Intro to RL & Policy Gradient

Xuchan (Jenny) Bao

CSC421 Tutorial 10, Mar 26/28, 2019

Outline:

- Brief intro to RL
- Policy Gradient
    - The log-derivative trick
    - Practical fixes: baseline & temporal structure
- OpenAI Gym
- Example: policy gradient on Gym environments
- References

Slides on intro & policy gradient are from / inspired by the Deep RL Bootcamp Lecture 4A: Policy Gradients by Pieter Abbeel https://www.youtube.com/watch?v=S_gwYj1Q-44

Brief Intro to RL

Represent agent with stochastic policy $\pi_\theta(a|s)$

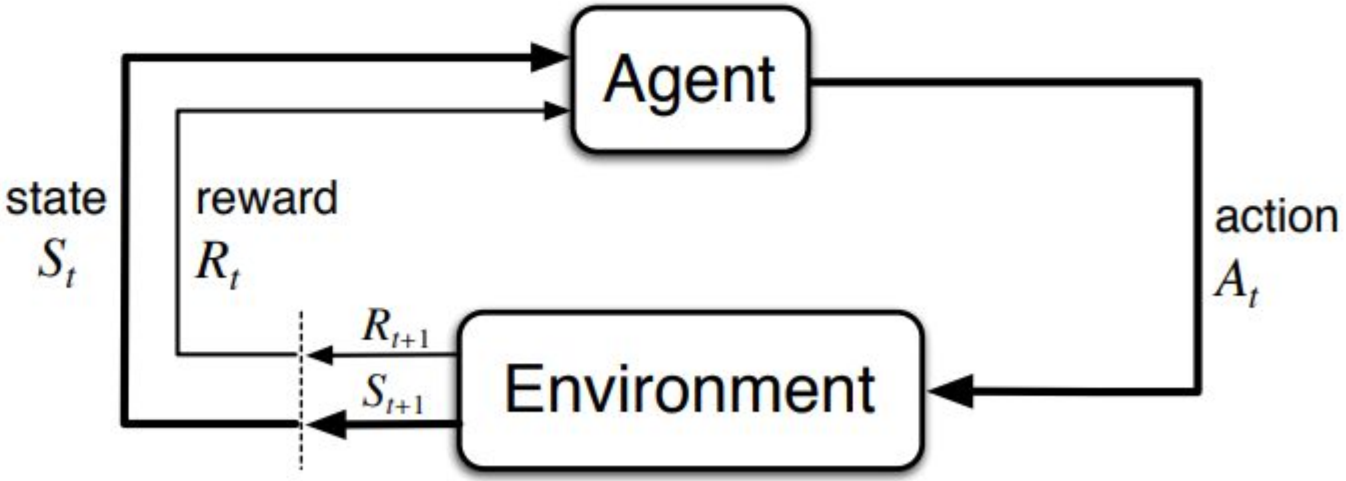
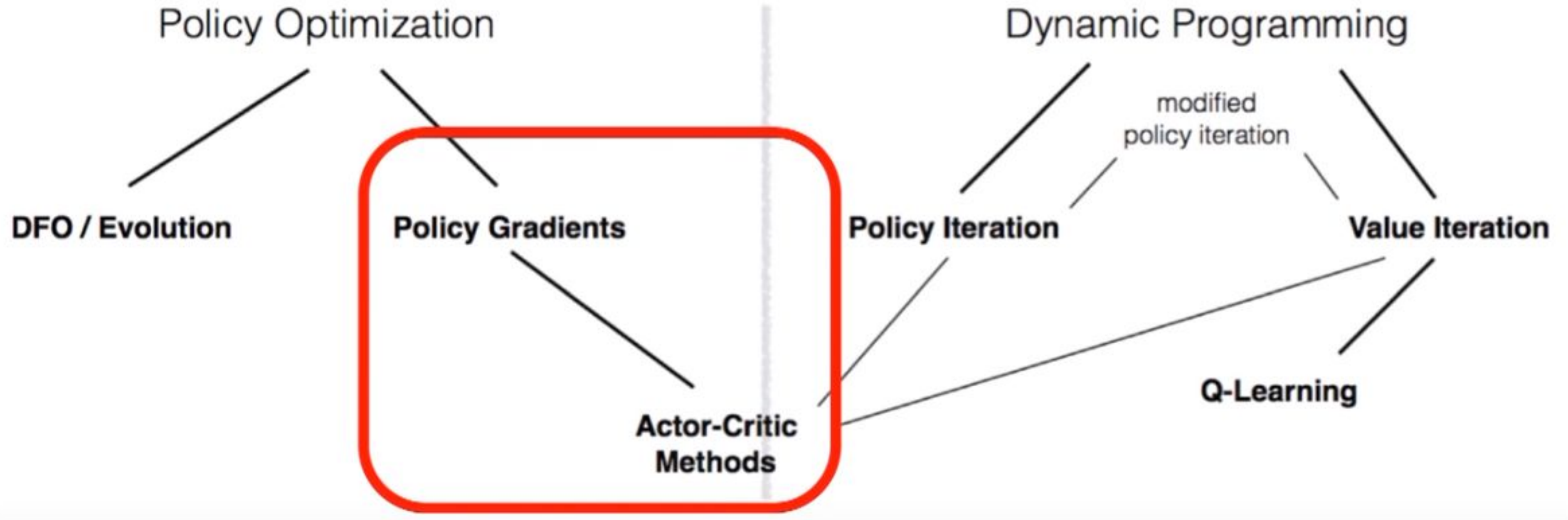


