Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 00 (2014) 000–7

Procedia Computer Science

Complex Adaptive Systems, Publication 4
Cihan H. Dagli, Editor in Chief
Conference Organized by Missouri University of Science and Technology
2014 - Philadelphia, PA

How to Rein in the Volatile Actor: A New Bounded Perspective

Abhijit Gosavi [*]

Missouri University of Science and Technology, 219 Engineering Management,Rolla, MO 65409, USA

Abstract

Actor-critic algorithms are amongst the most well-studied reinforcement learning algorithms that can be used to solve Markov decision processes (MDPs) via simulation. Unfortunately, the parameters of the so-called "actor" in the *classical* actor-critic algorithm exhibit great volatility — getting unbounded in practice, whence they have to be artificially constrained to obtain solutions in practice. The algorithm is often used in conjunction with Boltzmann action selection, where one may have to use a temperature to get the algorithm to work, but the convergence of the algorithm has only been proved when the temperature equals 1. We propose a new actor-critic algorithm whose actor's parameters are bounded. We present a mathematical proof of the boundedness and test our algorithm on small-scale MDPs for infinite horizon discounted reward. Our algorithm produces encouraging numerical results.

Keywords: Actor critics; adaptive critics; reinforcement learning; boundedness; stochastic policy search.

1. Introduction

Reinforcement Learning (RL) is an artificial intelligence technique that has been widely used within simulators to solve Markov decision processes (MDPs). One major advantage of RL is that it does not require the setting up of transition probabilities (TPs) underlying the MDP. Setting up the TPs is a key difficulty encountered in solving many real-world MDPs. In a simulation-based setting, one only needs the distributions of the input random variables — in order to use RL. Obviously, this is very advantageous for problems with numerous input random variables — where in general, the closed form of TPs involve integrals and complex expressions. Also, setting up these expressions and then computing them numerically is often tedious and time-consuming. As a consequence, RL is becoming increasingly popular for solving MDPs, because it bypasses this tedious step of generating TPs ; all it needs is a simulator. For relatively small-scale problems with up to 500 state-action pairs, it is possible to integrate the RL

[*]Tel: 573-341-4624; Email: gosavia@mst.edu

algorithm within popular simulation packages, e.g., ARENA [1], and this has increased the accessibility of RL in the operations research community.

The oldest RL algorithm is the so-called actor-critic [2]. It actually predates the more popular Q-Learning algorithm [3]. The actor-critic's most significant drawback, which has plagued it for years, preventing it from being as popular as Q-Learning, is the parameters estimated by the so-called actor becoming unbounded. A key question is: *how does one rein in this volatile actor in order to get the algorithm to work in practice?* One approach is to artificially bound the actor's values [4, 5]. This, usually causes sub-optimal solutions to be generated; the convergence of the algorithm has been proved to produce ϵ-optimal solutions in [5] when such a bounding is performed, which unfortunately does not guarantee the generation of an optimal solution. Another issue is that of action selection during the learning. In [5], the convergence is shown under the well-known Boltzmann selection rule under the specific case in which the temperature is constant and is set at 1. In this paper, we propose a new actor critic whose values remain naturally bounded and do not need the artificial constraining. The specific problem we study is an MDP for infinite time horizon and a discounted reward performance metric. We provide step-by-step details of the problem and numerical results on a small problem where the algorithm generates the optimal solution in every case and the value function it generates is also very close to that generated by Q-Learning.

Contributions of this paper: The contributions of this paper are threefold. First, we present a new actor-critic algorithm in which the actor, known for its "volatility," remains "stable" (bounded). The second contribution is to mathematically prove the boundedness of the actor's values. The third contribution is in showing how the new algorithm fares on small-scale MDPS; in particular, its performance is encouraging and it produces optimal solutions.

2. Reinforcement Learning: A Short Review

An MDP is a control-optimization problem in which one seeks to select the best action in each state such that some performance metric is optimized for the discrete-event system, driven by Markov chains, as a whole [6, 7]. The Markov chain is characterized by the so-called "states;" thus, in a Markov chain model, the system transitions from one state to another — typically in a random manner. The TPs are an inherent part of the model. In the MDP model, each action induces transitions according to its own set of TPs. Thus, these TPs depend on the state and the action chosen in the state. An important property in the context of the MDP is that the probability of transitioning from a given state to another, for any given action, does not depend on how many transitions have occurred thus far in the system. This is an important memoryless property that is key to the applicability of the solution algorithms rooted in dynamic programming (DP).

