Exponentiated Gradient Methods for Reinforcement Learning

Doina Precup
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
dprecup@cs.umass.edu

Richard S. Sutton
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
rich@cs.umass.edu

Abstract

This paper introduces and evaluates a natural extension of linear exponentiated gradient methods that makes them applicable to reinforcement learning problems. Just as these methods speed up supervised learning, we find that they can also increase the efficiency of reinforcement learning. Comparisons are made with conventional reinforcement learning methods on two test problems using CMAC function approximators and replacing traces. On a small prediction task, exponentiated gradient methods showed no improvement, but on a larger control task (Mountain Car) they improved the learning speed by approximately 25%. A more detailed analysis suggests that the difference may be due to the distribution of irrelevant features.

1 INTRODUCTION

Exponentiated gradient (EG) methods were first proposed by Littlestone (1988) in the form of the Winnow algorithm for training linear threshold classifiers. Kivinen and Warmuth (1994) proposed the first EG methods for on-line linear regression. The analogous conventional method for online linear regression is the Least-Mean Square (LMS) rule (Widrow & Hoff, 1960; also known as the Widrow-Hoff rule or the delta rule). The LMS rule makes additive updates in weight space, whereas EG methods perform multiplicative updates.

Kivinen and Warmuth (1994) established theoretical worst case loss bounds for both kinds of methods in the context of online linear regression. The theoretical and experimental evidence suggest that EG methods have a lower loss bound for problems in which only a few of the input features are relevant. Empirically, they converge faster for this class of problems. EG methods have also proved effective in applications. For example, several researchers have shown that they can be used successfully to train linear classifiers for information routing tasks with sparse targets (Lewis, Schapire, Callan & Papka, 1996; Papka, Callan & Barto, 1996).

EG methods have several characteristics that make them appealing for reinforcement learning. First, they are online methods, which is required for reinforcement learning. Second, many reinforcement learning tasks with continuous state spaces have been tackled using sparse coarse coded function approximators (e.g., Sutton, 1996, Watkins, 1989, Rummery and Niranjan, 1994). EG methods may be able to discriminate irrelevant features quickly in this type of input representation. Third, online reinforcement learning is particularly sensitive to the speed of learning, and EG methods can potentially improve it.

This paper presents EG methods for reinforcement learning with linear function approximators. EG methods are presented in section 2. Section 3 presents the extension to temporal-difference learning and reinforcement learning. Sections 4 and 5 describe experimental results on prediction and control tasks. Section 6 further analyzes the experimental results, and section 7 presents conclusions.

2 EXPONENTIATED GRADIENT METHODS

First, we review the derivation of conventional EG methods for online supervised learning. The learner is presented with a sequence of N-dimensional real-valued vectors, $\mathbf{x}_t$, and is asked to predict the scalar values, y_t, associated with them. At each time step, the learner's prediction, $\hat{y}_t$, depends on the input vector and the current hypothesis, represented by a vector of real-valued weights, $\mathbf{w}_t$:

$$\hat{y}_t = f(\mathbf{x}_t, \mathbf{w}_t).$$

The learner adjusts its hypothesis by choosing a new weight vector, $\mathbf{w}_{t+1}$, that minimizes the objective function

$$d(\mathbf{w}_{t+1}, \mathbf{w}_t) + \alpha E(y_t, \hat{y}_t), \tag{1}$$

where d is a distance metric and E is an error function. The parameter α determines how much emphasis is put on the corrective part of the objective function relative to the conservative part.

The objective function (1) can be used to derive weight update rules for different distance metrics, error measures, and forms of predictors. We focus on the case of linear predictors, in which $\mathbf{w}_t$ is an N-dimensional real vector, and the predicted output is given by $\hat{y}_t = \mathbf{w}_t^T \mathbf{x}_t = \sum_{i=1}^N w_t(i) x_t(i)$, where $w_t(i)$ are the individual components of $\mathbf{w}_t$, and $x_t(i)$ are the individual components of $\mathbf{x}_t$.

