)$  to search in a list of experts. Another feature is that the subdivision of the decision space will be simple since only two subsystems will compete in each node.

### 5.1 Generating responses

In section 4 we described the basic functions (i.e. response generation and calculation of predicted reward) of one subsystem or *node* as it will be called when used in a tree structure. Here we will discuss some issues in the response generation that are dependent or caused by the tree architecture. We will consider a fixed tree that is already adapted to the solution of the problem. The adaptive growing of the tree will be described in section 5.2.

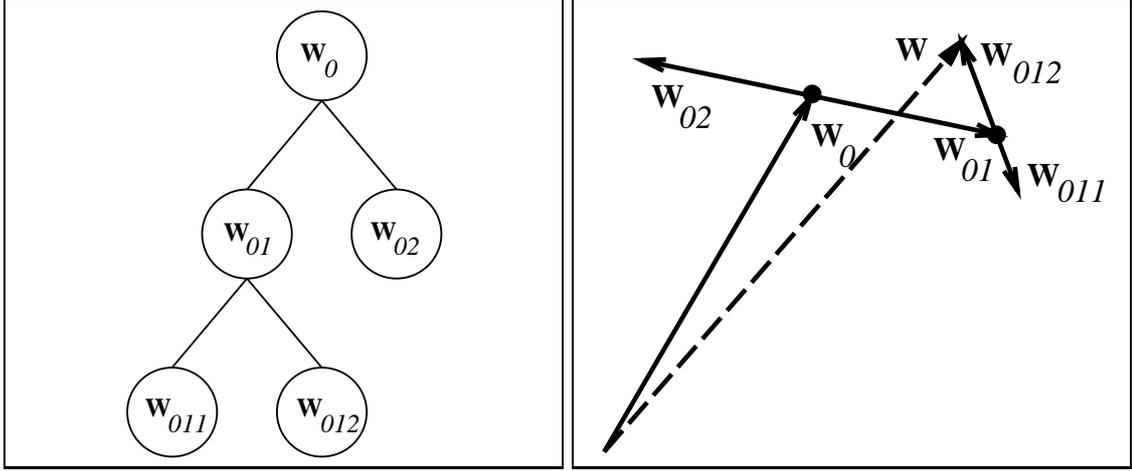


Figure 3: **Left:** An abstract illustration of a binary tree. **Right:** A geometric illustration of the weight matrices in a tree structure (representing three linear functions). Split nodes are marked with a dot ( $\bullet$ ).

Consider the tree in figure 3. The root node is denoted by  $\mathbf{W}_0$  and its two children  $\mathbf{W}_{01}$  and  $\mathbf{W}_{02}$  respectively and so on. A node without children will be called a *leaf node*. The weight matrix  $\mathbf{W}$  used in a node will be the sum of all matrices from the root to that node. In figure 3 node 012, for example, will use the weight matrix  $\mathbf{W} = \mathbf{W}_0 + \mathbf{W}_{01} + \mathbf{W}_{012}$ . Two nodes with the same father will be called *brothers*. Two brothers always have opposite directions since their father lies in the “center of mass” of the brothers (as will be further explained in section 5.2).

Now, consider a tree with the two solutions  $\mathbf{W}_A = \mathbf{W}_0 + \mathbf{W}_{01}$  and  $\mathbf{W}_B = \mathbf{W}_0 + \mathbf{W}_{02}$ . An input vector  $\mathbf{v}$  can generate one of the two output vectors  $\mathbf{q}_A$  and  $\mathbf{q}_B$  as

$$\mathbf{q}_A = \mathbf{W}_A \mathbf{v} = (\mathbf{W}_0 + \mathbf{W}_{01}) \mathbf{v} = \mathbf{W}_0 \mathbf{v} + \mathbf{W}_{01} \mathbf{v} = \mathbf{q}_0 + \mathbf{W}_{01} \mathbf{v} \quad (18)$$

and

$$\mathbf{q}_B = \mathbf{W}_B \mathbf{v} = (\mathbf{W}_0 + \mathbf{W}_{02}) \mathbf{v} = \mathbf{W}_0 \mathbf{v} + \mathbf{W}_{02} \mathbf{v} = \mathbf{q}_0 + \mathbf{W}_{02} \mathbf{v} \quad (19)$$

respectively, where  $\mathbf{q}_0$  is the solution of the root node. Since  $\mathbf{W}_{01}$  and  $\mathbf{W}_{02}$  will be of opposite directions we can write  $\mathbf{W}_{02} = -\alpha \mathbf{W}_{01}$ , where  $\alpha$  is a positive factor, and hence equation 19 can be written as

$$\mathbf{q}_B = \mathbf{q}_0 - \alpha \mathbf{W}_{01} \mathbf{v}. \quad (20)$$

Adding equations 18 and 20 gives

$$\mathbf{q}_0 = \frac{\mathbf{q}_A + \mathbf{q}_B - (1 - \alpha) \mathbf{W}_{01} \mathbf{v}}{2} \quad (21)$$

