Reinforcement Learning
Homework Sheet 1

Lawrence Kurowski 2019280743

1 April 2020

1

1.a

Let optimal value function starting from field s_i be denoted by v_i, and optimal value at G be v_G.

$$v_G = \sum_{t=0}^{\infty} \gamma^t = \frac{1}{1-\gamma}$$

since $0 < \gamma < 1$. At each s_i, we receive reward=0 at intermediate states, and the only reward comes from the final iterations at G. Since at each s_i the optimal policy is to go right, hence

$$v_{n-i} = \sum_{t=0}^{\infty} r_t \gamma^t = \sum_{t=0}^{n-i-1} 0 \times \gamma^{n-i-1} + \sum_{t=n-i}^{\infty} \gamma^t = \frac{\gamma^{n-i}}{1-\gamma}$$

for $1 \leq i \leq n-1$.

1.b

Optimal policy does depend on γ when $0 < \gamma$. If $\gamma = 0$, then $v_G = 1$ and $v_i = 0$ otherwise. Going to the right is still an optimal policy, but there are ∞ other optimal policies as well (all with optimal value 1).

1.c

Because we add extra reward c in both directions (for both actions a_0 and a_1), hence the optimal policy remains unchanged. Let the new optimal value at s_i be denoted by w_i, and the original reward function r_t, we have

$$w_i = \sum_{t=0}^{\infty} (r_t + c)\gamma^t = \sum_{t=0}^{\infty} \gamma^t r_t + \sum_{t=0}^{\infty} \gamma^t c = v_i + c \sum_{t=0}^{\infty} \gamma^t = v_i + \frac{c}{1-\gamma}$$

and

$$w_G = \sum_{t=0}^{\infty} (1 + c)\gamma^t = v_G + \frac{c}{1-\gamma} = \frac{1+c}{1-\gamma}$$

1.d

Let the new optimal value function at s_i be u_i, we have

$$u_i = \sum_{t=0}^{\infty} a(r_t + c)\gamma^t = a \sum_{t=0}^{\infty} \gamma^t r_t + a \sum_{t=0}^{\infty} \gamma^t c = a v_i + ac \sum_{t=0}^{\infty} \gamma^t = a v_i + \frac{ac}{1-\gamma}$$

and

$$u_G = \sum_{t=0}^{\infty} a(1 + c)\gamma^t = a v_G + \frac{ac}{1-\gamma} = \frac{a(1+c)}{1-\gamma}$$

We distinguish 3 cases:

$a = 0$ for all states, any policy is optimal and has value 0.

$a > 0$ optimal policy does not change vs. base case (a), and optimal value is given by u_i above.

$a < 0$ optimal policy is any policy that never gets to G and keeps "oscillating" in states $s_1 \dots s_{n-1}$ forever. This is because: suppose there is a policy π that is optimal and does end at G. Then we can add one more iteration of a_1 followed by a_0 to get policy π'. Now $u_{\pi'} \geq \gamma \times u_\pi > u_\pi$ since all u are negative. This is contradiction with π being optimal. Hence, any policy that gets to G is not optimal, or equivalently the optimal policy is to never get to G.

2

2.a

1st step - no reward. Thereafter: each step reward of 1 discounted by γ. Hence

$$V_{a_1}(s_0) = Q(a_1, s_0) = \sum_{t=0}^{\infty} r_t \gamma^t = 0 + \sum_{t=1}^{\infty} \gamma^t = \frac{\gamma}{1-\gamma}$$

2.b

1st step - reward of $\frac{\gamma^2}{1-\gamma}$. Thereafter: each step reward of 0. Hence

$$V_{a_2}(s_0) = Q(a_2, s_0) = \sum_{t=0}^{\infty} r_t \gamma^t = \frac{\gamma^2}{1-\gamma} \sum_{t=1}^{\infty} 0 \times \gamma^t = \frac{\gamma^2}{1-\gamma} = \gamma Q(a_1, s_0) < Q(a_1, s_0)$$

as $0 < \gamma < 1$, hence choosing a_1 is optimal.

