# Reinforcement Learning

Vincent Zhuang

April 18, 2016

## 1 Introduction

Reinforcement learning characterizes learning problems in which an *agent* seeks to determine its optimal actions given partial and delayed rewards from the *environment*. Typically, there is a state space $S$ and action space $A$. At each time step $t$, the agent performs an action $a_t \in A$ while the environment is in state $s_t \in S$, which then transitions to state $s_{t+1}$ and emits reward $r_{t+1}$ (generated from some unknown reward function $R : S \times A \to \mathbb{R}$).

Within an episode (see below), we typically define the reward as the total discounted future return starting at time $t$:

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots = \sum_{t' \geq t} \gamma^{t'-t} r_{t'}$$

for some $\gamma \in [0, 1]$.

The goal of the agent is to select the sequence of actions that maximizes its future reward. This motivates the two fundamental concepts of *policy* and *value*:

- Policy: $\pi : S \to \mathcal{P}(A)$ - how the agent behaves given the state of the environment.
- Value or Q-Function: $Q^\pi(s_t, a_t) = \mathbb{E}[R_t | s_t, a_t]$ - for some fixed policy $\pi$, the expected future reward given some state and action. Given a Q-function, the policy is then $\pi(s) = \arg\max_a Q^\pi(s, a)$.

The majority of reinforcement learning algorithms find the optimal policy by optimizing the value function, although some seek to learn the optimal policy independent of any value function.

**Episodic vs. Continuing RL.** We distinguish between two types of reinforcement learning problems - one in which the learning process can be separated into a series of fixed-length *episodes*, and one in which a single episode simply continues indefinitely. The former is called *episodic*, and the latter *continuing*.

In episodic tasks, the learning problem has clearly defined initial and terminal states. After each episode, the agent starts at the beginning of a new episode, and so on as its behavior improves. For example, playing a game (repeatedly) is more naturally modelled as an episodic RL problem, whereas a personalized home assistance robot is more naturally modelled as a continuing RL problem.

## 2 Markov Decision Processes

A common way to model reinforcement learning problems (and, in general, online decision making problems) is via the *Markov decision process* (MDP). A MDP treats the sequence of states as a Markov chain with the agent picking the transitions (actions) given the rewards emitted by the environment. Formally, we define a state transition probability kernel $P : S \times A \to \mathcal{P}(S)$ to model the environment's transitions.

A standard MDP operates as follows:

1. Begin at initial state $s_0$
2. Agent performs action $a_1$
3. Transition to state $s_1$ according to transition kernel $P$ on $(s_0, a_1)$
4. Receive reward $r_1$ according to reward function $R$
5. Repeat for $t = 1, \ldots$

**Markov Property.** The defining characteristic of MDPs is the *Markov property*:

$$Pr(s_{t+1}|s_t, a_t, \ldots, s_1, a_1) = Pr(s_{t+1}|s_t, a_t)$$

The Markov property allows us to (often compactly) parameterize the policy using only the current state, but also restricts us only being able to perfectly model situations where the transition dynamics between states is truly Markovian. Furthermore, the classical MDP setting also assumes perfect information, i.e., the agent alwys knows what state she is in. Nevertheless, relaxing this assumption still leads to good performance of MDP-based algorithms on many problems [3].

**POMDPs.** A *partially observable MDP* (POMDP) are MDPs in which the agent cannot see the actual state of the environment, but instead must infer the state from *observations* that are stochastically emitted from the state.

If the underlying state-space is discrete and take a Bayesian approach, then a POMDP can be viewed as a continuous-state MDP on the *belief states* via Bayes's rule, although in practice this is intractable and approximate algorithms are used instead (analogous to EM for HMMs).

**RL vs. Bandits.** Reinforcement learning is a generalization of standard online learning (e.g. multi-armed bandits) in the following two ways:

- **it involves a changing environment**: MAB, on the other hand, operates in a fixed state. There are two ways that a MAB problem can be viewed as a special case of a MDP or POMDP. First, the continuous RL setting modeled by a POMDP reduces to the MAB if we assume there is only a single state at all times, and the goal is only to learn the reward function of actions on that single state. Second, the episodic RL setting reduces to the MAB if we assume that the length of each episode is 1 (which trivially negates the effect of state transitions). In both cases the state transition is either non-existent (only one state) or unstructured (the prior probability of start states in episodic RL).
- **it incorporates long-term action-environment dynamics**: the contextual bandits setting is closer to general reinforcement learning than vanilla bandits, but still restricts the effect of the current action to the immediate reward. On the other hand, in general reinforcement learning the current action influences the future states as well as the immediate reward.

