Reinforcement Learning
Problem and Solution methods

Caroline Heidenreich

March 22, 2017

The following section aims at giving an overview to the standard Reinforcement learning problem and methods and is entirely based on the introduction provided in [3]. Reinforcement learning is a field of machine learning that tackles the problem of how a learning agent should choose actions in order to maximise some notion of cumulative reward. To learn an optimal policy the agent needs to interact with its environment by taking actions and collecting the reward following from these actions. Given a certain state s_t, the agent selects an action a_t according to some policy π_t and finds itself subsequently in a new state s_{t+1} while receiving some immediate reward r_{t+1}. Reinforcement learning algorithms provide different ways of how the agent should modify π to maximise its cumulative long-term reward, the return R.

The return is typically defined as

$$R_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1} \tag{1}$$

where T describes the horizon of the problem (that could also be ∞) and $\gamma \in [0, 1]$ denotes the discount factor that can be used to weigh sooner rewards more than later rewards.

An important framework to model reinforcement learning problems are Markov Decision Processes (MDP). To model a problem as an MDP it is necessary that the system fulfills the Markov property. That means that the system response at time $t+1$ only depends on the state and action at time t and not on the time before. An MDP then is defined by a quintuple (S, A, T, r, γ) where S is the state space and A is the action space. T describes the transition probabilities of arriving at a state s' given a state s and an action a i.e. $T(s, a, s') = p(s'|s, a)$. r is the reward function that assigns rewards to state-action pairs and γ is the discount factor.

Given a certain MDP, reinforcement learning algorithms typically try to estimate value functions that describe "how favourable" a state is, given that

starting from that state a policy π is applied. The state-value function V^π is defined in terms of expected return i.e.

$$V^\pi(s) = E_\pi\{R_t|s_t = s\} \tag{2}$$

which means that a state with higher future rewards is more favourable. Alternatively, the value of being in a state s, taking an action a and then following a policy π is defined by the action-value function

$$Q^\pi(s,a) = E_\pi\{R_t|s_t = s, a_t = a\}. \tag{3}$$

Value functions are used to determine how good a policy is. A policy is better than another if it achieves a higher expected return for all states and thus has a higher value $V^\pi(s)$ for all s. The optimal policy π^* is the one (or are the ones) with the optimal state-value function

$$V^*(s) = \max_\pi V^\pi(s). \tag{4}$$

The learning agent updates an initial estimate of the value functions from experience that is gained by interaction with the environment. A classical algorithm in the field of reinforcement learning, known as Q-learning, was presented by Watkins in 1989 [5]. In its simplest form the learning update of this algorithm is formulated as

$$Q_{t+1}(s_t,a_t) = Q_t(s_t,a_t) + \alpha_t\left[r_{t+1} + \gamma \max_a Q_t(s_{t+1},a) - Q_t(s_t,a_t)\right]. \tag{5}$$

In this expression, $Q(s,a)$ is the learned action-value function, α is the step size and r_{t+1} is the reward obtained by taking action a from state s and arriving in the succeeding state s_{t+1}. Q approximates the optimal action-value function Q^*. Convergence is ensured as long as the following two conditions on the step size α are fulfilled

$$\sum_{t=1}^{\infty} \alpha_t(a) = \infty \tag{6}$$

$$\sum_{t=1}^{\infty} \alpha_t^2(a) < \infty \tag{7}$$

and all state-action pairs are visited infinitely often as the number of transitions approaches infinity. The latter condition deals with the trade-off between exploration and exploitation. This describes the need to find a balance between applying the policy that is currently estimated to be best and exploring unknown regions of the state space. One way to ensure this balance is to employ an ϵ-greedy policy that makes the agent choose a random action with probability ϵ and the action maximising $Q(s,a)$ elsewise.

Classical Q-learning does converge under the above described conditions but it does so slowly. Therefore, several improvements to the original algorithm

have been proposed. Speedy Q-learning is a variant of traditional Q-learning that was introduced in 2011 [1] aimed at increasing the convergence speed. To achieve faster convergence there are slight changes in the update of the Q-values. Instead of only taking the Q-function of the current iteration into account, the update incorporates also the previous Q-function. The learning update becomes

$$\begin{aligned} Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha_k \left[r_{t+1} + \gamma \max_a Q_{t-1}(s_{t+1}, a) - Q_t(s_t, a_t) \right] \\ + (1 - \alpha_k) \left[r_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - r_{t+1} + \gamma \max_a Q_{t-1}(s_{t+1}, a) \right]. \end{aligned} \quad (8)$$

The second term goes to zero as the Q-values converge to the optimal values. Therefore, the usual step size conditions do not need to be fulfilled for this term and the algorithm achieves a larger step size while still ensuring convergence. More detailed convergence results can be found in [1].

Another variation of Q-learning is the Delayed Q-learning algorithm [2]. The largest change compared to traditional Q-learning is that the Q-values are only updated after m sample updates are collected so that the update rule becomes

$$Q_{t+1}(s_t, a_t) = \frac{1}{m} \sum_{i=1}^{m} \left(r_{k_i} + \gamma \max_a Q_{k_i}(s_{k_i}, a) \right) + \epsilon_1 \quad (9)$$

For such an update to be performed the change in the Q-value has to be larger than $2\epsilon_1$ where $\epsilon_1 \in (0, 1)$. Furthermore, the number of attempted updates is limited such that after the update of a state-action pair's Q-value has been attempted m times a successful update of any Q-value has to occur until the update is attempted again. The authors then provide a probabilistic upper bound on the sample complexity, i.e. the number of time steps that the agent executes a not ϵ-optimal policy. The details of the convergence proof can be looked up in [2].

A different approach to Q-learning for large state and action spaces is Q-learning with function approximation. In this method information from a limited subset of the state space is extended to cover the whole state space. In case of linear function approximation the Q-function becomes

$$Q_\theta(s, a) = \sum_{i=1}^{n} \theta_{i,a} \varphi_i(s) = \theta^T \varphi \quad (10)$$

where θ is the parameter matrix and φ is the vector of basis functions. φ_i can for instance be chosen to be a Gaussian radial basis function

$$\varphi_i(s) = \exp\left(\frac{-||s - m_i||^2}{2\sigma_i^2} \right) \quad (11)$$

where m_i is the center and σ_i is the standard deviation of the resulting Gaussian function. These are parameters that need to be tuned in order for the function approximation to accurately reflect the true function. The standard deviation

of the basis functions for instance determines over which distance generalization takes place. During the learning process a gradient-descent update of one column of the parameter matrix can then be performed in each iteration:

$$\delta_t = r_{t+1} + \gamma \max_a Q_\theta(s_{t+1}, a) - Q_\theta(s_t, a_t) \tag{12}$$

$$\theta_{\bullet, a_{t+1}} = \theta_{\bullet, a_t} + \alpha \cdot \delta_t \cdot \varphi_t(s_t). \tag{13}$$

A detailed convergence analysis of this method can be found in [4].

References

[1] M. G. Azar, R. Munos, M. Ghavamzadeh, H. J. Kappen, et al. Speedy q-learning. In *NIPS*, pages 2411–2419, 2011.

[2] A. L. Strehl, L. Li, E. Wiewiora, J. Langford, and M. L. Littman. Pac model-free reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 881–888. ACM, 2006.

[3] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[4] J. N. Tsitsiklis, B. Van Roy, et al. An analysis of temporal-difference learning with function approximation. *IEEE transactions on automatic control*, 42(5):674–690, 1997.

[5] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.