Reducing Uncertainty Propagation in Markov Decision Processes

Davide Tateo, Carlo D'Eramo, Alessandro Nuara, Marcello Restelli, Andrea Bonarini
Department of Electronics, Information and Bioengineering
Politecnico Di Milano, Milano, Italy
Email: {davide.tateo, carlo.deramo, alessandro.nuara, marcello.restelli, andrea.bonarini}@polimi.it

***Abstract*—In many real-world problems stochasticity is a critical issue for the learning process. The sources of stochasticity come from the transition model, the explorative component of the policy or, even worse, from noisy observations of the reward function. For a finite number of samples, traditional Reinforcement Learning (RL) methods provide biased estimates of the action-value function leading to a poor estimation of the action-value function that is propagated to other action-values by the application of the Bellman operator. We propose an approach that significantly mitigates this issue avoiding the propagation of bad estimates of the action-value function.**

I. Introduction

It is well known that a key issue of Reinforcement Learning (RL) problems is the accuracy of the estimation of the action-value with a limited number of samples. While most algorithms guarantee the convergence of the estimates to the optimal action-value, in practice the presence of stochastic components lead to poor performance. In fact, the majority of real-world problems have significant sources of stochasticity: the environment could have stochastic transition and this complicates the estimation of the effectiveness of an action; most of the times it is necessary to use stochastic policies to guarantee that all states are visited infinitely many times, that is required to guarantee the convergence of the algorithm; the policy could change during the learning process resulting in very different behaviors; the reward function is often corrupted by noisy observations and, in other cases, the reward function is stochastic itself. Moreover, it usually happens that some deterministic environments are partially observable and, thus, are perceived by the agent as stochastic decision processes (e.g. Blackjack).

Since Monte-Carlo estimates of action-values are affected by high variance of the returns, the most successful RL algorithms are based on bootstrapping (e.g. Q-Learning [1]), that trades off the variance of the estimation with a consistent but biased estimator. However with a finite number of samples the bias of the estimation could be significantly relevant when propagating the action-values to the next state and, recursively, it propagates to all the other states. Recent works tried to deal with this issue, in particular focusing on the estimation of the maximum expected value. It is well known [2], [3] that the maximum operator (used in the Q-Learning update equation) is positively biased, thus it overestimates the optimal action-value. In highly stochastic environments, this overestimation leads to unstable learning and poor convergence rates. In order to avoid this issue, the Double Q-Learning algorithm [4] has been proposed. This algorithm uses the double estimator [5] to compute the maximum in the updating formula of the Q-function. Providing a negatively biased estimation this value (i.e. it underestimates the optimal action-value),this approach can improve the performance especially in noisy environments. Another recently proposed approach is the Weighted Q-Learning [6] that, computing a weighted average of the action-value functions estimates, balances between underestimation and overestimation.

However, the correlation between bias of the estimation and performance is still unclear; indeed, as also shown in the empirical section of this paper, Speedy Q-Learning algorithm [7] has very good performance despite its extremely poor estimation of the action-value. This is due to the fact that most of the policies are not dependent on the accuracy of the action-values, but instead they rely on their ordering. Starting from this considerations, we try to address this problem from another point of view. Indeed, we do not focus on the estimation of the maximum expected value, but we care about avoiding the propagation of uncertain action-value estimates. (DA CAMBIARE)In fact, we propose this work starting from the consideration that it is not relevant whether the approximations of the point estimates of the maximum expected value are good or not, but how these information is propagated to the other states. Our idea is to propagate the information about the value of the best action only when there is sufficiently certainty about the estimation. In fact, we propose this work starting

The interesting aspect of this approach is that it is possible to use any maximum expected value estimator and, moreover, it can be easily extended in an on-policy scenario. Since we believe that in our approach the choice of the estimator is not a relevant issue, we choose the maximum operator as it is the simplest one.

