The Neuro Slot Car Racer: Reinforcement Learning in a Real World Setting

Tim C. Kietzmann
Neuroinformatics Group
Institute of Cognitive Science
University of Osnabrück
tkietzma@uni-osnabrueck.de

Martin Riedmiller
Machine Learning Lab
Computer Science Department
University of Freiburg
riedmiller@informatik.uni-freiburg.de

Abstract

This paper describes a novel real-world reinforcement learning application: The Neuro Slot Car Racer. *In addition to presenting the system and first results based on Neural Fitted Q-Iteration, a standard batch reinforcement learning technique, an extension is proposed that is capable of improving training times and results by allowing for a reduction of samples required for successful training. The* Neuralgic Pattern Selection *approach achieves this by applying a failure-probability function which emphasizes neuralgic parts of the state space during sampling.*

1. Introduction

One of the most important milestones in the application of reinforcement learning (RL) techniques is the capability of being able to directly learn on and control real-world systems. Although most RL algorithms are tested in simulated environments, experiments on real-world systems are able to provide valuable hints on potential problems and extensions that might not be obvious in simulations per se. Because of the benefits of experiments in real-world settings, low-cost applications that can be used as a benchmark are required. A system that serves this purpose, the Neuro Slot Car Racer, is introduced in this paper. Based on a Carrera Slot Car System, the task is to drive a slot car as fast as possible without going off track. The car is observed by a vision system, whereas the speed can be set via a USB device. The trained performance of our Slot Car Racer System is difficult to beat: during one week of racing with an earlier version of the system, which took part during the Hannover Fair exhibition in 2009, only 7 people were able to be faster than the RL controlled car.

Results of the system are based on batch reinforcement learning (RL) methods, which have recently shown their effectiveness and ability to directly learn controllers in real world applications [9, 1, 8]. In contrast to classical on-line RL, batch RL methods store and reuse information about system behaviour in form of (state, action, successorstate) - triples. The value function (or Q-value function) is then updated on all states (state-action pairs) simultaneously. This reuse of transition data makes batch learning methods particularly efficient.

The standard procedure of batch RL is to use the complete set of samples for each training iteration. As a result, training times increase with the amount of collected triples. A promising question is therefore whether it is possible to cleverly sample from the set of transition triples, such that the learning process is successful, even if the number of samples for learning is reduced or restricted. As an approach to this question, we propose a mechanism called Neuralgic Pattern Selection (NPS). The effectiveness of the approach is evaluated on the slot car racer system.

2. Real-World Testbed: The Slot Car Racer

The real-world training environment that is introduced in this paper and used for the experiments is based on an analog Carrera Evolution system shown in Figure 1. The car's motor is accelerated by actions that correspond to track voltages. The track setup used, with long straight passages as well as sharp inner curves and more relaxed outer curves, requires the trained policies to use a great range of different actions in order to succeed. The Carrera system was used as provided out of the box, only the color of the slot car (Ferrari 512 BB LM 2007) was changed in order to allow for improved tracking results. The goal of the learning procedure is to find a policy, which drives the car as fast as possible. A policy succeeds if it drives the car around the track without getting off-track (i.e. crashing) and fails otherwise. Additionally, a successful policy should be sensitive to the state of the car. A simple example of why this is needed is the initial start of the car. In this situation it is important to start as fast as possible. If only the position would be associated with an action, the car would not start in a position where the associated action would be to brake.

The main components of the Neuro Slot Car Racer system are the vision system and the controller. The vision system consists of an overhead camera (Point Grey Flea2) and a software which recognizes the setup of the track, the position of the car and its speed. Also, the vision system detects cases in which the car slides or gets off track. Currently, the system is able to deliver state information with 60Hz and automatically adapts to different lighting situations. A considerable challenge of the slot car system is given by the delay from the time of output of an action until the visibility of this very action in the state description (this includes mechanical inertia as well as the time needed for image processing and is estimated to be about 100 ms).

The output to the track is realized through a USB device, which is attached to the same computer as the vision system. This computer (an Apple iMac with a 2.4Ghz Intel Core 2 Duo processor and 2GB of RAM) runs the controlling cycles including the receiving of state input communicated between programs via TCP/IP, as well as selecting and sending the appropriate output via the USB controller. The actions, which are given as output, are encoded in percentage of the maximally possible voltage.

