Reinforcement Learning with Echo State Networks

István Szita, Viktor Gyenes, and András Lőrincz*

Eötvös Loránd University, Pázmány P. sétány 1/C, Budapest, Hungary, H-1117
{szityu@eotvos.elte.hu, gyenesvi@inf.elte.hu, andras.lorincz@elte.hu}
http://nipg.inf.elte.hu/

Abstract. Function approximators are often used in reinforcement learning tasks with large or continuous state spaces. Artificial neural networks, among them recurrent neural networks are popular function approximators, especially in tasks where some kind of of memory is needed, like in real-world partially observable scenarios. However, convergence guarantees for such methods are rarely available. Here, we propose a method using a class of novel RNNs, the echo state networks. Proof of convergence to a bounded region is provided for k-order Markov decision processes. Runs on POMDPs were performed to test and illustrate the working of the architecture.

1 Introduction

Reinforcement learning (RL) has become a popular method for training computer agents how to behave in complex scenarios, based on rewards and penalties received from the environment. As opposed to supervised learning, no explicit training examples are needed, instead, the agent is let to explore its environment. The trainer only needs to provide feedback on whether the actions had a positive or negative outcome in the form of possibly delayed rewards.

Traditional RL frameworks focus on Markov decision problems (MDPs) [1]; here, the state representation that forms the basis of an agent's decisions has the Markov property. The Markov property means that state signals retain all information relevant for decision making. An ideal state signal would summarize the past sensations compactly and yet, it would keep all relevant information.

Most learning methods in MDPs are concerned with learning a mapping from state-action pairs to values [1], which reflect the expected long term reward gained by choosing specific actions. In case of large state spaces that are intractable for tabular methods, function approximators are used to maintain the value mapping. Artificial neural networks are popular function approximators, and recurrent neural networks are spreading to be used in tasks which require memory, since they naturally retain information about previous states. However, proofs of convergence are only available in some special cases, like linear function approximators. Gordon's results about linear function approximators [2] ensure

* Corresponding author.

S. Kollias et al. (Eds.): ICANN 2006, Part I, LNCS 4131, pp. 830–839, 2006.

convergence to a bounded region when the Sarsa(λ) method is used. As far as we know, no positive result about ANNs have been reported so far.

In this paper, we propose a novel method utilizing echo state networks (ESN) [3] as function approximators in reinforcement learning. ESNs are novel kind of recurrent neural networks proposed by Jaeger. These networks were found to be particularly well suited for learning and predicting time series [4]. This way, ESNs are promising candidates for partially observable problems where information about the past may improve performance (e.g. k-order Markov processes).

An important aspect of these networks compared to other RNNs is that they are relatively easy to train. ESNs have random recurrent connections, and only feedforward output connections are trained in a supervised, least squares manner. Since ESNs are effectively linear function approximators acting on the *internal state* representation built from the previous observations, Gordon's results about linear function approximators [2] can be transferred to the ESN architecture. Building on this, we provide proof of convergence to a bounded region for ESN training in the case of k-order Markov decision processes. To the best of our knowledge, this is the first positive result about the convergence of artificial neural networks used for value prediction in reinforcement learning tasks.

2 Related Work

Artificial neural networks have widely been used as function approximators in RL for maintaining the value function of an agent [5, 6]. On the contrary, only limited work has already been done using recurrent neural networks, probably because of difficulties in training such networks. RNNs have the ability to retain state over time, because of their recurrent connections, and they are promising candidates for compactly storing moments of series of observations.

One of the first results with RNNs used for RL was achieved with Elman-style recurrent networks [7]. An Elman network [8] differs from a multi-layer feedforward neural network in that it has *context units*, which hold copies of the hidden unit activations of the previous time step. Because the hidden unit activations are partly determined by the context unit activations, the context units can, in principle, retain information from many time steps ago. Elman networks have also been used for RL-like tasks by Glickman et. al. [9]. They utilized an evolutionary algorithm to train the connection weights of the networks.

Perhaps the most similar work to ours is the work of Bakker [10, 11], who used two types of RNNs for RL tasks that require memory, focusing on tasks that are not fully observable, and investigated tasks with long term dependencies between events. He emphasizes the difficulty in discovering the correlation between a piece of information and the moment at which that information becomes relevant. As a solution, he introduced long short-term memory networks [10].