II. Preliminaries

—TODO preso dal mio paper, va parafrasato, solo modifiche minori– A Markov Decision Process (MDP) is defined by $\mathcal{M} = \langle \mathcal{X}, \mathcal{U}, \mathcal{P}, \gamma, \mathcal{R}, D \rangle$, where $\mathcal{X}$ is the state space, $\mathcal{U}$ is the action space, $\mathcal{P}$ is a Markovian transition model where $\mathcal{P}(x'|x,u)$ defines the transition density between state x and x' under action u, $\gamma \in [0,1)$ is the discount factor, $\mathcal{R}(r|x,u,x')$ defines the distribution of the reward, and D is the distribution of the initial state. A stochastic policy is defined by a density distribution $\pi(\cdot|x)$ that specifies for each state x the density distribution over the action space $\mathcal{U}$. — Given a policy π, we define the action value function as:

$$Q^{\pi}(x,u) = \underset{\substack{(x',u')\sim \\ \mathcal{P}(x'|x,u)\pi(u'|x')}}{\mathbb{E}}\left[r(x,u,x') + \gamma Q^{\pi}(x,u)(x',u')\right] \quad (1)$$

The optmimal policy $\pi*$ is the policy with the highest expected return in the MDP. It has been proven that the optimal policy can be always represented by a deterministic policy. The optmimal action velue function $Q^*(x,u)$ is the action-value function of the optmimal policy. It has been shown that the optimal policy in any MDP can be always a deterministic policy that at each state chooses the action with the highest $Q^*(x,u)$ value.

Given that we can always write the optimal action-value function as:

$$Q^*(x,u) = \underset{x'\sim\mathcal{P}(x'|x,u)}{\mathbb{E}}\left[r(x,u,x') + \gamma \max_{u'} Q^*(x',u')\right] \quad (2)$$

As the expected value is a linear operator, we can always write 2 as:

$$Q^*(x,u) = \underset{x'\sim\mathcal{P}(x'|x,u)}{\mathbb{E}}\left[r(x,u,x')\right] + \gamma \underset{x'\sim\mathcal{P}(x'|x,u)}{\mathbb{E}}\left[\max_{u'} Q^*(x',u')\right] \quad (3)$$

We now introduce two functions, $\widetilde{R}$ and $\widetilde{Q}$, defined as:

$$\begin{aligned}\widetilde{R}(x,u) &= \underset{x'\sim\mathcal{P}(x'|x,u)}{\mathbb{E}}\left[r(x,u,x')\right] \\ \widetilde{Q}(x,u) &= \underset{x'\sim\mathcal{P}(x'|x,u)}{\mathbb{E}}\left[\max_{u'} Q^*(x',u')\right]\end{aligned} \quad (4)$$

We can give an interpretation of this two functions. $\widetilde{R}(x,u)$ is the expected immediate reward of the action u in the state x. $\widetilde{Q}(x,u)$ is the expected discounted return of the states reached after performing action u in state x, i.e. the expected gain of the reached state.

We can now write the optimal value function as:

$$Q^*(x,u) = \widetilde{R}(x,u) + \gamma\widetilde{Q}(x,u) \quad (5)$$

Our approach shift the focus of the RL task from finding a good estimator for the optimal action-value function, to the task of finding good estimators for the $\widetilde{R}$ and $\widetilde{Q}$ functions. The main motivation is that the sources of uncertainty of the two components of the action-value function are different: the $\widetilde{R}$ function only depends on the transition and reward models, while $\widetilde{Q}$ also depends on the optmimal policy.

III. The Proposed Method

In the following section we will derive our method from 5. We will propose the general schema, using the maximum estimator and we will show the relations with the standard Q learning update.

