A Pontryagin Perspective on Reinforcement Learning

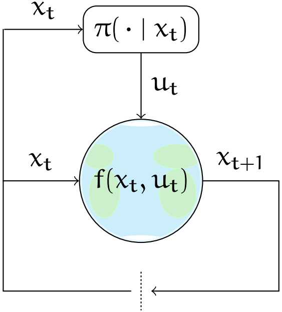
or: "Open-Loop Reinforcement Learning"

Onno Eberhard[1,2] · Claire Vernade[2] · Michael Muehlebach[1]

[1]Max Planck Institute for Intelligent Systems [2]University of Tübingen

April 19, 2024

Recap: reinforcement learning

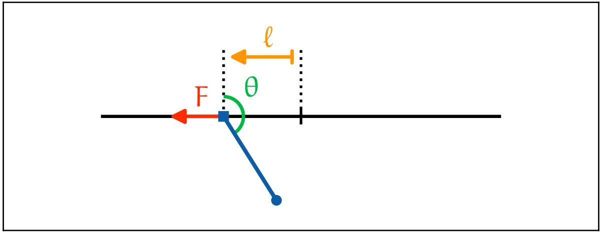


- Goal: learn a policy π that maximizes the sum of rewards

$$\pi^\star = \underset{\pi:\mathcal{X}\to\Delta_{\mathcal{U}}}{\arg\max}\ \mathbb{E}_\pi\left[\sum_{t=0}^{T-1} r(x_t, u_t) + r_T(x_T)\right]$$

- Challenge: the system is unknown, only simulations possible

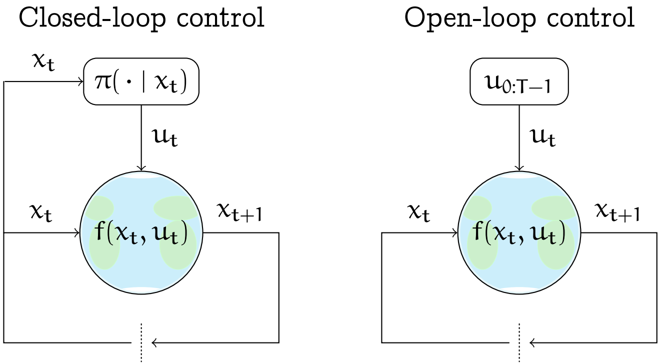
Example: inverted pendulum swing-up



- State: $x = (\ell, \dot{\ell}, \theta, \dot{\theta})$
- Action: $u = F$ (external force)
- Rewards: positive when upright and at rest, penalty for large force
- Classical benchmark in control theory

Open-loop control

- Pendulum is "easily" solvable with deep RL, e.g. soft actor-critic (SAC)
  - SAC uses multiple neural networks, $> 100,000$ parameters in total
  - Overkill! Why not just learn the actions that are necessary?



Pontryagin's principle

- Open-loop optimal control:

$$u^\star_{0:T-1} = \underset{u_{0:T-1}\in\mathcal{U}^T}{\arg\max} \underbrace{\sum_{t=0}^{T-1} r(x_t, u_t) + r_T(x_T)}_{J(u_{0:T-1})} \quad \text{s.t.} \quad x_{t+1} = f(x_t, u_t)$$

- Gradient-based optimization: how to compute $\nabla_{u_t} J(u_{0:T-1})$?

Pontryagin's principle for computing $\nabla_{u_t} J(u_{0:T-1})$

1. Forward pass: $x_{t+1} = f(x_t, u_t)$, where x_0 is given
2. Backward pass: $\lambda_t = \nabla_x r(x_t, u_t) + \nabla_x f(x_t, u_t)\lambda_{t+1}$, where $\lambda_T = \nabla r_T(x_T)$
3. Gradient: $\nabla_{u_t} J(u_{0:T-1}) = \nabla_u r(x_t, u_t) + \nabla_u f(x_t, u_t)\lambda_{t+1}$

Open-loop reinforcement learning

Theorem (informal)

Replace $\nabla_x f_t$ and $\nabla_u f_t$ in Pontryagin's equations by estimates A_t and B_t with sufficiently small errors $\|\nabla_x f_t - A_t\|$ and $\|\nabla_u f_t - B_t\|$ to get an approximate gradient $g \simeq \nabla J(u_{0:T-1})$. Gradient ascent on g produces iterates $u_{0:T-1}^{(0)}, \ldots, u_{0:T-1}^{(N-1)}$ that satify

