Stanford MS&E 236 / CS 225: Lecture 7
Reinforcement learning (Q-learning)

Ellen Vitercik[*]

April 28, 2024

In the last class, we introduced graph neural networks (GNNs), and in the next few classes, we will see how we can use them in the discrete optimization pipeline. There are many different ways to train GNNs for discrete optimization, ranging from supervised learning to reinforcement learning (RL). This class will lay the groundwork for RL, and in the next class, we will explore a natural connection between Q-learning and greedy algorithms for NP-hard problems.

1 Markov decision processes

A *Markov decision process (MDP)* models an agent interacting with their environment. They take *actions*, which move them between *states* in their environment. Their transition from state to state is stochastic, with the transition probabilities depending on the agents' actions. When they enter a state, they obtain a reward. Their goal is to learn which actions to take in each state in order to maximize their cumulative reward.

More formally, an MDP is defined by the following elements:

- A set S of states, which we assume for now to be discrete.
- A set A of actions.
- A transition probability distribution, where $P(s_{t+1} \mid s_t, a_t)$ is the probability the agent enters state $s_{t+1} \in S$ from state s_t after taking action $a_t \in A$. Notably, this probability depends only on the current state and action and not the entire history of states and actions (which is referred to as the *Markov assumption*).
- A reward function $R : S \to \mathbb{R}$.

The agent's goal is to compute a learn a policy $\pi : S \to A$ that maximizes its total (discounted) reward. In particular, if the agent starts in state s_0 and then visits states $s_1, s_2, \ldots$, then their discounted reward, with discount factor $\gamma \in (0, 1)$, is

$$\sum_{t=0}^{\infty} \gamma^t R(s_t).$$

[*]These notes are course material and have not undergone formal peer review. Please feel free to send me any typos or comments.
These notes are largely based on Zico Kolter's slides from CMU's 15-281 course.

The discount factor helps ensure the cumulative reward is bounded (if the rewards are bounded) and intuitively means that the agent prefers reward sooner rather than later (with the magnitude of γ corresponding to how myopic they are).

Of course, we must take into account that the states the agent visits are stochastic. This motivates the definition of a policy's *value function* $V^\pi(s)$, which is their expected discounted reward when they start in state s and follow the policy π. More formally,

$$V^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \,\middle|\, s_0 = s, a_t = \pi(s_t), (s_{t+1} \mid s_t, a_t) \sim P\right].$$

We can also define $V^\pi(s)$ recursively:

$$V^\pi(s) = R(s) + \gamma \sum_{s' \in S} P(s' \mid s, \pi(s)) V^\pi(s').$$

This famous equation is called the *Bellman equation.*

The optimal policy π^* achieves the highest value for every state: $V^{\pi^*}(s) = \max_\pi V^\pi(s)$. We use the simplified notation $V^*(s) := V^{\pi^*}(s)$. Its Bellman equation can be written as

$$V^*(s) = R(s) + \gamma \max_{a \in A} \sum_{s' \in S} P(s' \mid s, a) V^*(s'). \tag{1}$$

Correspondingly, the optimal policy π^* chooses the action in each state with the highest expected value:

$$\pi^*(s) = \text{argmax}_{a \in A} \sum_{s' \in S} P(s' \mid s, a) V^*(s').$$

Said another way, π^* acts greedily according to its value function.

There are many different ways to compute the optimal policy π^* and/or its value function V^*. For example, *value iteration* computes an approximation $\hat{V}$ of V^* iteratively as follows:

1. First, initialize the approximation $\hat{V}(s) \leftarrow 0$ for all states s.

2. Repeat the following update for all states $s \in S$ until $\hat{V}$ converges:

$$\hat{V}(s) \leftarrow R(s) + \gamma \max_{a \in A} \sum_{s' \in S} P(s' \mid s, a) \hat{V}(s') \tag{2}$$

One can prove that $\hat{V}$ will converge to V^*.

2 Reinforcement learning

In reinforcement learning, the twist is that we do not know the transition probability distribution P and/or the reward function R, or the state space is so large that they are impossible to enumerate. Therefore, it is not possible to compute updates like those in Equation (2). To overcome these challenges, we will begin by discussing *temporal different methods (TD methods)*, which will lead naturally to *Q-learning.*

Algorithm 1 Temporal difference learning algorithm

```
Input: Policy $\pi : S \to A$
1: Initialize $\hat{V}^\pi(s) \leftarrow 0$ for all states $s$
2: for $t = 1, 2, \ldots$ do
3:     Observe state $s_t$ and reward $r_t$
4:     Take action $a_t = \pi(s_t)$ and observe next state $s_{t+1}$
5:     Update $\hat{V}^\pi(s_t)$ as in Equation (5)
Output: $\hat{V}^\pi$
```

2.1 Temporal difference methods

Given a policy π, TD methods work by computing an estimate $\hat{V}^\pi$ of V^π, estimated over time as the agent follows the policy π and transitions from state to state. In particular, we certainly cannot compute the update in Equation (2) for all states $s \in S$, but can we update $\hat{V}^\pi(s_t)$ for the current state s_t that we are in?

The first thing we might try would be to set

$$\hat{V}^\pi(s_t) \overset{?}{\leftarrow} r_t + \gamma \sum_{s \in S} P(s' \mid s_t, a_t) \hat{V}^\pi(s), \tag{3}$$

but clearly, we cannot compute this sum since we do not know $P(s' \mid s_t, a_t)$. However, we know that the state s_{t+1} is a sample from the distribution $P(s' \mid s_t, a_t)$. Thus, a second try might be to set

$$\hat{V}^\pi(s_t) \overset{?}{\leftarrow} r_t + \gamma \hat{V}^\pi(s_{t+1}). \tag{4}$$

However, this update is too harsh: it assumes that s_{t+1} is the only possible next state (i.e., it corresponds to Equation (3) with $P(s_{t+1} \mid s_t, a_t) = 1$).

