CS 277: Control and Reinforcement Learning

Winter 2024

Lecture 9: Planning

Roy Fox
Department of Computer Science
School of Information and Computer Sciences
University of California, Irvine

Logistics

assignments

- Quiz 5 to be published soon, due next Monday
- Exercise 3 due following Monday

Today's lecture

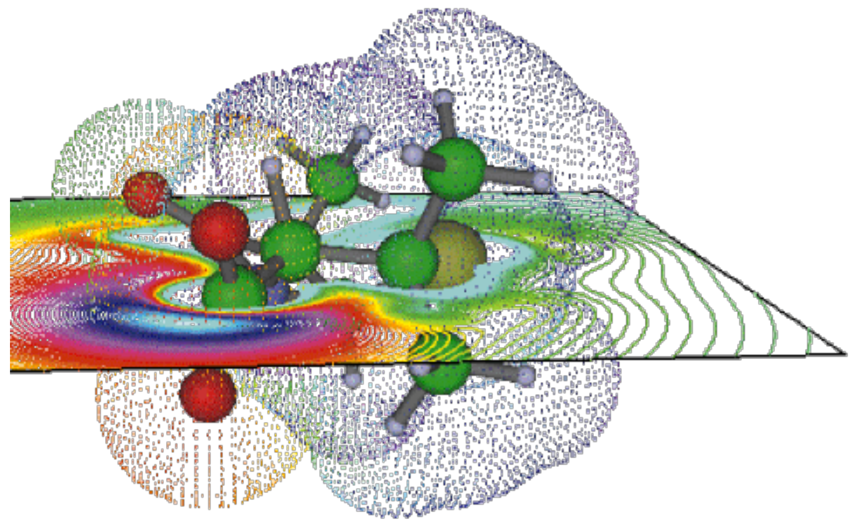
Planning

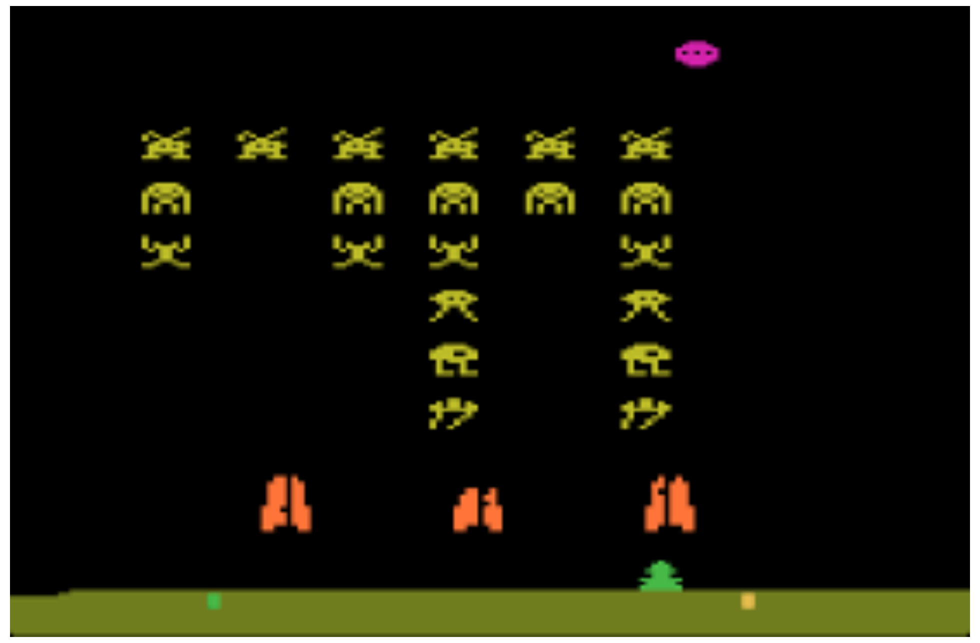
iLQR

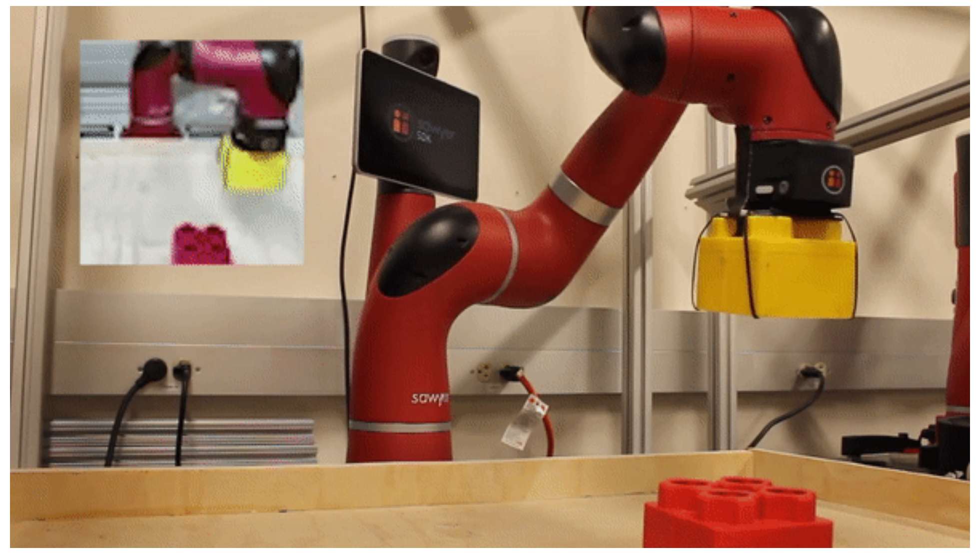
Model-based learning

Planning

- Planning: finding a good policy π when we "know" the MDP model
  - MDP = dynamics + reward function
- When do we "know" the model?
  - Well-modeled environments
    - Dynamics equations
  - Simulators
  - Learned models
  - System identification: the agent itself learns a model







Levels of "knowing" a model

- What does it mean to have a "known" model?
  - A really fast simulator