The LMS rule can be obtained from (1) by using the Euclidian distance metric, $d(\mathbf{w}_{t+1}, \mathbf{w}_t) = \sum_{i=1}^N (w_{t+1}(i) - w_t(i))^2$, and the squared error, $E(y_t, \hat{y}_t) = (y_t - \hat{y}_t)^2$. The resulting update rule is:

$$\mathbf{w}_{t+1} := \mathbf{w}_t + \alpha(y_t - \hat{y}_t)\mathbf{x}_t. \tag{2}$$

The corresponding EG method uses the squared error and the unnormalized relative entropy distance metric, $d(\mathbf{w}_{t+1}, \mathbf{w}_t) = \sum_{i=1}^N (w_{t+1}(i) - w_t(i) + w_t(i) \log(\frac{w_t(i)}{w_{t+1}(i)}))$. Assuming that the weights are restricted to positive values, the resulting update rule can be written:

$$\log \mathbf{w}_{t+1} := \log \mathbf{w}_t + \alpha(y_t - \hat{y}_t)\mathbf{x}_t, \tag{3}$$

where the natural logarithm is applied componentwise. Thus, in a sense, the EG method takes the same steps as the LMS rule, but in the space of the logarithm of the weights. This yields multiplicative instead of additive updates in weight space.

EG methods must take special steps to allow negative as well as positive weights. Otherwise, update rule (3) would require strictly positive weights. The solution adopted by Kivinen and Warmuth (1994) is to duplicate the input vector $\mathbf{x}_t$, replacing it by $(x_t(1), ...x_t(N), -x_t(1), ..., -x_t(N))$. Thus, although each weight is strictly positive, the difference of the weights corresponding to each input feature can have any sign.

Both EG and LMS methods are of linear complexity in the number of weights ($O(N)$). However, the EG method needs twice as many weights, and is slightly more expensive computationally: it requires an exponentiation operation for each weight on each time step, in addition to the computation required by the LMS rule.

The EG method is also more sensitive than LMS to the initial magnitude of the weights. If this magnitude is too small, the algorithm will converge very slowly. Big initial magnitudes can lead to instability and overflow errors.

In order to keep the weight magnitudes bounded, Kivinen and Warmuth (1994) proposed a variation of the EG method that applies a normalization step after each weight update. The normalization linearly rescales all weights so that they sum to a constant:

$$\sum_{i=1}^{2*N} w_t(i) = W.$$

Normalization introduces a supplementary parameter, W, which needs to be tuned. Our initial experiments found normalization to be sometimes counterproductive and never very helpful (Precup & Sutton, 1996). We did not use it in the experiments presented here.

3 EXTENSION TO REINFORCEMENT LEARNING

It is natural to extend the EG idea to reinforcement learning in two steps. First we extend it to the case of long-term prediction by temporal-difference learning, and then we extend it to the full case of reinforcement learning, including control.

The long term prediction problem most frequently encountered in reinforcement learning is that of predicting the discounted cumulative future reward. The learner is presented with a time sequence of input vectors, $\mathbf{x}_t$, and scalar rewards, r_t. At each time t, the learner seeks to predict

$$y_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \tag{4}$$

where $\gamma \in [0, 1]$ is called the *discount-rate* parameter. The prediction at time t is denoted $\hat{y}_t$ and is formed as a linear function of $\mathbf{x}_t$ and a real-valued parameter vector, $\mathbf{w}_t$, just as in the supervised case.

The standard learning rule for this problem is linear TD(λ) (Sutton, 1988). It is given by

$$\mathbf{w}_{t+1} := \mathbf{w}_t + \alpha\delta_t\mathbf{e}_t, \tag{5}$$

where δ_t is the temporal-difference error at time t:

$$\delta_t = r_{t+1} + \gamma\mathbf{w}_t^T\mathbf{x}_{t+1} - \hat{y}_t,$$

and $\mathbf{e}_t$ is a vector of eligibility traces. In our experiments we found that EG worked much more reliably with replacing traces (Singh & Sutton, 1996) than with conventional accumulating traces. Replacing traces are defined by

$$e_t(i) = \begin{cases} 1 & \text{if } x_t(i) = 1 \\ \gamma\lambda e_{t-1}(i) & \text{if } x_t(i) = 0, \end{cases}$$

where $e_t(i)$ is the ith component of $\mathbf{e}_t$. Replacing traces require that the input features be binary, $x_t(i) \in \{0, 1\}$.

