CS 188
Spring 2019

Section Handout 4 Solutions

Temporal Difference Learning

Temporal difference learning (TD learning) uses the idea of *learning from every experience*, rather than simply keeping track of total rewards and number of times states are visited and learning at the end as direct evaluation does. In policy evaluation, we used the system of equations generated by our fixed policy and the Bellman equation to determine the values of states under that policy (or used iterative updates like with value iteration).

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

Each of these equations equates the value of one state to the weighted average over the discounted values of that state's successors plus the rewards reaped in transitioning to them. TD learning tries to answer the question of how to compute this weighted average without the weights, cleverly doing so with an **exponential moving average**. We begin by initializing $\forall s,\ V^\pi(s) = 0$. At each timestep, an agent takes an action $\pi(s)$ from a state s, transitions to a state s', and receives a reward $R(s, \pi(s), s')$. We can obtain a **sample value** by summing the received reward with the discounted current value of s' under π:

$$sample = R(s, \pi(s), s') + \gamma V^\pi(s')$$

This sample is a new estimate for $V^\pi(s)$. The next step is to incorporate this sampled estimate into our existing model for $V^\pi(s)$ with the exponential moving average, which adheres to the following update rule:

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

Above, α is a parameter constrained by $0 \leq \alpha \leq 1$ known as the **learning rate** that specifies the weight we want to assign our existing model for $V^\pi(s)$, $1-\alpha$, and the weight we want to assign our new sampled estimate, α. It's typical to start out with learning rate of $\alpha = 1$, accordingly assigning $V^\pi(s)$ to whatever the first *sample* happens to be, and slowly shrinking it towards 0, at which point all subsequent samples will be zeroed out and stop affecting our model of $V^\pi(s)$.

Let's stop and analyze the update rule for a minute. Annotating the state of our model at different points in time by defining $V_k^\pi(s)$ and $sample_k$ as the estimated value of state s after the k^{th} update and the k^{th} sample respectively, we can reexpress our update rule:

$$V_k^\pi(s) \leftarrow (1-\alpha)V_{k-1}^\pi(s) + \alpha \cdot sample_k$$

This recursive definition for $V_k^\pi(s)$ happens to be very interesting to expand:

$$\begin{aligned}
V_k^\pi(s) &\leftarrow (1-\alpha)V_{k-1}^\pi(s) + \alpha \cdot sample_k \\
V_k^\pi(s) &\leftarrow (1-\alpha)[(1-\alpha)V_{k-2}^\pi(s) + \alpha \cdot sample_{k-1}] + \alpha \cdot sample_k \\
V_k^\pi(s) &\leftarrow (1-\alpha)^2 V_{k-2}^\pi(s) + (1-\alpha) \cdot \alpha \cdot sample_{k-1} + \alpha \cdot sample_k \\
&\vdots \\
V_k^\pi(s) &\leftarrow (1-\alpha)^k V_0^\pi(s) + \alpha \cdot [(1-\alpha)^{k-1} \cdot sample_1 + \ldots + (1-\alpha) \cdot sample_{k-1} + sample_k] \\
V_k^\pi(s) &\leftarrow \alpha \cdot [(1-\alpha)^{k-1} \cdot sample_1 + \ldots + (1-\alpha) \cdot sample_{k-1} + sample_k]
\end{aligned}$$

Because $0 \leq (1-\alpha) \leq 1$, as we raise the quantity $(1-\alpha)$ to increasingly larger powers, it grows closer and closer to 0. By the update rule expansion we derived, this means that older samples are given exponentially less

weight, exactly what we want since these older samples are computed using older (and hence worse) versions of our model for $V^{\pi}(s)$! This is the beauty of temporal difference learning - with a single straightfoward update rule, we are able to:

- learn at every timestep, hence using information about state transitions as we get them since we're using iteratively updating versions of $V^{\pi}(s')$ in our samples rather than waiting until the end to perform any computation.
- give exponentially less weight to older, potentially less accurate samples.
- converge to learning true state values much faster with fewer episodes than direct evaluation.

Q-Learning

Both direct evaluation and TD learning will eventually learn the true value of all states under the policy they follow. However, they both have a major inherent issue - we want to find an optimal *policy* for our agent, which requires knowledge of the q-values of states. To compute q-values from the values we have, we require a transition function and reward function as dictated by the Bellman equation.

