Algorithms for control

reading: Sutton & Barto, Chap 10

Policy Iteration from data

- We have seen how to learn V^π from data (TD)
- If we can learn Q^π, then we can do control (policy optimization) by running policy iteration
- How to learn Q^π? similar idea

!! $\mathcal{T}^\pi Q^\pi$

- Bellman eq for Q^π: $Q^\pi(s, a) = \boxed{R(s, a) + \gamma \mathbb{E}_{s' \sim P(s,a)}[Q^\pi(s', \pi(s'))]}$
- Given $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$ where all actions are taken according to π, update rule for learning Q^π: "SARSA"
  $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma Q(s_{t+1}, \boxed{a_{t+1}}) - Q(s_t, a_t))$
  - Do you need a_{t+1}? check out: expected Sarsa.
- In TD (for learning V^π), we require that each state is visited sufficiently often

$V^\pi(s) = Q^\pi(s, \pi)$

- Similarly, here we require that each state-action pair is visited sufficiently often
- π must be stochastic! (so we cannot run PI exactly)

$Q^\pi \in \mathbb{R}^{S \times A}$. $\quad Q^\pi(s,a) = (\mathcal{T}^\pi Q^\pi)(s,a)$

$$(\mathcal{T}^\pi Q_{k-1})(s,a) \overset{\Delta}{=} R(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)}[Q_{k-1}(s', \pi)].$$

$$= \mathbb{E}[r + \gamma \cdot f(s', \pi) \mid s, a].$$

$$\mathbb{E}_{\substack{r, s' | s,a \\ a' \sim \pi}}[r + \gamma f(s', a')]$$

$\{(r_i, s_i')\}_{i=1}^{n} \sim (s,a)$.

$$\approx \frac{1}{n} \sum_{i=1}^{n} (r_i + \gamma Q_{k-1}(s_i', \pi)).$$

online.

$$Q_k(s,a) \leftarrow Q_k(s,a) + \alpha(r_i + \gamma Q_{k-1}(s_i', \pi) - Q_k(s,a)).$$

(s_t, a_t, r_t, s_{t+1}).

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma Q(s_{t+1}, \pi) - Q(s_t, a_t)).$$

Expected Sarsa.

off-policy.

$$(s_t, a_t, r_t, s_{t+1}, a_{t+1}).$$

s_t: π, a_{t+1}: π.

$$(s, a, r, s', a')$$

s, a: anything. a': π.

---

a. r_a.

$a \sim \pi$. $r \sim R(\cdot \mid a)$.

$$\frac{1}{n}\sum_{i=1}^{n} r_i = V^{\pi}(\underline{s}).$$

$$(\underline{a_i}, r_i)$$

$$\hat{R}(a) = \frac{\sum_i r_i \mathbb{I}[a_i = a]}{\sum_i \mathbb{I}[a_i = a]}.$$

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

$$s_t \xrightarrow{\pi} a_t \longrightarrow \begin{matrix} r_t \\ s_{t+1} \end{matrix} \Bigg| \, s_t .$$

