Reinforcement Learning 2015/2016

Tutorial 3

1. Consider the gambler's problem (example 4.3 in S+B). Why does the optimal policy for the gambler's problem have such a curious form? In particular, for capital of 50 it bets it all on one flip, but for capital of 51 it does not. Why is this a good policy?

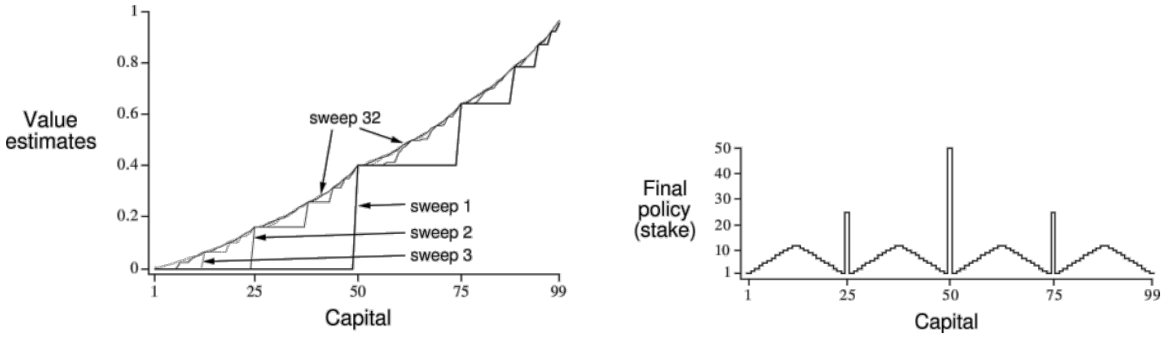

Figure 1: Solution of the Gamblers problem

2. Implement (in Matlab; if you like Lisp, use the program from http://webdocs.cs.ualberta.ca/~sutton/book/code/gambler.lisp) value iteration for the gambler's problem and solve it for $p = 0.25$ and $p = 0.55$. In programming, you may find it convenient to introduce two dummy states corresponding to termination with capital of 0 and 100, giving them the values of 0 and 1 respectively. Show your results graphically. Comment on the stability of your results as $\varepsilon \to 0$.

3. The Bellman equation
$$V^{\pi}(s) = \sum_{a} \pi(s;a) \sum_{s'} P^{a}_{ss'}[R^{a}_{ss'} + \gamma V^{\pi}(s')]$$
must hold at each state for the value function V^{π}. Show, numerically in figure below, that this equation holds for the centre state, valued at +0.7, with respect to its four neighbouring states, valued at +2.3, +0.4, -0.4, and +0.7.

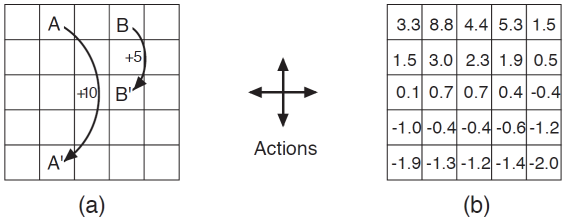

Figure from Sutton and Barto, 2nd ed.

4. Suppose that you are restricted to considering only policies that are ε-soft, meaning that the probability of selecting each action in each state, s, is at least $\frac{\varepsilon}{|\mathcal{A}(s)|}$, where $\mathcal{A}(s)$ is the set of actions that can be applied in state s. Describe qualitatively the changes that would be required in each of the steps 3, 2, 1, in that order, of the policy iteration algorithm for V in Fig. 4.3 in the Sutton and Barto book (see below).

1. Initialization
   $v(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
      $\Delta \leftarrow 0$
      For each $s \in \mathcal{S}$:
         $temp \leftarrow v(s)$
         $v(s) \leftarrow \sum_{s'} p(s'|s, \pi(s)) \Big[r(s, \pi(s), s') + \gamma v(s') \Big]$
         $\Delta \leftarrow \max(\Delta, |temp - v(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
      $temp \leftarrow \pi(s)$
      $\pi(s) \leftarrow \arg\max_a \sum_{s'} p(s'|s, a) \Big[r(s, a, s') + \gamma v(s') \Big]$
      If $temp \neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return v and π; else go to 2

Figure from Sutton and Barto, 2nd ed.

5. The simple Reinforcement Learning algorithms that we have considered so far may serve as methods for solving small-scale MDP problems, but they find few practical applications unless the underlying assumptions are relaxed, the relevant concepts are generalised and/or problem-specific adaptations are taken. S. Singh and John Langford (see umichrl.pbworks.com/w/page/7597585/Myths of Reinforcement Learning, or just search for "Myths of Reinforcement Learning") have contrasted the standard form of RL (which we have so far focused on in class) and the state-of-the-art algorithms (which are designed by specialists and often incorporate domain knowledge) by formulating what they call the ***Myths of RL***.
   Go through the list below and discuss whether the myths could be "busted" by reasonable modifications of the algorithms that we have covered until now.

   (a) RL is TD or perhaps Q-learning
   (b) RL is about learning optimal policies
   (c) RL is tabula rasa
   (d) RL is model-free
   (e) RL is table look-up
   (f) Large state spaces are hard for RL
   (g) RL is slow
   (h) Non-Markovianity invalidates standard RL methods
   (i) RL does not have (m)any success stories since TD-Gammon[1]
   (j) Value function approximation does not work (and so we should do something else - the favourite alternative seems to be policy search)
   (k) RL does not work well with function approximation[2]
   (l) POMDPs are hard for RL to deal with[3]

[1] We will discuss TD-Gammon later, in the context of function approximation for RL, for the moment check wikipedia.

[2] Function approximation for RL will be an important topic later in the course, the idea being that the value function is not learned separately for every value separately, but as a parameteric function, such that, if there are fewer parameters than values the learning process is sped up considerably.

[3] POMDPs are a generalisation of MDPs, where the state of the system is not generally observable. The idea of the solution of such problems it to generate an internal state that provides the information for a correct decision. Obviously, the internal state is an imprecise estimate of the true state of the system, such that learning is certainly harder. Can you think of an example when this approach relevant?