Now, if  $\mathbf{W}_A$  and  $\mathbf{W}_B$  have equal masses (i.e. the solutions are equally good and equally common) then  $\alpha = 1$  and

$$\mathbf{q}_0 = \frac{\mathbf{q}_A + \mathbf{q}_B}{2}, \quad (22)$$

i.e.  $\mathbf{q}_0$  will be the average of the two output vectors. If  $\mathbf{W}_A$  is better or more common (i.e. more likely a priori to appear) than  $\mathbf{W}_B$  then  $\alpha > 1$  ( $\mathbf{W}_0$  is closer to  $\mathbf{W}_A$ ) and  $\mathbf{q}_0$  will approach  $\mathbf{q}_A$ . Equation 18 also implies that the response can be generated in a hierarchical manner starting with the coarse and global solution of of the root and then refining the response gradually by just adding to the present output vector the new vectors that are

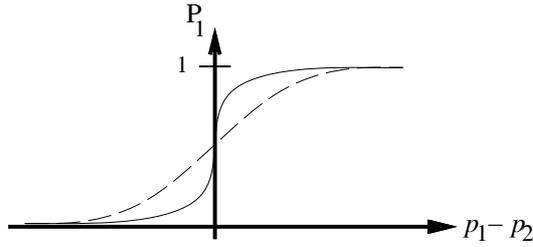


Figure 4: Illustration of equation 23 for two different cases of  $\sigma_i$  and  $n_i$ . The black line shows a case where the variation is smaller and/or the number of data is larger than in the case shown by the broken line.

generated when traversing the tree. In this way an approximate solution can be generated very fast since this solution don't have to wait until the best leaf node is found.

The climbing of the tree in search of the best leaf node can be described as a recursive procedure.

1. Start with the root node:  $\mathbf{W} = \mathbf{W}_0$
2. Let the current matrix be the father:  $\mathbf{W}_f = \mathbf{W}$
3. Select one of the children  $[\mathbf{W}_{f1}, \mathbf{W}_{f2}]$  as winner.
4. Add the winner to the father:  $\mathbf{W} = \mathbf{W}_f + \mathbf{W}_{f, \text{winner}}$
5. If  $\mathbf{W}$  is not a leaf node goto 2.
6. Generate response:  $\mathbf{q} = \mathbf{W}\mathbf{v}$ .

This scheme does not describe the hierarchical response generation mentioned above. This is however easily implemented by moving the response generating step into the loop and adding each new output vector to the previous.

Now, how should the winner be selected? Well, the most obvious way is to always select the child with the highest prediction  $p$ . This could, however, cause some problems. Of reasons explained in section 5.2 the winner will be selected in a probabilistic way. Consider two children with predictions  $p_1$  and  $p_2$  respectively. The probability  $P_1$  of selecting child 1 as winner is

$$P_1 = \frac{1}{2} \left[ \operatorname{erf} \left( \frac{p_1 - p_2}{\sqrt{\frac{\sigma_{high}^2}{n_{high}} + \frac{\sigma_{low}^2}{n_{low}}}} \right) + 1 \right], \quad (23)$$

Where  $\sigma_{high}$  and  $\sigma_{low}$  are the variances of the rewards for the choice of the child with highest prediction and the lowest prediction respectively.  $n_{high}$  and  $n_{low}$  are the sample sizes of these estimates. When the two predictions are equal the probability for each child to win is 0.5 (see figure 4). When the variances decreases and the sample sizes increases the significance increases in the hypothesis that it is better to chose the node with the highest prediction as winner than to chose at random. This leads to a sharpening of the error function and a decrease in the probability of selecting 'wrong' child. This function is similar to the test of hypothesis on the means of two normal distributions using the *paired t-test* statistics. Here we have used the normal distribution instead of the t-distribution<sup>1</sup> to be able to use the error function.

<sup>1</sup>The effect of this simplification is probably not important since both distributions are unimodal and symmetric with their maximum at the same point. The exact shape of the distribution is of less importance.

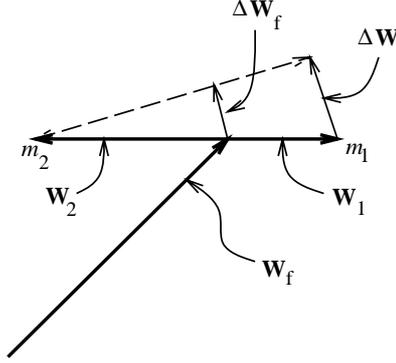


Figure 5: A geometric illustration of the update vectors of a father node and its two child nodes.