SARSA with epsilon-greedy policy

Q^π

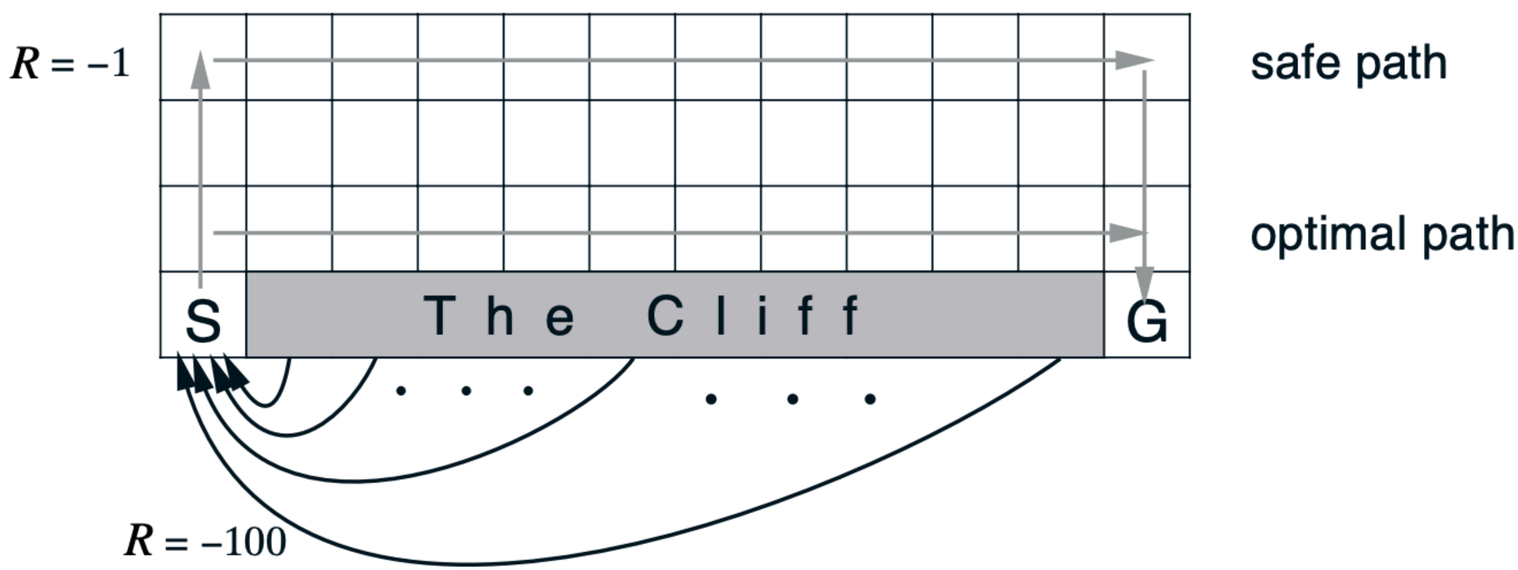
- $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$
- Take epsilon-greedy policy w.r.t the current Q-estimate
    - At each time step t, with probability ε, choose a_t from the action space uniformly at random. otherwise, $a_t = \mathbf{argmax}_a\, Q(s_t, a)$
- Greedy part: "no-wait" version of policy improvement. Take greedy action w.r.t. Q every time step!
    - the policy being evaluated is constantly changing
    - "ε-greedy policy" is not a fixed policy
- ε part: make sure to explore all actions
- Precisely speaking, this is SARSA(0)
    - Can be extended to SARSA(λ) just as TD

$1-\varepsilon$.

$r_t + \gamma r_{t+1} + \gamma^2 Q(s_{t+2}, a_{t+2})$.

Does SARSA converge to optimal policy?

- The epsilon part can prevent convergence!
- The cliff example (pg 132 of Sutton & Barto)
  - Deterministic navigation, high penalty when falling off the cliff
  - Optimal policy: walk near the cliff
  - Unless epsilon is super small, SARSA will avoid the cliff
- Will need to reduce ε over time—but small ε does not sufficiently explore, and Q-value estimates converge slower



SARSA with epsilon-greedy policy

- ε-greedy can be replaced by softmax: chooses action a with probability $\boxed{\frac{e^{Q(s_t,a)/T}}{\sum_{a'} e^{Q(s_t,a')/T}}}$, here T is temperature and needs to decrease over time (playing a role similar to ε in ε-greedy)
- Can use other stochastic policy that assigns most probability to the greedy action and explore all other actions at the same time
- Exercise: derive SARSA with function approximation

temperature.

$$\pi(a|s_t) \propto e^{Q(s_t,a)/T}$$

$$Q_\theta(s,a)$$

$$Q^\pi \approx \phi(s,a)^T\theta.$$

$f_k \leftarrow \mathcal{T} f_{k-1}$

Q-learning

$(\mathcal{T}f)(s,a) = \mathbb{E}\left[\boxed{r + \gamma \max_{a'} f(s',a')}\ \Big|\ s,a\right].$

- We've seen how to derive a control algorithm (SARSA) based on the idea of policy iteration (or Bellman eq. for policy eval)
- How about value iteration (Bellman optimality eq.)?
- $Q^\star(s,a) = R(s,a) + \gamma \mathbb{E}_{s' \sim P(s,a)}[\max_{a'} Q^\star(s',a')]$ $=: (\mathcal{T} Q^*)(s,a)$
- Update rule:

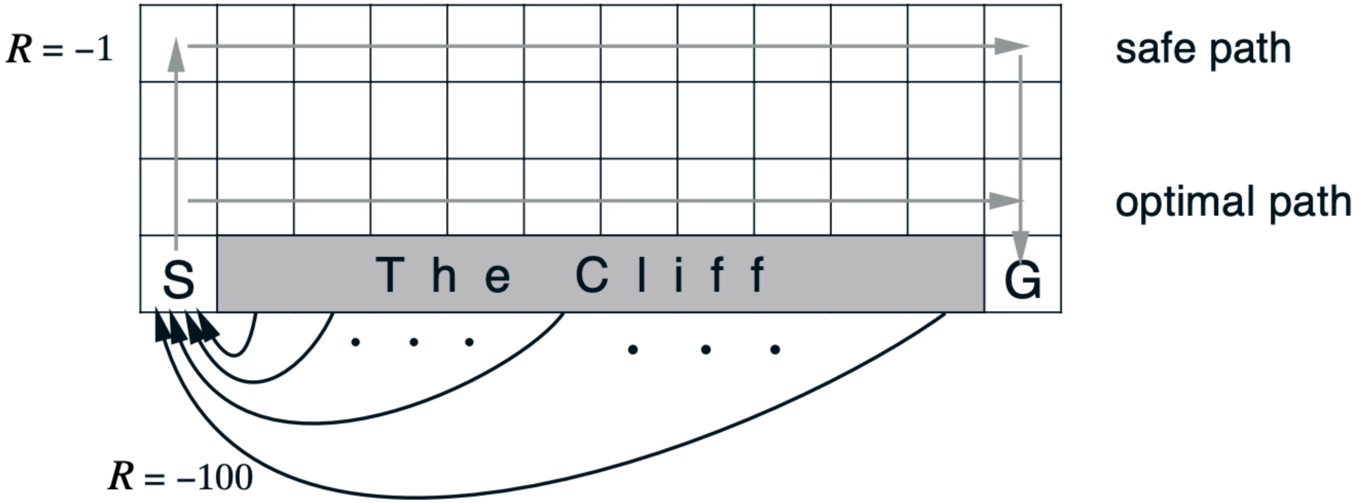
$$Q(s_t, a_t) \leftarrow \underline{Q(s_t, a_t)} + \alpha(\boxed{r_t + \gamma \max_{a'} Q(s_{t+1}, a')} - \underline{Q(s_t, a_t)})$$

- Algorithms for control always have a "max" somewhere
  - the max in Q-learning is explicit in the update rule
  - Exercise: where is the "max" in SARSA?

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

Q-learning

- Q-learning does not specify how a_t should be taken
  - Q-learning is *off-policy*: how we take actions have nothing to do with our current Q-estimate (or its greedy policy)
  - Learning rule is completely disentangled from the exploration rule (how to take actions during data collection). Explore however you want using a “behavior policy”
  - e.g., uniformly random action, or ε-greedy (here you do not need to reduce ε)
- Exercise: think about how Q-learning behaves in the cliff example



Connection between Q-learning and SARSA

- Expected sarsa: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma Q(s_{t+1}, \pi) - Q(s_t, a_t))$
  - recall that when π is stochastic, $Q(s, \pi) := \mathbb{E}_{a \sim \pi(\cdot|s)}[Q(s, a)]$
- Expected sarsa can be run off-policy! $\pi = \pi_\theta$
  - Sarsa needs to be on-policy because we use a_{t+1} from data; this action needs to be consistent with π according to Bellman equation
  - If we replace it with the expected (i.e., "imagined" action that is not actually taken in the environment), it removes any restriction on the behavior policy
  - (Insight due to Rich Sutton): Q-learning is a special case of expected Sarsa! Which policy are we evaluating?

$$Q(s_t, a_t) \dashrightarrow \mathbb{E}\left[r_t + \gamma \max_{a'} Q(s_{t+1}, a') \mid s_t, a_t\right]$$

$$= (\mathcal{T}Q)(s_t, a_t)$$

Exercise: Multi-step Q-learning?