Figure 3.1: The agent–environment interaction in a Markov decision process.

From Sutton & Barto “Reinforcement Learning: An Introduction”, 1998

Policy Optimization in the RL Landscape



From Deep RL Bootcamp Lecture 4A: Policy Gradients, Pieter Abbeel https://www.youtube.com/watch?v=S_gwYj1Q-44

	Policy Optimization	Dynamic Programming
Conceptually:	Optimize what you care about	Indirect, exploit the problem structure, self-consistency
Empirically:	More compatible with rich architectures (including recurrence) More versatile More compatible with auxiliary objectives	More compatible with exploration and off-policy learning More sample-efficient when they work

From Deep RL Bootcamp Lecture 4A: Policy Gradients, Pieter Abbeel https://www.youtube.com/watch?v=S_gwYj1Q-44

Policy Gradient

Suppose we have a trajectory: $\tau = (s_0, a_0, s_1, a_1, \ldots, s_{H-1}, a_{H-1}, s_H)$

And represent the reward for the whole trajectory: $R(\tau) = \sum_{t=0}^{H-1} R(s_t, a_t)$

The expected reward under policy π_θ (utility function):

$$U(\theta) = \mathbb{E}[\sum_{t=0}^{H} R(s_t, a_t); \pi_\theta] = \sum_{\tau} P(\tau; \theta) R(\tau)$$

The goal is to find the optimal parameters to max the utility function.

$$\max_\theta U(\theta) = \max_\theta \sum_{\tau} P(\tau; \theta) R(\tau)$$

Policy Gradient: the log-derivative trick

Take the gradient:

$$
\begin{aligned}
\nabla_\theta U(\theta) &= \nabla_\theta \sum_\tau P(\tau;\theta)R(\tau) \\
&= \sum_\tau \nabla_\theta P(\tau;\theta)R(\tau) \\
&= \sum_\tau \frac{P(\tau;\theta)}{P(\tau;\theta)} \nabla_\theta P(\tau;\theta)R(\tau) \\
&= \sum_\tau P(\tau;\theta) \frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)} R(\tau) \\
&= \sum_\tau P(\tau;\theta) \nabla_\theta \log P(\tau;\theta) R(\tau)
\end{aligned}
$$

Policy Gradient: approximate gradient with samples

Now we can approximate the gradient using Monte Carlo Samples!

$$\nabla_\theta U(\theta) = \sum_\tau P(\tau;\theta)\nabla_\theta \log P(\tau;\theta)R(\tau)$$

$$\approx \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)};\theta)R(\tau^{(i)})$$

Where $\tau^{(i)}$ are sample rollout trajectories under policy π_θ

Policy Gradient: approximate gradient with samples

Take a moment to appreciate this:

$$\nabla_\theta U(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)}; \theta) R(\tau^{(i)})$$

This gradient approximation is valid even when:

- The reward is discontinuous / unknown
- Sample space is a discrete set

Policy Gradient: intuition

$$\nabla_\theta U(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)}; \theta) R(\tau^{(i)})$$

The gradient tries to:

- Increase probability of paths with positive rewards
- Decrease probability of paths with negative rewards

Does NOT try to change the paths themselves.