Various other recurrent neural network approaches have also been proposed. The interested reader is referred to a detailed review of Schmidhuber [12], whose work in the field precedes Bakker's work considerably. However, it must be noted, that none of these works provide convergence guarantees.

3 Short Introduction to ESNs

Recurrent neural networks are efficient black-box models of nonlinear systems. Their feedback connections are able to maintain an ongoing activation even in the absence of inputs. In principle, general RNNs can learn to mimic a target system with arbitrary accuracy. However, training methods available for RNNs (back-propagation through time, real-time recurrent learning, extended Kalman-filtering, etc., [3]) have not been widely employed in technical applications because of their slow convergence.

In the ESN approach, a larger-than-normal layer of neurons is used with random recurrent connections. They are not modified during training, they serve as a dynamic 'reservoir'. The network has an output layer, and may also have an input layer. Input-to-hidden layer connections are randomly generated and are not modified during training. The hidden-to-output connections are trained and training becomes a simple linear regression task. Upon tuning, the output weights minimize the squared error on the training sequence.

The idea behind the ESN approach is that the activations in the hidden layer will mimic systematic variations of the teacher sequence: traces of the hidden layer activations echo spatial and temporal combinations of the previous inputs. It is important that the 'echo' signals be richly varied. This is ensured by a sparse connectivity in the hidden layer, the reservoir. For more details, see [3].

4 Reinforcement Learning with ESNs

A large family of RL algorithms uses value function estimation: they estimate how good it is for the agent to perform a given action in a given state. The notion of 'how good' is defined in terms of expected cumulated future rewards. Value functions are defined with respect to particular policies. The value of taking action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$ under policy $\pi = \pi(s, a)$ is the expected return starting from s, taking action a, and thereafter following policy π and it is denoted by $Q^\pi(s, a)$:

$$Q^\pi(s, a) = E_\pi\{R_t | s_t = s, a_t = a\}, \tag{1}$$

where policy $\pi(s, a)$ is a probability distribution function $\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ that determines the probability of each action in each state, r_t is the immediate reward received in time step t, $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$ is the cumulated discounted reward received after time step t, and γ is the discount factor.

Agents can either use value tables, or parameterized function approximators to maintain the value function. A popular update method in RL is the so called SARSA update [1], in which the value function is updated by bootstrapping, using the immediate reward and our estimate for the next step:

$$Q(s_t, a_t) \leftarrow (1 - \alpha)Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1})], \tag{2}$$

The temporally extended version of the update is the SARSA(λ) update, which virtually looks ahead more than one steps to incorporate future rewards discounted by λ into the immediate reward. For $\lambda = 0$, it yields the SARSA update, whereas for $\lambda = 1$, it yields the Monte Carlo update, which takes a whole episode into account. For details, see [1]. Equations for training linear function approximators of the form $\hat{V}_t(s) = \boldsymbol{\theta_t}^T \boldsymbol{\phi_s}$ by gradient descent are also available:

$$\boldsymbol{\theta_{t+1}} \leftarrow \boldsymbol{\theta_t} + \alpha_t(q_t - Q_t(s_t, a_t))\boldsymbol{\phi_{s_t}}, \tag{3}$$

where $\boldsymbol{\theta_t}$ is the weight vector to be tuned, α_t is a learning rate, q_t is the t^{th} training example, $Q_t(s_t, a_t)$ is our estimated value as a function of $\boldsymbol{\theta_t}$ and $\boldsymbol{\phi_{s_t}}$ is the vector of features corresponding to state s_t.

ESNs can be viewed as linear function approximators acting on an *internal state* developed from a series of previous inputs, thus incorporating the past into the state representation. The observation at time step t alone is not sufficient to choose an optimal action, but the internal representation should be more adequate since it is more likely to have the Markov property.