    - Analytic model, fast implementation, parallelization, approximate (high-level) model
  - A simulator that can be reset to any given state
    - Sample $p(s' \mid s, a)$ for any (s, a), rather than an entire trajectory $p_\pi(\xi)$ with $s \sim p_\pi$
  - An analytic model (e.g. equations) that can be manipulated symbolically
  - A differentiable model
    - Backprop gradients through p

How to use a really fast simulator

- Any RL algorithm can benefit from more data

Algorithm MC model-free RL

Initialize some policy π

repeat

  Initialize some value function Q

  repeat to convergence

    Sample $\xi \sim p_\pi$ ← need to add exploration

    Update $Q(s_t, a_t) \to R_{\geq t}(\xi)$ for all $t \geq 0$

  $\pi(s) \leftarrow \arg\max_a Q(s, a)$ for all s

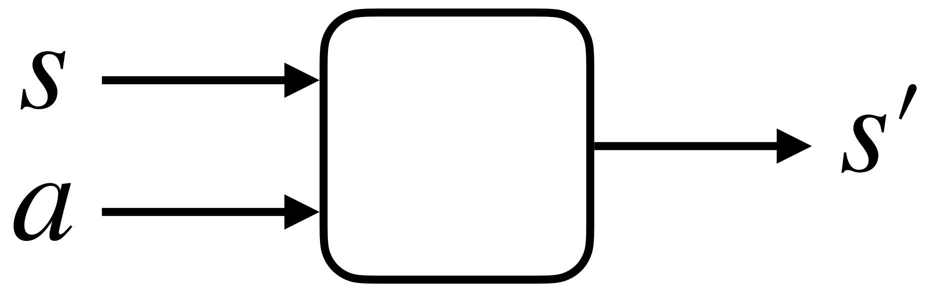
- Simple, unbiased, consistent algorithm