In order to estimate baseline performance, the maximally possible action of the car was selected, such that it was able to complete the whole track at once without failure. The corresponding ground action, which was found to be $a = 25\%$ of the maximum, results in a lap time of 6.10 seconds. The best human performance is $4.50s$.

Because of the need for a dynamic control of the car, the temporal delay and the setup of the track, this real-world application is particularly difficult to control and poses a great challenge to model-free machine learning approaches.

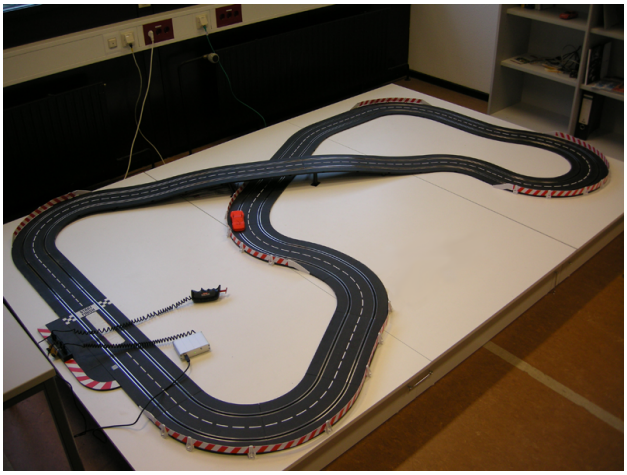


Figure 1. The real world slot car system used in the experiments. The state of the car is extracted from the image of an overhead camera.

3. Batch Reinforcement Learning

The classical approach towards model-free reinforcement learning is Q-learning in which an optimal value function of state-action pairs is learned iteratively and online [10]. This means that the Q-function is updated after each transition of the system. Typically, this approach requires thousands of iterations until successful policies are found. The classical Q-learning update rule is $Q_{k+1}(s,a) := (1-\alpha)Q(s,a) + \alpha(c(s,a) + \gamma \min_b Q_k(s',b))$. s is the origin state of the transition, a is the chosen action and s' is the successor state. $c(s,a)$ describes the immediate costs associated with action a applied in state s. γ is the discounting factor.

The crucial difference of batch reinforcement learning to the classical approach is to perform the update of the Q-function based on sets of past transitions instead of singular state transitions. Because of this, the general approach of batch reinforcement learning consists of three steps [3]. First, sampling is done according to the currently active policy via interaction with the real system. Then a training pattern set is generated, and finally the actual batch supervised learning is applied.

One way of dealing with continuous state spaces, as in the current system, is to use multilayer perceptrons (MLPs) as function approximators for the value function. Because MLPs approximate the value function in a global way, an online change of one value might have unpredictable impacts on the outcome of arbitrary other state-action pairs [8, 4]. Batch training ensures global consistency and is thus a perfect match for MLPs as function approximators. In addition, batch training has the advantage that more advanced and powerful learning methods exist than for standard gradient descent techniques. One of these methods, which is also used in the current work, is the Rprop algorithm [7], which allows for very fast training times and has also been shown to find very good solutions and to be very insensitive with respect to its parameter settings.

Although other batch reinforcement learning techniques exist that are capable of learning in real-world settings, the current work focuses on and extends Neural Fitted Q-Iteration (NFQ) [6]. Training is based on sets of patterns and an MLP is used to approximate the value function. The data for the supervised training of the neural network consists of state-action pairs as input and the target value. The latter is calculated by the sum of the immediate transition costs and the expected minimal path costs for the successor state. The minimal expected path costs are estimated by the policy which is stored in the preceding MLP.

The standard mode of using NFQ, alternating batch, divides sampling and training into two distinct but alternating processes. The current policy is applied greedily to collect new (state, action, successorstate) - triples, which are

then used together with earlier samples to learn a new policy. The best policy is kept by a pocket algorithm. Thus, the provided results always refer to the performance of the best trained policy. Interaction with the system is required because the car has to be put back on the track upon failure.

As an alternative to alternating batch, NFQ can also be applied in an offline mode. In this setting, it is assumed that the set of transition triples is given a priori and training is thus purely based on collections of previously sampled data. All samples are available from the first iteration and either all or subsets of samples can be used for training. It has to be noted that this data can be obtained in various ways: it can be sampled from existing controllers, human interaction with the system or even from a random policy.