## 3 Q-Learning

In *Q-learning*, we seek to learn the optimal action-value function: $Q^*(s, a) = \max_\pi Q^\pi(s, a)$, i.e. the best value achievable by any policy.

The standard online Q-learning algorithm uses a one-step *value iteration* update given by the *Bellman equation*, which is just depth-one expansion of the recursive definition of $Q$:

$$Q(s, a) = r + \gamma \max_{a'} Q(s', a')$$

**Algorithm 1** Q-learning

```
1: Init learning rate $\alpha$
2: while not converged do
3:     $t \leftarrow t + 1$
4:     pick and do action $a_t$ according to current policy (e.g. $\epsilon$-greedy)
5:     receive reward $r_t$
6:     observe new state $s'$
7:     update $Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_t + \gamma \max_{a'} Q(s', a') - Q(s_t, a_t)]$
8: end while
```

**Action selection.** Instead of using the obvious greedy policy $\max_a Q(s, a)$, it is generally better to introduce exploration by occasionally visiting suboptimal states. Simple action selection strategies for introducing exploration including $\epsilon$-greedy and softmax (Boltzmann) exploration.

**Function approximation.** Vanilla Q-learning stores a table of the $Q$ function estimates. However, Q-learning can also be combined with powerful functional approximation methods, such as deep convolutional neural networks.

**Convergence.** Given ergodicity assumptions (specifically that each state must be visited infinitely often) is simple to prove that Q-learning always converges (to optimal $Q$ and $\pi$) via stochastic approximation techniques [2] [1]. See [5] for an alternative proof in a slightly more restrictive setting.

**Comparison to Bandit Algorithms.** The main conceptual difference between Q-Learning (and learning in RL problems in general) and bandit algorithms is the fact that in RL on needs to do exploration over sequences of states. In the bandit setting, there are no state transitions, so the goal is to just explore the rewards for actions in that single state as efficiently as possible. In RL problems, exploration must take into account the potential impact of future rewards as well.

## 4 SARSA

SARSA, or "state action reward state action," is essentially just Q-learning in which the optimistic Q-update is replaced with a depth-one actual reward update:

$$Q(s_t, a_t) = (1 - \alpha)Q(s_t, a_t) + \alpha[r_t + \gamma Q(s_{t+1}, a_{t+1})]$$

where $\alpha$ is the learning rate.

The distinction between Q-learning and SARSA lies primarily in the role of the agent's policy in the value updates. To wit, Q-learning does not depend on what actions the agent performs and is an *off-policy* algorithm, whereas SARSA does and is *on-policy.*

## 5 Other RL algorithms

We primarily focused on value-based reinforcement learning methods, and in particular, *model-free* algorithms. Other common RL algorithms also seek to learn the policy directly, either alongside a value function (e.g. actor-critic) or without the value function (e.g. policy gradient).

**Model-based vs. Model-free RL** If one has domain knowledge about the environment (i.e. the transition probabilities of the MDP), then one can use *model-based* dynamic programming algorithms such as value iteration and policy iteration to solve the MDP exactly. If one does not have *a priori* information about the MDP, one can still use model-based approaches that simultaneously learn a model of the environment, which can then be used to model future behavior, etc. However, it is in general difficult to learn a good model, and using a bad model can negatively impact performance w.r.t. a model-free approach.

## Acknowledgements

Thanks to Yisong and Stephan for comments and suggestions.

## References

[1] Jaakkola, T., Jordan, M. I., & Singh, S. P. (1994). On the convergence of stochastic iterative dynamic programming algorithms. *Neural computation*, 6(6), 1185-1201.

[2] Melo, F. S. (2001). Convergence of Q-learning: A simple proof. *Institute Of Systems and Robotics, Tech. Rep.*

[3] Sutton, R. S., & Barto, A. G. (2011). *Reinforcement learning: An introduction.* The MIT Press (March 1998).

[4] Szepesvári, C. (2009). Algorithms for reinforcement learning. *Morgan and Claypool.*

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