## 5.2 Growing the tree

Since the size of the tree in general can not be known in advance we will have to start with a single node and grow the tree until a satisfying solution is obtained. Hence, we start the learning by trying to solve the problem with a single linear function. This node will now try to approximate the optimal function as good as possible. If this solution is not good enough (i.e. a low average reward is obtained) the node will be split. To find out when the node has converged to a solution (optimal or not optimal) a measure of consistency of the changes of  $\mathbf{W}$  is calculated as

$$c = \left| \frac{\sum_k \gamma^k \Delta \mathbf{W}_k}{\sum_k |\gamma^k \Delta \mathbf{W}_k|} \right|, \quad (24)$$

where  $\{\gamma : 0 < \gamma \leq 1\}$  is the decay factor that makes the measure more sensitive to recent changes than older ones. These sums are accumulated continuously during the learning process. Note that the consistency measure is normalized so that it does not depend upon the step size in the learning algorithm.

Now, if a node is split two new nodes will be created, each with a new weight matrix. The nodes will be selected with a probability that depends on the predicted reward. (See equation 23). Consider the case where the problem can be solved with two linear models. The solution developed by the first node (the root) will converge to an interpolation of the two linear models. This solution will be in between the two optimal solutions in the decision space ( $V \otimes Q$ ). If the node is split into two child nodes, they will find their solutions on each side of the father's solution. If the child nodes start in the same position as their father in the decision space (i.e. they are initialized with zeros) the competition will be very simple. As soon as one child becomes closer to one solution the other child will be closer to the other solution.

The mass  $m$  of a node is defined as the mean reward obtained by that node over all decisions made by that node *or* its brother, i.e. a node that often loses in the competition with its brother will get a low mass (even if it gets high rewards when it *does* win). The mass of a node will indicate the nodes share of the total success of itself *and* its brother. This means that the masses of a winner node and its brother are updated as

$$m_{winner} := m_{winner} + \gamma(r - m_{winner}) \quad (25)$$

and

$$m_{loser} := m_{loser} + \gamma(0 - m_{loser}) = (1 - \gamma)m_{loser} \quad (26)$$

where  $\gamma$  is a memory decay factor. When a node's matrix and mass are updated the father will be updated too so that it remains in the center of mass of its two children. In this way the father will contain a solution that is a coarser, more global approximation of the solutions of its two children. Consider figure 5 and suppose  $\mathbf{W}_1$  is the winner child and  $\mathbf{W}_2$  is the loser. One experience gives:

- $\Delta\mathbf{W}$ : The total change of the vectors according to the update rule in equation 16 in section 4.
- $\Delta m_1$ : The change in mass of the winner node
- $\Delta m_2$ : The change in mass of the loser node

We have three prerequisites:

- $\mathbf{W}_f$  should remain in the center of mass of its children.  $\Rightarrow$

$$\begin{aligned} \mathbf{W}_f &= \frac{m_1(\mathbf{W}_f + \mathbf{W}_1) + m_2(\mathbf{W}_f + \mathbf{W}_2)}{m_1 + m_2} = \mathbf{W}_f + \frac{m_1\mathbf{W}_1 + m_2\mathbf{W}_2}{m_1 + m_2} \Rightarrow \\ & m_1\mathbf{W}_1 + m_2\mathbf{W}_2 = 0 \end{aligned} \quad (27)$$

- The loser should not be moved.  $\Rightarrow$

$$\Delta\mathbf{W}_f + \Delta\mathbf{W}_2 = 0 \quad (28)$$

- The total change of the winner node should be  $\Delta\mathbf{W}$ .  $\Rightarrow$

$$\Delta\mathbf{W}_f + \Delta\mathbf{W}_1 = \Delta\mathbf{W}. \quad (29)$$

These prerequisites implies that when  $\Delta\mathbf{W}$  has been calculated the father should be altered according to

$$\Delta\mathbf{W}_f = \frac{\Delta\mathbf{W}m'_1 + \mathbf{W}_1\Delta m_1 + \mathbf{W}_2\Delta m_2}{m'_1 + m'_2} \quad (\text{See appendix ??}) \quad (30)$$

where  $m'$  is the new mass, i.e.  $m' = m + \Delta m$ . If  $\mathbf{W}_f$  is not the root node the same calculation will be made at the next level with  $\Delta\mathbf{W}_f$  as the new  $\Delta\mathbf{W}$ . Hence the changes of the weight matrices will be propagated up the tree.

If a node is split accidently and that node's children converge to the same solution the node itself will also contain this solution. If this is the case the two redundant children could be pruned away. To detect this situation a significance measure  $s$  of the use of two children is calculated. Since the system sometimes chose the child with the higher and sometimes the child with the lower prediction the distributions of the rewards for these two cases could be estimated. From this statistic the significance can be calculated as

$$s = \frac{1}{2} \left[ \operatorname{erf} \left( \frac{(1 - \epsilon)\mu_{high} - \mu_{low}}{\sqrt{\frac{\sigma_{high}^2}{n_{high}} + \frac{\sigma_{low}^2}{n_{low}}}} \right) + 1 \right], \quad (31)$$

where  $\mu_{high}$  and  $\mu_{low}$  are the estimated mean rewards for the choice of the child with high and low prediction respectively,  $\sigma$  and  $n$  are the corresponding variances and sample sizes respectively and  $\epsilon$  is a threshold. Equation 31 gives a significance measure  $s$  of the of the hypothesis that  $\mu_{high}$  is  $\epsilon$  better than  $\mu_{low}$  (relatively). If the significance is low, i.e.

it does not matter which child is chosen, the solutions are obviously very similar to each other and the child nodes can be pruned away.

