Policy Gradient

Policy Gradient (PG)

- Given a class of parameterized policies π_θ, optimize $J(\pi_\theta) := \mathbb{E}_{s \sim d_0}[V^{\pi_\theta}(s)]$
  - We will often make the dependence of π_θ on θ implicit, i.e., when we write π we mean π_θ in this part of the course
- Simple idea: can run (stochastic) gradient descent if we can obtain (an unbiased estimate of) $\nabla_\theta J(\pi_\theta)$
  - will abbreviate as $J(\pi)$
- Beautiful result: an unbiased estimate can be obtained from a single on-policy trajectory, without using knowledge of P and R of the MDP!
- Has a strong connection to IS
- “Vanilla” PG (e.g., REINFORCE) is considered a Monte-Carlo method—it does not leverage Bellman equation

Why PG?

- RL methods can be categorized according to what we try to approximate: model-based RL, value-based RL, policy search
- Eventually we only care about a good policy!
- value-based RL is indirect (model-based even more)
- If a value function induces a good greedy policy, but the function itself severely violates Bellman equation, you won't be able to find such a policy via value-based methods
- In other words, policy search is agnostic against misspecification of function approximation
  - Apart from difficulties in optimization, there is nothing that prevents policy search from finding the best policy in class
- Value- (and model-) based methods have their advantages—will come back later

Example of policy parametrization

- Linear + softmax:
  - Featurize state-action: $\phi : S \times A \to \mathbb{R}^d$
  - Policy: $\pi(a \mid s) \propto e^{\theta^\top \phi(s,a)}$
- Recall that in SARSA we've also seen the softmax policy
- There we include a temperature parameter, $\pi(a \mid s) \propto e^{\theta^\top \phi(s,a)/T}$
- Why the difference?
  - In TD, we want $\theta^\top \phi(s,a) \approx Q^\pi(s,a)$. We don't have the freedom to rescale it; i.e., if $\theta^\top \phi(s,a) \approx Q^\pi(s,a)$, then $(2\theta)^\top \phi(s,a) \neq Q^\pi(s,a)$.
  - We need an additional knob (T) to control the stochasticity of π
  - In PG, $\theta^\top \phi(s,a)$ does not carry any meaning—it's totally possible that eventually we find a θ but $\theta^\top \phi(s,a) \neq Q^{\pi_\theta}(s,a)$!
  - That's why we can absorb the temperature parameter in θ
  - Reflection of the agnosticity of PG

Derivation of PG

- Use $\tau := (s_1, a_1, r_1, \ldots, s_H, a_H, r_H)$ to denote a trajectory (episodic)
- Use $\tau \sim \pi$ as a shorthand for distribution induced by π
- Let $R(\tau) := \sum_{t=1}^{H} \gamma^{t-1} r_t$
- Ver 1: $\nabla J(\pi) = \mathbb{E}_{\tau\sim\pi}[R(\tau) \sum_{t=1}^{H} \nabla \log \pi(a_t \,|\, s_t)]$
  - Will derive using a "MC"-style proof
- Ver 2: $\nabla J(\pi) = \frac{1}{1-\gamma} \mathbb{E}_{s\sim d^\pi, a\sim\pi(s)}[Q^\pi(s,a) \nabla \log \pi(a \,|\, s)]$
  - d^π is the normalized occupancy (from d_0 as init distribution)
  - Possible implementation: (1) roll out $\tau \sim \pi$, (2) pick a random time step t w.p. $\propto \gamma^{t-1}$, (3) $(\sum_{t'=t}^{H} \gamma^{t'-1} r_t) \nabla \log \pi(a_t \,|\, s_t)$
    - Note that $\mathbb{E}[\sum_{t'=t}^{H} \gamma^{t'-1} r_t \,|\, s_t, a_t] = Q^\pi(s_t, a_t)$
    - Take expectation over step (2) gives an alternative form: $\nabla J(\pi) = \mathbb{E}_{\tau\sim\pi}[\sum_{t=1}^{H} (\sum_{t'=t}^{H} \gamma^{t'-1} r_{t'}) \nabla \log \pi(a_t \,|\, s_t)]$
  - Will derive using a "DP"-style proof; can also be derived using the MC-style proof for ver 1

