1 MDPs: Racing

Consider a modification of the racing robot car example seen in lecture. In this game, the car repeatedly moves a random number of spaces that is equally likely to be 2, 3, or 4. The car can either Move or Stop if the total number of spaces moved is less than 6.

If the total spaces moved is 6 or higher, the game automatically ends, and the car receives a reward of 0. When the car Stops, the reward is equal to the total spaces moved (up to 5), and the game ends. There is no reward for the Move action.

Let's formulate this problem as an MDP with the states $\{0, 2, 3, 4, 5, Done\}$.

(a) What is the transition function for this MDP? (You should specify discrete values for specific state/action inputs.)

$T(s, Stop, Done) = 1$
$T(0, Move, s') = \frac{1}{3}$ for $s' \in \{2, 3, 4\}$
$T(2, Move, s') = \frac{1}{3}$ for $s' \in \{4, 5, Done\}$
$T(3, Move, 5) = \frac{1}{3}$
$T(3, Move, Done) = \frac{2}{3}$
$T(4, Move, Done) = 1$
$T(5, Move, Done) = 1$
$T(s, a, s') = 0$ otherwise.

(b) What is the reward function for this MDP?

$R(s, Stop, Done) = s, s \leq 5$
$R(s, a, s') = 0$ otherwise

(c) Recall the value iteration update equation:

$$V_{k+1}(s) = \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V_k(s')]$$

Perform value iteration for 4 iterations with $\gamma = 1$.

States	0	2	3	4	5
V_0					
V_1					
V_2					
V_3					
V_4					

States	0	2	3	4	5
V_0	0	0	0	0	0
V_1	0	2	3	4	5
V_2	3	3	3	4	5
V_3	$\frac{10}{3}$	3	3	4	5
V_4	$\frac{10}{3}$	3	3	4	5

(d) You should have noticed that value iteration converged above. What is the optimal policy?

States	0	2	3	4	5
π^*					

States	0	2	3	4	5
π^*	Move	Move	Stop	Stop	Stop

(e) How would our results change with $\gamma = 0.1$?

By increasing the discount (decreasing γ), we focus more and more on immediate rewards (this is discussed more in 3a). This effectively makes our algorithm more greedy, valuing short-term rewards more than long-term ones.
For this game, with a discount factor of 0.1, value iteration converges in fewer iterations, but upon a suboptimal policy (Move, Stop, Stop, Stop, Stop). For state 2, the algorithm preferred the short-term reward of Stopping over the long-term reward of Moving.
In the most extreme case with $\gamma = 0$, the values converge immediately and yield a policy to Stop at all states.

(f) Now recall the policy evaluation and policy improvement equations, which together make up policy iteration:

$$V_{k+1}^{\pi}(s) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V_k^{\pi}(s')]$$

$$\pi_{new}(s) = \underset{a}{\text{argmax}} \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^{\pi_{old}}(s')]$$

Perform two iterations of policy iteration for one step of this MDP, starting from the fixed policy below. Use the initial $\gamma = 1$.

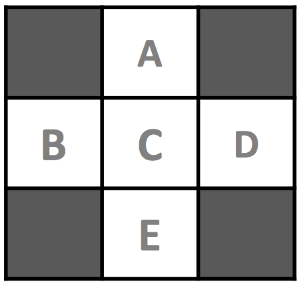
States	0	2	3	4	5
π_0	Move	Stop	Move	Stop	Move
V^{π_0}					
π_1					
V^{π_1}					
π_2					

States	0	2	3	4	5
π_0	Move	Stop	Move	Stop	Move
$V_0^{\pi_0}$	0	0	0	0	0
$V_1^{\pi_0}$	0	2	0	4	0
$V_2^{\pi_0}$	2	2	0	4	0
$V_3^{\pi_0}$	2	2	0	4	0
π_1	Move	Stop	Stop	Stop	Stop
$V_0^{\pi_1}$	0	0	0	0	0
$V_1^{\pi_1}$	0	2	3	4	5
$V_2^{\pi_1}$	3	2	3	4	5
$V_3^{\pi_1}$	3	2	3	4	5
π_2	Move	Move	Stop	Stop	Stop

Side note: above when evaluating π_1, we started policy evaluation off with all 0s again - this is called a cold start (terminology not super important). We also could have started off instead with the optimal values we'd converged upon last round of policy evaluation ($V_3^{\pi_0}$), which would converge upon the optimal values for π_1 faster.