A. Decomposition of the TD Error

Standard Q-Learning algorithm computes the temporal difference error given the tuple (x,u,r,x') w.r.t. the current action-value estimates, and then updates such estimate proprotionally to the error. The amount of correction in the direction of the new sample is measured by the learning rate: if the learning rate is 1, the new sample substitutes the old estimate, if the learning rate is close to 0, the new sample is discarded, and the old estimate is kept unchanged. As shown in 5 the action-value function could be decomposed in two different components. Our method is based on the idea to give separate estimates for this two components, Instead of computing a TD error, we compute the error w.r.t. each component of the action-value function:

$$\widetilde{R}(x,u) \leftarrow \tilde{R}(x,u) + \alpha(R(x,u,x') - \widetilde{R}(x,u)) \quad (6)$$
$$\widetilde{Q}(x,u) \leftarrow \tilde{Q}(x,u) + \beta(\max_{u'} Q(x',u') - \widetilde{Q}(x,u)) \quad (7)$$

Separating the two components of the value function can be useful, as the two components has inherently different sources of stochasticity. Moreover, the information stored in $\widetilde{R}$ is local to each state-action pair, and doesn't contain the uncertainty of the estimation of others states. The information stored in $\widetilde{Q}$ instead depends only on the action-value function of the states that could be reached after performing the action u in the state x, which, depends, recursively, on the others action-value functions. It is clear that the propagation of uncertain values only affects the $\widetilde{Q}$ component. As the actual action value function is the sum of the two estimates, we can write an equivalent update for the Q function:

$$\begin{aligned}Q(x,u) \leftarrow\ & \tilde{R}(x,u) + \alpha(R(x,u,x') - \tilde{R}(x,u)) \\ & + \gamma\left(\tilde{Q}(x,u) + \beta(\max_{u'} Q(x',u') - \tilde{Q}(x,u))\right) \\ =\ & Q(x,u) + \alpha(R(x,u,x') - \tilde{R}(x,u)) \\ & + \gamma\beta(\max_{u'} Q(x',u') - \tilde{Q}(x,u))\end{aligned} \quad (8)$$

notice that this update cannot be used in practice in the algorithm, as it is not keeping the current values of the single components. However 8 is useful to analyze the relations to standard Q-Learning algorithm.

B. Analysis of the Decomposed Update

We will discuss the relationship of our method with standard temporal difference methods. Let t be the learning step. As a first step of our analysis we can consider the simplest case $\alpha(t) = \beta(t)$, $\forall t$. by combining 8 and 5 we obtain:

$$\begin{aligned}Q(x,u) \leftarrow\ & Q(x,u) + \alpha(R(x,u,x') + \gamma\max_{u'} Q(x',u') \\ & - Q(x,u))\end{aligned} \quad (9)$$

That is the classical Q-Learning update.

We consider now the setting $\alpha(t) \geq \beta(t) > 0$, $\forall t$. Let $\beta(t) = \delta(t)\alpha(t)$, we obtain:

$$\begin{aligned}
Q(x,u) &\leftarrow Q(x,u) + \alpha(R(x,u,x') + \gamma\delta \max_{u'} Q(x',u') \\
&\quad - (\tilde{R}(x,u) + \gamma\delta\tilde{Q}(x,u))) \\
&= Q(x,u) + \alpha(R(x,u,x') + \gamma' \max_{u'} Q(x',u') \\
&\quad - (\tilde{R}(x,u) + \gamma'\tilde{Q}(x,u))) \\
&= Q(x,u) + \alpha((R(x,u,x') + \gamma' \max_{u'} Q(x',u')) \\
&\quad - Q'(x,u)) \qquad (10)
\end{aligned}$$

With $\gamma' = \gamma\delta$. Notiche that $Q'(x,u)$ is the current Q function, but computed with a different discount factor. If the condition above is satisfied, then, we can see our method as a variable discount factor learning. If we consider $\delta(t)$ that increase monotonically in the interval $[0,1]$, our method is increasing each step the effective horizon, starting from trying to solve a greedy miopyc problem, and moving towards the real one. This approach, it has been used in practice to solve infinite horizon problems when the discount factor approach to 1, [].