The natural way to extend TD(λ) to an EG form is to perform the same update (5), except in the logarithm of the weights, just as we did in the supervised case. The result is a new update rule:

$$\log\mathbf{w}_{t+1} := \log\mathbf{w}_t + \alpha\delta_t\mathbf{e}_t, \tag{6}$$

which we refer to as EG-TD(λ).

We now briefly describe the further extension of the EG ideas to the full reinforcement learning case, including control. Our extension assumes a discrete action set, and is based on the Sarsa algorithm (Rummery & Niranjan, 1994; Sutton, 1996), although analogous extensions to other methods are probably straightforward. The EG extension we tested is identical to the Sarsa algorithm described by Sutton (1996) except that steps were made in the logarithm of the weights. We call this algorithm EG-Sarsa.

In brief, the full reinforcement learning methods make a separate prediction of the discounted future reward for each possible action. The learner prefers to take at each step the action with the greatest prediction of future reward. Each prediction is a linear function of a feature vector, $\mathbf{x}_t$, describing the current state. The reader is referred to earlier papers for further details (Singh & Sutton, 1996; Sutton, 1996).

4 CONTINUOUS RANDOM WALK (PREDICTION) EXPERIMENT

In this task, the observed data, $\mathbf{x}_t$ and r_t, are generated by a system whose state is a point taking a random walk in the interval from 0 to 1. The time sequence is broken into a series of episodes, each of which begins with the state in the middle, at 0.5, and ends with the state reaching or exceeding one of the bounds at 0 and 1. On each time step, the state moves up or down by a uniformly selected random step in the continuous interval $[-0.2, +0.2]$. The reward, r_t, is zero on all time steps except those on which the the state exceeds one of the bounds. Exceeding either bound ends the episode and the time period over which the sum (4) is accumulated. The reward at the end of the episode is equal to the position at the end of the trial. For instance, if the position at the end of a trial is -0.02, the final reward will also be -0.02. The discount-rate parameter is $\gamma = 1$. Thus, the correct prediction for any state is the position of the state inside the walking interval.

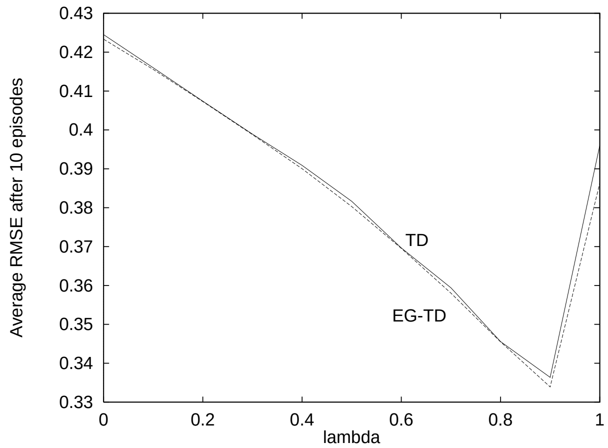

Figure 1: Summary performance on continuous random-walk task

A sparse coarse coding technique, CMAC (Albus, 1981; Miller, Glanz & Kraft, 1990), was used to construct the binary feature vector, $\mathbf{x}_t$, from the real-valued state. The state space, $[0, 1]$, was divided into 10 equal-sized intervals. An additional intervel was added in order to allow for an offset of the whole tiling. This was repeated to obtain 10 tilings, each offset by a different randomly selected fraction of an interval. For each interval, there was a corresponding feature that took on the value 1 when the state was within that tile, and 0 otherwise. Thus, there were $11 * 10 = 110$ tiles and 110 input features. At each step, exactly 10 features are active (one for each tiling), representing the current state.

We applied EG-TD(λ) and conventional TD(λ) to this problem, each with a variety of values for α and λ.

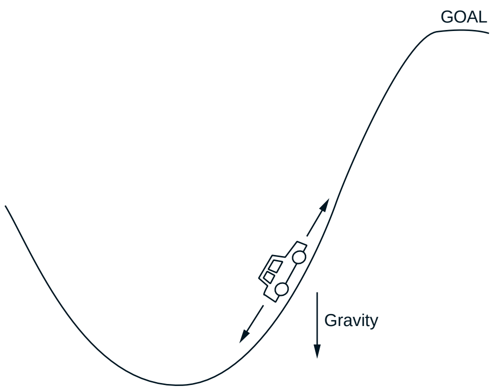

Figure 2: Mountain-car task

For TD(λ) the weights were initialized to 0, and for EG-TD(λ) they were initialized to 1, which produces the same initial predictions (zero everywhere). For each combination of algorithm and parameter value we ran 10 episodes. To obtain a performance measure for each run, we evaluated the final learned prediction function at 21 sample points along the interval from 0 to 1, evenly spaced at 0.05. For this problem we know the correct predictions are equal to the position. The performance measure for a run was the square root of the mean-squared error (RMSE) between the correct predictions and the final learned predictions over the 21 sample points. This performance was then averaged over 100 independent runs to provide a measure of the performance of each algorithm and set of parameter values.