For a formal description, let x_t be the input to the ESN, and let u_t be the internal representation of the network at time step t. Let G be the input weight matrix, and F be the recurrent matrix of the ESN, which are defined to be random sparse matrices with a spectral radius less than one [3]. Then the internal state is calculated by the following equation:

$$u_t = \sigma(Fu_{t-1} + Gx_t), \tag{4}$$

where σ is some nonlinearity, usually the tanh function. Suppose we have k actions $a_1, ..., a_k$ to choose from. Then the network will have k outputs, one corresponding to each of the actions. We want to train an output matrix H, so that the i^{th} output yields the Q value of the i^{th} action in state u:

$$Q(u, A_i) = H_i u \tag{5}$$

where H_i denotes the i^{th} row of H. To achieve this, from (3) we can derive the following update rule:

$$H_i^{t+1} \leftarrow H_i^t + \alpha_t(training\_value^t - esn\_output_i^t)u_t. \tag{6}$$

5 Theoretical Results

Although proofs of convergence are available for many tabular RL algorithms, the case of function approximators seems to be somewhat more problematic. Even in the simplest linear function approximator case, learning may diverge [13]. The use of neural networks seems even more difficult.

There is a positive result about linear function approximators: the TD(λ) method can be used in a convergent manner to evaluate a fixed strategy. If the strategy is adapted (instead of evaluating a fixed strategy), then care must be taken, because even Q-learning, the simplest extension of the TD method, may diverge [14]. On the other hand, Gordon had shown [15] that using the SARSA algorithm, the value function converges to a bounded region. At the same time,

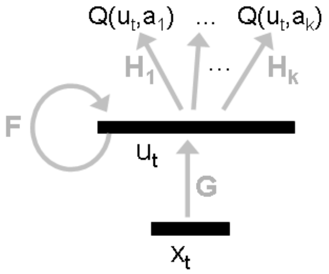

Fig. 1. Value learning with ESN. G is the input matrix, F is the matrix of recurrent weights, which are random sparse matrices. The output matrix H is tuned by the least square method to yield the Q-values.

he showed [2], that using a linear function approximator with SARSA, the value function may oscillate, and also gave a sufficient condition for the algorithm to converge.

Building on these results, we show that using an ESN as a nonlinear function approximator with the SARSA(λ) algorithm, the value function also converges to a bounded region, if the task to learn is an MDP. What is more, we also show that this result holds for k-order MDPs, too. This extension is made available by the memory present in the ESN representation. As far as we know, this is the first such result for (recurrent) neural networks.

Theorem 1 (Gordon). *Assume that a finite MDP is given and SARSA(λ) learning is being used with a linear function approximator. If the learning rates satisfy the Robbins-Monro conditions ($\alpha_t > 0$, $\sum_{t=0}^{\infty} \alpha_t = \infty$, $\sum_{t=0}^{\infty} \alpha_t^2 < \infty$), then the weight vector sequence generated by the algorithm converges to some bounded region R with probability 1.*

We note that the proof for finite MDPs can trivially be extended for continuous state space. Now, let us consider what the ESN-SARSA algorithm does: (1) the observations x_t are nonlinearly mapped to the continuous internal representation u_t, (2) on this representation a linear function approximator is trained using the SARSA method. This means that if the process u_t has the Markov property, Gordon's theorem can be applied.

Let us suppose that the input vectors x_t only contain ±1 entries.

Definition 2 (k-step unambiguous ESN). *Given is an ESN with initial state u_0. Let us suppose, that the input sequence $x_0, \ldots, x_t$ results in an internal state u, and the input sequence $x'_0, \ldots, x'_{t'}$ results in an internal state u'. We say, that the ESN is k-step unambiguous, if $u = u'$ implies that $x_{t-i} = x'_{t'-i}$ for all $i = 0 \ldots k-1$.*

Definition 3 (unambiguous input matrix). *The matrix G of size $n \times m$ is said to be an unambiguous input matrix, if for any nonzero vector $z \in \mathbb{R}^m$ such that $z_i \in \{-1, 0, 1\}$ $(i \in \{1, \ldots, m\})$, Gz is also nonzero.*

Lemma 4. *Let G be a matrix of size $n \times m$ ($n > m$), whose entries are uniform random values from the set $\{-C, 0, C\}$. The probability that G is an unambiguous input matrix, is at least $1-(1/3)^{n-m}$.*

Proof. Let z be a fixed vector with entries 0 or ± 1, and let us suppose that its first entry is nonzero. Let the i^{th} row of G be denoted by G_i. The first entry of the row can have three possible values (0 or $\pm C$), from which at most one makes $G_i z = 0$. This means that at most one third of the row's possible settings make $G_i z = 0$. Having n rows, the probability that for all i, $G_i z = 0$ is at most $(1/3)^n$. Finally z can take $3^m - 1$ different values, thus the probability that for any of these values, $Gz = 0$ for a randomly selected matrix G, is at most $(1/3)^n \cdot (3^m - 1) < (1/3)^{n-m}$.