DP, developed by Bellman [8] and Howard [9], is a field that provides algorithms for solving MDPs. The equation developed by Bellman is called the Bellman equation of optimality, while the same developed by Howard goes by names such as Poisson equation or Bellman equation for a given policy. DP is preferred when the TPs can be estimated, which is typically possible when one has a few states, e.g., up to 1000. Beyond that, DP breaks down. The reason for the breakdown is that it is not possible to store matrices exceeding a million elements in computers. It is also the case that developing TPs for large-scale systems is never straightforward. A number of real-world systems can be modeled via Markov chains and MDPs. While industrial engineers have been using Markov chains for years in modeling production systems, more recently computer scientists have used them in robotics. Of course, computer scientists sought to solve MDPs without generating TPs, which has led to the birth of RL. The use of Markov chains in robotics has been an exciting development in building thinking robots that can move from one spot to another avoiding obstacles. Most of these robots tend to use versions of Q-Learning. The use of Markov chains in the PageRank algorithm [10] has perhaps significantly enhanced the popularity of Markov chains in the world of computing.

The power of RL is that it can be used on problems with bigger state spaces because RL does not require TPs [11, 4, 12], i.e., RL uses a version of DP that bypasses the TPs. RL combines stochastic approximation with DP and uses a format of the Bellman equation that is devoid of the TPs. Of course, because of the stochastic approximation underlying the equation, RL can be used within simulators. In the artificial intelligence community, RL is not used within simulators but rather in an online sense where each state transition is the result of a real trial in the actual system.

RL techniques have now been widely employed for solving problems in supply chains [13, 14, 15], semi-conductor manufacturing [16], and preventive maintenance [1, 17]. RL and its close relatives have also been used in wide variety of other areas such irrigation control [18], sensor placement [19], vehicle cruise control [20], and variable speed-limit control [21].

3. Actor-critics: Preliminaries

In this section, we present some preliminaries of the actor-critic framework. We begin with some mathematical notation that will be needed in the rest of this paper: $\mathcal{S}$ will denote the set of states in the MDP; $\mathcal{A}(i)$, the set of actions that can be selected in state i; μ, a deterministic stationary policy; $\mu(i)$, the action selected in state i under policy μ; $p(i,a,j)$, the probability of a transition from i to j when action $a \in \mathcal{A}(i)$ is selected; $r(i,a,j)$, the immediate reward earned in one transition from i to j when action $a \in \mathcal{A}(i)$ is selected; $\bar{r}(i,a) = \sum_{j \in \mathcal{S}} p(i,a,j) r(i,a,j)$, the mean reward earned in one transition from i to j when action $a \in \mathcal{A}(i)$ is chosen; $\mathbf{P_a}$, the transition probability matrix (TPM) when action a is selected in every state such that the (i,j)th element of this matrix is denoted by $P_a(i,j)$ and equals $p(i,a,j)$; and $\mathbf{R_a}$, the transition reward matrix (TRM) when action a is selected in every state such that the (i,j)th element of this matrix is denoted by $R_a(i,j)$ and equals $r(i,a,j)$. The objective underlying the problem here is to maximize the (total) discounted reward for an infinite horizon MDP [7] can be shown to be as follows: Maximize over all policies μ, for each starting state state i: $\Lambda_\mu(i) = \liminf_{K \to \infty} \mathsf{E}\left[\sum_{k=1}^{K} \lambda^{k-1} r(x_k, \mu(x_k), x_{k+1}) | x_1 = i\right]$, where λ denotes the discount factor, x_k is the state from which the kth jump of the Markov chain occurs in the trajectory, and $\mathsf{E}[.]$ denotes the expectation over the trajectory.