Figure 1 shows performance as a function of λ, using the best α value for each combination of algorithm and λ value. TD(λ) and EG-TD(λ) performed similarly at all the values of λ. None of the differences are statistically significant.

5 MOUNTAIN CAR (CONTROL) EXPERIMENT

The mountain-car problem is a minimum-time control problem in which the learning agent seeks to drive a vehicle to the top of a hill (Figure 2). The reward is -1 for all time steps until the goal is reached. There is no discounting ($\gamma = 1$). At each time step there are three possible actions: accelerate forward, accelerate backward, and no acceleration. It is not possible to simply accelerate up the hill because gravity is stronger than the engine. The only solution is to accelerate backwards first and then thrust forwards towards the goal. The detailed physics we use in these experiments are

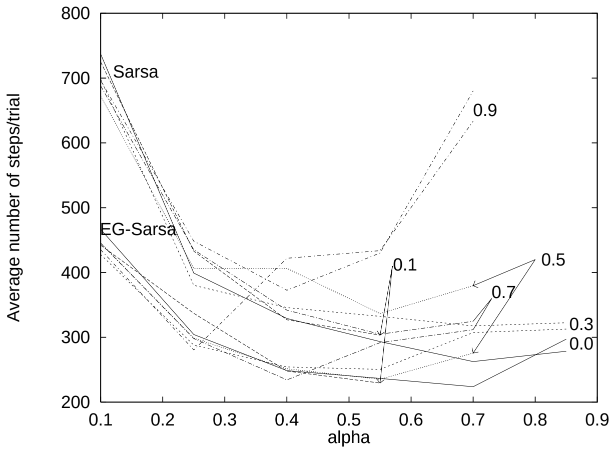

Figure 3: Performance on mountain-car task

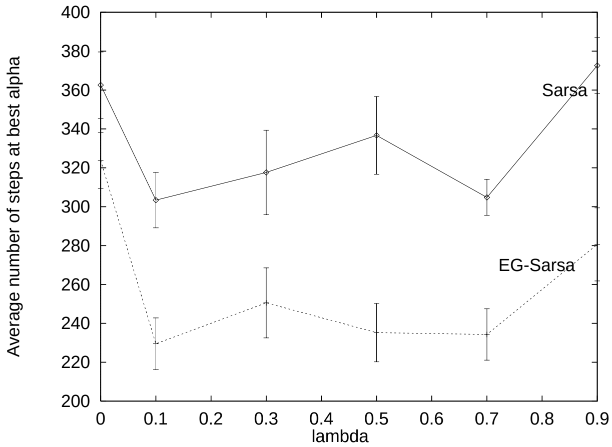

Figure 4: Summary performance on mountain-car task, using the best α for each λ value

as given by Singh and Sutton (1996). The software we used was derived from that provided by Mahadevan.

We applied the Sarsa and EG-Sarsa algorithms, with replacing traces, as described earlier. The details of mapping the algorithms onto the mountain car task were exactly as described in (Singh & Sutton, 1996). This task has a continuous state space, with two state variables: position and velocity. We used a set of three CMACs, one for each action. Each CMAC had 5 tilings with random offsets. Each variable was evenly divided in 8 intervals, adding one additional interval in order to accomodate a random offset. Thus, there were $9 * 9 * 5 = 405$ input features, five of which are active at any one time.