There is another reason for selecting the winner in a probabilistic manner. Just after a node has been split there is a risk that one child gets a prediction function that is larger than the brother's function for *all* input. This is in particular a risk if the father is split too early so that the father and one child will leave the other child behind. If a deterministic selection of the winner is made in such a case the child that is left behind will never get a chance to find a solution and it will be a dead end. The brother would then converge to the same solution as the father. This would not necessary affect the final solution but it would give a redundant level in the tree which costs memory space and computing time. By using a probabilistic choice this problem is avoided.

Using this tree is computationally very simple. The response generation is of the order  ${}^2\log n$  multiplications of an  $M \times N$  matrix with an  $M$ -dimensional vector, where  $n$  is the number of nodes,  $M$  is the dimension of the input vector and  $N$  is the dimension of the output vector. The learning process takes another  ${}^2\log n$  additions of  $M \times N$  matrices.

## 6 Experiments

In this section some preliminary results are presented. These are not intended to demonstrate the ultimate capacity of these algorithms but rather to illustrate the principles. There are still several practical problems to deal with before the ideas can be used in any "real" situations. In the first subsection the use of channel representation will be illustrated. In the next subsection the use of a tree structure is illustrated on the same problem.

### 6.1 The Use of Different Number of Channels

In the first experiments the use of different number of channels in the representation of the input variable is demonstrated. We have used three different representations (see figure 6). The first one uses two channels to represent the input  $x$ :

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} \cos\left(\frac{\pi}{80}x\right) \\ \cos\left(\frac{\pi}{80}(x-40)\right) \end{bmatrix}, \quad 0 \leq x \leq 40 \quad (32)$$

The second representation uses five channels with a  $\cos^2$  shape:

$$v_i = \begin{cases} \cos^2\left(\frac{\pi}{30}(x-x_i)\right) & \text{if } \frac{\pi}{30}|x-x_i| < \frac{\pi}{2} \\ 0 & \text{otherwise} \end{cases} \quad (33)$$

where  $0 \leq x \leq 40$  and  $x_i = 10i, i \in \{0..4\}$ . The last one uses nine channels with a  $\cos^2$  shape:

$$v_i = \begin{cases} \cos^2\left(\frac{2\pi}{30}(x-x_i)\right) & \text{if } \frac{2\pi}{30}|x-x_i| < \frac{\pi}{2} \\ 0 & \text{otherwise} \end{cases} \quad (34)$$

where  $0 \leq x \leq 40$  and  $x_i = 10i, i \in \{0..9\}$ . Note that the sum of the squares of the channels is *one* in these three representations, i.e. the norm of the channel vectors are constant, for all  $x$  except at the lowest and highest values in the  $\cos^2$  representations. The output  $y$  was represented by two channels in all three cases:

$$\mathbf{q} = \begin{bmatrix} q_1 \\ q_2 \end{bmatrix} = \begin{bmatrix} \cos\left(\frac{\pi}{80}y\right) \\ \cos\left(\frac{\pi}{80}(y-40)\right) \end{bmatrix}, \quad 0 \leq y \leq 40 \quad (35)$$