2 Reinforcement Learning

Consider the Gridworld example that we looked at in lecture. We would like to use TD learning to find the values of these states.



Suppose we observe the following transitions:

$(B$, East, C, 2), (C, South, E, 4), (C, East, A, 6), (B, East, C, 2)

The initial value of each state is 0. Let $\gamma = 1$ and $\alpha = 0.5$.

(a) What are the learned values for each state from TD learning after all four observations?

$V(B) = 3.5$; $V(C) = 4; V(s) = 0 \ \forall s \in \{A, D, E\}$
Here are our intermediate computations - the values of each state after each transition are shown below:

Transitions	A	B	C	D	E
(initial)	0	0	0	0	0
(B, East, C, 2)	0	1	0	0	0
(C, South, E, 4)	0	1	2	0	0
(C, East, A, 6)	0	1	4	0	0
(B, East, C, 2)	0	3.5	4	0	0

(b) In class, we presented the following two formulations for TD-learning:

$$V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + (\alpha)sample \tag{1}$$

$$V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(sample - V^{\pi}(s)) \tag{2}$$

Mathematically, these two equations are equivalent. However, they represent two conceptually different ways of understanding TD value updates. How might we explain each of these equations?

The first equation takes a weighted average between our current values and our new sample. We might think of this as computing an expected value.

The second equation updates our current values towards the new sample value, scaled by a factor of our learning rate, α. This is where the "temporal difference" term is motivated (for those of you familiar, this is gradient descent, where $(sample - V^{\pi}(s))$ is the gradient. We'll cover this later on too!).

3 Discussion-Based Questions

(a) In class, we learned that the Bellman Equations can be used to characterize optimal utility in MDPs. For reference, recall that this equation is given as:

$$V^*(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

What do we call γ in this equation? Why is it necessary? What happens as γ grows larger? As it grows smaller?

γ is referred to as the *discount factor*, and is $\in [0,1]$. We need it to guarantee that our algorithms will converge (and to prevent against infinite rewards if our game lasts forever, for example)! Also, intuitively, it makes sense to value immediate rewards more highly than rewards obtained at a later time.

A smaller γ indicates smaller "horizon," or a shorter term focus. As γ increases to 1 (no discount), we begin to act as though rewards at any given point in time are equally valuable. As γ decreases to 0, we begin to only count our immediate rewards.

(b) What are the key distinctions between the value iteration and policy iteration algorithms, and when might you prefer one to the other?

Possible answers: policy iteration is focused on evaluating the policies themselves, while value iteration evaluates states or state-action pairs and implicitly derives a policy from there.
Policy iteration often converges faster than value iteration, so we would generally prefer policy iteration over value iteration.

(c) When does policy iteration end? Immediately after policy iteration ends (without performing additional computation), do we have the values of the optimal policy?

Policy iteration ends when the policy converges, ie. when $\pi_{new} = \pi_{old}$ after running policy improvement.

We do have the values for the optimal policy. Since we necessarily ran policy evaluation on the most recent iteration of policy iteration, we have the value function $V^{\pi_{old}}$ corresponding to π_{old}. We know that $\pi_{new} = \pi_{old}$, implying that $V^{\pi_{old}} = V^{\pi_{new}}$. Therefore, we have $V^{\pi_{new}}$, which is the value function corresponding to the optimal policy π_{new}.

(d) What changes if during policy iteration, you only run one iteration of policy evaluation instead of running it until convergence? Do you still get an optimal policy?

Yes, you will still get an optimal policy. The key observation here is that this procedure is effectively the same as value iteration, since value iteration involves one step of evaluation as well. In this case, we update values based on our current best policy, and keep doing that until convergence of a policy!

Recall that the equation for value iteration is:

$$V_{k+1}(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_k(s')] \tag{3}$$

And the equation for policy evaluation is:

$$V_{k+1}^{\pi_i}(s) = \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V_k^{\pi_i}(s')] \tag{4}$$

given some fixed policy π that we update in the policy improvement step, given by:

$$\pi_{i+1}(s) \leftarrow \arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_k^{\pi_i}(s')] \tag{5}$$

which looks very similar to policy extraction done in value iteration.

(e) **(Bonus)** Think of a problem that seems like it can be modeled using an MDP, and formulate it (what are its states, reward function, and transition function)? Once you're complete, swap with a partner and verify your formulation. Try to get creative! There's quite a lot that can be modeled as an MDP.

Answers will vary as a function of the chosen problem.