The main result underlying the solution of the discounted reward MDP is as follows:

Theorem 1. [7, 6] *There exists a value function $v^* : \mathcal{S} \to \Re$ satisfying the following system of equations for all $i \in S$,*

$$v^*(i) = \max_{a \in \mathcal{A}(i)} \left[\bar{r}(i,a) + \lambda \sum_{j \in S} p(i,a,j) v^*(j) \right], \quad (1)$$

such that the greedy policy μ^* formed by selecting actions that maximize the right-hand side of Eqn. (1) is optimal.

The actor critic seeks to solve the Bellman equation above without generating the TPs needed in the above. The key idea underlying the actor-critic is as follows. The so-called actor is a somewhat "volatile" agent that seeks all possible actions in each state visited, while the critic is a less reactive agent that updates its database only when it sees a sensible (greedy) action from the actor. As time passes, the actor becomes less and less volatile, and the critic and the actor together hone in on the optimal policy.

In more mathematical terms, the critic stores the value function, $J(i)$ for each $i \in \mathcal{S}$, while the actor stores an action-value function $H(i,a)$ for each $\mathcal{A}(i)$ and $i \in \mathcal{S}$. The original actor-critic had two main steps:

- **Actor Update**: Update $H(i,a)$ using step-size α, which depends on k:

$$H(i,a) \leftarrow H(i,a) + \alpha \left[r(i,a,j) + \lambda J(j) - J(i) \right]. \quad (2)$$

  If $H(i,a) > \bar{\mathsf{H}}$, set $H(i,a) \leftarrow \bar{\mathsf{H}}$. If $H(i,a) < -\bar{\mathsf{H}}$, set $H(i,a) \leftarrow -\bar{\mathsf{H}}$. The scalar $\bar{\mathsf{H}}$ is a positive number fixed at the start of the algorithm. It is the artificial bound on the actor's value alluded to above and this is the artificial constraining that we discussed above.

- **Critic Update**: Update $J(i)$, via step-size β, which depends on k: $J(i) \leftarrow (1-\beta)J(i) + \beta \left[r(i,a,j) + \lambda J(j) \right]$.

The action selection is done using the Gibbs softmax method which works as follows: An action a is selected in state i with the probability $p(i,a)$ where

$$p(i,a) = \frac{e^{H(i,a)}/T}{\sum_{b \in \mathcal{A}(i)} e^{H(i,b)}/T}, \quad (3)$$

where T denotes the so-called temperature. In the above, $H(.,.)$ can become unbounded, and hence it is bounded using $\bar{\mathsf{H}}$, which is a constant that should equal the greatest possible positive value — such that the computer does not overflow when it attempts to compute $e^{\bar{\mathsf{H}}}$. This poses a severe limitation in practice and can lead to sub-optimal policies [5]; in practice, the algorithm often requires a value much greater than the one that can be stored in the computer, and under those conditions, the resulting policy is sub-optimal.

The convergence of the actor-critic, shown in [5], holds when $T = 1$. Using $T > 1$ can help compute the exponential term in practice, but note:

1. The convergence proof is only for the case $T = 1$, and setting $T > 1$ implies sub-optimal policies potentially.
2. Using $T > 1$ does not eliminate the difficulties posed by $H(.,.)$ becoming unbounded.

In the above (i.e., the classical actor critic), clearly, as one action starts dominating, e.g., with two actions, action 1 starts dominating action 2 when $H(.,1) > H(.,2)$ is consistently true, the probabilities tend to one for the dominating action. In every transition that occurs from i to j under the action selected, a, both the actor and the critic are updated. A review of numerous actor-critic algorithms has appeared in [22].

4. A New Bounded Actor Critic

For the new algorithm that we propose we seek to keep the actor's iterates, $H(.,.)$ bounded. The change that we propose in the classical algorithm is as follows:

Update $H(i,a)$ using a step size α: $H(i,a) \leftarrow H(i,a) + \alpha\left[r(i,a,j) + \lambda J(j) - H(i,a)\right]$. The difference is in the temporal difference: Compare the update above to Equation (2). Within the square brackets of the update above, $J(i)$ of Equation (2) is replaced by $H(i,a)$. This leads to $H(.,.)$ being naturally bounded. We will present a mathematical proof later.