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

Resultingly, TD learning or direct evaluation are typically used in tandem with some model-based learning to acquire estimates of T and R in order to effectively update the policy followed by the learning agent. This became avoidable by a revolutionary new idea known as **Q-learning**, which proposed learning the q-values of states directly, bypassing the need to ever know any values, transition functions, or reward functions. As a result, Q-learning is entirely model-free. Q-learning uses the following update rule to perform what's known as **q-value iteration**:

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')]$$

Note that this update is only a slight modification over the update rule for value iteration. Indeed, the only real difference is that the position of the max operator over actions has been changed since we select an action before transitioning when we're in a state, but we transition before selecting a new action when we're in a q-state.

With this new update rule under our belt, Q-learning is derived essentially the same way as TD learning, by acquiring **q-value samples**:

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

and incoporating them into an exponential moving average.

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha \cdot sample$$

As long as we spend enough time in exploration and decrease the learning rate α at an appropriate pace, Q-learning learns the optimal q-values for every q-state. This is what makes Q-learning so revolutionary - while TD learning and direct evaluation learn the values of states under a policy by following the policy before determining policy optimality via other techniques, Q-learning can learn the optimal policy directly even by taking suboptimal or random actions. This is called **off-policy learning** (contrary to direct evaluation and TD learning, which are examples of **on-policy learning**).

1 Odds and Ends

(a) Can all MDPs be solved using expectimax search? Justify your answer.

No, MDPs with self loops lead to infinite expectimax trees. Unlike search problems, this issue cannot be addressed with a graph-search variant.

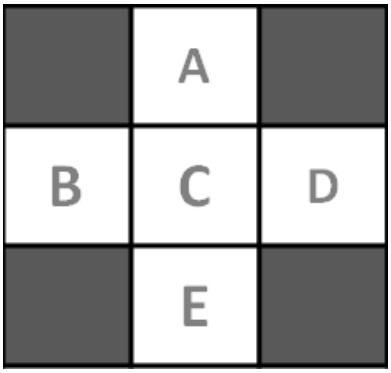
(b) Why might Q-learning be superior to TD learning of values?

(a) If you use temporal difference learning on the values, it is hard to extract a policy from the learned values. Specifically, you would need to know the transition model T and reward function R. For Q-learning, the policy can be extracted directly by taking $\pi(s) = \arg\max_a Q(s, a)$.

(b) While TD learning learns values for a particular policy (online policy evaluation), Q-learning is *off-policy*: it will converge to Q-values that give you the optimal policy, even if the actions you used to explore were sub-optimal. The caveats to this are that you need to explore enough, and you need conditions on the learning rate α (has to eventually be small enough, but not decrease too fast)

2 Learning in Gridworld

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



Suppose that we have the following observed 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. Assume that $\gamma = 1$ and $\alpha = 0.5$.

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

$V(B) = 3.5$
$V(C) = 4$
All other states have a value of 0.

(b) What are the learned Q-values from Q-learning after all four observations?

$Q(B, East) = 3$
$Q(C, South) = 2$
$Q(C, East) = 3$
All other q-states have a value of 0.

Q3. MDPs and RL: Mini-Grids

The following problems take place in various scenarios of the gridworld MDP (as in Project 3). In all cases, A is the start state and double-rectangle states are exit states. From an exit state, the only action available is *Exit*, which results in the listed reward and ends the game (by moving into a terminal state X, not shown).

From non-exit states, the agent can choose either *Left* or *Right* actions, which move the agent in the corresponding direction. There are no living rewards; the only non-zero rewards come from exiting the grid.

Throughout this problem, assume that value iteration begins with initial values $V_0(s) = 0$ for all states s.

First, consider the following mini-grid. For now, the discount is $\gamma = 1$ and legal movement actions will always succeed (and so the state transition function is deterministic).

(a) What is the optimal value $V^*(A)$?

10

Since the discount $\gamma = 1$ and there are no rewards for any action other than exiting, a policy that simply heads to the right exit state and exits will accrue reward 10. This is the optimal policy, since the only alternative reward if 1, and so the optimal value function has value 10.

(b) When running value iteration, remember that we start with $V_0(s) = 0$ for all s. What is the first iteration k for which $V_k(A)$ will be non-zero?

2
The first reward is accrued when the agent does the following actions (state transitons) in sequence: Left, Exit. Since two state transitions are necessary before any possible reward, two iterations are necessary for the value function to become non-zero.

