ECE433/COS435 Spring 2024 Introduction to RL

Lecture 9: Q Learning

1. Q Function and Bellman Equation

The environment is represented as a Markov decision process (MDP) $\mathcal{M} = \{S, A, \gamma, r, P\}$.

Optimal Q Function

- An optimal state-action value function (aka Q function) returns the maximum achievable value to each (s, a) pair

$$Q^*(s,a) = \max_\pi Q^\pi(s,a) = Q^{\pi^*}(s,a)$$

- Once we have Q^*, the agent can act optimally

$$\pi^*(s) = \operatorname*{argmax}_a Q^*(s,a)$$

- Optimal value function and optimal Q function are related by

$$V^*(s) = \max_a Q^*(s,a), \qquad Q^*(s,a) = r(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s')$$

Bellman Equation for Q

The Bellman Equation can be written equivalently into a recursive formula for the optimal action-value function.
The Optimal Bellman Equation for Q^* is given by

$$Q^*(s,a) = \mathbb{E}_{P(s'|s,a)}\left[r + \gamma \max_{a'} Q^*\left(s', a'\right) \mid s, a \right]$$

The optimal policy π^* is the one that maximizes the expected discounted return, and the optimal action-value function Q^* is the action-value function for π^*.
The optimal Bellman equation for Q^π is

$$Q^\pi(s,a) = r(s,a) + \gamma \mathbb{E}_{P(s'|s,a)\pi(a'|s')}\left[Q^\pi\left(s', a'\right) \right]$$

- There are various Bellman equations, and most RL algorithms are based on repeatedly applying one of them.
- This system of equations characterizes the optimal action-value function. So maybe we can approximate Q^* by trying to solve the optimal Bellman equation.
- Bellman equation is **necessary and sufficient** for optimal policy and optimal value function.

Value Iteration and Q Iteration

Recall the value iteration takes for the form: $V_0(s) = 0$ for all s, and

$$V_{k+1}(s) = \max_a \sum_{s'} P\left(s' \mid s, a\right)\left[r(s,a) + \gamma V_k\left(s'\right) \right], \qquad \forall s$$

Value iteration can be viewed as using dynamic program to solve MDP with increasing horizon (length of episode):

- Each V_k is the value function corresponding to initial states of a k-horizon MDP where each state includes a time step that decrement from k to 0: The initiate state takes of the form (s, k) and can transition to a next state $(s', k-1)$, then $(s'', k-2)$ and so on.
- Value iteration is equivalent to using dynamic programming to find values for $(k+1)$-level states using values of k-level states. By virtual of dynamic programing, V_k are always optimal values in the k-horizon MDP, and the actions attaining argmax are also optimal for the k-horizon MDP.
- Letting $k \to \infty$, the k-horizon MDP approximates the ∞-horizon MDP. At the same time, both V_k, V_{k+1} in the Bellman equation converges to V^*. Thus we have obtained the optimal Bellman equation in the limit.
- Since all V_k and actions chosen are optimal for k-horizon MDP, in the limit, V^* is optimal for the infinite-horizon MDP.

- This is an alternative proof of the optimal Bellman equation.

A variant of value iteration is the Q-Iteration expressed as:

$$Q_{k+1}(s,a) = \sum_{s'} P(s' \mid s,a) \left[r(s,a) + \gamma \max_{a'} Q_k(s',a') \right]$$

- $Q_{k+1}(s,a)$: Q-value of state s and action a at iteration $k+1$
- The righthandside of Q iteration takes the form of an expectation over s', which amenable to sampling-based estimation.

Value Iteration and Q Iteration cannot work in RL! They require full knowledges of r, P to implement. In what next, we try to develop RL methods in the spirit of value iteration that do not require these knowledges.

2. Temporal Difference (TD) Learning

Temporal Difference (TD) Learning represents a class of model-free algorithms which learn by bootstrapping from the current estimate of the value function. Unlike Monte Carlo methods, TD learning does not wait until the end of an episode to update the value estimate. Instead, it updates estimates based in part on other learned estimates, without waiting for a final outcome (hence "temporal difference").