We now present a detailed description of our algorithm. In the algorithm, the action selection is done via an exploration strategy where the exploration is decayed with the number of iterations. An example of a decaying exploration strategy works as follows: Select an action u in state i in the kth iteration with probability p^k, where

$$u = \arg\max_{b \in \mathcal{A}(i)} H(i,b) \text{ and } \mathsf{p}^k = 1 - \frac{B^k}{k},$$

and select any of the other actions — each with a probability of $\frac{B^k}{(|\mathcal{A}(i)|-1)k}$, where for instance $B^k = 0.5$ for all values of k. In the above, action u is called the greedy action since it maximizes $H(.,.)$. Every non-greedy action is also called an exploratory action. In our action selection, if we have multiple greedy actions, we break ties randomly; thus, for each state i, we have 1 greedy action and $|\mathcal{A}(i)| - 1$ non-greedy actions. Another simple mechanism to reduce the rate of exploration is to use a geometric rate for the exploration probability, i.e., the probability of *not* selecting a greedy action. Thus, if we define q^k to be the probability of not selecting the greedy action, then we use the following rule: For each state i:

$$\mathsf{q}^1 = \frac{|\mathcal{A}(i)| - 1}{|\mathcal{A}(i)|}; \;\; \mathsf{p}^k = 1 - \mathsf{q}^k; \text{ and } \mathsf{q}^{k+1} = \mathsf{l}\mathsf{q}^k$$

where l equals a positive number slightly less than 1, e.g., 0.999. Each of the exploratory actions is chosen with the probability $\frac{q^k}{|\mathcal{A}(i)|-1}$. In general, the exploratory strategy must satisfy the following: every action is selected with the same probability at first, but gradually the algorithm starts preferring the greedy action with respect to the $H(.,.)$ values.

Step 1. Set all J-values and H-values to 0, i.e., for all l, where $l \in \mathcal{S}$ and $u \in \mathcal{A}(l)$, set $J(l) \leftarrow 0$ and $H(l,u) \leftarrow 0$. Further, set k, the number of iterations, to 0. Choose a value for $k_{\max}$ iterations, where $k_{\max}$ denotes the maximum number of iterations for which the algorithm is run. The system simulation is started at any arbitrary state.

Step 2. Let the current state be i. Select action a via an exploratory strategy, where the rate of exploration is decayed with iterations. If $a \in \arg\max_{u \in \mathcal{A}(i)} H(i,u)$, set $\phi = 0$. Otherwise set $\phi = 1$.

Step 3. (**Actor Update**) Simulate action a. Let the next state be j. Let $r(i,a,j)$ be the immediate reward earned in going to j from i under a. Update $H(i,a)$ using a a step size, α, which depends on k:

$$H(i,a) \leftarrow H(i,a) + \alpha\left[r(i,a,j) + \lambda J(j) - H(i,a)\right].$$

Step 4. (**Critic Update**) If $\phi = 1$ go to Step 5. Otherwise update $J(i)$ via the following equation using the step-size, β, which also depends on k:

$$J(i) \leftarrow (1-\beta)J(i) + \beta\left[r(i,a,j) + \lambda J(j)\right].$$

Step 5. If $k < k_{\max}$, increment k by 1, set $i \leftarrow j$, and then go to Step 2. Otherwise go to Step 6.

Step 6. For each $l \in \mathcal{S}$, select $d(l) \in \arg\max_{b\in\mathcal{A}(l)} H(l,b)$. The policy delivered is d. Stop.

As is clear, the critic updates its own iterate only when a greedy action is selected. Figure 1 explains the working mechanism pictorially. The key intuition underlying development of our new algorithm is as follows: Ideally, $H(i,a)$ should converge to the following:

$$\sum_{j\in\mathcal{S}} p(i,a,j)\left[r(i,a,j) + \lambda J(j)\right];$$

since in the limit, the algorithm selects only the greedy policy in $H(.,.)$, the action selected in updating $J(i)$ will be

$$\arg\max_{a\in\mathcal{A}(i)} \sum_{j\in\mathcal{S}} p(i,a,j)\left[r(i,a,j) + \lambda J(j)\right].$$

This suggests that $J(.)$ should converge to a solution of the following: For all $i \in \mathcal{S}$,

$$v(i) = \max_{a\in\mathcal{A}(i)}\left[\sum_{j\in\mathcal{S}} p(i,a,j)\left[r(i,a,j) + \lambda v(j)\right]\right].$$

The above is the Bellman optimality equation, which implies that the algorithm should converge to the optimal solution. Of course, all of this needs to be made mathematically precise using analysis, which is a topic for future research.