For Sarsa, the initial weight values were all 0, which gives an optimistic estimate. This ensures exploration

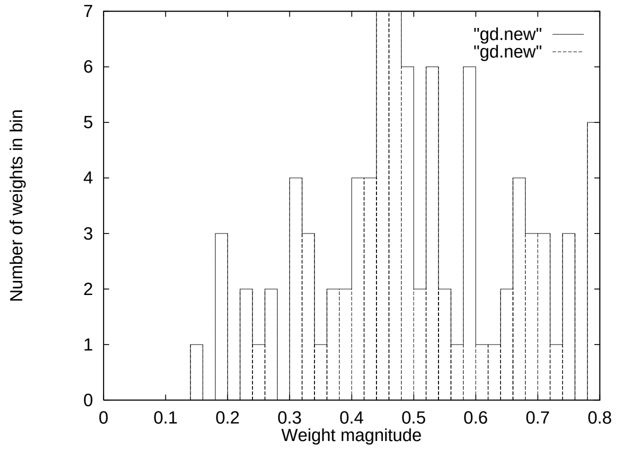

Figure 5: Weight histogram for the Random Walk, TD(λ)

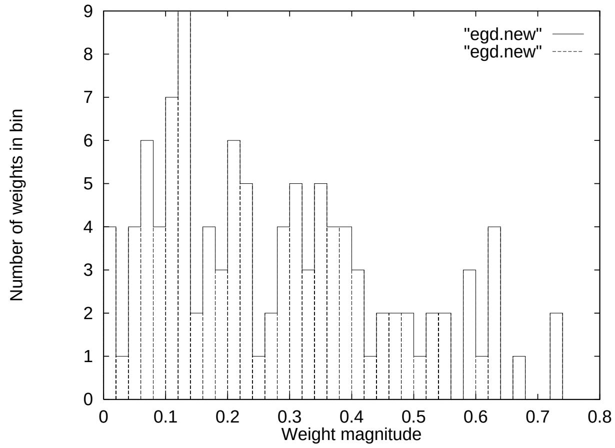

Figure 6: Weight histogram for the Random Walk, EG-TD(λ)

in the beginning, even when using a greedy policy for picking actions, as we did in these experiments. For EG-Sarsa, both the weights for both positive and negative inputs were initialized to 1, which produces the same optimistic initial predictions (zero everywhere).

Each algorithm was run with a variety of values for α and λ. For each algorithm and parameter values, we ran 20 episodes, and then repeated for 30 independent runs. The performance measure for each combination of algorithm and parameter values was the number of time steps to reach the goal, averaged over the first 20 episodes and then over the 30 runs. All the algorithms used the same random starting positions for the episodes.

Figure 3 shows the performance in detail, for all values of λ and α. Figure 4 is a summary of the results, showing only the best performance of each algorithm at each λ value. EG-Sarsa generated shorter episodes than conventional Sarsa at almost all combinations of parameter settings. Moreover, for all λ values, EG-Sarsa was about 25% faster at the best α, and the differences are statistically significant.

6 DISCUSSION

Why does EG speed up learning on the mountain-car task, but not on the random walk task? In order to answer this question, we generated histograms of the weights that were learned on one run for one pair of λ and α values, after a longer learning period.

Figures 5 and 6 present the histograms for the two algorithms on the random walk task after 100 walks.