Algorithm 1 NFQ main loop (adapted from [4])

input: a set of transition samples D, # of iterations N
output: neural Q-value function Q_N
$k = 0$;
init MLP() $\rightarrow Q_0$;
while ($k < N$) **do**
  generate pattern set P from D
  for $l = 1$ to size(pattern set) **do**
    $input^l$ = (s^l , a^l);
    $target^l$ = $c(s^l, a^l, s'^l) + \gamma * min_b Q_k(s'^l, b)$;
    P(l) = ($input^l$, $target^l$);
  end for
  Rprop training(P) $\rightarrow Q_{k+1}$
  $k+ = 1$
end while

4. Finding Neuralgic Parts of the State Space: Neuralgic Pattern Selection

Whenever dealing with large amounts of patterns, the question arises whether all patterns need to be used in every training step, or whether there exists a way in which only parts of the patterns can be selected while still allowing for successful training and good policies [5]. There are two reasons for this. First, it is clearly beneficial from a computational point of view to reduce the time needed for training by reducing the numbers of patterns. Also, large amounts of patterns might even become computationally intractable [2]. Second, some parts of the state space are learned quickly, they are comparably safe and do not need excessive training. However, many reinforcement learning problems also contain parts which are harder to learn than others (called *neuralgic states* hereafter). These areas form the crucial points of failure and demand more training than others. Because of this, it is reasonable to concentrate the learning process on exactly these more difficult parts of the problem space.

The idea of Neuralgic Pattern Sampling (NPS) is to learn a probability distribution of failure for the state space (or parts of it) and to use this distribution as basis for pattern sampling. The initial distribution assigns equal probability to each part of the state space because the neuralgic parts are not yet known. Whenever a failure occurs, the value which corresponds to the position of failure (x_{fail}) and its neighboring positions are increased. This way, the probability distribution shapes over time, revealing the neuralgic parts of the state space.

For the experiments with the slot car system, we used the position on the track as basis for the probability distribution. Note, however, that NPS is applicable to multiple state dimensions (leading to multi-dimensional Gaussian distributions). In the offline version of the algorithm, each position of failure out of all patterns is first surrounded by a Gaussian distribution. This is done in order to increase sampling in a greater area around the failure. Then, the resulting values are added to the distribution. This can also be seen as forming a histogram of failure, with Gaussian smoothing of each pattern in a bin. Finally, the distribution is normalized to form a probability distribution.

In the described form, NPS is closely connected to the current application because of its relying on 'failures' during training. Still, NPS can be extended to cope with problems that do not contain explicit failure states but nevertheless exhibit neuralgic parts. In this case, sampling could be based on the amount or variance of the expected costs with higher sampling rates at cost-intensive areas or regions which still have high variance in the expected costs.

Algorithm 2 Neuralgic Pattern Sampling

$state\_hist$ = split_space($hist\_granularity$);
for all $x \in X_{fail}$ **do**
  for $shift$ = 0 to $shift\_max$ **do**
    $pos = x - shift$;
    $state\_hist(pos)$ += $\frac{1}{\sigma\sqrt{2\pi}} exp(-\frac{(pos-x)^2}{2\sigma^2})$
  end for
end for
$state\_hist$ = $state\_hist/sum(state\_hist)$;

5. Experiments & Results

5.1. Training Setup

In its most basic version, the state description of the slot car contains information about the position of the car, its velocity and whether it is on- or off-track. The position of the car is calculated with regard to the overall length of the track and given in percentage. To overcome the problem of delayed state information, an action history in form of the

last selected action is included in the state description. The resulting state space contains four dimensions (position, velocity, on/off track and action history). Because a sampling frequency of 15 Hz is sufficient for successful training and produces less redundant patterns, it was used for the experiments.

The actions of the system were chosen to be multiples of the baseline action ($a = 25\%$ of the maximal action). Together with the 'brake' action, $a = 0\%$, the action set was $A = [0, 25, 50, 75, 100]$. In order to let the training lead to time-optimal policies, the associated immediate costs were selected to favor fast over slow actions. Failure, however, was punished with maximal costs. The immediate costs associated with the given actions are $c = [0.06, 0.03, 0.01, 0.003, 0]$, failure induces costs of $c = 1$. γ was set to 0.95.