$(\mathcal{T}(\mathcal{T}Q))(s_t, a_t)$

- Does the target $\mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 \max_{a'} Q(s_{t+2}, a') \,\middle|\, s_t, a_t \right]$ work? If not, why?
  - Consider the expected target conditioned on s_t, a_t . Express it using standard Bellman update operators
  - Give away: the expected target is $(\mathcal{T}^{\pi}(\mathcal{T}Q))(s_t, a_t)$, where π is behavior policy

$(\mathcal{T}^{\pi}\mathcal{T})(\mathcal{T}^{\pi}\mathcal{T})(\mathcal{T}^{\pi}\mathcal{T}) \cdots Q_0$

(s, a, r, s') :

$Q(s,a) \leftarrow Q(s,a) + \alpha \left(r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right)$

Q-learning with experience replay

- So far most algorithms we see are "one-pass"
  - i.e., use each data point once and discard them
  - # updates = # data points
- Concern 1: We need many updates for optimization to converge. Can we separate optimization from data collection?
- Concern 2: Need to reuse data if sample size is limited
- Q-learning as an example: suppose we are given a bag of (s, a, r, s') tuples and we cannot collect further data, what to do?
- Sample (with replacement) a tuple randomly from the bag, and apply the Q-learning update rule.
  - # updates >> # data points
- Converges with appropriate learning rate
  - Guess what it converges to?
  - Model-based RL!

Q-learning with function approximation

- As before, we first derive the batch version
- Approximate Q^* using a (parameterized) function class $\mathscr{F}$
- Want to approximate Bellman update operator using data (a bag of (s, a, r, s') tuples)
- Fitted Q-Iteration (FQI): $f_{k+1} \leftarrow \arg\min_{f_\theta \in \mathscr{F}} \sum_{(s,a,r,s')} (f_\theta(s, a) - r - \gamma \max_{a'} f_k(s', a'))^2$
- Q-learning with function approximation
- $\theta \leftarrow \theta - \alpha \cdot (f_\theta(s, a) - r - \gamma \max_{a'} f_\theta(s', a')) \nabla f_\theta(s, a)$
- Exercise: this is Q-learning when using tabular function class
- Similar to TD, we only take gradient on $f_\theta(s, a)$ and ignore $f_\theta(s', a')$, because the latter is treated as a constant (it plays the role of f_k)

$$Q_k \leftarrow \mathcal{T} Q_{k-1}$$

$$(\mathcal{T} Q_{k-1})(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q_{k-1}(s',a') \,\middle|\, s,a \right].$$

$$= \operatorname*{argmin}_{f : S \times A \to \mathbb{R}} \mathbb{E}\left[\left(f(s,a) - r - \gamma \max_{a'} \cdots \right)^2 \right].$$

$$\approx \operatorname*{argmin}_{f_\theta \in \mathcal{F}} \underline{\mathbb{E}\left[\left(f_\theta(s,a) - \boxed{} \right)^2 \right]}.$$

sample (s,a,r,s').

$$\theta \leftarrow \theta - \alpha \left(f_\theta(s,a) - \quad \right) \cdot \nabla f_\theta(s,a).$$

Quick Recap of the TD Part

How to go from a Bellman update operator to a learning rule?

1. Write down the Bellman up op for the thing you want to learn
   - e.g., $Q_{k+1} \leftarrow \mathcal{T}^{\pi} Q_k$ if we want to learn Q^{π}
2. Write down the detailed equation for a single s (or (s,a))
   - $Q_{k+1}(s, a) \leftarrow R(s, a) + \gamma \mathbb{E}_{s' \sim P(s,a)}[Q_k(s', \pi(s'))]$
3. Replace the expectations with their sampled version to form the target (assuming data is (s, a, r, s', a'))
   - target: $r + \gamma Q(s', \pi(s'))$ (expected Sarsa)
   - alternative target: $r + \gamma Q(s', a')$ if on-policy $(a' \sim \pi(s'))$
4. Online tabular ver: Plug into the template
   - $Q(s, a) \leftarrow Q(s, a) + \alpha(\text{target} - Q(s, a))$
5. Batch function approximation ver: run least sq regression on
   - $\{(s, a) \mapsto \text{target}\}$

Quick Recap of the TD Part

Another example: TD(0)

1. Write down the Bellman up op for the thing you want to learn
   - $V_{k+1} \leftarrow \mathcal{T}^{\pi} V_k$
2. Write down the detailed equation for a single s (or (s,a))
   - $V_{k+1}(s) \leftarrow R(s, \pi(s)) + \gamma \mathbb{E}_{s' \sim P(s,\pi(s))}[V_k(s')]$
3. Replace the expectations with their sampled version to form the target (assuming data is (s, a, r, s'))
   - target: $r + \gamma V(s')$
   - Be careful! This is only a sampled version of above if on-policy $(a \sim \pi(s))$
   - Difference between learning V and Q: learning V^{π} has to be on-policy (for now), but learning Q^{π} can be easily off-policy (expected sarsa)