Finally, we can observe that, if the reward function and the transition model are deterministic, we can fix α to the value 1, while considering only $\beta(t) = \delta(t)$,

C. Variance dependent learning rate

To have better performance on the estimation, we would like to weight the error of each sample w.r.t. the current estimate depending on how much we are sure about the current value of our estimate. We propose then a learning rate that depends on the variance of the current variable estimate.

First of all we need to compute the variance of each estimator. To perform such computation we have to made the assumption that the learning rates are independent from the data. We will, of course, violate this assumption, but it is needed in order to have a closed form for the variance of the estimator, that can be used in practice. We will suppose also that the samples X_i are i.i.d., with mean μ and variance σ^2. Consider the general form of the estimator:

$$\widetilde{X}_{n+1} = (1-\alpha(t))\widetilde{X}_n + \alpha(t) * \widetilde{X} \qquad (11)$$

We now compute the expected value and the variance of this estimator:

$$\mathbb{E}\left[\widetilde{X}_{n+1}\right] = \mu \sum_i^n \alpha(i) \prod_{j=i+1}^n (1-\alpha(j)) \qquad (12)$$

$$\operatorname{Var}\left[\widetilde{X}_{n+1}\right] = \sigma^2 \sum_i^n \alpha(i)^2 \prod_{j=i+1}^n (1-\alpha(j))^2 = \sigma^2\omega \qquad (13)$$

With $\omega = \sum_i^n \alpha(i)^2 \prod_{j=i+1}^n (1-\alpha(j))^2$. Notice also that we can use the sample covariance S_{n-1} of the random variable X to estimate the real covariance σ^2. It is possible to compute in an incremental way both the sample covariance, in the traditional way, and ω:

$$\omega_{n+1} = (1-\alpha(n))^2\omega_n + \alpha(n)^2 \qquad (14)$$

A weak assumption of this model, is that the variables are identically distribuited. While this should be true for the reward function, if the MDP is stationary, this is not true for the Q function values, whose distribution is affected by the policy and by the others states current estimates. However, a good approximation, could be to consider a window lenght in which the distribution is approximatively stationary: using such approach, we can compute the variance of the process in a given time window, forgetting old values that can lead to a biased estimation of the current variance. While this approach is not formally correct, as the derivation of the variance estimates makes the assumption of i.i.d. variables, this approximation has in practice have very good results.

Finally, we can choose a learning rate that depends on the covariance. Let $\sigma_e^2(t)$ be an estimate of $\operatorname{Var}\left[\widetilde{X}_t\right]$. We propose the following learning rate for each component of the action-value function:

$$\alpha(t) = \frac{\sigma_e^2(t)}{\sigma_e^2(t) + \eta} \qquad (15)$$

Where η is the amount of the estimator variance for wich the learning rate is 0.5. It can be seen as a soft treshold to tune the speed of the decrease of the learning rate w.r.t. the estimator variance.

If we consider the case $\beta(t) = \alpha(t)\delta(t)$, then we have tu use a different learning rate. As we want an increase of the discount factor faster than the decrease of the general learning rate, we can use an exponentialy increasing learning rate for the delta parameter:

$$\delta(t) = 1 - e^{\frac{\sigma^2}{\eta}\log(0.5)} \qquad (16)$$

Where η has the same iterpretation of the previous scenario.

D. Convergence Properties

TODO – chiedere anche a restelli. Non banale!

IV. Experimental Results

As we said before, QDec-Learning is helpful in stochastic environments. In this section, we highlight the main advantages of the adaptation of the learning rate according to the uncertainty of the estimates in three noisy discrete MDPs. Moreover, we compare the performance of QDec-Learning against Q-Learning and some of its variants that only use a single decaying learning rate.

A. Noisy Grid World

This environment consists in a 3×3 grid with the initial position in the lower-left cell and the goal state in the upper-right cell. Each action performed in a non-goal state obtains a reward -12 and 10 with equal probability. In the goal state, every action obtains a reward of 5 and terminates the episode.