Pros & Cons of PG, and beyond

- Standard PG is fully on-policy, and it's hard to reuse data
  - after each update step, the policy changes and we need to generate MC trajectories from the new policy
- in practice, it suffers from noisy gradient estimate
- Blend PG with value-based method:
  - $\nabla J(\pi) = \frac{1}{1-\gamma} \mathbb{E}_{s \sim d^\pi, a \sim \pi(s)} [Q^\pi(s, a) \nabla \log \pi(a \mid s)]$
  - Instead of using MC estimate $\sum_{t'=t}^{H} \gamma^{t'-1} r_t$ for $Q^\pi(s_t, a_t)$, use an approximate value-function $\hat{Q}^\pi(s_t, a_t)$, often trained by TD
  - e.g., using expected Sarsa—can leverage previous (off-policy) data to learn $\hat{Q}^\pi(s_t, a_t)$
  - "Actor-critic": the parametrized policy is called the actor, and the value-function estimate is called the critic

Baseline in PG

- $\nabla J(\pi) = \frac{1}{1-\gamma}\mathbb{E}_{s\sim d^\pi, a\sim\pi(s)}[Q^\pi(s,a)\nabla \log \pi(a\,|\,s)]$
- For any $f : S \to \mathbb{R}$, $\nabla J(\pi) = \frac{1}{1-\gamma}\mathbb{E}_{s\sim d^\pi, a\sim\pi(s)}[(Q^\pi(s,a) - f(s))\nabla \log \pi(a\,|\,s)]$
  - for any s, $\mathbb{E}_{a\sim\pi(s)}[f(s)\nabla \log \pi(a\,|\,s)] = f(s)\cdot\mathbb{E}_{a\sim\pi(s)}[\nabla \log \pi(a\,|\,s)] = \mathbf{0}$
  - proof: $\mathbb{E}_{a\sim\pi(s)}[\nabla \log \pi(a\,|\,s)] = \sum_a \pi(a\,|\,s)\nabla \log \pi(a\,|\,s)$
    $= \sum_a \nabla \pi(a\,|\,s) = \nabla \sum_a \pi(a\,|\,s) = \nabla 1 = \mathbf{0}$
- One choice: $f = V^\pi(s)$
  - $\nabla J(\pi) = \frac{1}{1-\gamma}\mathbb{E}_{s\sim d^\pi, a\sim\pi(s)}[A^\pi(s,a)\nabla \log \pi(a\,|\,s)]$
  - recall that A is the advantage function

