Reinforcement Learning

From an optimal control perspective

Recall:

Optimal Control Problem

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

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

$$\int RL \text{ is } DC \text{ without an a prive known model}$$

$$\int do \text{ a bunch of random relieves}$$

$$\int vsc \text{ that to optimize your actions}$$

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

Q-Learning

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

- To find optimal action

Actor Critic Methods

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

Model based RL

- Learn a model from data.

win
$$\mathbb{E}_{\sigma_{i}, \nu_{i}} \left(f_{\sigma}(\lambda, \nu) - \lambda' \right)$$

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

Problem -s Objective Mismetch

learning a lot of unnecessary info

Q-Learning

Use dynamic programming :

$$V(x) = \min_{u} l(x, u) + V(f(x, u))$$

$$\lim_{v \to u} l(x, u) + V(f(x, u))$$

$$\lim_{v \to u} l(x, u) + \lim_{v \to u} l(x, u)$$

$$\lim_{v \to u} l(x, u) + \lim_{v \to u} l(x, u)$$

Q-Learning

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

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

- Then take an argmin to compute optimal actions

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

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

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

Deadly Triad

Q (n, un)

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-leavy - use old Q-volve estimates is tigets

- two tayet returnes (TD3) max (Q, Q2)

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?

Policy gradient methods

We want to minimize :

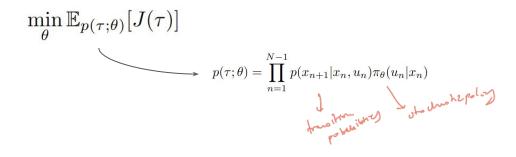
$$\min_{\theta} \mathbb{E}\left[J(\tau(\pi_{\theta}))\right]$$

Compute zeroth order gradients!

- But this is inefficient so we Couple of modifications
 - Use the policy gradient explicitly $\nabla_{\theta}\pi_{\theta}(u|x)$ stockestic p by
 - Exploit the sequential nature of the problem. actions at time to only affects costs at time to onwards

Policy gradient methods

Rewriting the problem.



Policy gradient trick!

- How do we compute gradients through the sampling process?

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

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

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

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

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

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

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!

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

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

Finally! We have :

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

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,

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

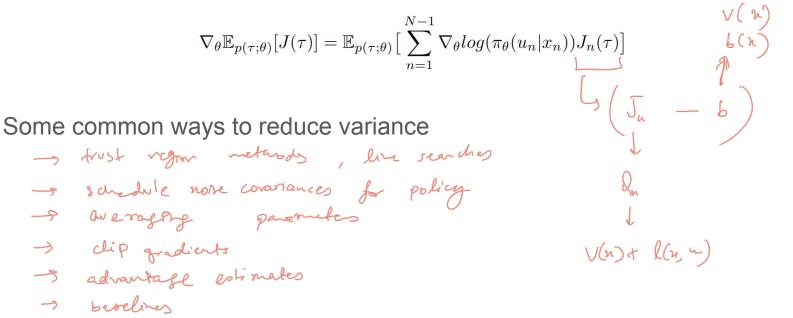
$$\pi_{\theta}(u_n|x_n) = C \exp(-\frac{1}{2}(u_n + Kx_n)^T V^{-1}(u_n + Kx_n))$$

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

Variance reduction

-> bereliner

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



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.