CS 188
Spring 2024

Regular Discussion 13 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 $0 \le \alpha \le 1$ 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 $0 \le (1-\alpha) \le 1$, 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 incorporating 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**).

Q1. Reinforcement Learning

Imagine an unknown environments with four states (A, B, C, and X), two actions (← and →). An agent acting in this environment has recorded the following episode:

s	a	s'	r	Q-learning iteration numbers (for part b)
A	→	B	0	1, 10, 19, ...
B	→	C	0	2, 11, 20, ...
C	←	B	0	3, 12, 21, ...
B	←	A	0	4, 13, 22, ...
A	→	B	0	5, 14, 23, ...
B	→	A	0	6, 15, 24, ...
A	→	B	0	7, 16, 25, ...
B	→	C	0	8, 17, 26, ...
C	→	X	1	9, 18, 27, ...

(a) Consider running model-based reinforcement learning based on the episode above. Calculate the following quantities:

$$\hat{T}(B, \rightarrow, C) = \boxed{\frac{2}{3}}$$

$$\hat{R}(C, \rightarrow, X) = \boxed{1}$$

(b) Now consider running Q-learning, repeating the above series of transitions in an infinite sequence. Each transition is seen at multiple iterations of Q-learning, with iteration numbers shown in the table above.

After which iteration of Q-learning do the following quantities first become nonzero? (If they always remain zero, write *never*).

$Q(A, \rightarrow)$? $\boxed{14}$

$Q(B, \leftarrow)$? $\boxed{22}$

(c) True/False: For each question, you will get positive points for correct answers, zero for blanks, and negative points for incorrect answers. Circle your answer **clearly**, or it will be considered incorrect.

(i) [[✓ *true*] or *false*] In Q-learning, you do not learn the model.
Q learning is model-free, you learn the optimal policy explicitly, and the model itself implicitly.

(ii) [*true* or [✓ *false*]] For TD Learning, if I multiply all the rewards in my update by some nonzero scalar p, the algorithm is still guaranteed to find the optimal policy.
TD Learning does not necessarily find the optimal policy, it only learns the value of the states following some given policy.

(iii) [*true* or [✓ *false*]] In Direct Evaluation, you recalculate state values after each transition you experience.

In order to estimate state values, you calculate state values from episodes of training, not single transitions.

(iv) [*true* or [✓ *false*]] Q-learning requires that all samples must be from the optimal policy to find optimal q-values.
Q-learning is off-policy, you can still learn the optimal values even if you act suboptimally sometimes.

Q2. RL

Pacman is in an unknown MDP where there are three states [A, B, C] and two actions [Stop, Go]. We are given the following samples generated from taking actions in the unknown MDP. For the following problems, assume $\gamma = 1$ and $\alpha = 0.5$.

(a) We run Q-learning on the following samples:

s	**a**	**s'**	**r**
A	Go	B	2
C	Stop	A	0
B	Stop	A	-2
B	Go	C	-6
C	Go	A	2
A	Go	A	-2

What are the estimates for the following Q-values as obtained by Q-learning? All Q-values are initialized to 0.

(i) $Q(C, Stop) =$ ___0.5___

(ii) $Q(C, Go) =$ ___1.5___

For this, we only need to consider the following three samples.

$$Q(A, Go) \leftarrow (1-\alpha)Q(A, Go) + \alpha(r + \gamma \max_a Q(B, a)) = 0.5(0) + 0.5(2) = 1$$
$$Q(C, Stop) \leftarrow (1-\alpha)Q(C, Stop) + \alpha(r + \gamma \max_a Q(A, a)) = 0.5(0) + 0.5(1) = 0.5$$
$$Q(C, Go) \leftarrow (1-\alpha)Q(C, Go) + \alpha(r + \gamma \max_a Q(A, a)) = 0.5(0) + 0.5(3) = 1.5$$

(b) For this next part, we will switch to a feature based representation. We will use two features:

- $f_1(s, a) = 1$
- $f_2(s, a) = \begin{cases} 1 & a = \text{Go} \\ -1 & a = \text{Stop} \end{cases}$

Starting from initial weights of 0, compute the updated weights after observing the following samples:

s	**a**	**s'**	**r**
A	Go	B	4
B	Stop	A	0

What are the weights after the first update? (using the first sample)

(i) $w_1 =$ ___2___

(ii) $w_2 =$ ____2____

$$\begin{aligned}Q(A, Go) &= w_1 f_1(A, Go) + w_2 f_2(A, Go) = 0\\ difference &= [r + max_a Q(B, a)] - Q(A, Go) = 4\\ w_1 &= w_1 + \alpha(difference) f_1 = 2\\ w_2 &= w_2 + \alpha(difference) f_2 = 2\end{aligned}$$

What are the weights after the second update? (using the second sample)

(iii) $w_1 =$ ____4____

(iv) $w_2 =$ ____0____

$$\begin{aligned}Q(B, Stop) &= w_1 f_1(B, Stop) + w_2 f_2(B, Stop) = 2(1) + 2(-1) = 0\\ Q(A, Go) &= w_1 f_1(A, Go) + w_2 f_2(A, Go) = 2(1) + 2(1) = 4\\ difference &= [r + max_a Q(A, a)] - Q(B, Stop) = [0 + 4] - 0 = 4\\ w_1 &= w_1 + \alpha(difference) f_1 = 4\\ w_2 &= w_2 + \alpha(difference) f_2 = 0\end{aligned}$$