2.1. TD update

In TD learning, the agent learns the value function of a given policy based on the TD error, a measure of the difference between the predicted value and the observed reward plus the estimated value of the next state.

- Given a sample transition (s, a, r, s'), the **TD error** is computed as

$$\delta_t = r + \gamma V(s') - V(s),$$

where $r + \gamma V(s')$ is referred to as the **TD target**.

- Update value using

$$V(S_t) \leftarrow V(S_t) + \alpha \delta_t$$

where α is the learning rate.

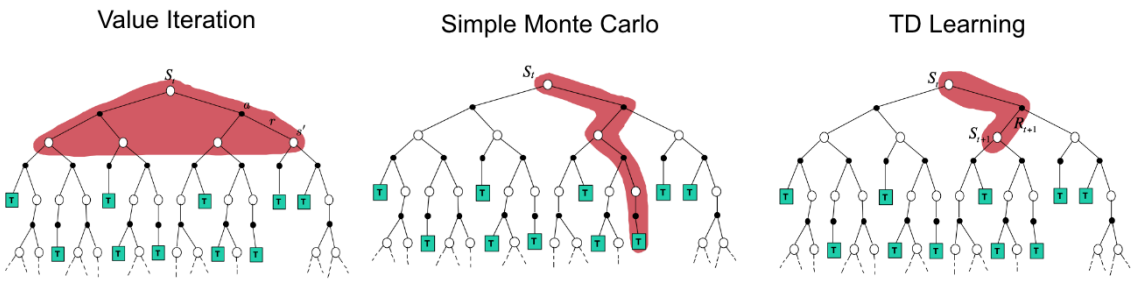
Suppose we have a sequence of transitions $s_1, r_1, s_2, r_2, s_3, \ldots$ following policy π. One can estimate the corresponding value function using the TD updates, which can be rewritten as:

$$V(s_t) \leftarrow (1 - \alpha_t) V(s_t) + \alpha_t (r_t + \gamma V(s_{t+1}))$$

This is an on-policy learning method. The value estimate would converge to value of the policy being run, i.e., V^π.

Comparison between value iteration, simple Monte Carlo and TD learning:

- Value iteration requires evaluating expectation of value of next states. It needs knowledges of transition probabilities
- Simple Monte Carlo means to update value estimate using sum of rewards from a single trajectory. A lot of work for a single noisy sample
- TD learning makes update using a single state transition. It combines Monte Carlo estimation and value iteration.



2.2. TD learning as a stochastic approximation

For general root finding problem $g(x) = 0$, stochastic approximation presents a class of algorithm of the form

$$x \leftarrow x + \alpha \cdot g(x, w),$$

where $g(x, w)$ is a random sample such that $\mathbb{E}[g(x, w)] = g(x)$. Under mild conditions, stochastic approximation is guaranteed to converge to root x^* solving $g(x^*) = 0$.

Bellman equation for a fixed policy is equivalent to a root finding problem of the form

$$(R + \gamma P)V - V = 0,$$

where R is a vector representing rewards and P is the state-to-state transition probability matrix.

Now we verify that TD learning is a stochastic approximation for finding the root of Bellman equation.
Suppose we have a state transition (s, r, s').

- Assume for simplicity $s \sim Unif(S)$.
- Here s' is random next state following probability transition matrix P.
- The observed reward r is one entry of reward vector R, i.e., $r = R(s)$.