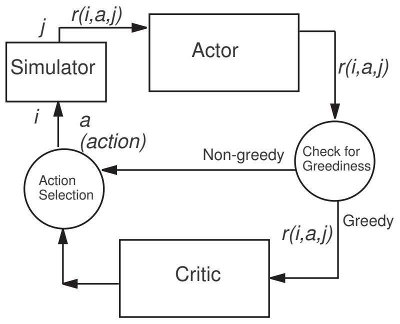

Figure 1. Working mechanism of the actor-critic

5. Boundedness of the Actor's Values

We now present our main result on boundedness. The proof follows along the lines of [23]. The main idea underlying the proof is as follows. We first show that the critic's values remain bounded. Then, the boundedness of the actor's values follows directly from there. Let J^k denote the vector of values computed by the critic and H^k denote the same by the actor in the kth iteration. Technically, $H(.,.)$ is a matrix, but we can map it into a one-dimensional vector.

Theorem 2. *The sequence $\{J^k, H^k\}_{k=1}^{\infty}$ remains bounded.*

Proof We will first prove that the vector of iterates J^k remain bounded.

Lemma 1. *The sequence $\{J^k\}_{k=1}^{\infty}$ remains bounded.*

Proof We claim that for every state i:

$$|J^k(i)| \leq M(1+\lambda+\lambda^2+\cdots+\lambda^k), \tag{4}$$

$$\text{where } M \text{ is a positive finite number defined as follows: } M = \max\left\{r_{\max}, \max_{i\in\mathcal{S}}|J^1(i)|\right\}, \tag{5}$$

$$\text{where } r_{\max} = \max_{i,j\in\mathcal{S},a\in\mathcal{A}(i)} |r(i,a,j)|. \tag{6}$$

Since $r(.,.,)$ is bounded, $r_{\max}$ must be bounded. Since we start with finite values for J, then M too must be bounded. Then, from the above claim (4), boundedness follows since if $k \to \infty$,

$$\limsup_{k\to\infty} |J^k(i)| \leq M\frac{1}{1-\lambda}$$

for all $i \in \mathcal{S}$, since $0 \leq \lambda < 1$. We now prove our claim in (4) via induction.

In asynchronous updating, the J-value of only one state is updated in a given iteration, while the other J-values remain un-updated. Hence, in the kth iteration of the asynchronous algorithm, the update for $J^k(i)$ is either according to Case 1 or Case 2.

Case 1: The state-action pair is updated in the kth iteration: $J^{k+1}(i) = (1-\beta)J^k(i) + \beta\left[r(i,a,j) + \lambda J^k(j)\right]$.

Case 2: The state-action pair is not updated in the kth iteration: $J^{k+1}(i) = J^k(i)$.

Now, if the update is carried out as in Case 1:

$$\begin{aligned} |J^2(i)| &\leq (1-\beta)|J^1(i)| + \beta|r(i,a,j) + \lambda J^1(j)| \\ &\leq (1-\beta)M + \beta M + \beta\lambda M \text{ (from (6) and (5))} \\ &\leq (1-\beta)M + \beta M + \lambda M = M(1+\lambda) \text{ (since } \beta \leq 1) \end{aligned}$$

Now, if the update is carried out as in Case 2: $|J^2(i)| = |J^1(i)| \leq M \leq M(1+\lambda)$. From the above, our claim in (4) is true for $k = 1$. Now assuming that the claim is true when $k = m$, we have that for all $i \in \mathcal{S}$.