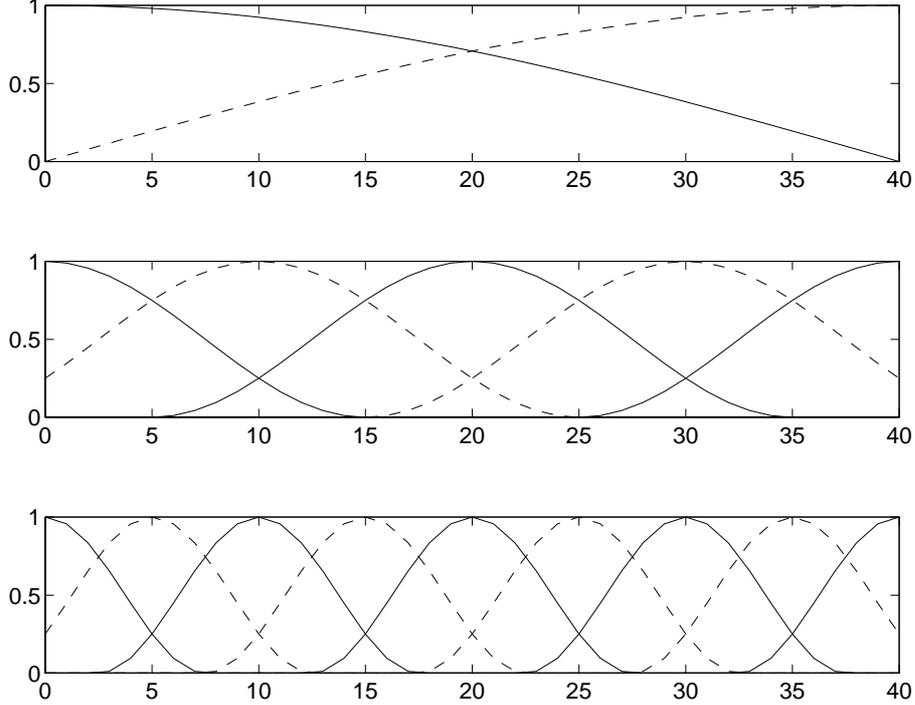


Figure 6: *The input scalar represented by two channels (top), five channels (middle) and nine channels (bottom). For clarity, every second channel is drawn with a broken line.*

The system described in section 4 was trained on a fairly simple but nonlinear problem using these three representations of the input variable  $x$ . Note that no tree structure was used here. Only one single linear system was used. The goal for the system was to learn a nonlinear function  $y = f(x)$  that contained a discontinuity:

$$y = \begin{cases} 20 & \text{for } 0 \leq x \leq 20 \\ 10 & \text{for } 20 < x \leq 40 \end{cases} \quad (36)$$

See the top left plot in figure 7. Random  $x$ -values was generated from a uniform distribution on the interval  $0 \leq x \leq 40$  and a reward signal  $r$  was calculated as

$$r = \max \left[ 0, \left( 1 - \frac{y - y_{correct}}{2} \right)^2 \right] \quad (37)$$

where  $y$  is the output from the system decoded into a scalar and  $y_{correct}$  is the desired output according to equation 36. The plots in figure 7 illustrates the predictions made by the system for all combinations of input and output as iso-curves according to equation 5 in section 4. The maximum prediction for each input is indicated with a dotted line and this is the response that the system would choose. Note that the channel representation enables the linear system  $\mathbf{q} = \mathbf{W}\mathbf{v}$  to implement a nonlinear function  $y = f(x)$  and the more channels that are used the more accurate is the approximation of the optimal function as well as the approximation of the reward function.

## 6.2 The Use of a Tree Structure

In the following experiment the tree structure described in section 5 was used. Each node consists of a linear system with two input channels and two output channels like the first system described in the previous subsection.