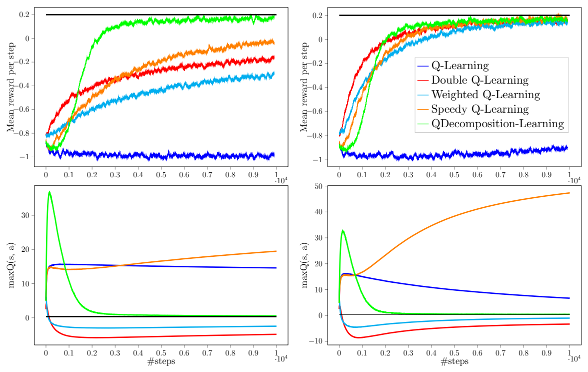


Fig. 1. ..

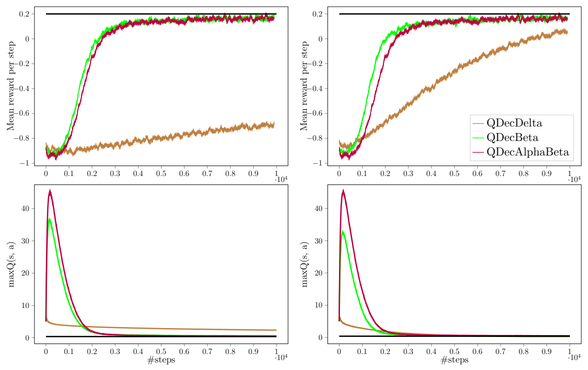


Fig. 2. ..

The policy is ε-greedy with $\varepsilon = \frac{1}{\sqrt{n(s)}}$, where $n(s)$ is the number of visits of the state s. We analyse the performance of QDec-Learning with different choices of learning rates, on the other hand for all the other algorithms we use a decaying learning rate $\alpha(s,a) = \frac{1}{n(s,a)^k}$ where $n(s,a)$ and k are respectively the number of updates for each action a in state s[1] and a coefficient to tune the rate of decay.

Figure **??** shows the mean reward per step and the approximation of the maximum action-value in the initial state computed by the empirically best setting of QDec-Learning and the other algorithms. Figure **??** shows the same QDec-Learning performance of the previous figure in comparison with other less effective QDec-Learning settings. In this figures, the optimal average reward per step is 0.2 and the maximum action-value function of the starting state is $5\gamma^4 - \sum_{k=0}^{3} \gamma^k \approx 0.36$. QDec-Learning is able to deal with the highly stochasticity of the reward function overcoming the performance of the other algorithm both in the

B. Trap Gridworld

We tested our algorithm on a GridWorld problem variant, introducing some traps in the grid. In the 5x5 grid shown in Figure 4 we have 4 traps in the central row. The agent receives a reward equal to zero in all white cells, a positive +10 reward if she reaches the goal, while she receives a negative

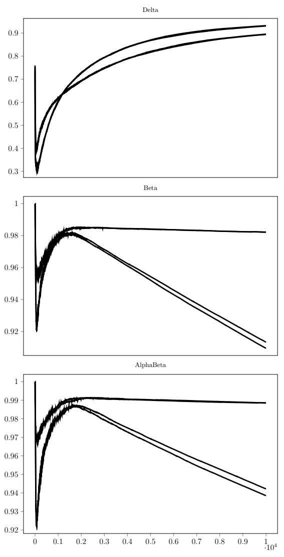


Fig. 3. ..