$$|J^m(i)| \leq M(1+\lambda+\lambda^2+\cdots+\lambda^m). \tag{7}$$

Now, if the update is carried out as in Case 1:

$$\begin{aligned} |J^{m+1}(i)| &\leq (1-\beta)|J^m(i)| + \beta|r(i,a,j) + \lambda J^m(j)| \\ &\leq (1-\beta)M(1+\lambda+\lambda^2+\cdots+\lambda^m) + \beta M + \beta\lambda M(1+\lambda+\lambda^2+\cdots+\lambda^m) \text{ (from (7))} \\ &= M(1+\lambda+\lambda^2+\cdots+\lambda^m) - \beta M(1+\lambda+\lambda^2+\cdots+\lambda^m) + \beta M + \beta M(\lambda+\lambda^2+\cdots+\lambda^{m+1}) \\ &= M(1+\lambda+\lambda^2+\cdots+\lambda^m) + \beta M\lambda^{m+1} \\ &\leq M(1+\lambda+\lambda^2+\cdots+\lambda^m) + M\lambda^{m+1} = M(1+\lambda+\lambda^2+\cdots+\lambda^m+\lambda^{m+1}). \end{aligned}$$

Now, if the update is carried out as in Case 2:

$$|J^{m+1}(i)| = |J^m(i)| \leq M(1+\lambda+\lambda^2+\cdots+\lambda^m) \leq M(1+\lambda+\lambda^2+\cdots+\lambda^m+\lambda^{m+1}).$$

From the above, the claim in (4) is proved for $k = m+1$. ■

We will now show that $H(.,.)$ is bounded by R where $R = \max\left\{r_{\max} + \lambda\frac{M}{1-\lambda}, \max_{i,a}|H^1(i,a)|\right\}$. Again, we will use induction. Our claim is $|H(i,a)| < R$ for all (i,a) pairs.

From Step 3 of the actor-critic, for $k = 1$, for any (i,a) pair,

$$\begin{aligned} |H^2(i,a)| &\leq (1-\alpha^1)|H^1(i,a)| + \alpha^1|r(i,a,j) + \lambda J^1(j)| \\ &\leq (1-\alpha^1)R + \alpha^1\left|r_{\max} + \lambda\frac{M}{1-\lambda}\right| \\ &\leq (1-\alpha^1)R + \alpha^1 R = R. \end{aligned}$$

Assuming the result is true for $k = m$, i.e., $H^m(i,a) \leq R$, from Step 3 we have for any (i,a),

$$\begin{aligned} |H^{m+1}(i,a)| &\leq (1-\alpha^m)|H^m(i,a)| + \alpha^m |r(i,a,j) + \lambda J^m(j)| \\ &\leq (1-\alpha^m)R + \alpha^m \left| r_{\max} + \lambda \frac{M}{1-\lambda} \right| \\ &\leq (1-\alpha^m)R + \alpha^m R = R. \quad \blacksquare \end{aligned}$$

6. Numerical Results

We ran our algorithm on four different MDPs, where each problem has two states and two actions allowed in each. We provide the data for each problem below:
Case 1: The TPMs and TRMs are as follows:

$$\mathbf{P}_1 = \begin{bmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{bmatrix}; \mathbf{P}_2 = \begin{bmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{bmatrix}; \mathbf{R}_1 = \begin{bmatrix} 6 & -5 \\ 7 & 12 \end{bmatrix}; \mathbf{R}_2 = \begin{bmatrix} 10 & 17 \\ -14 & 13 \end{bmatrix}.$$

For the remaining cases, we list only those values where the problem differs from Case 1. **Case 2:** $r(1,1,2) = 5$; $r(2,2,1) = 14$; **Case 3:** $r(1,2,1) = 12$; **Case 4:** $r(1,1,1) = 16$. Also, $\lambda = 0.8$ for all cases.

We ran the algorithm for $k_{\max} = 10,000$ with the following learning rates: $\alpha^k = \frac{log(k+1)}{k+1}$; $\beta^k = \frac{150}{300+k}$. We use the geometric rule for exploration decay with l = 0.999. The optimal solution (policy) was determined for each case via DP (value iteration). Also where the policy is denoted by (a_1, a_2), a_1 denotes the action to be selected in state 1 and a_2 the action in state 2. The optimal policy is shown as μ^* in Table 1. Table 1 also shows the value function, $J(.)$, obtained from the new algorithm and $J^*(.)$ obtained from DP. As is clear, the value functions from our new algorithm are quite close to the same from DP. Table 2 shows the values of the actor for each case. Note that the policy delivered by the new algorithm is obtained from the actor's values. We explain how the policy is obtained for Case 1. For state 1, $H(i = 1, a = 2) = 53.00 > H(i = 1, a = 1) = 43.95$, and hence the action suggested by the algorithm for state $i = 1$ must be 2. Similarly, since $52.07 > 39.48$, the action suggested by the algorithm for state 2 must be 1. Thus, the policy $(2,1)$ is the policy delivered by the algorithm, which coincides with the optimal policy — shown in Table 1. Our actor-critic delivers the optimal policy in each case.