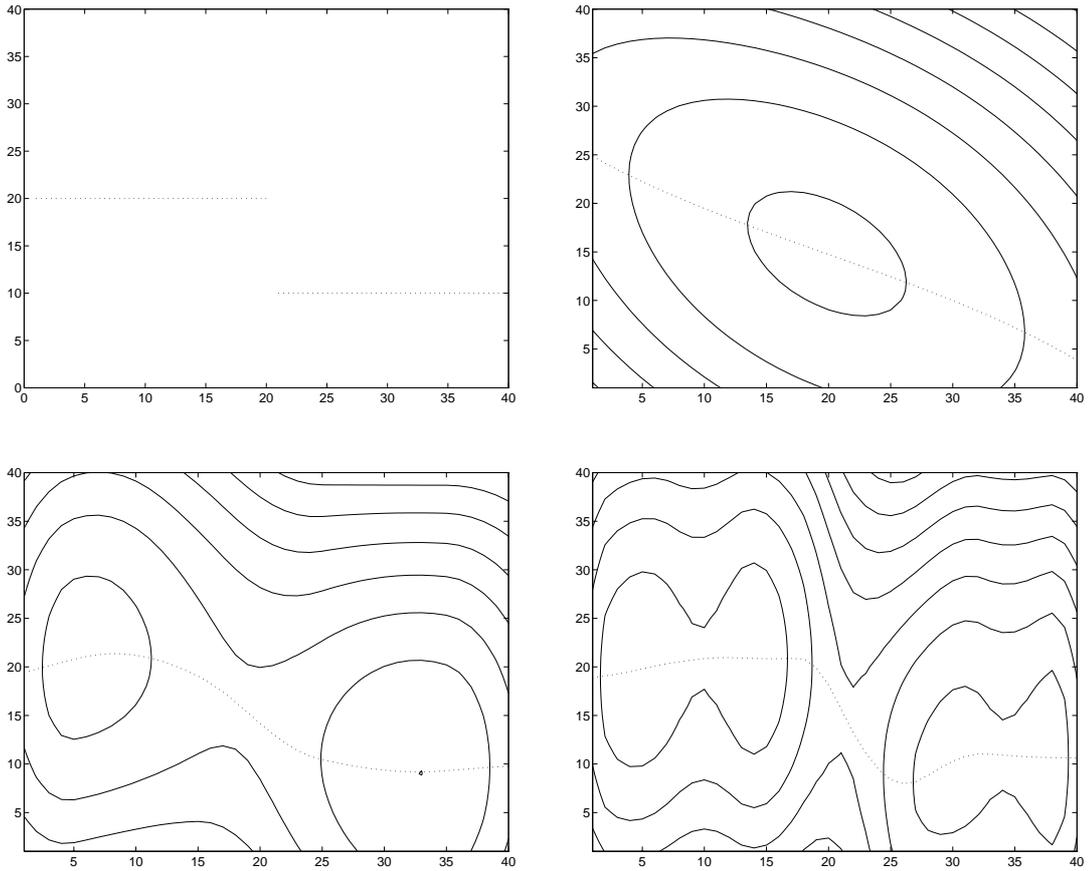


Figure 7: *The top left figure displays the optimal function. The other three figures illustrates the predictions as iso-curves with the maximum prediction at the dotted line for two, five and nine channels in top right, bottom left and bottom right image respectively.*

The system was trained on the same problem as in the previous subsection (equation 36). The input variable was generated in the same way as above and the reward function was the same as in equation 37. Each iteration consists of one input, one output and a reward signal. The result is plotted in figure ?? . It is obvious that this problem can be solved with a two-level tree and that is what the algorithm converged to. The discontinuity explains the fact that this solution with only two channels is better than the solutions in the previous experiments with more channels but with only one level.

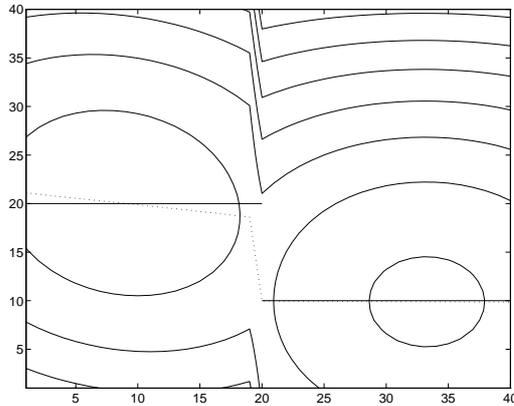


Figure 8: *The result using a tree structure. The predictions are illustrated by iso-curves with the maximum prediction at the dotted line. The optimal function is drawn as a solid line.*

### 6.3 The XOR-problem

The aim of this experiment is to demonstrate the use of the tree structure in a problem that is impossible to solve even approximately on one single level, no matter how many channel are fused. The problem is to implement a function similar to the Boolean XOR-function. The input consists of two parameters,  $x_1$  and  $x_2$ , that each are represented by nine channels. The two channel vectors are concatenated to form an input channel vector  $\mathbf{v}$  with 18 channels.

The function the system is to learn is

$$y = \begin{cases} 1 & \text{if } (x_1 > 0.5 \text{ AND } x_2 \leq 0.5) \text{ OR } (x_1 \leq 0.5 \text{ AND } x_2 > 0.5) \\ 0 & \text{otherwise} \end{cases} \quad (38)$$

wich is illustrated left in figure ??.

## 7 Discussion

A new reinforcement learning algorithm has been presented that take advantage of the channel representation. It can use simple linear models to implement non-linear functions. Since it generates predictions of the reward signals it can also be used for handling delayed rewards like the TD-methods.

A dynamic binary tree structure for this type of reinforcement learning systems has been developed. In this structure the nodes are competing experts that specialize on regions of the decision space where the function can be described in a simple (i.e. linear) way. The tree is dynamic in the sense that it continuously tests split- and prune criteria

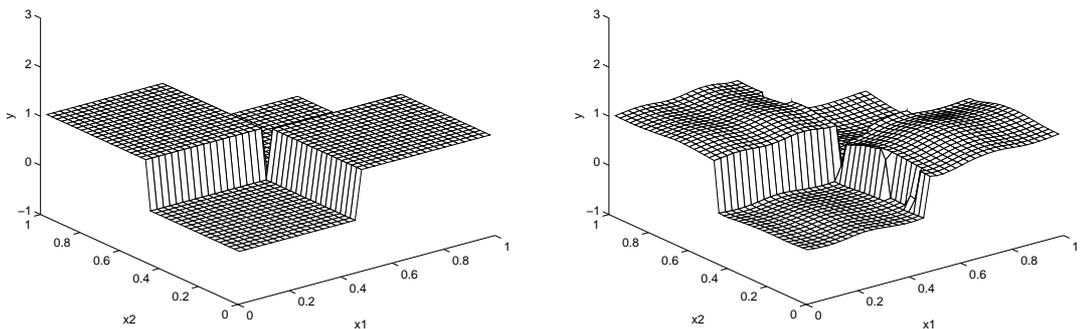


Figure 9: **Left:** The correct XOR-function. **Right:** The result after 4000 iterations.

and therefore it does not have a predetermined size. It can also have different depths in different parts of the decision space depending on the local complexity of the solution.