- High variance ⇒ with fast simulator, can sample many trajectories

How to use an arbitrary-reset simulator

- Arbitrary-reset simulator allows sampling from $(s' \mid s, a) \sim p$ for any (s, a) we want
- Small state space — can run Value Iteration with tabular parametrization:

$$V(s) \leftarrow \max_a r(s,a) + \gamma \mathbb{E}_{(s'|s,a)\sim p}[V(s')]$$



- Large state space — should we use Fitted Value Iteration?

$$\mathscr{L}_\theta(s) = (\min_a r(s,a) + \gamma \mathbb{E}_{(s'|s,a)\sim p}[V_{\bar\theta}(s')] - V_\theta(s))^2$$

  - Problem: must have $s \sim p_\theta(\xi)$, or suffer covariate shift (train–test mismatch)
    - $p_\theta(\xi)$ requires sampling entire trajectories, starting from s_0, arbitrary-reset is no help (for this)
- Simulator does enable data augmentation: perturb $s_t \sim p_\theta(\xi)$ and see how it evolves

Deterministic dynamics

- With deterministic dynamics, we can fully predict future states
  - Open-loop control: policy doesn't depend on observations = sequence of actions

$$\max_{\vec{a}} R(\vec{a}) = \max_{\vec{a}} r(s_0, a_0) + \gamma r(f(s_0, a_0), a_1) + \gamma^2 r(f(f(s_0, a_0), a_1), a_2) + \cdots$$

- Use any black-box optimizer; e.g. stochastic optimization:

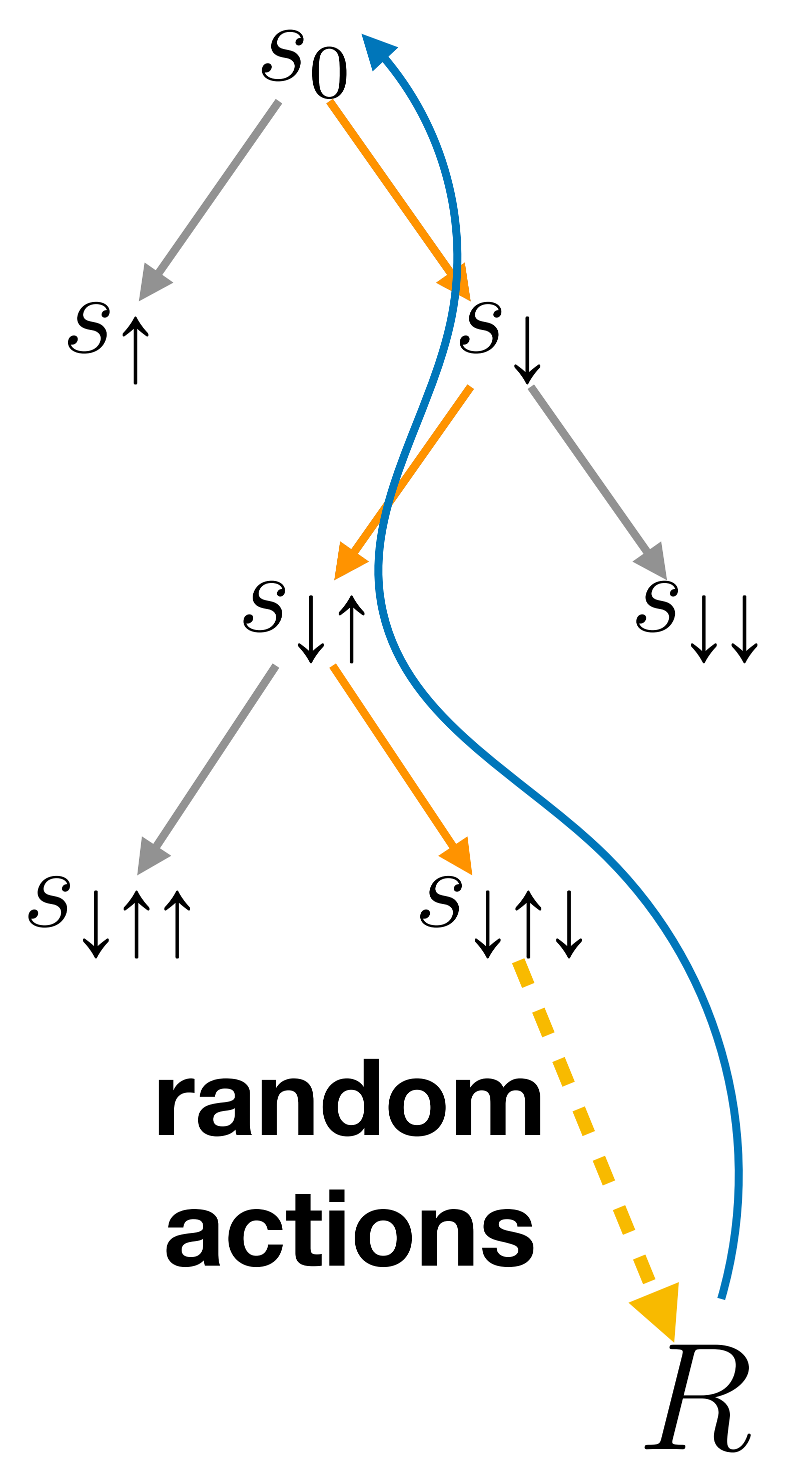
Algorithm Stochastic optimization

Initialize π
repeat
  Sample $\vec{a}_1, \dots, \vec{a}_k \sim \pi$
  Run model to get returns $R_1, \dots, R_k$
  Select k/c top returns
  Fit π to these "elites"

- Scales poorly with the dimension of $\vec{a}$

Discrete action space: optimal exploration

- Action sequences have a tree structure
  - Shallow (short) prefixes are visited often ⇒ possible to learn their value
  - Deep (long) sequences are visited rarely ⇒ we can only explore
- Monte Carlo Tree Search (MCTS):
  - Select leaf of the already-learned subtree
  - Explore to end of episode
  - Update nodes along branch to leaf



- Selecting a leaf: recursively maximize

exploration bonus

$$\begin{cases} \infty & \text{if } N_{\text{visits}}(\text{child}) = 0 \\ V(\text{child}) + C\sqrt{\dfrac{\log N_{\text{visits}}(\text{self})}{N_{\text{visits}}(\text{child})}} & \text{otherwise} \end{cases}$$

Today's lecture

Planning

iLQR

Model-based learning

How to use a differentiable model

- Suppose we have differentiable $x_{t+1} = f(x_t, u_t)$ and $c(x_t, u_t)$
- Taylor expansion for ϵ-perturbation $(\delta x, \delta u)$ around a trajectory $(\hat{x}, \hat{u})$:

interesting dependence on x_t and u_t

$$\hat{f}(x_t, u_t) = \hat{f}(\hat{x}_t, \hat{u}_t) + O(\epsilon)$$

How to use a differentiable model

- Suppose we have differentiable $x_{t+1} = f(x_t, u_t)$ and $c(x_t, u_t)$
- Taylor expansion for ϵ-perturbation $(\delta x, \delta u)$ around a trajectory $(\hat{x}, \hat{u})$:

captures linear dependence on x_t and u_t

$$\hat{f}(x_t, u_t) = \hat{f}(\hat{x}_t, \hat{u}_t) + \delta x_t \nabla_x \hat{f}_t + \delta u_t \nabla_u \hat{f}_t + O(\epsilon^2)$$