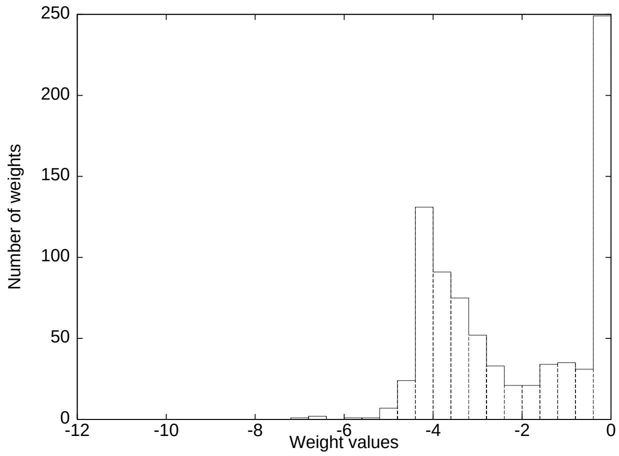

Figure 7: Q-value histogram for one action, Sarsa

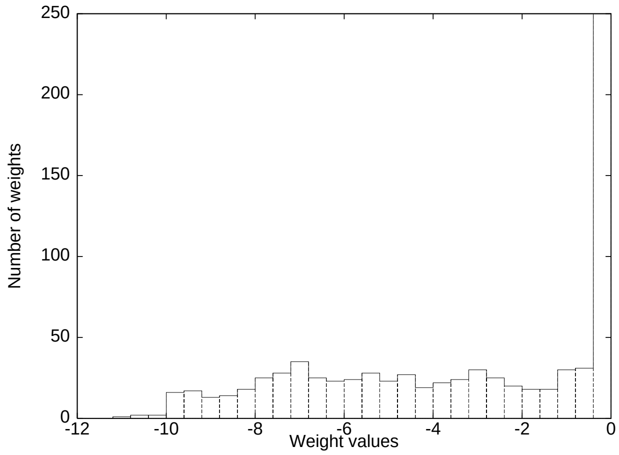

Figure 8: Q-value histogram for one action, EG-Sarsa

Figures 7 and 8 present the same histograms for the mountain car task after 500 episodes. The shapes of the two sets of histograms are different. In the random walk, the weights are more evenly distributed. Moreover, their is no preponderance of small valued weights. In the mountain-car task, the distribution is quite uneven, with a large number of weights close to 0. The weight distributions generated by the two learning algorithms also appear to be different.

In supervised learning, EG methods have been shown to converge faster than gradient descent in problems with many irrelevant input features. These results support the hypothesis that a similar situation arises when using EG methods for reinforcement learning.

7 CONCLUSIONS

We have presented a straightforward extension of exponentiated gradient methods to reinforcement learning. The empirical results suggest that exponentiated gradient variations of reinforcement learning methods can yield significantly faster learning when used with linear function approximators. This improvement depends on the task that needs to be solved. More precisely, exponentiated gradient methods can be expected to learn faster if the representation is such that many input features are irrelevant. More experiments with more tasks are needed to determine how often and how easily this characteristic occurs.

When using exponentiated gradient methods, one needs to pay slightly more attention to prevent overflow of the weights. The use of replacing traces, instead of accumulating traces, helps deal with this issue. The algorithms are also sensitive to the initial weight magnitudes. Small initial values can lead to slow convergence. Overall, however, we did not find EG methods significantly harder to tune than conventional reinforcement learning methods.

Acknowledgments

The authors thank Manfred Warmuth, Paul Utgoff, and Jyrki Kivinen for encouragement, ideas, and comments contributing to this paper, and Amy McGovern, for proof-reading. This research was supported by NSF Grant ECS-9511805 to Andrew G. Barto and Richard S. Sutton. Doina Precup also acknowledges the support of the Fulbright Foundation.

References

Albus, J. S. (1981). *Brain, behaviour and robotics*, chapter 6, pp. 139-176. Byte Books.

Kivinen, J., & Warmuth, M. K (1994). Exponentiated gradient versus gradient descent for linear predictors, Technical Report UCSC-CRL-94-16, CSE Dept., University of California, Santa Cruz.

Lewis, D. D., Schapire, R. E., Callan, J. P., & Papka, R. (1996). Training algorithms for linear text classifiers. *Proceedings of the Nineteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (pp. 298-315). Zurich, Switzerland.

Littlestone, N. (1988). Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning, 2*, 285-318.

Miller, W. T., Glanz, F. H., & Kraft, L. G. (1990). CMAC: An associative neural network alternative to backpropagation. *Proceedings of the IEEE, 78*, 1561-1567.

Papka, R., Callan, J. P., & Barto, A. G. (1996). *Text-based information retrieval using exponentiated gradient descent*, to appear in Neural Information Processing Systems, 1996.

Precup, D., & Sutton, R. S. (1996). Empirical comparison of gradient descent and exponentiated gradient descent in supervised and reinforcement learning, Technical Report 96-70, CS Dept., University of Massachusetts, Amherst.

Rummery, G. A., & Niranjan, M. (1994). *On-line Q-learning using connectionist systems*, Technical report CUED/F-INFENG/TR 166 Cambridge University Engineering Department.

Singh, S.P., & Sutton, R.S. (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning , 22*, 123-158.

Sutton, R. S. (1988). Learning to predict by the method of temporal differences. *Machine Learning, 3*, 9-44.

Sutton, R.S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in Neural Information Processing Systems 8* (pp. 1038-1044,). MIT Press.

Watkins, C.J.C.H. (1989). *Learning with delayed rewards.* Doctoral dissertation, Psychology Department, Cambridge University.