(c) What will $V_k(A)$ be when it is first non-zero?

1
As explained above, the first non-zero value function value will come from exiting out of the left exit cell, which accrues reward 1.

(d) After how many iterations k will we have $V_k(A) = V^*(A)$? If they will never become equal, write *never*.

4
The value function will equal the optimal value function when it discovers this sequence of state transitions: Right, Right, Right, Exit. This will happen in 4 iterations.

Now the situation is as before, but the discount γ is less than 1.

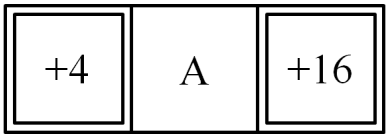
(e) If $\gamma = 0.5$, what is the optimal value $V^*(A)$?

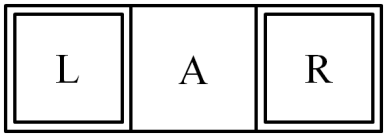
The optimal policy from A is Right, Right, Right, Exit. The rewards accrued by these state transitions are: 0, 0, 0, 10. The discount values are $\gamma^0, \gamma^1, \gamma^2, \gamma^3$, which is 1, $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$. Therefore, $V^*(A) = 0 + 0 + 0 + \frac{10}{8}$.

(f) For what range of values γ of the discount will it be optimal to go *Right* from A? Remember that $0 \leq \gamma \leq 1$. Write *all* or *none* if all or no legal values of γ have this property.

The best reward accrued with the policy of going left is $\gamma^1 * 1$. The best reward accrued with the policy of going right is $\gamma^3 * 10$. We therefore have the inequality $10\gamma^3 \geq \gamma$, which simplifies to $\gamma \geq \sqrt{1/10}$. The final answer is $1/\sqrt{10} \leq \gamma \leq 1$

Finally, consider the following mini-grid (rewards shown on left, state names shown on right).





In this scenario, the discount is $\gamma = 1$. The failure probability is actually $f = 0$, but, now we do not actually know the details of the MDP, so we use reinforcement learning to compute various values. We observe the following transition sequence (recall that state X is the end-of-game absorbing state):

s	a	s'	r
A	*Right*	R	0
R	*Exit*	X	16
A	*Left*	L	0
L	*Exit*	X	4
A	*Right*	R	0
R	*Exit*	X	16
A	*Left*	L	0
L	*Exit*	X	4

(g) After this sequence of transitions, if we use a learning rate of $\alpha = 0.5$, what would temporal difference learning learn for the value of A? Remember that $V(s)$ is intialized with 0 for all s.

3. Remember how temporal difference learning works: upon seeing a s, a, r, s' tuple, we update the value function as $V_{i+1}(s) = (1 - \alpha)V_i(s) + \alpha(r + V_i(s'))$. To get the answer, simply write out a table of states, all initially with value 0, and then update it with information in each row of the table above. When all rows have been processed, see what value you ended up with for A.

(h) If these transitions repeated many times and learning rates were appropriately small for convergence, what would temporal difference learning converge to for the value of A?

10. We are simply updating the value function with the results of following this policy, and that's what we will converge to. For state A, the given tuples show the agent going right as often as it goes left If the agent goes left as often as it goes right from A, the value of being in A is only $16/2 + 4/2 = 10$.

(i) After this sequence of transitions, if we use a learning rate of $\alpha = 0.5$, what would Q-learning learn for the Q-value of (A, *Right*)? Remember that $Q(s, a)$ is initialized with 0 for all (s, a).

4. The technique is the same as in problem (o), but use the Q-learning update (which includes a max). How do you get the max? Here's an example:

The sample sequence: $(A, Right, R, 0)$.

$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha(r + \gamma \max_{a'}(s', a'))$.

$Q(A, right) \leftarrow (1 - \alpha)Q(A, right) + \alpha(r + \gamma \max_{a'}(R, a'))$.

But since there is only one exit action from R, then:

$Q(A, right) \leftarrow (1 - \alpha)Q(A, right) + \alpha(r + \gamma Q(R, Exit))$.

Note that this MDP is very small – you will finish the game in two moves (assuming you have to move from A).

(j) If these transitions repeated many times and learning rates were appropriately small for convergence, what would Q-learning converge to for the Q-value of (A, *Right*)?

16. Q-learning converges to the optimal Q-value function, if the states are fully explored and the convergence rate is set correctly.