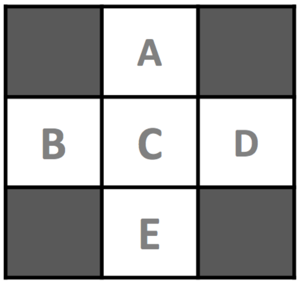
15-281: AI: Representation and Problem Solving Fall 2019

RL Worksheet October 25

1 Temporal Difference and Q-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 $(s, a, s', R(s, a, s'))$* transitions and rewards:

$$(B, \text{East}, C, 2), (C, \text{South}, E, 4), (C, \text{East}, A, 6), (B, \text{East}, C, 2)$$

**Note that the $R(s, a, s')$ in this notation refers to observed reward, not a reward value computed from a reward function.*

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?

For $(B$, East, C, 2), we update $V^\pi(B)$:
$V^\pi(B) \leftarrow V^\pi(B) + \alpha(R(s, a, s') + \gamma V^\pi(C) - V^\pi(B)) = 0 + 0.5(2 + 1 * 0 - 0) = 1.$
Following the same computation, we get final values: $V(B) = 3.5$; $V(C) = 4; V(s) = 0\ \forall s \in \{A, D, E\}$
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:

$$V^\pi(s) \leftarrow (1 - \alpha)V^\pi(s) + (\alpha)sample \quad (1)$$

$$V^\pi(s) \leftarrow V^\pi(s) + \alpha(sample - V^\pi(s)) \quad (2)$$

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

The first equation takes a weighted average between our current values and our new sample. We can 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.).

(c) What are the learned Q-values from Q-learning after all four observations? Use the same $\alpha = 0.5, \gamma = 1$ as before.

$Q(C, South) = 2; Q(C, East) = 3; Q(B, East) = 2;$ $Q(s, a) = 0$ for all other Q-states (s, a).
Here are our intermediate computations - the values of each Q-state after each transition are shown below (Q-states for which values did not change are omitted):

Transitions	$(B, East)$	$(C, South)$	$(C, East)$
(initial)	0	0	0
(B, East, C, 2)	1	0	0
(C, South, E, 4)	1	2	0
(C, East, A, 6)	1	2	3
(B, East, C, 2)	3	2	3

2 RL: Conceptual Questions

Recall that in Q-learning, we continually update the values of each Q-state by learning through a series of episodes, ultimately converging upon the optimal policy.

(a) What's the main shortcoming of TD learning that Q-learning resolves?

TD value learning provides a value for each state for a given policy π. It is impossible to get the optimal policy directly from the learned values because the state values are learned for the given policy π. And if we want to follow policy iteration to extract an improved policy from these values, we would need to use the R and T functions (which we don't have). With Q-learning, we can get values of Q-states (i.e., (state, action) pairs) of the optimal policy, from which we can extract an optimal policy simply by taking the action corresponding to the maximum Q-value from each state.

(b) We are given a pre-existing table of current estimate of Q-values (and its corresponding policy), and asked to perform ϵ-greedy Q-learning. Individually, what effect does setting each of the following constants to 0 have on this process?

(i) α:

$Q(s,a) = Q(s,a) + \alpha[r + \gamma max_{a'} Q(s',a') - Q(s,a)]$ becomes $Q(s,a)$.
We put 0 weight on newly observed samples, never updating the Q-values we already have.
Additional remarks about the value of α: α is the learning rate or step size determining to what extent newly acquired information overrides old information. When the environment is stochastic, the algorithm converges under some technical conditions on the learning rate that require it to decrease to zero. In practice, sometimes a constant learning rate is used, such as $\alpha_t = 0.1$ for all t. If you want to learn more about learning rate in Q-learning, you can search for research papers, e.g., Even-Dar and Mansour, JMLR 2005 (http://www.jmlr.org/papers/volume5/evendar03a/evendar03a.pdf).

(ii) γ:

$Q(s,a) = Q(s,a) + \alpha[r + \gamma max_{a'} Q(s',a') - Q(s,a)]$ becomes $(1-\alpha)Q(s,a) + \alpha r$.
Our valuation of reward becomes short-sighted, as we weight Q-values of successor states with 0. Continue the Q-learning process with $\gamma = 0$ and gradually decreasing α will eventually lead to Q-values of $Q(s,a) = \sum_{s'} T(s,a,s')R(s,a,s')$ because we only care about immediate reward.

(iii) ϵ:

By definition of an ϵ-greedy policy, we randomly select actions with probability 0 and select our policy's recommended action with probability 1; we exclusively exploit the policy we already have.

(c) Consider a variant of the ϵ-greedy Q-learning algorithm that is changed such that instead of using the policy extracted from our current Q-values, we use a fixed policy instead. We still perform exploration with probability ϵ. If this fixed policy happens to be optimal, how does the performance of this algorithm compare to normal ϵ-greedy Q-learning?

Both algorithms will result in finding the optimal Q-values eventually. However, normal ϵ-greedy Q-learning makes more mistakes along the way, racking up more *regret* (the difference between actual yielded rewards and the optimal expected rewards).
In practice, normal ϵ-greedy Q-learning with a small ϵ may lead to a policy that is "pretty good" but not necessarily optimal, thus making it very unlikely for it to change unless given an extremely high number of iterations to allow for random chance to find a better policy. This result is known as a local optimum. ϵ-greedy Q-learning is in spirit similar to the simulated annealing algorithm in local search.

(d) Let's revisit the CandyGrab code from recitation 1 (https://www.cs.cmu.edu/~15281/recitations/rec1/candygrab.zip). What RL strategies does `AgentRL` employ? Does it evaluate states or Q-states?

`AgentRL` plays out each game (either randomly playing each round with probability ϵ if `explore_mode` is on, or by exploiting its learned policy) and records the lose/win rate for each state, action pair seen along the way.
`AgentRL` uses direct evaluation, with an option to execute ϵ-greedy exploration. It evaluates Q-states.

(e) Contrast the following pairs of reinforcement learning terms:

(i) Off-policy vs. on-policy learning

An off-policy learning algorithm learns the value of the optimal policy independently of the policy based on which the agent chooses actions. Q-learning is an off-policy learning algorithm. An on-policy learning algorithm learns the value of the policy being carried out by the agent.

(ii) Model-based vs. model-free

In model-based learning, we estimate the transition and reward functions by taking some actions, then solve the MDP using them. In model-free learning, we don't attempt to model the MDP, and instead just try to learn the values directly.

(iii) Passive vs. active

Passive learning involves using a fixed policy as we try to learn the values of our states, while active learning involves improving the policy as we learn.