These two attempts motivate the TD update rule:

$$\hat{V}^\pi(s_t) \leftarrow (1-\alpha)\hat{V}^\pi(s_t) + \alpha \left(r_t + \gamma \hat{V}^\pi(s_{t+1}) \right).$$

The parameter α allows us to update with our estimate using r_t and s_{t+1} as in Equation (4), but without completely erasing the previous estimate. Another way to define this update is to set

$$\text{difference}_t = \underbrace{r_t + \gamma \hat{V}^\pi(s_{t+1})}_{\text{new “guess” of } V^\pi(s_t)} - \underbrace{\hat{V}^\pi(s_t)}_{\text{old “guess”}}$$

and define

$$\hat{V}^\pi(s_t) \leftarrow \hat{V}^\pi(s_t) + \alpha \cdot \text{difference}_t. \tag{5}$$

As we can see from this formulation, the TD update shifts the model in the direction of the new estimate of $V^\pi(s_t)$. The magnitude of the shift is proportional to the magnitude of the difference between the new and old estimates. The basic TD learning algorithm is summarized in Algorithm 1. One can prove that $\hat{V}^\pi(s)$ will converge to $V^\pi(s)$ for all states s visited “often enough.”

Algorithm 2 Q-learning algorithm

```
1: Initialize $\hat{Q}^*(s,a) \leftarrow 0$ for all states $s \in S$, $a \in A$
2: Choose initial action $a_1$
3: for $t = 1, 2, \ldots$ do
4:     Take action $a_t = \pi(s_t)$ and observe reward $r_t$ and next state $s_{t+1}$
5:     Choose action $a_{t+1} = \text{argmax}_a \hat{Q}^*(s_{t+1}, a)$
6:     Update $\hat{Q}^*(s_t, a_t)$ as follows:
```

$$\text{difference}_t \leftarrow \underbrace{r_t + \gamma \hat{Q}^*(s_{t+1}, a_{t+1})}_{\text{new "guess" of } Q^*(s_t,a_t)} - \underbrace{\hat{Q}^*(s_t, a_t)}_{\text{old "guess"}}$$

$$\hat{Q}^*(s_t, a_t) \leftarrow \hat{Q}^*(s_t, a_t) + \alpha \cdot \text{difference}_t$$

2.2 Q-learning

Although Algorithm 1 provides a good estimate of V^π, the question remains: what do we do with this estimate when our ultimate goal is to learn the optimal policy? One option might be to execute execute the greedy policy π' using $\hat{V}^\pi(s)$:

$$\pi'(s) \leftarrow \text{argmax}_a \sum_{s' \in S} P(s' \mid s, a) \hat{V}^\pi(s'),$$

but ... we still don't know $P(s' \mid s, a)$! This motivates the notion of a *Q-function.*

Q-functions are similar to value functions, but they are defined over state-action pairs. In particular, given a policy π,

$$Q^\pi(s,a) = R(s) + \gamma \sum_{s' \in S} P(s' \mid s, a) Q^\pi(s', \pi(s'))$$

is the value of starting in state s, taking the action a, and then acting thereon according to the policy π. The optimal policy π^* acts greedily according to its Q-function Q^*:

$$\pi^*(s) = \text{argmax}_{a \in A} Q^*(s,a).$$

Naturally, the recursive form of Q^* is nearly identical to that of the value function V^* (Equation (1)):

$$Q^*(s,a) = R(s) + \gamma \sum_{s' \in S} P(s' \mid s, a) \underbrace{\max_{a'} Q^*(s', a')}_{V^*(s')}.$$

The basic Q-learning algorithm (Algorithm 2) maintains an estimate $\hat{Q}^*$ of Q^*, which it updates as the agent moves from state to state, and selects new actions greedily according to $\hat{Q}^*$.

2.3 Approximate Q-learning

For very large (or continuous) action spaces, storing $\hat{Q}^*(s,a)$ can be infeasible. Instead, one can use a function $f_{\boldsymbol{\theta}}(s,a)$ parameterized by $\boldsymbol{\theta}$, such as a neural network, to approximate

$\hat{Q}^*(s,a)$. We can run Algorithm 2 using $f_{\boldsymbol{\theta}}(s,a)$ instead of $\hat{Q}^*(s,a)$, but Step 6 must change. In particular, we will define

$$\text{difference}_{\boldsymbol{\theta},t} \leftarrow \underbrace{r_t + \gamma f_{\boldsymbol{\theta}}(s_{t+1}, a_{t+1})}_{\text{new ``guess'' of } Q^*(s_t, a_t)} - \underbrace{f_{\boldsymbol{\theta}}(s_t, a_t)}_{\text{old ``guess''}}$$

and update the function $f_{\boldsymbol{\theta}}$ via its parameters, setting

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \cdot \frac{df_{\boldsymbol{\theta}}}{d\boldsymbol{\theta}}(s_t, a_t) \cdot \text{difference}_{\boldsymbol{\theta},t}.$$

This equation mirrors the intuition we gave for Equation (5), where we are shifting the parameters in the direction of the gradient, with the magnitude of the shift proportional to the magnitude of the difference between the new and old estimates of $Q^*(s_t, a_t)$.