The TD update can be written in a vector form

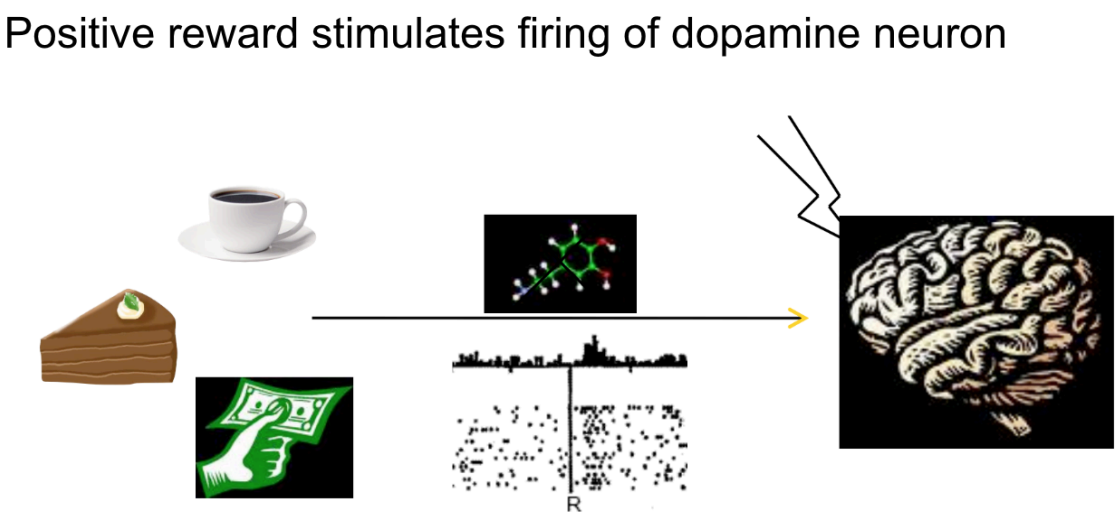
$$V \leftarrow V + \alpha \mathbf{1}_s (r + \gamma V(s') - V(s)),$$

where $\mathbf{1}_s$ is a vector with value 1 at state s and zeros elsewhere. Taking this update in expectation, we have

$$\mathbb{E}[V] \leftarrow V + \frac{\alpha}{S}(R + \gamma PV - V).$$

In other words, the TD update is a noisy estimate of $R + \gamma PV - V$. Thus TD learning is a special case of stochastic approximation of the Bellman equation for the value function under a given policy. Convergence of TD learning follows from stochastic approximation theory for root finding.

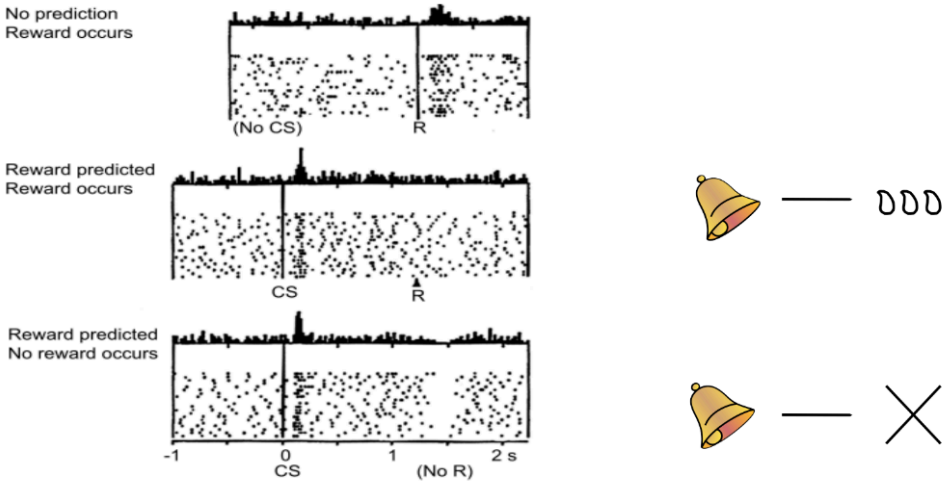
2.3. TD and Neuroscience



Dopamine is a neurotransmitter that plays a significant role in the brain's reward system. Neuroscience studies have found that dopamine neurons increase firing rates above baseline when an unexpected reward is received (positive prediction error), and decrease firing rates below baseline when an expected reward is omitted (negative prediction error):

$$\delta = r + V(s') - V(s)$$

Dopamine encodes error prediction (TD difference) and drives learning (Rescorla & Wagner (1972))



The idea is that the brain updates its predictions about future rewards based on the difference between expected and received rewards, akin to the way Q/TD learning algorithms update their predictions.
This discovery has been influential in the fields of neuroscience and psychology, particularly in understanding how the brain processes rewards and makes decisions.

3. Q-Learning

Q-Learning is an off-policy, model-free reinforcement learning algorithm. It focuses on learning the value of an action in a particular state, aiming to estimate the optimal action-value function $Q^*(s, a)$.