The following lemma states that if the input weights are significantly greater than the recurrent weights, then the $x \to u$ mapping is unambiguous.

Lemma 5. *Let the entries of the G matrix of the ESN be randomly chosen uniformly from the set $\{0, \pm C\}$, where $C > \sqrt{n}$. Let the recurrent matrix F be a sparse random matrix with $\|F\| < 1$. Then the ESN is 1-step unambiguous with probability $1-(1/3)^{n-m}$.*

Proof. The input matrix is unambiguous with probability $1-(1/3)^{n-m}$, so let us suppose for now that G is unambiguous. Indirectly, let us suppose that there exists two input series whose last elements are different, but they transfer the ESN to the same internal state. This boils down to the existence of such $x_1 \neq x_2$ and u_1, u_2 that

$$u_t = \sigma(Fu_1 + Gx_1) = \sigma(Fu_2 + Gx_2).$$

Then

$$F(u_1 - u_2) = G(x_2 - x_1),$$

and by denoting $z = (x_2 - x_1)/2$

$$F\frac{u_1 - u_2}{2} = Gz.$$

G is an unambiguous input matrix, thus at least one component of Gz is nonzero. However, the nonzero terms of Gz have the form $(\pm C) \cdot (\pm 1)$, so $\|Gz\| \geq C$. On the other hand, the entries of u_i are real values between -1 and 1, thus

$$\|F\frac{u_1 - u_2}{2}\| \leq \|F\| \|\frac{u_1 - u_2}{2}\| < 1 \cdot \sqrt{n},$$

which is a contradiction.

Lemma 6. *If the ESN is 1-step unambiguous, then it is k-step unambiguous for all $k \geq 1$.*

Proof. We proceed by induction on k: for $k = 1$ the lemma holds, and by supposing that it holds for all $n < k$, we show that it also holds for $k + 1$. For $j = 0, 1, \ldots, t$, let

$$u_{t-j} = \sigma(Fu_{t-j-1} + Gx_{t-j}),$$

and similarly, for $j = 0, 1, \ldots, t'$, let

$$u'_{t'-j} = \sigma(Fu_{t'-j-1} + Gx_{t'-j}),$$

and let us suppose that $u_t = u'_{t'}$. Since the ESN is 1-step unambiguous, then $x_t = x'_{t'}$, and this implies $u_{t-1} = u'_{t'-1}$. Using our supposition that the ESN is k-step unambiguous, it must hold for the previous k observations that $x_{t-1} = x'_{t'-1}, \ldots, x_{t-k} = x'_{t'-k}$, which means that the ESN is $k+1$-step unambiguous.

Definition 7 (induced ESN decision process). *Given is the following decision process, denoted by M: X is the finite state space, A is the finite action space, $(X \times A)^*$ denotes the space of series made of elements of $X \times A$, $R : X \to \mathbb{R}$ is the reward function, $P : (X \times A)^* \times X \to [0,1]$ is the transition probability function, that gives the probabilities of the next states based on the trajectory of states travelled and actions applied so far. The ESN-decision process induced by the decision process M is the following: $U = \mathbb{R}^n \cup Z$ is the state space, where Z is an abstract terminal state, A is the action space; and $P' : U \times U \to [0,1]$, $R' : U \to \mathbb{R}$. P' and R' are defined as follows: for any $u \in \mathbb{R}^n$ let us observe the sequence set*

$$S(u) = \{s = (x_0, a_0, x_1, \ldots, a_{t-1}, x_t, a_t) \mid \text{ESN input } s \text{ results in internal state } u\}.$$

The following three cases must be treated separately:

- *$S(u) = \emptyset$: then let $P'(Z|u,a) := 1$, $P'(v|u,a) := 0$ and $R'(u) := 0$ for all a and $v \neq Z$.*
- *for all $s \in S(u)$ the probabilities $P(.|s)$ are all the same. Then*

$$P'(v|u,a_t) := \begin{cases} P(x|s), & \text{if } \exists x : v = \sigma(Fu + Gx) \\ 0, & \text{otherwise.} \end{cases}$$

- *if the value of P is ambiguous on the elements of $S(u)$, then let $P'(Z|u,a) := 1$, $P'(v|u,a) := 0$ and $R'(u) := 0$ for all a and $v \neq Z$.*

If the last case does not occur for any $u \in \mathbb{R}^n$, then the induced decision process is said to be well defined.

