Reinforcement Learning

From an optimal control perspective

Recall :

- Optimal Control Problem

$$\min_{x,u} \sum_{n=1}^{N-1} \ell(x_n, u_n) + \ell(x_N) \longrightarrow \text{cost}$$

$$\cancel{s.t \quad x_{n+1} = f(x_n, u_n)} \longrightarrow \text{dynamics}$$

→ RL is OC without an a priori known model

→ do a bunch of random rollouts

↳ use that to optimize your actions

Why do we care?

- We sometimes don't have great models of the environment/surroundings
- RL approaches naturally handle nonlinear, non-differentiable, partially observable and stochastic dynamics without special treatment
- RL methods parallelize super well
- RL approaches (have been made to) play well with deep networks

Reinforcement Learning

Policy Gradient methods

Directly learn the policy by estimating gradients using zeroth order methods

$\pi_\theta(x) \sim K$

$-K\delta x$

Problems → high variance

↳ instability

Q-Learning

- Learn an action-value function to approximate the cost to go

$$Q_\phi(x, u) \sim J_n(x, u)$$

- To find optimal action

$$u^* = \min_u Q_\phi(x, u)$$

Problems → generalization issues

↳ high bias

Model based RL

- Learn a model from data.

$$\min_\theta \mathbb{E}_{(x,u,x')}\left[\left(f_\theta(x, u) - x' \right)^2 \right]$$

- solve the OC problem with learnt model
- typically using sampling based methods for Nnet models

Problem → Objective Mismatch

↳ learning a lot of unnecessary info

Actor Critic Methods

Actor-critic methods aim to balance bias and variance by simultaneously learning policy and value function

Q-Learning

- Use dynamic programming :

$$V(x) = \min_u l(x,u) + V(\underbrace{f(x,u)}_{\text{dynamics}})$$

dynamics → we don't have access to it

$$Q(x,u) = l(x,u) + \min_{u'} Q(x',u')$$

Q-Learning

- Use dynamic programming to learn Q : minimize the following residual

$\begin{bmatrix} x \\ u \end{bmatrix}^T L \begin{bmatrix} x \\ u \end{bmatrix}$

$$min_{\phi}\ \mathbb{E}_{(x_n, u_n, x_{n+1})}\left[\left(Q_{\phi}(x_n, u_n) - (\ell(x_n, u_n) + \gamma \min_{u} Q_{\bar{\phi}}(x_{n+1}, u))\right)^2\right]$$

- Then take an argmin to compute optimal actions

$\max_{\phi, \psi_2}\left(\min_{u} Q_{\bar{\phi}_1}(\ ),\ \min_{u} Q_{\bar{\phi}_2}(\ )\right)$

$$u_n^* = \operatorname*{argmin}_{u} Q_{\phi}(x_n, u)$$

We perform rollouts using a stochastic policy

- Could do random exploration - but it's very inefficient

- Typically, we add noise to the controls/actions while collecting rollouts

$$u_n^* = \underset{u}{\operatorname{argmin}}\, Q_\phi(x_n, u) + \epsilon$$

$\hookrightarrow N(0,V)$

- Useful for exploration
- The noise helps mitigate some of the bias issues.

Deadly Triad

$Q(x_n, u_n)$

- Leads to "overestimation bias" or in this case "underestimation bias" since we are modelling cost.

- Three factors contribute
    - Function approximation
    - Bootstrap updates (updating using the current Q-value estimates as targets)
    - Off-policy updates

- Solutions

→ Double Q-learning → use old Q-value estimates as targets

→ two target networks (TD3) max (Q_1, Q_2)

Q-Learning for continuous actions

- For discrete actions this actually works great!

- For continuous actions
    - computing the argmin/min is annoying for expressive Q functions
    - We typically don't use this vanilla algorithm
    - Instead, we use some actor-critic based derivatives of this algorithm for continuous control problems.

Policy gradient methods

- Can we directly learn a policy?

$$\min_{\theta} \mathbb{E}_{\tau}\left[J\left(\tau\left(\pi_\theta \right) \right) \right]$$

→ zeroth order updates to policy
↳ sample random trajectories by perturbing θ
↳ move θ along zeroth order gradient directions

Policy gradient methods

- We want to minimize :

$$\min_{\theta} \mathbb{E}\left[J(\tau(\pi_\theta))\right]$$

- Compute zeroth order gradients!

- But this is inefficient - so we Couple of modifications
  - Use the policy gradient explicitly - $\nabla_\theta \pi_\theta(u|x)$ → stochastic policy
  - Exploit the sequential nature of the problem. → actions at time t only affects costs at time t+1 onwards

Policy gradient methods

- Rewriting the problem.

$$\min_\theta \mathbb{E}_{p(\tau;\theta)}[J(\tau)]$$

$$p(\tau;\theta) = \prod_{n=1}^{N-1} p(x_{n+1}|x_n, u_n)\pi_\theta(u_n|x_n)$$

transition probability

stochastic policy

Policy gradient trick!