The system was trained and results are presented for the alternating- and offline batch mode. Each training sequence included 150 NFQ iterations. The network setup of the MLP included five input neurons (four state dimensions plus one action dimension), two hidden layers with ten units each, and one output layer. Each network was trained for 300 epochs using Rprop. In offline mode, training was based on patterns that were sampled during an earlier alternating batch training. In total, about 30.000 transition samples (s, a, s') were collected.

In addition to the baseline-experiments, the performance of NPS was estimated. NPS is generally applicable in the alternating- as well as in the offline mode of NFQ. However, the offline mode is especially well suited for the current experiments because it allows for using the same set of patterns in different conditions and therefore permits direct comparison (alternating batch mode creates a different set of samples in each experiment). Because of this, the presented experiments on NPS were conducted in offline mode.

5.2. Results

Alternating & Offline NFQ: Baseline Results

As a first proof of concept and in order to provide the required data for later offline learning experiments, the system was first trained in alternating-batch mode. After only 104 episodes, the time required for a complete lap was improved to only $4.45s$ as compared to $6.1s$ at ground speed. Figure 2 shows the selected actions of two exemplary policies: (a) shows the behavior of the first successful policy, (b) shows the behavior of a later policy. Whereas the early policy is still highly random and clearly suboptimal, the later policy is much more homogenous. More importantly, this policy clearly shows that the car acts according to the varying difficulties of the track. In corners with provided cushions, the car drives at higher speeds than in more insecure curves. (c) shows the actions of the car taken during one run selected according to the policy described in (b). The success of the alternating batch training, which found a very good policy in only 104 iterations, demonstrates the efficiency of the overall approach.

In a second experiment, offline NFQ was trained with all patterns available. Ten training runs were performed on the complete set, leading to ten sets of 150 policies. In contrast to alternating batch mode, in which pattern sampling automatically gives an estimate of the performance of the latest policy, the resulting policies of offline NFQ need to be tested after training. Because each of the ten training runs created 150 policies, testing every single one of them in the real system is clearly too time-consuming. Instead, five policies were randomly selected for policy screening on the real system out of the 150 policies of each run.

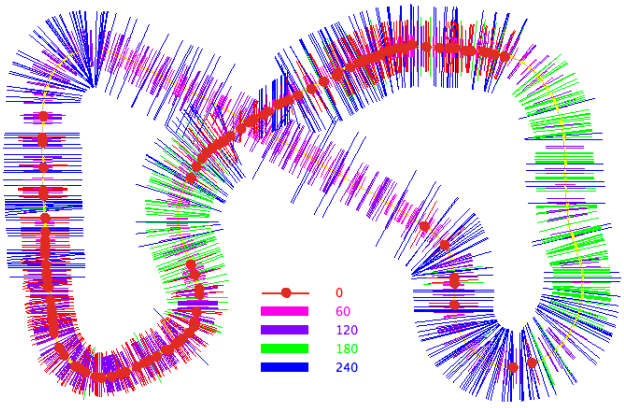
Of all the tested policies, 79% were successful. The best performance of the tested policies was found to be $4.53s$, the mean performance of all successful policies is $4.91s$. This again is a major improvement compared to the one-action baseline of $6.1s$. Also, these results are comparable to the current human record of $4.50s$.

Pattern Selection using NPS

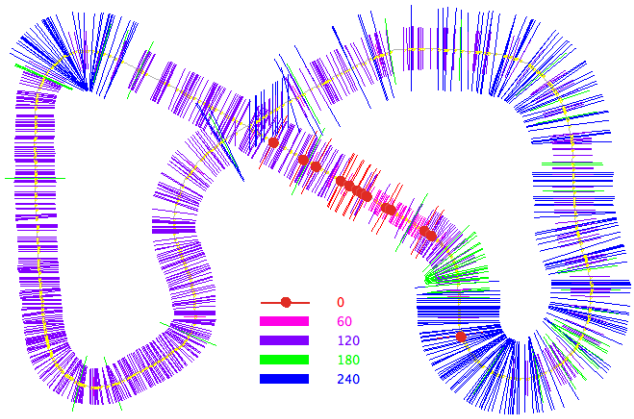
In the experimental implementation of NPS, the track positions were separated into bins with a granularity of 0.01% of the track's length. The standard deviation of the Gaussian distribution was set to 3. Also, the increase in probability of positions around x_{fail} were only applied to 10 bins prior to x_{fail}. Once the distribution is calculated, it can be used as basis for pattern sampling. Figure 3 shows the estimated sampling distribution which results in a more intense sampling of difficult parts of the track as compared to easier ones.