./imgs/gridHole/grid_hole.jpg

Fig. 4. GridWorld with traps

reward -10 if she falls in the traps. When he reaches the goal or fall in a trap the episode ends. The parameters settings used in this environment are the same described in the van Hasselt GridWorld. Figure IV-C shows the performance in terms of average reward per step and Q-function estimation of all tested algorithms. As shown in the plot in Figure 5 the QD-learning provides the best Q-function estimation in few steps. However, an inaccurate estimation of the Q-function does not necessary imply a worse optimal policy estimation. Indeed, as we can see in Figure IV-C, Q-learning and SpeedyQ-learning have the better performance in terms of average reward per step, while QD-learning outperforms the Double Q-learning and the WQ-learning algorithms.

[1]Double Q-Learning splits $n(s,a)$ in $n_A(s,a)$ and $n_B(s,a)$ that are the number of updates for each action a in state s in the two Q-Tables.

./imgs/gridHole/reward.pdf

Fig. 5. Reward per step averaged on 1000 experiments

./imgs/gridHole/maxQ.pdf

Fig. 6. maxQ(s,a) estimation averaged on 1000 experiments

Fig. 7. Results Gridworld traps

V. Conclusion

References

[1] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[2] J. E. Smith and R. L. Winkler, "The optimizer's curse: Skepticism and postdecision surprise in decision analysis," *Management Science*, vol. 52, no. 3, pp. 311–322, 2006.

[3] E. Van den Steen, "Rational overoptimism (and other biases)," *American Economic Review*, pp. 1141–1151, 2004.

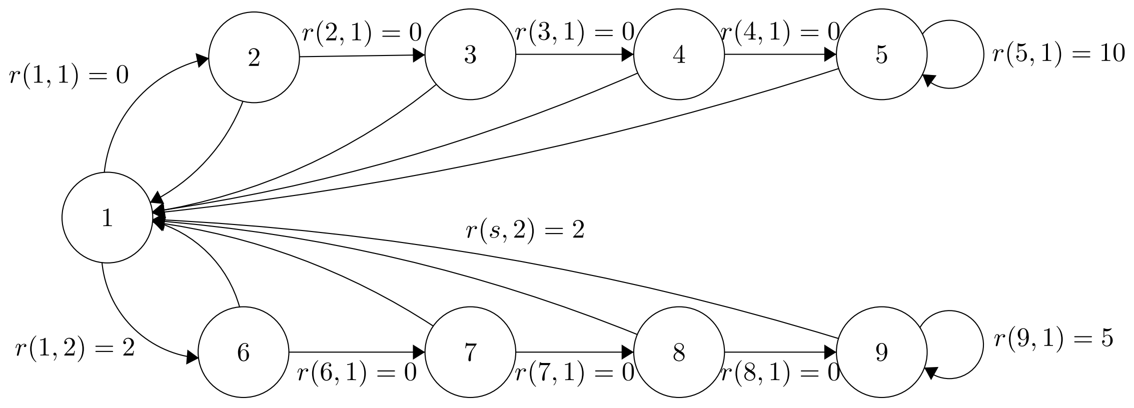
[4] H. van Hasselt, "Double q-learning," in *Advances in Neural Information Processing Systems*, 2010, pp. 2613–2621.

[5] ——, "Estimating the maximum expected value: an analysis of (nested) cross-validation and the maximum sample average," *arXiv preprint arXiv:1302.7175*, 2013.

[6] C. D'Eramo, M. Restelli, and A. Nuara, "Estimating maximum expected value through gaussian approximation," in *International Conference on Machine Learning*, 2016, pp. 1032–1040.

[7] M. Ghavamzadeh, H. J. Kappen, M. G. Azar, and R. Munos, "Speedy q-learning," in *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2011, pp. 2411–2419. [Online]. Available: http://papers.nips.cc/paper/4251-speedy-q-learning.pdf

C. Double Chain



$r(1,1) = 0$
$r(2,1) = 0$
$r(3,1) = 0$
$r(4,1) = 0$
$r(5,1) = 10$
$r(s,2) = 2$
$r(1,2) = 2$
$r(6,1) = 0$
$r(7,1) = 0$
$r(8,1) = 0$
$r(9,1) = 5$
1
2
3
4
5
6
7
8
9