The previous lemmas imply our main theorem:

Theorem 8. *Given is a k-step unambiguous ESN, and given is an $M := (X, A, P, R, x_0)$ k-order MDP, then the decision process induced by the ESN is well defined, furthermore, the decision process is an MDP, on which the value function sequence generated by the ESN-SARSA algorithm converges to a bounded region.*

Note: The theorem gives the important result that the ESN-SARSA algorithm is convergent to a region (that is, the algorithm cannot diverge) for *any k*. This is because it only states that the algorithm is convergent, it does not tell anything about the speed of convergence and how good the resulting value function will be. As k is increased, the effect of earlier steps decreases exponentially, which means that the temporal resolution of the function approximator about the far past will become poorer.

6 Test Runs

There are several benchmark tasks to test RL algorithms for partially observed environments. These tasks typically require some amount of memory. We have tested the ESN-RL architecture on some of these tasks. In these tests, the ESN was always trained 'on-line'.

The first problem was described by Littman et al. [16]. You stand in front of two doors: behind one door is a tiger and behind the other is a vast reward. You may open either door, receiving a large penalty if you chose the one with the tiger and a large reward if you chose the other. You have the additional option of simply listening. If the tiger is on the left, then with probability $1 \geq p > 0.5$ you will hear the tiger on your left and with probability $(1 - p)$ you will hear it on your right; symmetrically for the case in which the tiger is on your right. If you listen, you will pay a small penalty. The problem is this: How long should you stand and listen before you choose a door?

By varying the value of p the difficulty of the task varies; as p decreases, the task gets more difficult, more listening is needed to safely determine the position of the tiger (note, that at $p = 0.5$, one can not do better than to guess randomly). Figure 2 shows our results on the tiger problem. It can be seen, that as p increases, the ESN learns that less listening is needed, and its performance also increases up to 1, when the task becomes deterministic. In this case, the ESN learns to answer after 1 round of listening. The number of listenings learned by the ESN is similar to the results in [16]. At $p = 0.85$, their system learned to listen until it hears two more roars from one side then the other. This is comparable to the ESN's result of listening 2.37 times on average.

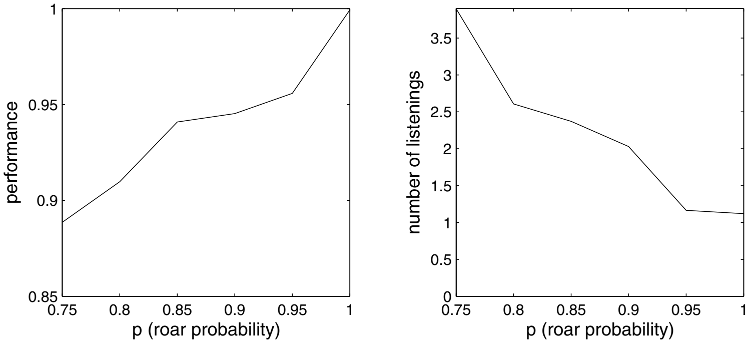

Fig. 2. Results on the tiger problem. Left: performance, right: number of listenings needed to safely open the right door. Results are averages of 100 runs

The second problem was a simple 4x3 maze example proposed by Russell and Norvig [17]. The maze has an obstacle in the middle, and has two special states, one which gives a reward of +1, one which gives a penalty of -1. The actions, N, S, E, W, have the expected result 80% of the time, and transition in a direction perpendicular to the intended with a 10% probability for each direction. Observation is limited to two wall detectors that can detect when a wall is to the left or right. The task is to find the +1 state repeatedly and avoid the -1 state, starting from a random position. As Figure 3 shows, the ESN was also able to learn this navigation problem. It must be noted that adding the agent's own previous action to the observations in the next time step increased the stability of the ESN, which might be because of the heavy partially observable nature of the task, probably being compensated by considering the previous moves.