To ensure the functioning of the overall approach and to yield a first estimate of the performance, some preliminary experiments were performed comparing NPS to a random sampling method. Table 1 shows the experimental results. In this test setting, the random approach required sampling of up to 22500 patterns to yield a successful policy whereas NPS needed only 5000. Note, however, that the results are based on ten randomly selected networks based on one training set each.

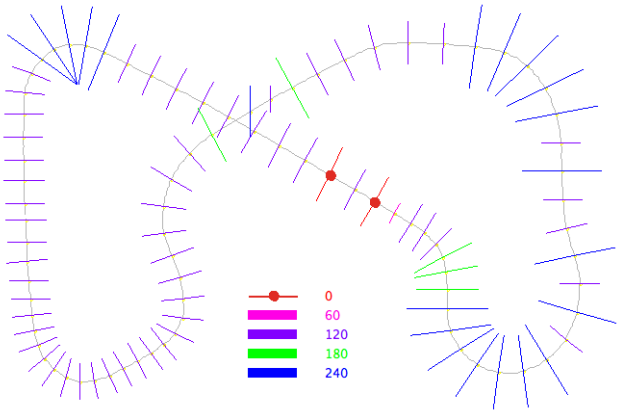
Having ensured the functioning of the overall approach, more detailed experiments were performed. As in the offline NFQ baseline experiments, performance was evaluated based on ten sets. However, instead of using the complete set, the selected number of patterns was sampled with the corresponding technique. Then, these sets were used as basis for training. Again, results were estimated from randomly selecting five networks out of all trained ones. Table 2 shows results from training with randomly selected patterns (naive approach) and results from the same amount of



(a)



(b)



(c)

Figure 2. This figure shows the selected actions of two policies. The action 'brake' is encoded by a circle, for the other actions the size of the bars correspond to the power of the action (longer bars correspond to faster actions). (a) An early successful strategy. (b) The actions selected by a later and faster policy. The selected actions are more homogenous and directly match the difficulty of the corresponding parts of the track. (c) The actions selected during one round.

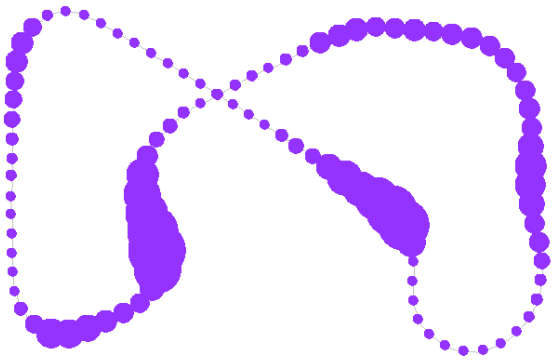


Figure 3. This figure shows a mapping from the estimated sampling distribution to the track. It can be seen that the sampling density directly corresponds to the varying difficulty of the track.

# of Samples	Random (s)	NPS (s)
5000	-	4.50
15000	-	4.55
22500	4.55	4.37
30000	4.57	4.57

Table 1. Results of the preliminary experiments. This table shows the best performance out of 10 randomly selected policies after training. The training sets were either randomly sampled or based on NPS. In the case of the complete set no actual selection is required.

patterns selected by NPS. Performance is also compared to the results from training with the complete data set. The results clearly show the advantage of using NPS instead of a random selection method. Not only the performance is closer to the performance of training with all patterns, but also the success rate of the trained networks is significantly improved (4 % vs 16 % successful policies). With regard to training times, one iteration of NFQ on the complete set of patterns takes 48 s whereas the training on a set of 5000 samples can be accomplished in only 8 seconds.