Figure 2 shows values of $J(1)$ and $J(2)$ as they are estimated during the run-time of the algorithm in the simulator for Case 2. As is clear, the values approach the optimal values much before the algorithm is terminated. Also, the actor's values always remained bounded in all our experiments.

Table 1. Value function obtained from actor critic and DP and actor's values from actor critic

Case #	μ^*	$J(1)$	$J(2)$	$J^*(1)$	$J^*(2)$	$H(1,1)$	$H(1,2)$	$H(2,1)$	$H(2,2)$
Case 1	(2,1)	52.93	51.67	53.03	51.86	43.95	53.00	52.07	39.48
Case 2	(2,2)	55.38	61.16	55.77	61.45	50.11	55.81	48.40	61.71
Case 3	(2,1)	60.80	56.59	60.83	56.66	49.32	60.81	56.70	43.08
Case 4	(1,1)	49.90	49.35	48.97	49.36	48.87	38.17	49.03	38.75

7. Conclusions

The actor critic or the adaptive critic is a long-standing algorithm in RL [24, 2, 25] with roots in artificial intelligence. It is based on ideas in policy iteration. Nonetheless, in practice, it has always suffered from the deficiency of the actor's values getting unbounded. The work of [5] sought to analyze the convergence properties of the algorithm; they also explicitly explain the unboundedness phenomenon by constraining the actor's values in their algorithm. Our work in this paper was directed at overcoming this difficulty of unboundedness. We redefined how the actor's values are updated; of course, this resulted in a significantly different transformation, which requires further mathematical analysis. We provided a mathematical proof of boundedness of the actor's values. Some of the future work in this area that we intend to carry out is testing the algorithm on larger problems.

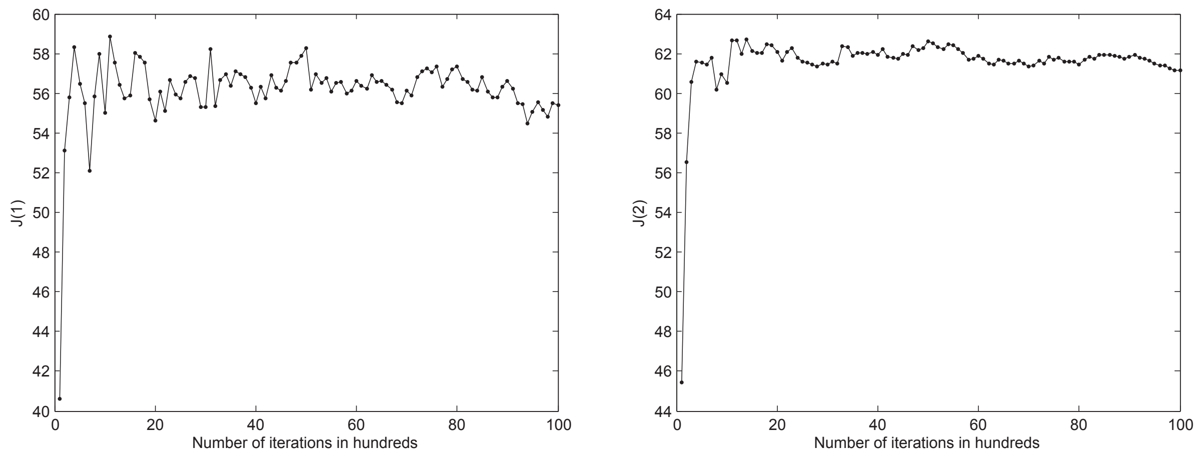

Figure 2. Run-time performance of the algorithm: Values of $J(1)$ and $J(2)$ in simulating Case #2

References

[1] T. Das, A. Gosavi, S. Mahadevan, N. Marchalleck, Solving semi-Markov decision problems using average reward reinforcement learning, Management Science 45(4) (1999) 560–574.