We argue that reinforcement learning together with a dynamic tree structure will be useful in the design of a learning autonomous system. Some experiments have been presented to illustrate the ideas applied on a simple but illustrative problem. There are, however, some difficulties left to solve before this method can be used on more complex problems. We will end this report by describing a few problems more explicitly.

## 7.1 Breakpoint Instability

A function as the one described in figure ?? could theoretically be represented exactly by a two level tree with  $x$  represented by two channels. One reason that this solution in practice does not yield perfect results is due to the instability of the breakpoint between the two child nodes. This problem is related to the fact that the *shape* of the prediction function is fixed when only two channels are used. Hence, while this function is sufficient for a subsystem to select a best response for a certain stimulus, it is not completely reliable when compared to a prediction function of another subsystem.

The reward prediction  $p$  as a function of the input variable  $x$  can, simplified, be described as a quadratic function in this case. The breakpoint  $x$  between two competing models can then be described by

$$a - (x - x_a)^2 = b - (x - x_b)^2. \quad (39)$$

See figure ?? This gives the following relation for the breakpoint:

$$x = \frac{b - a + x_a^2 - x_b^2}{2(x_a - x_b)} \quad (40)$$

If this equation is differentiated with respect to  $a$  we get

$$\frac{\partial x}{\partial a} = -\frac{1}{2(x_a - x_b)}. \quad (41)$$

This indicates that the position of the breakpoint is very unstable when the centers of the reward functions lie close to each other. This is the case short after a split when the two child nodes contain almost the same solution. The factors  $a$  and  $b$  in this discussion represent the norm of the matrices  $\mathbf{W}_A$  and  $\mathbf{W}_B$ .

The easiest way to deal with this problem is to use more channels to represent the input parameters.

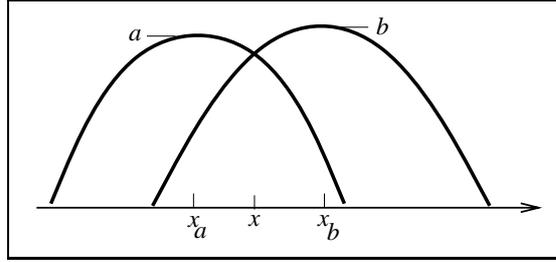


Figure 10: *The breakpoint  $x$  for two quadratic functions*

## 7.2 Overlapping Prediction Functions

There is another problem that is related to the problem with breakpoint instability. If two models are close to each other one of the models can completely cover the other model. This problem was mentioned in section 5.2. In that section we argued that the probabilistic choice of child node was a way to deal with that problem. A more robust way to handle this problem would also here be to use more channels for the input representation.

## 7.3 Splitting at Discontinuities

If the function that is to be approximated contains a discontinuity there will be a region near this discontinuity where the algorithm will make a lot of errors. This is because the breakpoint is not stable, and even if it was, it would not lie exactly at the discontinuity. There will always be an interval where the wrong model is chosen. In this interval a low reward will be received and the node will get a low mean reward. If the valid region for that model is small compared to the interval where it is not valid, the mean reward will get so low that the node will split. One of the new nodes created will face the same problem, and it will in fact get even worse, since this node has got an even smaller valid region than its father. This will lead to an infinite number of splittings around a discontinuity. Another way to see this problem is to consider the use of piecewise linear functions to approximate an arbitrary function. The number of linear functions needed in a certain interval will be in some proportion to the second derivative of the approximated function in that interval.

One way to handle this problem is to take less notice of the reinforcement for decisions near the breakpoint. This is fairly straight forward. Since the predictions are almost the same in this region, the difference between the predictions can be used as a confidence measure to moderate the update step.

## A Proofs

In this section proofs of some equations are presented that would have made the text uncomfortable to read. The proofs are presented in the same order as the equations occur in the text. The numbers in parenthesis over some relations refer to equations that imply the relations.

**The prediction function (equation 9, page 5)**

$$p \stackrel{(7)}{=} \langle \hat{\mathbf{q}} | \mathbf{W} \hat{\mathbf{v}} \rangle \stackrel{(8)}{=} \left\langle \frac{\mathbf{W} \mathbf{v}}{|\mathbf{W} \hat{\mathbf{v}}|} \middle| \mathbf{W} \hat{\mathbf{v}} \right\rangle = \frac{|\mathbf{v}| \langle \mathbf{W} \hat{\mathbf{v}} | \mathbf{W} \hat{\mathbf{v}} \rangle}{|\mathbf{v}| |\mathbf{W} \hat{\mathbf{v}}|} = \frac{|\mathbf{W} \hat{\mathbf{v}}|^2}{|\mathbf{W} \hat{\mathbf{v}}|} = |\mathbf{W} \hat{\mathbf{v}}| \quad \square$$

