Reinforcement Learning, Part 2: Unknown Environment

Data 102, Lecture 22

Spring 2023

Weekly Outline

- **Announcements:**
  - ✦ Midterm 2 on 04/13
  - ✦ Office hours on Thursday
- **So far: Reinforcement learning intro**
  - ✦ Dynamic programming
  - ✦ Markov Decision Processes: transitions and rewards are known
  - ✦ Value iteration (Bellman equation)
- **Today: More on reinforcement learning**
  - ✦ Unknown environment (transitions and/or rewards)
  - ✦ Q-Learning, RL in practice, exploration
  - ✦ Next time: game theory

Agenda

- Q-Learning
- Reinforcement Learning in practice

Online Decision Making in Unknown Environment

- So far we have learned how to make optimal decisions in a fully known environment:
  - ✦ Both transition $T(s, a, s')$ and reward $R(s, a, s')$ functions were known.
  - ✦ **Question:** Is value iteration an offline or online algorithm? offline

    $T(\cdot,\cdot,\cdot)$ $R(\cdot,\cdot)$
- In reality, the environment is unknown: either or both transition and reward functions may be unknown.
- **Examples:** self-driving car, learning to play a game (e.g. minesweeper). Other examples?
- We can no longer user value iteration.
- Instead, we need to **learn about the environment online** as data about rewards and transitions arrive.

Q-Learning

- If we don't known $T(s, a, s')$ and $R(s, a, s')$ functions, then we need to try (explore) actions and observe **trajectories**.
- **Trajectory**: one sequence of state/action/reward from "running" the MDP

policy π policy π

$$s_0,\ a_0,\ r_0,\ s_1,\ a_1,\ r_1,\ s_2,\ a_2,\ r_2,\ \ldots$$

$T(s_0, a_0, s_1)$ $T(s_1, a_1, s_2)$

- Learn Q^* from trajectories: "Q-Learning"
- To construct a trajectory, we follow an exploration policy
- A good exploration policy should visit all possible states and rewards
- **Question:** In a trajectory, what do we control? What does the nature control?

we control $\{a_i\}_{i=0}^{\infty}$, nature controls $\{s_i\}_{i=0}^{\infty}$ & $\{r_i\}_{i=0}^{\infty}$

Q-Learning

- **Trajectory**: a sequence of triplets $\{(s_i, a_i, r_i)\}_{i=0}^{\infty}$

$$s_0, a_0, r_0,\ s_1, a_1, r_1,\ s_2, a_2, r_2, \ldots$$

- A trajectory effectively gives us a sample from transition and reward functions
  - ✦ Our samples look like $T(s_i, a_i, s_{i+1})$ and $R(s_i, a_i, s_{i+1})$
- How do we use these samples to learn the Q^* function?
  - ✦ Intuitively, we should use the observed rewards r_i to get information about Q^*
  - ✦ If we observe that reward r is usually high for a pair of state/action (s, a), then $Q^*(s, a)$ must be high.
- [Example trajectories]

SARSA Algorithm

- Generate trajectories according to our exploration policy
- Initialize $Q^*(s, a)$ as an array (we'll talk about how to initialize later)
- For each (s_i, a_i, r_i, s_{i+1}) in each trajectory update our current Q^* function:

$$Q^*(s_i, a_i) = (1 - \alpha)\underbrace{Q^*(s_i, a_i)}_{\text{old value}} + \alpha \overbrace{\underbrace{\left[r_i + \gamma \max_{a' \in A} Q^*(s_{i+1}, a')\right]}_{\text{"noisy" target value (a new sample from } Q^*)}}^{\text{new obs}}$$

S A R S A

- **Intuition:** "mixing" new observations with what we think so far about $Q^*(s, a)$
- α is the step size parameter:
  - ✦ Large values: big updates but noisy
  - ✦ Small values: smaller updates but more robust

$\alpha \to 0$

SARSA Algorithm

- Recall Q-Iteration:

$$Q^*(s,a) = \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma \max_{a' \in A} Q^*(s',a')\right]$$

- **Question:** How is SARSA update related to Q-Iteration?

$$Q^*(s_i,a_i) = (1-\alpha)Q^*(s_i,a_i) + \alpha\left[r_i + \gamma \max_{a' \in A} Q^*(s_{i+1},a')\right]$$

SARSA Algorithm

- Recall Q-Iteration:

$$Q^*(s,a) = \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma \max_{a' \in A} Q^*(s',a')\right]$$

- **Question:** How is SARSA update related to Q-Iteration?

$$Q^*(s_i,a_i) = (1-\alpha)Q^*(s_i,a_i) + \alpha\left[r_i + \gamma \max_{a' \in A} Q^*(s_{i+1},a')\right]$$

- The right-hand term in SARSA update is equal to Q-iteration update on average:

$Q^*(s,a)$

$$E_{s_{i+1}}[r_i + \gamma \max_{a' \in A} Q^*(s_{i+1},a')] = \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma \max_{a' \in A} Q^*(s',a')\right]$$

- Hence, the SARSA update incorporate a "noisy" target into our current estimate of Q^* function.