- Let Q be an action-value function which hopefully approximates Q^*.
- The Bellman error is the update to our expected return when we observe the next state s'.

$$\underbrace{r(s_t, a_t) + \gamma \max_a Q(s_{t+1}, a)}_{\text{inside } \mathbb{E} \text{ in RHS of Bellman eqn}} - Q(s_t, a_t)$$

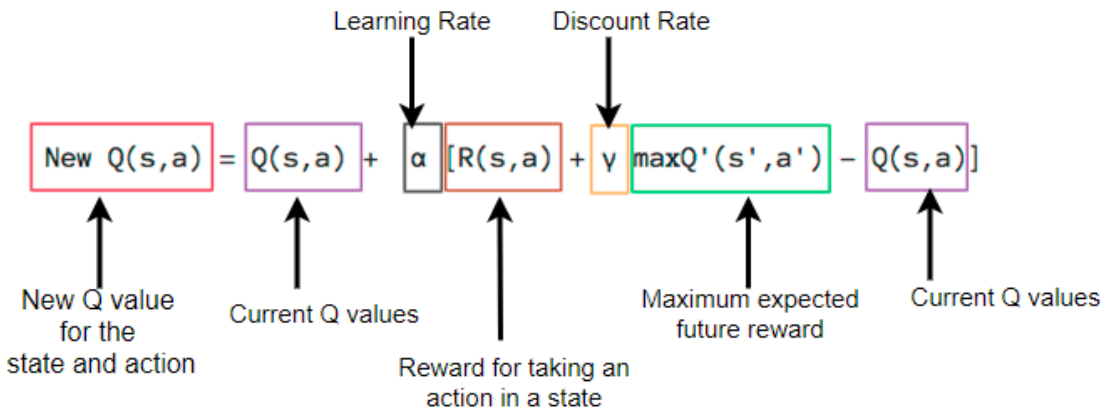
- The Bellman equation says the Bellman error is 0 in expectation

Q-learning is an algorithm that repeatedly adjusts Q to minimize the Bellman error

- Each time we sample consecutive states and actions (s_t, a_t, s_{t+1}) :
- Update a single Q value by

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \underbrace{\left[r(s_t, a_t) + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right]}_{\text{Bellman error}}$$

- In online RL, we also need to specify how to choose actions



Exploration-exploitation tradeoff

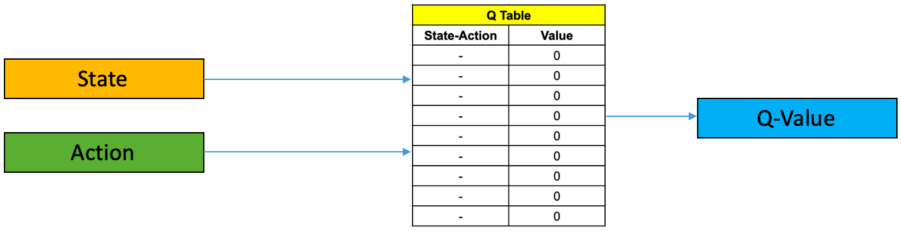
Q-learning only learns about the states and actions it visits. The agent should sometimes pick suboptimal actions in order to visit new states and actions. Similar to multi-arm bandit, Q learning has to balance the exploration-exploitation tradeoff.

A simple go-to solution: ϵ-greedy policy

- With probability $1-\epsilon$, choose the optimal action according to $a = \text{argmax}_{a'} Q(s, a')$
- With probability ϵ, choose a random action uniformly from the action set $a \sim Unif(A)$.

There are more advanced methods, which we will discuss in the second half of the semester.

The form of Q learning we have discussed so far is also known as **tabular Q learning**. Here **tabular** means that the method requires keeping a Q table and recording values of every (s, a) pair. The space complexity of tabular Q learning is thus $O(S \times A)$



For a demo of tabular Q learning on grid world, see https://youtu.be/AMnW-OsOcl8

Observations:

- Each transition updates one Q values.
- It takes a very long time to converge (much slower than value iteration)
- Optimal action get to update much less frequently
- Everything happens on the Q table