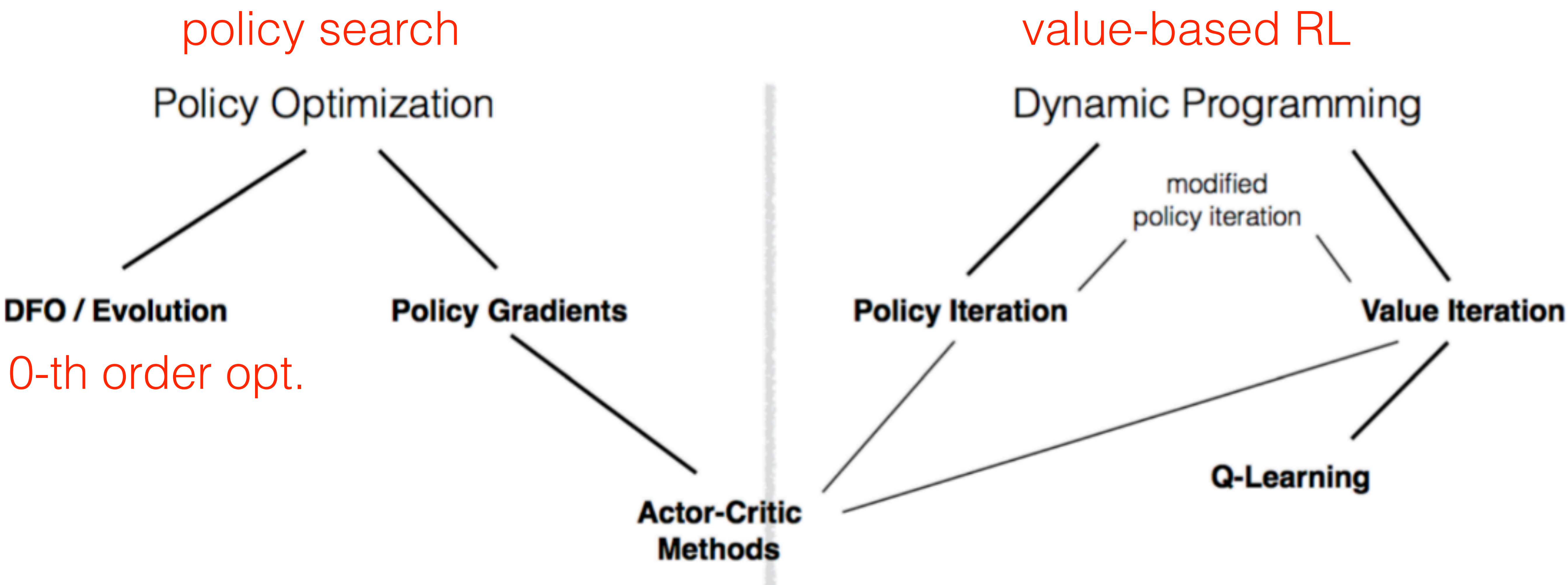
Comparing AC with Policy Iteration

- $\nabla J(\pi) \approx \frac{1}{1-\gamma}\mathbb{E}_{s\sim d^{\pi}, a\sim\pi(s)}[\hat{Q}^{\pi}(s,a)\nabla\log\pi(a\,|\,s)]$
- A different but related procedure: freeze π, update the parameter of another policy π' (whose parameters are θ') by
  $\theta' \leftarrow \theta' + \alpha \cdot \dfrac{1}{1-\gamma}\mathbb{E}_{s\sim d^{\pi}, a\sim\pi(s)}[\hat{Q}^{\pi}(s,a)\nabla\log\pi'(a\,|\,s)]$
  - gradient = $\mathbf{0}$ at $\pi' = \pi_{Q^{\pi}}$ => policy iteration
- This can run into serious issues
  - Tabular PI theory assumes that we get $\hat{Q}^{\pi}$ that is accurate for every single state-action pair
  - Simply unrealistic if problem is complex and we can only roll-out trajectories (instead of sweeping the entire state space)
  - in the middle of learning, part of the state space may be under-explored
  - at best we can hope $\hat{Q}^{\pi}$ to be accurate under distribution of state space we have data for

Comparing AC with Policy Iteration

- $\nabla J(\pi) \approx \frac{1}{1-\gamma} \mathbb{E}_{s \sim d^\pi, a \sim \pi(s)}[\hat{Q}^\pi(s,a) \nabla \log \pi(a \mid s)]$
- A different but related procedure: freeze π, update the parameter of another policy π' (whose parameters are θ') by
  $\theta' \leftarrow \theta' + \alpha \cdot \frac{1}{1-\gamma} \mathbb{E}_{s \sim d^\pi, a \sim \pi(s)}[\hat{Q}^\pi(s,a) \nabla \log \pi'(a \mid s)]$
  - gradient = $\mathbf{0}$ at $\pi' = \pi_{Q^\pi}$ => policy iteration
- This can run into serious issues
  - (cont.) if π' visits new states, $\hat{Q}^\pi$ may be highly inaccurate in those states, and policy improvement no longer holds
- Perhaps better idea: move π' a little more but not too far from π, so that their state occupancies are still similar.
- Theory: CPI [Kakade & Langford'02]
- Modern implementations & variants: TRPO, PPO, etc

RL Algorithms Landscape



Slide Credit: Pieter Abbeel

Practical considerations

- Recall that one way to implement PG/AC is:
  1. roll out $\tau \sim \pi$,
  2. gradient from step *t*: $Q^{\pi}(s_t, a_t) \nabla \log \pi(a_t \mid s_t)$
  3. sum up the gradients from all time steps, with weight $\propto \gamma^{t-1}$,
- What if a trajectory length >> $1/(1-\gamma)$?
  - Most of the data points are wasted!
- Deep RL implementation in Atari games:
  - Trajectory length = ~5 min
  - Effective horizon = secs

$\gamma = 0.99$, frame rate 60Hz $\Rightarrow$ effective horizon = O(1/(1-γ) * 1/60)) = ~ sec

Practical considerations

- Actual implementation:
    1. roll out $\tau \sim \pi$,
    2. gradient from step t: $Q^{\pi}(s_t, a_t) \nabla \log \pi(a_t \mid s_t)$
    3. put equal weights on gradients from all time steps
- Pro: use all data points; Con: biased gradient.
- Is there no discounting then?
    - $Q^{\pi}(s_t, a_t)$ is still learned using γ (e.g., by TD in actor-critic)
- How to understand/make sense of this?