Still, compared to the performance based on the complete set of patterns, a smaller amount of networks succeeded. A reason for this can be given by the fact that sampling is only dependent on the position of the pattern, but not on its success. Thus, the different sets contained changing amounts of positive and negative patterns. In order to increase the success rate, a straightforward solution would be to add all negative patterns and sample the rest ac-

Method	Success (%)	Mean (s)	Best (s)
30000	79	4.91	4.53
Rand 5000	4	5.07	4.78
NPS 5000	16	5.01	4.66
NPS^{neg} 5000	87	6.54	5.20

Table 2. This table shows the success rates and mean and best performance of the tested policies. Results are given for the complete training set, 5000 randomly selected patterns and 5000 patterns selected by NPS. Finally, results of NPS including all negative training patterns are given (NPS^{neg}).

cording to the described procedure. Results are also given in Table 2 (NPS^{neg}). As expected, the stability of the policies increased dramatically. However, the average lap time is decreased because the small sample size does not contain enough positive patterns. Put differently, the overhead of negative training instances lead the system to behave overly cautious.

6. Conclusions

The paper introduced a novel real-world RL application, the *Slot Car Racer System*, and an extension of batch RL addressing the problem of selecting appropriate patterns for training. The effectiveness of the overall approach and system was demonstrated via standard alternating- and offline batch learning. After training the system for only 45 minutes, the resulting policies are equal to human performance, the lap time of the best policy ($4.37s$) is clearly better than the best observed human performance. The reported results can be used as benchmark for later experiments with the system.

Neuralgic Pattern Sampling (NPS) was applied to decrease the amount of needed patterns and to concentrate the training on sensible parts of the state space. It was shown that the number of patterns could be reduced from 30.000 to only 5.000 while still obtaining comparable results and that NPS is highly preferable over a random selection of training patterns. The inclusion of all negative samples in the sampling process was shown to lead to an increased amount of successful policies. However, this was achieved at the cost of performance. Future research will have to further investigate how an optimal amount of negative patterns can be selected.

Apart from its application in offline mode, NPS can also be applied in an online learning environment. In addition to providing a means of selecting promising subsets of the sampled data for learning, the evolving probability distribution of neuralgic states can be used to drive learning process to promising states. In an active learning paradigm, the system could safely drive the car to selected positions and like this concentrate the training and sampling on the more difficult parts of the track.

Remark: In this paper, we have introduced a novel, comparably low-cost real-world reinforcement learning problem. The slot car system consists of a Carrera Evolution Package, the vision software and the USB controller. In order to allow for comparability of approaches across groups, the software as well as the controller design can be provided upon request.

7. Acknowledgements

The authors would like to thank Christian Müller and Sascha Lange for developing the vision tool and the Harting Technology Group for supporting this project and for providing an opportunity to exhibit the system on the Hannover fair.

References

[1] M. Deisenroth, J. Peters, and C. Rasmussen. Approximate dynamic programming with gaussian processes. In *Proceedings of the 2008 American Control Conference (ACC 2008), Seattle, WA, USA, June*, pages 4480–4485, 2008.

[2] D. Ernst. Selecting concise sets of samples for a reinforcement learning agent. In *Proceedings of CIRAS*, 2005.

[3] D. Ernst, P. Geurts, and L. Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6(1):503–556, 2006.

[4] R. Hafner and M. Riedmiller. Neural reinforcement learning controllers for a real robot application. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 07)*, pages 2098–2103, 2008.

[5] M. Plutowski and H. White. Selecting concise training sets from clean data. *IEEE Transactions on Neural Networks*, 4(2):305–318, 1993.

[6] M. Riedmiller. Neural fitted q iteration-first experiences with a data efficient neural reinforcement learning method. *Lecture Notes in Computer Science*, 3720:317–328, 2005.

[7] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: the rprop algorithm. In *Neural Networks, 1993., IEEE International Conference on*, pages 586–591, 1993.

[8] M. Riedmiller, M. Montemerlo, and H. Dahlkamp. Learning to drive a real car in 20 minutes. In *Proceedings of the FBIT 2007 conference, Jeju, Korea. Special Track on autonomous robots*, pages 645–650. Springer, 2007.

[9] A. Rottmann, C. Plagemann, P. Hilgers, and W. Burgard. Autonomous blimp control using model-free reinforcement learning in a continuous state and action space. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 1895–1900, 2007.

[10] C. Watkins. *Learning from Delayed Rewards*. Cambridge University, 1989.