How to use a differentiable model

- Suppose we have differentiable $x_{t+1} = f(x_t, u_t)$ and $c(x_t, u_t)$
- Taylor expansion for ϵ-perturbation $(\delta x, \delta u)$ around a trajectory $(\hat{x}, \hat{u})$:

$$\hat{f}(x_t, u_t) = \hat{f}(\hat{x}_t, \hat{u}_t) + \delta x_t \nabla_x \hat{f}_t + \delta u_t \nabla_u \hat{f}_t + O(\epsilon^2)$$

$$\hat{c}(x_t, u_t) = \hat{c}(\hat{x}_t, \hat{u}_t) + O(\epsilon)$$

interesting dependence on x_t and u_t

How to use a differentiable model

- Suppose we have differentiable $x_{t+1} = f(x_t, u_t)$ and $c(x_t, u_t)$
- Taylor expansion for ϵ-perturbation $(\delta x, \delta u)$ around a trajectory $(\hat{x}, \hat{u})$:

$$\hat{f}(x_t, u_t) = \hat{f}(\hat{x}_t, \hat{u}_t) + \delta x_t \nabla_x \hat{f}_t + \delta u_t \nabla_u \hat{f}_t + O(\epsilon^2)$$
$$\hat{c}(x_t, u_t) = \hat{c}(\hat{x}_t, \hat{u}_t) + \delta x_t \nabla_x \hat{c}_t + \delta u_t \nabla_u \hat{c}_t + O(\epsilon^2)$$

linear dependence on x_t and u_t

optimal control: ∞

How to use a differentiable model

- Suppose we have differentiable $x_{t+1} = f(x_t, u_t)$ and $c(x_t, u_t)$
- Taylor expansion for ϵ-perturbation $(\delta x, \delta u)$ around a trajectory $(\hat{x}, \hat{u})$:

NOW we can neglect these

$$\hat{f}(x_t, u_t) = \hat{f}(\hat{x}_t, \hat{u}_t) + \delta x_t \nabla_x \hat{f}_t + \delta u_t \nabla_u \hat{f}_t + O(\epsilon^2)$$

$$\hat{c}(x_t, u_t) = \hat{c}(\hat{x}_t, \hat{u}_t) + \delta x_t \nabla_x \hat{c}_t + \delta u_t \nabla_u \hat{c}_t$$

$$+\frac{1}{2}(\delta x_t^\intercal (\nabla_x^2 \hat{c}_t) \delta x_t + \delta u_t^\intercal (\nabla_u^2 \hat{c}_t) \delta u_t + 2 \delta x_t^\intercal (\nabla_{xu} \hat{c}_t) \delta u_t) + O(\epsilon^3)$$

Iterative LQR (iLQR)

Algorithm iLQR

Initialize $\hat{x}, \hat{u}$

repeat

Set $A, B \leftarrow \nabla_x \hat{f}_t, \nabla_u \hat{f}_t$ — linearize dynamics around current trajectory $(\hat{x}, \hat{u})$

Set $Q, R, N, q, r \leftarrow \nabla_x^2 \hat{c}_t, \nabla_u^2 \hat{c}_t, \nabla_{xu} \hat{c}_t, \nabla_x \hat{c}_t, \nabla_u \hat{c}_t$ — quadratic cost approximation around $(\hat{x}, \hat{u})$

$\hat{L}_t, \hat{\ell}_t \leftarrow$ LQR on $\delta x_t = x_t - \hat{x}_t, \delta u_t = u_t - \hat{u}_t$ ← place "origin" at $(\hat{x}, \hat{u})$

$\delta\hat{x}, \delta\hat{u} \leftarrow$ execute policy $\delta u_t = \hat{L}_t \delta x_t + \hat{\ell}_t$ in env — roll out to get new trajectory $(\hat{x}, \hat{u})$

$\hat{x} \leftarrow \hat{x} + \delta\hat{x}, \hat{u} \leftarrow \hat{u} + \delta\hat{u}$