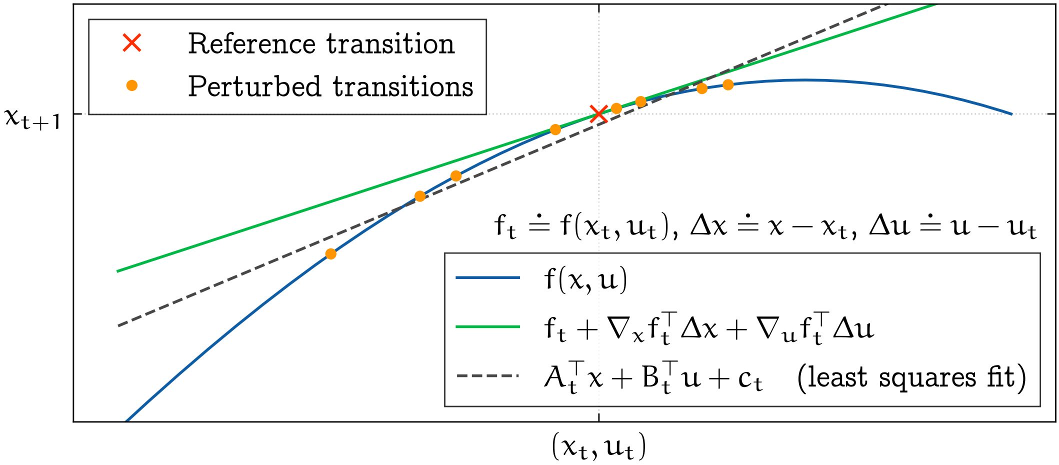
$$\frac{1}{N}\sum_{k=0}^{N-1}\|\nabla_{u_t} J(u_{0:T-1}^{(k)})\|^2 \leqslant \frac{J^\star - J(u_{0:T-1}^{(0)})}{\alpha\eta N}.$$

How should we choose A_t and B_t?

- Model-based open-loop RL: given a model $\tilde{f}$, use $A_t \doteq \nabla_x \tilde{f}_t$ and $B_t \doteq \nabla_u \tilde{f}_t$
- Model-free open-loop RL: estimate $\nabla_x f_t$ and $\nabla_u f_t$ directly

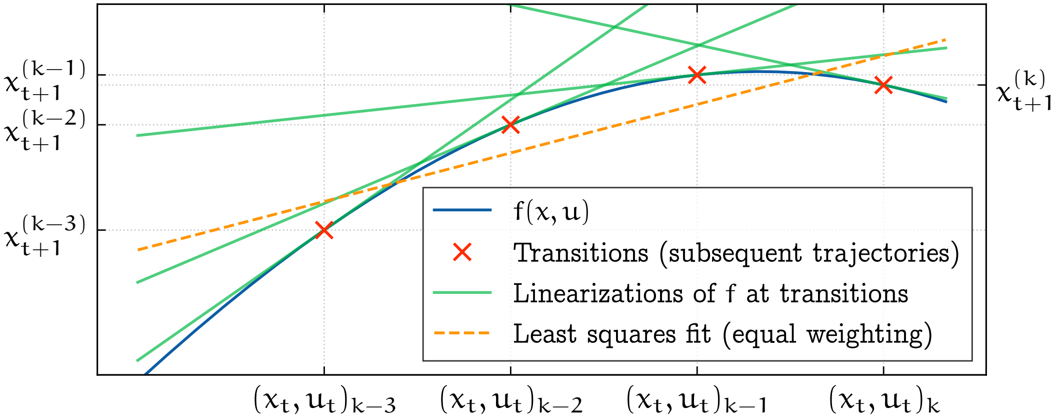
Model-free open-loop RL

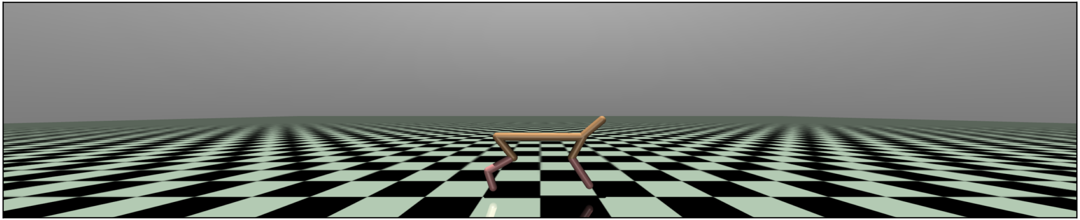
- The Jacobians $\nabla_x f_t$ and $\nabla_u f_t$ measure how x_{t+1} changes if (x_t, u_t) is perturbed

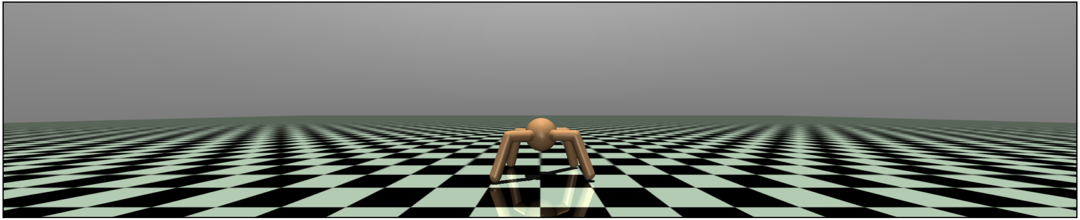


Off-trajectory open-loop RL

- Subsequent trajectories are similar, no need to throw all data away!
- Algorithm: recursive least squares with forgetting







Fin.