2.c

Bellman value iteration is $V^{(n)}(s_0) = \max_{i \in 1,2} Q^n(a_i, s_0)$. At iteration n,

$$\begin{aligned} Q^{(n)}(a_2, s_0) =& \frac{\gamma^2}{1-\gamma} \\ Q^{(n)}(a_1, s_0) =& 0 + \sum_{t=1}^{n-1} \gamma^t = \frac{\gamma(1-\gamma^n)}{1-\gamma} \end{aligned}$$

The algorithm will choose a_1 for all $k \geq n*$ such that

$$\begin{aligned} Q^{(n*)}(a_2, s_0) \leq& Q^{(n*)}(a_1, s_0) \\ \frac{\gamma^2}{1-\gamma} \leq& \frac{\gamma(1-\gamma^{n^*})}{1-\gamma} \\ \gamma \leq& 1 - \gamma^{n^*} \\ \frac{\log(1-\gamma)}{\log(\gamma)} \leq& n^* \end{aligned}$$

Q.E.D.

3

Let $\tilde{V}$ be some value function, V_g be the greedy value function and V^* be the optimal value function. Let $\mathbb{S}$ and $\mathbb{A}$ be the state and action spaces, respectively. Let $\epsilon > 0$.

For value function V and $s \in \mathbb{S}$ define $L_V(s) = |V(s) - V^*(s)|$ which is the value loss if we choose a sub-optimal policy.

CLAIM $|\tilde{V} - V^*| < \epsilon \Rightarrow < \frac{2\epsilon\gamma}{1-\gamma}$.

PROOF Let $\bar{s} = \arg\max_{s\in\mathbb{S}} L_{\tilde{V}}(\bar{s})$. This exists by the assumption in CLAIM. Let $a\mathbb{A}$ be the optimal choice $\pi^*(\bar{s}) = a$ and b be the greedy choice $\pi_g(\bar{s}) = b$. Because b is chosen with greedy policy, it has to be at least as good as a:

$$\tilde{V}(s) = R(\bar{s}, a) + \gamma \sum_{s'\in\mathbb{S}} P(s'|\bar{s}, a)\tilde{V}(s') \leq R(\bar{s}, b) + \sum_{s'\in\mathbb{S}} P(s'|\bar{s}, b)\tilde{V}(s') \tag{1}$$

By assumed property, $V^*(s) - \epsilon < \tilde{V}(s) < V^*(s) + \epsilon, \forall s \in \mathbb{S}$, hence (1) gives

$$R(\bar{s}, a) + \gamma\gamma \sum_{s'\in\mathbb{S}} P(s'|\bar{s}, a) \left(V^*(s') - \epsilon\right) \leq R(\bar{s}, b) + \gamma \sum_{s'\in\mathbb{S}} P(s'|\bar{s}, b) \left(V^*(s') + \epsilon\right)$$

hence

$$R(\bar{s}, a) - R(\bar{s}, b) \leq 2\gamma\epsilon + \gamma \sum_{s'\in\mathbb{S}} \left(P(s'|\bar{s}, b)V^*(s') - P(s'|\bar{s}, a)V^*(s')\right) \tag{2}$$

Because $\bar{s}$ was chosen so that it maximizes $L_V(s)$ for an arbitrary value function V, in particular it maximizes the loss for the greedy policy value function L_{V_g}, hence, using the fact that g is the greedy policy choice, as well as (2) we have

$$\begin{aligned}
L_{V_g}(\bar{s}) =& V^*(\bar{s}) - V_g(\bar{s}) \\
=& R(\bar{s}, a) - R(\bar{s}, b) + \gamma \sum_{s'\in\mathbb{S}} \left(P(s'|\bar{s}, a)V^*(s') - P(s'|\bar{s}, b)V_g^*(s')\right) \\
\leq& 2\epsilon\gamma + \gamma \sum_{s'\in\mathbb{S}} \left(P(s'|\bar{s}, b)V^*(s') - P(s'|\bar{s}, a)V^*(s') + P(s'|\bar{s}, a)V^*(s') - P(s'|\bar{s}, b)V_g^*(s')\right) \\
=& 2\epsilon\gamma + \gamma \sum_{s'\in\mathbb{S}} \left(P(s'|\bar{s}, b)V^*(s') - P(s'|\bar{s}, b)V_g(s')\right) \\
=& 2\epsilon\gamma + \gamma \sum_{s'\in\mathbb{S}} P(s'|\bar{s}, b)L_{V_g}(s') \\
\leq& 2\epsilon\gamma + \gamma \sum_{s'\in\mathbb{S}} P(s'|\bar{s}, b)L_{V_g}(\bar{s})
\end{aligned}$$

rearranging we get that for every $s \in \mathbb{S}$

$$L_{V_g}(s) =\leq L_{V_g}(\bar{s}) \leq \frac{2\epsilon\gamma}{1-\gamma}$$

Q.E.D.