[2] A. Barto, R. Sutton, C. Anderson, Neuronlike elements that can solve difficult learning control problems, IEEE Transactions on Systems, Man, and Cybernetics 13 (1983) 835–846.

[3] C. Watkins, Learning from delayed rewards, Ph.D. thesis, Kings College, Cambridge, England (May 1989).

[4] R. Sutton, A. G. Barto, Reinforcement Learning: An Introduction, The MIT Press, Cambridge, MA, USA, 1998.

[5] V. Konda, V. S. Borkar, Actor-critic type learning algorithms for Markov decision processes, SIAM Journal on Control and Optimization 38(1) (1999) 94–123.

[6] M. L. Puterman, Markov Decision Processes, Wiley Interscience, New York, NY, USA, 1994.

[7] D. Bertsekas, Dynamic Programming and Optimal Control, 3rd Edition, Athena Scientific, Belmont, MA, USA, 2007.

[8] R. E. Bellman, Dynamic Programming, Princeton University Press, Princeton, NJ, 1957.

[9] R. Howard, Dynamic Programming and Markov Processes, MIT Press, Cambridge, MA, 1960.

[10] L. Page, S. Brin, R. Motwani, T. Winograd, The pagerank citation ranking: Bringing order to the web., Technical Report 1999-66 (1999). URL `http://ilpubs.stanford.edu:8090/422/`

[11] D. Bertsekas, J. Tsitsiklis, Neuro-Dynamic Programming, Athena Scientific, Belmont, MA, USA, 1996.

[12] A. Gosavi, Simulation-Based Optimization: Parametric Optimization Techniques and Reinforcement Learning, Springer, New York, 2003.

[13] S. Chaharsooghi, J. Heydari, S. Zegordi, A reinforcement learning model for supply chain ordering management: An application to the beer game, Decision Support Systems 45 (2008) 949959.

[14] S. Reveliotis, Uncertainty management in optimal disassembly planning through learning-based strategies, IIE Transactions 39(6) (2007) 645–658.

[15] Z. Sui, A. Gosavi, L. Lin, A reinforcement learning approach for inventory replenishment in vendor-managed inventory systems with consignment inventory, Engineering Management Journal 3(4) (2007) 44–53.

[16] J. A. Ramirez-Hernandez, E. Fernandez, A case study in scheduling re-entrant manufacturing lines: Optimal and simulation-based approaches, in: Proceedings of 44th IEEE Conference on Decision and Control, IEEE, 2005, pp. 2158–2163.

[17] A. Gosavi, Reinforcement learning for long-run average cost, European Journal of Operational Research 155 (2004) 654–674.

[18] N. Schutze, G. Schmitz, Neuro-dynamic programming as a new framework for decision support for deficit irrigation systems, in: International Congress on Modelling and Simulation, Christchurch, New Zealand, 2007, pp. 2271–2277.

[19] T. Ben-Zvi, J. Nickerson, Decision analysis: Environmental learning automata for sensor placement, IEEE Sensors Journal 11(5) (2011) 1206–1207.

[20] A. Malikopoulos, P. Papalambros, D. Assanis, A real-time computational learning model for sequential decision-making problems under uncertainty, Journal of Dynamic Systems, Measurement, and Control (ASME) 131 (2009) 041010–1–041010–8.

[21] F. Zhu, S. Ukkusuri, Accounting for dynamic speed limit control in a stochastic traffic environment: A reinforcement learning approach, Transportation Research Part C 41 (2014) 30–47.

[22] I. Grondman, L. Busoniu, G. Lopes, R. Babuska, A survey of actor-critic reinforcement learning: Standard and natural policy gradients, IEEE Transactions on SMC: Part C 42(6) (2012) 1291–1307.

[23] A. Gosavi, Boundedness of iterates in Q-learning, Systems and Control Letters 55 (2006) 347–349.

[24] I. Witten, An adaptive optimal controller for discrete time Markov environments, Information and Control 34 (1977) 286–295.

[25] P. J. Werbös, Building and understanding adaptive systems: A statistical/numerical approach to factory automation and brain research, IEEE Transactions on Systems, Man., and Cybernetics 17 (1987) 7–20.