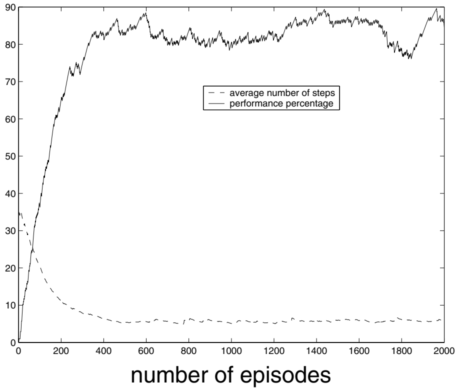

Fig. 3. An example run on the 4x3 maze problem. The performance goes up to 90 percent, and the average number of steps of the agent is around 5: the agent has learned the routes to the goal

We also tested the architecture on a T-maze problem well suited to test the state retaining properties in memory required tasks [10]. At the beginning of a T-shaped maze, the agent is shown a sign indicating whether it should to turn left or right at the end. By varying the length of the T-shape, an algorithm can be tested how long it is able to retain previous observations. Whether the ESN approach was able to learn the task seemed to depend on the randomly generated G and F matrices. With proper matrices, the agent easily learned the task. The matrices are thought to be proper, if due to the random connections, the activation resulting from observing the sign is maintained long enough to effect the decision at the end. The larger the k in the Markov process, the less probable that the random matrices will be proper. For $k < 10$, almost 100% of the matrices were proper; training was always successful. As k was increased, this percentage slowly fell, and reached 0 at $k = 25$, being around 50% at $k = 20$.

7 Conclusions

An echo state network approach for maintaining the value function of an agent in reinforcement learning tasks was proposed. In principle, the ESN is able to compactly summarize recent observations in an internal state, upon which the networks output weights can be trained to act as a linear function approximator. Theoretical results ensure convergence to a bounded region when the task is a k-order MDP. Tests of the architecture on artificial problems predict that the ESN approach may be a simple method for partially observable MDP tasks.

References

[1] Sutton, R., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA (1998)

[2] Gordon, G.J.: Chattering in SARSA(lambda) - a CMU Learning Lab Internal Report (1996)

[3] Jaeger, H.: Tutorial on training recurrent neural networks, covering BPTT, RTRL, EKF and the 'echo state network' approach. Technical Report GMD Report 159, German National Research Center for Information Technology (2002)

[4] Jaeger, H., Haas, H.: Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless telecommunication. Science (2004) 78–80

[5] Tesauro, G., Sejnowski, T.J.: A parallel network that learns to play backgammon. Artificial Intelligence **39** (1989) 357–390

[6] Bertsekas, D.P., Tsitsiklis, J.N.: Neuro-Dynamic Programming. Athena Scientific, Belmont, MA (1996)

[7] Lin, L.J., Mitchell, T.M.: Memory approaches to reinforcement learning in non-markovian domains. Technical Report CMU-CS-92-138, Carnegie Mellon University, Pittsburgh, PA (1992)

[8] Elman, J.L.: Finding structure in time. Cognitive Science **14** (1990) 179–211

[9] Glickman, M.R., Sycara, K.: Evolution of goal-directed behavior from limited information in a complex environment. In: Proc. of the Genetic and Evol. Comp. Conf., Orlando, Florida, USA, Morgan Kaufmann (1999) 1281–1288

[10] Bakker, B.: Reinforcement learning with long short-term memory. Advances in Neural Information Processing Systems **14** (2002) 1475–1482

[11] Bakker, P.B.: The State of Mind - Reinforcement Learning with Recurrent Neural Networks. PhD thesis, Universiteit Leiden (2004)

[12] Schmidhuber, J.: Making the world differentiable. Technical Report TR-FKI-126-90, Institut für Informatik, Technische Universität München (1990)

[13] Baird, L.C.: Residual algorithms: Reinforcement learning with function approximation. In: International Conference on Machine Learning. (1995) 30–37

[14] Watkins, C.J.C.H.: Learning from Delayed Rewards. PhD thesis, Cambridge University, Cambridge, UK (1989)

[15] Gordon, G.J.: Reinforcement learning with function approximation converges to a region. In: Advances in Neural Information Processing Systems. Volume 13., Cambridge, MA, MIT Press (2001) 1040–1046

[16] L. P. Kaelbling, A.R.C., Littman, M.L.: Acting optimally in partially observable stochastic domains. In: Proc. of the 12th Nat'l Conf. on Artif. Intell. (1994)

[17] Russell, S.J., Norvig, P.: Artificial Intelligence: a Modern Approach. Prentice-Hall, Englewood Cliffs, New Jersey (1994)