See any problems here?

Decomposing the paths into states & actions

$$\nabla_\theta U(\theta) \approx \frac{1}{m}\sum_{i=1}^{m} \boxed{\nabla_\theta \log P(\tau^{(i)};\theta)} R(\tau^{(i)})$$

$$\boxed{\nabla_\theta \log P(\tau^{(i)};\theta)} = \nabla_\theta \log[\prod_{t=0}^{H-1} P(s_{t+1}^{(i)}|s_t^{(i)}, a_t^{(i)})\pi_\theta(a_t^{(i)}|s_t^i)]$$

$$= \nabla_\theta [\boxed{\sum_{t=0}^{H-1} \log P(s_{t+1}^{(i)}|s_t^{(i)}, a_t^{(i)})} + \sum_{t=0}^{H-1} \log \pi_\theta(a_t^{(i)}|s_t^i)]$$

Not a function of θ

$$= \nabla_\theta [\sum_{t=0}^{H-1} \log \pi_\theta(a_t^{(i)}|s_t^i)]$$

Policy Gradient: problems and fixes

The vanilla policy gradient estimator is unbiased, but very noisy.

- Requires lots of samples to make it work

Fixes:

- Baseline
- Temporal Structure
- Other (e.g. KL trust region)

Policy Gradient: baseline

$$\nabla_\theta U(\theta) \approx \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)};\theta)(R(\tau^{(i)}) - b)$$

Subtract the reward with a baseline (b) does not change the optimization problem.

- The gradient estimation is still unbiased, but with lower variance

Intuition: we want to adjust path probabilities based on how the path reward compares to the **average**, not the path reward itself.

- Increase probability if the path reward is higher than average
- Decrease probability if the path reward is lower than average

Policy Gradient: temporal structure

Put together what we have:

$$\nabla_\theta U(\theta) \approx \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)};\theta)(R(\tau^{(i)}) - b)$$

$$= \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta [\sum_{t=0}^{H-1} \log \pi_\theta(a_t^{(i)}|s_t^i)][\sum_{t=0}^{H-1} R(s_t^{(i)}, a_t^{(i)}) - b]$$

$$= \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta [\sum_{t=0}^{H-1} \log \pi_\theta(a_t^{(i)}|s_t^i)(\sum_{k=0}^{H-1} R(s_t^{(i)}, a_t^{(i)}) - b)]$$

Past reward does not affect current action

$$= \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta [\sum_{t=0}^{H-1} \log \pi_\theta(a_t^{(i)}|s_t^i)(\boxed{\sum_{k=0}^{t-1} R(s_t^{(i)}, a_t^{(i)})} + \sum_{k=t}^{H-1} R(s_t^{(i)}, a_t^{(i)}) - b)]$$

$$= \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta [\sum_{t=0}^{H-1} \log \pi_\theta(a_t^{(i)}|s_t^i)(\sum_{k=t}^{H-1} R(s_t^{(i)}, a_t^{(i)}) - b)]$$

OpenAI Gym

https://gym.openai.com/

Widely-used testing platform for RL algorithms.

- `pip install gym`

Different kinds of environments, including discrete / continuous control, pixel-input Atari games, etc.

You can also create your own environments, following the Gym interface.

OpenAI Gym environments

Create an environment:

- env = gym.make(“<environment_name>”) ← e.g. gym.make(“CartPole-v1”)

Env methods you will need the most:

- state = env.reset()
- next_state, reward, done, info = env.step(action)
- env.seed(seed=None)
- env.close()

Useful attributes:

- env.observation_space
- env.action_space

More documentation at https://gym.openai.com/docs/

Example: Policy Gradient in PyTorch on a Gym Environment (CartPole-v1)

References

- Sutton & Barto “Reinforcement Learning: An Introduction”, 1998
- Williams, “Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning”, 1992
- Sutton et al, “Policy Gradient Methods for Reinforcement Learning with Function Approximation”, 1999
- Pieter Abbeel, Deep RL Bootcamp Lecture 4A: Policy Gradients https://www.youtube.com/watch?v=S_gwYj1Q-44