SARSA Algorithm

- **Question:** Why do we need to mix new target with our old estimate of $Q*$ function? Why not use the following simple update?

$$Q*(s_i, a_i) = r_i + \gamma \max_{a' \in A} Q*(s_{i+1}, a')$$

SARSA Algorithm

- **Question:** Why do we need to mix new target with our old estimate of Q^* function? Why not use the following simple update?

$$Q^*(s_i, a_i) = r_i + \gamma \max_{a' \in A} Q^*(s_{i+1}, a')$$

- This update rule would throw away whatever we have learned so far in each step!

- **Target in each step is noisy,** our estimate of Q^* will always remain noisy.

$$Q^*(s_i, a_i) = (1 - \alpha) Q^*(s_i, a_i) + \alpha \left[r_i + \gamma \max_{a' \in A} Q^*(s_{i+1}, a') \right]$$

- Mixing with rate α allows us to "average" many noisy updates together

- Mixing stabilizes our estimate of Q^* function

Convergence of SARSA

$$\underbrace{Q^*(s_i, a_i)}_{\text{new}} = (1-\alpha)\underbrace{Q^*(s_i, a_i)}_{\text{old}} + \alpha\left[r_i + \gamma \max_{a' \in A} \underbrace{Q^*(s_{i+1}, a')}_{\text{old}}\right]$$

- Theoretically we need the step sizes to become smaller over updates: $\alpha \to 0$
  - ✦ In practice, we use fixed small α or use a decay schedule

- We also need sufficient exploration of all states and actions. This is similar to Multi Armed Bandits where we needed to explore all arms sufficiently.
  - ✦ In theory, we need to visit all states/actions infinitely often
  - ✦ In practice, we need a good exploration policy $\pi(s)$ that tries each action enough times

Q-Learning: Exploration vs Exploitation

- Two related questions:
  - ✦ How do we initialize $Q*$?
  - ✦ How do we choose a policy to make "good" trajectories?

- Baseline idea:
  - ✦ Initialize $Q*$ array to 0
  - ✦ **Policy:** choose action with the highest $Q*$
  - ✦ Generate trajectories using the above policy
  - ✦ Use SARSA to update $Q*$, and repeat

- **Question:** why is this a bad idea?

Q-Learning: Exploration vs Exploitation

- Two related questions:
  - ✦ How do we initialize $Q*$?
  - ✦ How do we choose a policy to make "good" trajectories?

- Baseline idea:
  - ✦ Initialize $Q*$ array to 0
  - ✦ **Policy:** choose action with the highest $Q*$
  - ✦ Generate trajectories using the above policy
  - ✦ Use SARSA to update $Q*$, and repeat

- **Question:** why is this a bad idea?
  - ✦ After first update, $Q* > 0$. Keeps trying the same trajectory repeatedly.
  - ✦ No Exploration!

Exploration Policy

- Good exploration strategies:
  - ✦ Randomly choose and explore actions at the beginning
  - ✦ In each time step, take a random action with probability ϵ
  - ✦ Initialize $Q*$ to some large value, and in each step choose action with the highest $Q*$

- The last option is similar to exploration in UCB algorithm:
  - ✦ Initializing $Q*$ to large values incentives us to try actions which we have not visited enough yet
  - ✦ Large initial values for $Q*$ capture the idea that we have a lot of uncertainty about unexplored actions

Agenda

- Q-Learning
- Reinforcement Learning in practice

Function Approximation

- In grid world and road trip the number of states were small
  - ✦ About 10 in grid world and 20 in road trip

- But in most problems, the state space is very large:
  - ✦ About 10^{150} in chess and 10^{170} in Go

- When the state space is large, we can't use an array for $Q*$ to store its values

- **Idea**: use function approximation
  - ✦ Let $Q*$ be a function and approximate that function
  - ✦ $Q*$ will be a predictive model (e.g., regression, neural network, etc.)
  - ✦ Intuitively, when we update one step, we also update similar states

Function Approximation

- Instead of a general function Q^*, use a parametric function Q^*_θ
  - ✦ **Examples:** Linear in feature space $Q^*_\theta(s,a) = \theta^T \phi(s,a)$ where $\phi(s,a)$ are features of the state/action pair, Output of a Neural Network feature (s,a)
- Recall SARSA update:

$$Q^*(s_i, a_i) = (1-\alpha)Q^*(s_i, a_i) + \alpha\Big[r_i + \gamma \max_{a' \in A} Q^*(s_{i+1}, a')\Big]$$

$$= Q^*(s_i, a_i) + \alpha\Big[r_i + \gamma \max_{a' \in A} Q^*(s_{i+1}, a') - Q^*(s_i, a_i)\Big]$$

- Now, instead of updating Q^* update its parameters θ over the trajectory:

$$\theta = \theta + \alpha\Big[r_i + \gamma \max_{a' \in A} Q^*_\theta(s_{i+1}, a') - Q^*_\theta(s_i, a_i)\Big] \nabla_\theta Q^*_\theta(s_i, a_i)$$

Reward Hacking

- If we are not careful about how we describe the reward function, we could end up with a policy that ignores the problem structure
  - ✦ Agent would cheat the reward structure by finding irrelevant actions that artificially increase the reward
  - ✦ Agent would learn a policy that is not helpful to us

- **Example:** Misspecified reward function leads to to reward hacking and odd behavior in CoastRunners