- How do we compute gradients through the sampling process?

$$\nabla_\theta \mathbb{E}_{p(\tau;\theta)}[J(\tau)] = \int J(\tau)\nabla_\theta p(\tau;\theta)d\tau$$

$$= \int J(\tau)\left[\frac{\nabla_\theta p(\tau;\theta)}{p(\tau;\theta)}\right]p(\tau;\theta)d\tau$$

$$= \int J(\tau)\left[\nabla_\theta log(p(\tau;\theta))\right]p(\tau;\theta)d\tau$$

$$= \mathbb{E}_{p(\tau;\theta)}[J(\tau)\nabla_\theta log(p(\tau;\theta))]$$

$$\nabla_\theta\Big[\sum_{n=1}^{N-1} \cancel{log(p(x_{n+1}|x_n,u_n))} + log(\pi_\theta(u_n|x_n))\Big]$$

n

$$\nabla_\theta \mathbb{E}_{p(\tau;\theta)}[J(\tau)] = \mathbb{E}_{p(\tau;\theta)}\Big[J(\tau)\sum_{n=1}^{N-1}\nabla_\theta log(\pi_\theta(u_n|x_n))\Big]$$

Policy gradient

- Reminder that $$J(\tau) = \sum_{i=1}^{N-1} \ell(x_i, u_i) + \ell_N(x_N)$$

- We can use the sequential structure of the problem!

$$J_n(\tau) = \sum_{i=n}^{N-1} \ell(x_i, u_i) + \ell_N(x_N)$$

$$\mathbb{E}_{p(\tau;\theta)}\Big[J(\tau) \sum_{n=1}^{N-1} \nabla_\theta log(\pi_\theta(u_n|x_n))\Big] = \mathbb{E}_{p(\tau;\theta)}\Big[\sum_{n=1}^{N-1} \nabla_\theta log(\pi_\theta(u_n|x_n)) J_n(\tau)\Big]$$

- Finally! We have :

$$\nabla_\theta \mathbb{E}_{p(\tau;\theta)}[J(\tau)] = \mathbb{E}_{p(\tau;\theta)}\Big[\sum_{n=1}^{N-1} \nabla_\theta log(\pi_\theta(u_n|x_n)) J_n(\tau)\Big]$$

Intuitively, we are weighting the gradient corresponding to each sample by the cost to go!

Policy gradient for the LQR problem

$$\min_{\theta=K} \mathbb{E}_{v\sim N(0,V)} \left[\sum_{n=1}^{N-1} \frac{1}{2} x_n^T Q x_n + \frac{1}{2} u_n^T R u_n + \frac{1}{2} x_N^T Q x_N \right]$$

$$s.t \quad u_n = K x_n + v_n$$

$$x_n = A x_n + B u_n$$

- For this problem,

$$J_n(\tau) = \sum_{i=n}^{N-1} \frac{1}{2} x_i^T Q x_i + \frac{1}{2} u_i^T R u_i + \frac{1}{2} x_N^T Q x_N$$

$$\pi_\theta(u_n|x_n) = C\, exp(-\frac{1}{2}(u_n + K x_n)^T V^{-1} (u_n + K x_n))$$

$$\nabla_\theta log(\pi_\theta(u_n|x_n)) = -(u_n + K x_n)^T V^{-1} (x_n^T \otimes I)$$

Variance reduction

- One of the most important determinants of sample efficiency and stability!

$$\nabla_\theta \mathbb{E}_{p(\tau;\theta)}[J(\tau)] = \mathbb{E}_{p(\tau;\theta)}\Big[\sum_{n=1}^{N-1} \nabla_\theta log(\pi_\theta(u_n|x_n)) J_n(\tau)\Big]$$

$V(x)$

$b(x)$

$(J_n - b)$

Q_n

$V(x) + R(x,u)$

- Some common ways to reduce variance
  - → trust region methods, line searches
  - → schedule noise covariances for policy
  - → averaging parameters
  - → clip gradients
  - → advantage estimates
  - → baselines

Takeaways!

- Policy gradient methods suffer from high variance
- Q-learning methods suffer from high bias

- So we typically never use Q-learning or policy gradients as is for most continuous control tasks.
    - We instead typically use actor-critic approaches (independently or along with learnt models) - But the core design considerations flow from the bias-variance issues we discussed

- But ultimately, a lot of the solutions you'll see to handle these bias-variance issues will look like a bag of tricks/hacks (which in a lot of cases they are) - but that's just something we have to learn to accept?

Parting thoughts

- A lot of the utility of these RL approaches stem from the fact that they work well with deep networks!
    - Using auxiliary datasets - transferring vision models or natural language models
    - In a lot of cases we actually don’t even have a good cost function
        - But the auxiliary datasets can provide weak supervision/costs
- Use optimal control methods as primitives for low level physics based reasoning, while using RL with neural nets for higher level reasoning
- Solve problems outside robotics!
    - Navigating the web
    - Interactive teaching
    - Multi-agent/human coordination etc …
- If you have a ‘good’ simulator and a ‘good’ cost function - things are pretty much solved.