### Derivation of the update rule (equation 14, page 5)

$$\begin{aligned}
 r &\stackrel{(10,13)}{=} \langle \mathbf{W} + a\hat{\mathbf{q}}\hat{\mathbf{v}}^T \mid \hat{\mathbf{q}}\hat{\mathbf{v}}^T \rangle = \\
 &= \langle \mathbf{W} \mid \hat{\mathbf{q}}\hat{\mathbf{v}}^T \rangle + a\langle \hat{\mathbf{q}}\hat{\mathbf{v}}^T \mid \hat{\mathbf{q}}\hat{\mathbf{v}}^T \rangle = \\
 &= p + a(\hat{\mathbf{q}}^T \hat{\mathbf{q}} \hat{\mathbf{v}}^T \hat{\mathbf{v}}) = \\
 &= p + a(\hat{\mathbf{v}}^T \hat{\mathbf{v}}) \quad \square
 \end{aligned}$$

### The change of the father node (equation 30, page 11)

After the update of  $m_i$  and  $\mathbf{W}_i$  we get

$$(m_1 + \Delta m_1)(\mathbf{W}_1 + \Delta \mathbf{W}_1) + (m_2 + \Delta m_2)(\mathbf{W}_2 + \Delta \mathbf{W}_2) \stackrel{(27)}{=} 0$$

and this equation together with equations 27, 28 and 29 gives

$$(m_1 + \Delta m_1)(\mathbf{W}_1 + \Delta \mathbf{W} - \Delta \mathbf{W}_f) + (m_2 + \Delta m_2)(\mathbf{W}_2 - \Delta \mathbf{W}_f) = 0.$$

This leads to

$$\begin{aligned}
 &\Delta \mathbf{W}_f(m_1 + \Delta m_1 + m_2 + \Delta m_2) = \\
 &= (\mathbf{W}_1 + \Delta \mathbf{W})(m_1 + \Delta m_1) + \mathbf{W}_2(m_2 + \Delta m_2) = \\
 &= \underbrace{\mathbf{W}_1 m_1 + \mathbf{W}_2 m_2}_{=0 \text{ (27)}} + \Delta \mathbf{W}(m_1 + \Delta m_1) + \mathbf{W}_1 \Delta m_1 + \mathbf{W}_2 \Delta m_2
 \end{aligned}$$

which with the substitution  $m' = m + \Delta m$  gives the result:

$$\Delta \mathbf{W}_f = \frac{\Delta \mathbf{W} m'_1 + \mathbf{W}_1 \Delta m_1 + \mathbf{W}_2 \Delta m_2}{m'_1 + m'_2} \quad \square$$

## References

- [1] D. H. Ballard. *Vision, Brain, and Cooperative Computation*, chapter Cortical Connections and Parallel Processing: Structure and Function. MIT Press, 1987. M. A. Arbib and A. R. Hanson, Eds.
- [2] 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.
- [3] 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.
- [4] T. Denooux and R. Lengellé. Initializing back propagation networks with prototypes. *Neural Networks*, 6(3):351–363, 1993.
- [5] D. J. Field. What is the goal of sensory coding? *Neural Computation*, 1994. in press.
- [6] G. H. Granlund. Magnitude representation of features in image analysis. In *The 6th Scandinavian Conference on Image Analysis*, pages 212–219, Oulu, Finland, June 1989.

- [7] V. Gullapalli. A stochastic reinforcement learning algorithm for learning real-valued functions. *Neural Networks*, 3:671–692, 1990.
- [8] D. H. Hubel. *Eye, Brain and Vision*, volume 22 of *Scientific American Library*. W. H. Freeman and Company, 1988.
- [9] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3:79–87, 1991.
- [10] H. Knutsson. Representing local structure using tensors. In *The 6th Scandinavian Conference on Image Analysis*, pages 244–251, Oulu, Finland, June 1989. Report LiTH-ISY-I-1019, Computer Vision Laboratory, Linköping University, Sweden, 1989.
- [11] C-S. Lin and H. Kim. CMAC-based adaptive critic self-learning control. *IEEE Trans. on Neural Networks*, 2(5):530–533, 1991.
- [12] 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.
- [13] K. Nordberg, G. Granlund, and H. Knutsson. Representation and learning of invariance. Report LiTH-ISY-I-1552, Computer Vision Laboratory, S-581 83 Linköping, Sweden, 1994.
- [14] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.
- [15] 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.
- [16] R. S. Sutton. *Temporal credit assignment in reinforcement learning*. PhD thesis, University of Massachusetts, Amherst, MA., 1984.
- [17] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.
- [18] Chris Watkins. *Learning from delayed rewards*. PhD thesis, Cambridge University, 1989.
- [19] P. J. Werbos. Consistency of HDP applied to a simple reinforcement learning problem. *Neural Networks*, 3:179–189, 1990.
- [20] 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.