4

4.a

We implement `policy_evaluation`and `policy_improvement`to run the `policy_itertation`algorithm.

`policy_evaluation`evaluates the update for value function at each state, according to Bellman equation. `policy_improvement`uses the `policy_evaluation`to greedily choose the new action at each state.

Full code is attached. The result is shown in 1 (a).

4.b

We implement `value_iteration`algorithm. We don't evaluate policy as before, instead we just select action that maximises the value function update according to Bellman equation.

Full code is attached. The result is shown in 1 (b).

We also present computing time (code executed on our laptop) for comparison. Evidently, **value iteration is significantly faster than policy iteration**, since in policy iteration we need to perform "policy evaluation" for each updated value function. Note that the optimal policy found in the end is the same for both cases, which assures us that the code is correct and both methods converge to the same optimal policy.

(a)

```
-------------------------
Beginning Policy Iteration
-------------------------

SFFF
FHFH
FFFH
HFFG
  (Down)
SFFF
FHFH
FFFH
HFFG
  (Down)
SFFF
FHFH
FFFH
HFFG
  (Right)
SFFF
FHFH
FFFH
HFFG
  (Down)
SFFF
FHFH
FFFH
HFFG
  (Right)
SFFF
FHFH
FFFH
HFFG
  (Right)
SFFF
FHFH
FFFH
HFFG
Episode reward: 1.000000
Policy iteration computation time: 190.18560300000001
```

(b)

```
-------------------------
Beginning Value Iteration
-------------------------

SFFF
FHFH
FFFH
HFFG
  (Down)
SFFF
FHFH
FFFH
HFFG
  (Down)
SFFF
FHFH
FFFH
HFFG
  (Right)
SFFF
FHFH
FFFH
HFFG
  (Down)
SFFF
FHFH
FFFH
HFFG
  (Right)
SFFF
FHFH
FFFH
HFFG
  (Right)
SFFF
FHFH
FFFH
HFFG
Episode reward: 1.000000
Value iteration computation time: 0.6372270000000242
```

Figure 1: Optimal policy found via (a) policy iteration and (b) value iteration. Below each table we present the computing time for comparison.

4.c

We now re-run both value and policy iteration -based algorithm for the stochastic case (non-deterministic actions).

1. The value iteration: takes longer to converge and the optimal policy for stochastic case is different than in deterministic case (12 steps versus 7 steps).

2. The policy iteration: might not converge / needs much longer to converge. After the first run of 100 episodes it did not reach an optimal policy (ended up in a HOLE field instead of the target field). We re-ran it over 500 episodes for comparison and it did not converge either, hence we conclude it might either not converge at all, or do so asymptotically in which case computing the optimal policy is not computationally efficient.

5

We are concerned with a finite-horizon MDP, where agent is uncertain about the true dynamics of the MDP, and learns the system dynamics through exploration.

[1] develops posterior sampling for reinforcement learning (PSRL) which expands on the idea of Thompson sampling[2], and provides a bound on expected regret not based on optimism. The paper provides several experiments proving this approach to significantly outperform existing algorithms with similar regret bounds.

Majority of previous algorithms would adopt the OFU (*optimism in the face of uncertainty*) rule. This means that the unknown parameters in the system are assigned maximum statistically possible value. This assumption is arbitrary. The agent is exploring poorly known policies, until it converges to optimal policy.

However, while useful in practice, there is no reason why OFU should be taken for granted. [1] develops an alternative approach, PSRL, where we optimize over MDP samples, taken at each iteration from the new, updated posterior distribution.

In summary, OFU explores the whole MDP by exploration of poorly understood policies. PSLR maximizes over sample each time, without prioritizing a particular policy. In this, PSLR is less biased in a way than OFU. It turns out that PSLR approach converges faster than OFU.

As a test, [1] applies PSRL to the River Swim problem, as well as several randomly generated MDPs.

As a further expansion, more testing could be conducted on examples were OFU is commonly used, to verify that PSRL is indeed a better approach. While providing an improved theoretical upper bound on expected regret, more experiments could better establish that this bound is indeed useful in practice (that the algorithm reaches close to the bound).

References

[1] Osband I., Russo D., Van Roy B., "(More) efficient reinforcement learning via posterior sampling", *Advances in Neural Information Processing Systems*, 2013, available at ArXiv: 1306.0940 (accessed 2020-03-28).

[2] Thomson, W.R., "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples", *Biometrika*, 25(3/4):285-294, 1933