CS 188
Spring 2020

Section Handout 6 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**).

Q1. Pacman with Feature-Based Q-Learning

We would like to use a Q-learning agent for Pacman, but the size of the state space for a large grid is too massive to hold in memory. To solve this, we will switch to feature-based representation of Pacman's state.

(a) We will have two features, F_g and F_p, defined as follows:

$$F_g(s,a) = A(s) + B(s,a) + C(s,a)$$
$$F_p(s,a) = D(s) + 2E(s,a)$$

where

$$\begin{aligned} A(s) &= \text{number of ghosts within 1 step of state } s \\ B(s,a) &= \text{number of ghosts Pacman touches after taking action } a \text{ from state } s \\ C(s,a) &= \text{number of ghosts within 1 step of the state Pacman ends up in after taking action } a \\ D(s) &= \text{number of food pellets within 1 step of state } s \\ E(s,a) &= \text{number of food pellets eaten after taking action } a \text{ from state } s \end{aligned}$$

For this pacman board, the ghosts will always be stationary, and the action space is $\{left, right, up, down, stay\}$.

calculate the features for the actions $\in \{left, right, up, stay\}$

$$\begin{aligned} F_p(s,up) &= 1 + 2(1) = 3 \\ F_p(s,left) &= 1 + 2(0) = 1 \\ F_p(s,right) &= 1 + 2(0) = 1 \\ F_p(s,stay) &= 1 + 2(0) = 1 \\ F_g(s,up) &= 2 + 0 + 0 = 2 \\ F_g(s,left) &= 2 + 1 + 1 = 4 \\ F_g(s,right) &= 2 + 1 + 1 = 4 \\ F_g(s,stay) &= 2 + 0 + 2 = 4 \end{aligned}$$

(b) After a few episodes of Q-learning, the weights are $w_g = -10$ and $w_p = 100$. Calculate the Q value for each action $\in \{left, right, up, stay\}$ from the current state shown in the figure.

$$\begin{aligned} Q(s,up) &= w_p F_p(s,up) + w_g F_g(s,up) = 100(3) + (-10)(2) = 280 \\ Q(s,left) &= w_p F_p(s,left) + w_g F_g(s,left) = 100(1) + (-10)(4) = 60 \\ Q(s,right) &= w_p F_p(s,right) + w_g F_g(s,right) = 100(1) + (-10)(4) = 60 \\ Q(s,stay) &= w_p F_p(s,stay) + w_g F_g(s,stay) = 100(1) + (-10)(4) = 60 \end{aligned}$$

(c) We observe a transition that starts from the state above, s, takes action *up*, ends in state s' (the state with the food pellet above) and receives a reward $R(s, a, s') = 250$. The available actions from state s' are *down* and *stay*. Assuming a discount of $\gamma = 0.5$, calculate the new estimate of the Q value for s based on this episode.

$$\begin{aligned} Q_{new}(s, a) &= R(s, a, s') + \gamma * \max_{a'} Q(s', a') \\ &= 250 + 0.5 * \max\{Q(s', down), Q(s', stay)\} \\ &= 250 + 0.5 * 0 \\ &= 250 \end{aligned}$$

where

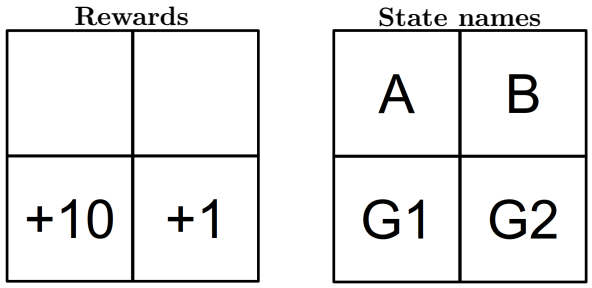
$$\begin{aligned} Q(s', down) &= w_p F_p(s, down) + w_g F_g(s, down) = 100(0) + (-10)(2) = -20 \\ Q(s', stay) &= w_p F_p(s, stay) + w_g F_g(s, stay) = 100(0) + (-10)(0) = 0 \end{aligned}$$

(d) With this new estimate and a learning rate (α) of 0.5, update the weights for each feature.

$$\begin{aligned} w_p &= w_p + \alpha * (Q_{new}(s, a) - Q(s, a)) * F_p(s, a) = 100 + 0.5 * (250 - 280) * 3 = 55 \\ w_g &= w_g + \alpha * (Q_{new}(s, a) - Q(s, a)) * F_g(s, a) = -10 + 0.5 * (250 - 280) * 2 = -40 \end{aligned}$$

Q2. Q-learning

Consider the following gridworld (rewards shown on left, state names shown on right).



From state A, the possible actions are right($\rightarrow$) and down($\downarrow$). From state B, the possible actions are left($\leftarrow$) and down($\downarrow$). For a numbered state (G1, G2), the only action is to exit. Upon exiting from a numbered square we collect the reward specified by the number on the square and enter the end-of-game absorbing state X. We also know that the discount factor $\gamma = 1$, and in this MDP all actions are **deterministic** and always succeed.

Consider the following episodes:

Episode 1 ($E1$)

s	a	s'	r
A	$\downarrow$	$G1$	0
$G1$	exit	X	10

Episode 2 ($E2$)

s	a	s'	r
B	$\downarrow$	$G2$	0
$G2$	exit	X	1

Episode 3 ($E3$)

s	a	s'	r
A	$\rightarrow$	B	0
B	$\downarrow$	$G2$	0
$G2$	exit	X	1

Episode 4 ($E4$)

s	a	s'	r
B	$\leftarrow$	A	0
A	$\downarrow$	$G1$	0
$G1$	exit	X	10

(a) Consider using temporal-difference learning to learn $V(s)$. When running TD-learning, all values are initialized to zero.
For which sequences of episodes, if repeated infinitely often, does $V(s)$ converge to $V^*(s)$ for all states s?

(Assume appropriate learning rates such that all values converge.)
Write the correct sequence under "Other" if no correct sequences of episodes are listed.

☐ $E1, E2, E3, E4$ ☐ $E1, E2, E1, E2$ ☐ $E1, E2, E3, E1$ ■ $E4, E4, E4, E4$
☐ $E4, E3, E2, E1$ ☐ $E3, E4, E3, E4$ ☐ $E1, E2, E4, E1$

■ Other See explanation below

TD learning learns the value of the executed policy, which is $V^\pi(s)$. Therefore for $V^\pi(s)$ to converge to $V^*(s)$, it is necessary that the executing policy $\pi(s) = \pi^*(s)$.

Because there is no discounting since $\gamma = 1$, the optimal deterministic policy is $\pi^*(A) = \downarrow$ and $\pi^*(B) = \leftarrow$ ($\pi^*(G1)$ and $\pi^*(G2)$ are trivially exit because that is the only available action). Therefore episodes $E1$ and $E4$ act according to $\pi^*(s)$ while episodes $E2$ and $E3$ are sampled from a suboptimal policy.

From the above, TD learning using episode $E4$ (and optionally $E1$) will converge to $V^\pi(s) = V^*(s)$ for states A, B, $G1$. However, then we never visit $G2$, so $V(G2)$ will never converge. If we add either episode $E2$ or $E3$ to ensure that $V(G2)$ converges, then we are executing a suboptimal policy, which will then cause $V(B)$ to not converge. Therefore none of the listed sequences will learn a value function $V^\pi(s)$ that converges to $V^*(s)$ for all states s. An example of a correct sequence would be $E2, E4, E4, E4, ...$; sampling $E2$ first with the learning rate $\alpha = 1$ ensures $V^\pi(G2) = V^*(G2)$, and then executing $E4$ infinitely after

ensures the values for states A, B, and $G1$ converge to the optimal values.

We also accepted the answer such that the value function $V(s)$ converges to $V^*(s)$ for states A and B (ignoring $G1$ and $G2$). TD learning using only episode $E4$ (and optionally $E1$) will converge to $V^\pi(s) = V^*(s)$ for states A and B, therefore the only correct listed option is $E4, E4, E4, E4$.

(b) Consider using Q-learning to learn $Q(s, a)$. When running Q-learning, all values are initialized to zero. For which sequences of episodes, if repeated infinitely often, does $Q(s, a)$ converge to $Q^*(s, a)$ for all state-action pairs (s, a)

(Assume appropriate learning rates such that all Q-values converge.)
Write the correct sequence under "Other" if no correct sequences of episodes are listed.

- [x] $E1, E2, E3, E4$
- [ ] $E1, E2, E1, E2$
- [ ] $E1, E2, E3, E1$
- [ ] $E4, E4, E4, E4$
- [x] $E4, E3, E2, E1$
- [x] $E3, E4, E3, E4$
- [ ] $E1, E2, E4, E1$
- [ ] Other _____

For $Q(s, a)$ to converge, we must visit all state action pairs for non-zero $Q^*(s, a)$ infinitely often. Therefore we must take the exit action in states $G1$ and $G2$, must take the down and right action in state A, and must take the left and down action in state B. Therefore the answers must include $E3$ and $E4$.