Newton's method

- Compare iLQR with Newton's method for optimizing $\min_x f(x)$

Algorithm Newton's method

---

repeat

$g \leftarrow \nabla_x f(\hat{x})$

$H \leftarrow \nabla_x^2 f(\hat{x})$

$\hat{x} \leftarrow \operatorname{argmin}_x \frac{1}{2}\delta x^\intercal H \delta x + g^\intercal \delta x$

- iLQR approximates this method for $\min_u \mathcal{J}(u)$
- This would be exact if we expanded the dynamics to 2nd order
  - Similar to iLQR, called Differential Dynamic Programming (DDP)

Today's lecture

Planning

iLQR

Model-based learning

Learning vs. planning

- Model = dynamics + reward function
  - Planning = finding a good policy with access to a model
- Learning = improving performance using data
  - Are rollouts from the model considered "data"?
    - If yes, planning can involve learning
- Model-based learning = methods that explicitly learn the model
  - Unlike planning, access to a model is not given; it is learned
  - Usually, focus on dynamics p, because reward function r is simulated

Model-based learning

- Is a learning algorithm $\mathscr{A}$ model-based?
- In tabular representation — just count parameters:
  - Model-free = $O(|\mathcal{S}| \cdot |\mathscr{A}|)$ (to represent $\pi(a|s)$ or $Q(s,a)$)
  - Model-based = $\Omega(|\mathcal{S}|^2 \cdot |\mathscr{A}|)$ (to represent $p(s'|s,a)$)
- Not always clear-cut:
  - If intermediate features of DQN $Q_\theta(s,a)$ are informative of s', is this model-free?
- Not to be confused with ML terminology calling anything learned a “model”

Model-based learning: benefits

- Dynamics p has “more parameters” than $\pi \Rightarrow$ harder to learn? not always
  - p can have simpler form and generalize better to unseen states and actions and
  - p can be learned locally; π or Q encode global knowledge (long-term planning)
- Model-based methods produce transferable knowledge
  - Useful if MDP changes only slightly / partially (non-stationary environment)
    - E.g. only the task changes, i.e. r changes but not p
    - Can generalize across environment changes, e.g. friction or arm length
    - Can help transfer learning in an inaccurate simulator to the real world (sim2real)

How to learn a model

- Interact with environment to get trajectory data
  - Deterministic continuous dynamics / reward: minimize MSE loss

$$\mathscr{L}_\phi(s, a, r, s') = \|s' - f_\phi(s, a)\|_2^2 + (r - r_\phi(s, a))^2$$

  - Stochastic dynamics: minimize NLL loss

$$\mathscr{L}_\phi(s, a, s') = -\log p_\phi(s' | s, a)$$

- Data can be off-policy ⇒ unbiased estimate, but with covariate shift
  - Random policy is often used
- Another possibility discussed later

How to use a learned model

- Recall how planning benefitted from access to a model:
  - As a fast simulator
  - As an arbitrary-reset simulator
  - As a differentiable model

How to use a learned model

- Recall how planning benefitted from access to a model:
  - As a fast simulator
  - As an arbitrary-reset simulator
  - As a differentiable model

Policy Gradient through the model

- Model is often learned with SGD ⇒ must be differentiable

$$\hat{J}_\theta = \sum_t \gamma^t \hat{c}(x_t, u_t) = \sum_t \gamma^t \hat{c}(\hat{f}(\cdots \hat{f}(x_0, \pi_\theta(x_0)) \cdots, \pi_\theta(x_{t-1})), \pi_\theta(x_t))$$

- Just do Policy Gradient over $\hat{J}_\theta$?
  - Chain rule ⇒ back-propagation through time (BPTT)
- $\nabla_\theta \hat{J}_\theta$ can be bad approximation of $\nabla_\theta J_\theta$; also, $\hat{J}_\theta$ is ill-conditioned for SGD:
  - Perturbing one action individually may change $\hat{J}_\theta$ unreasonably little / much
    - Vanishing / exploding gradients
  - Second-order methods can help, but Hessian is even nastier — for the same reason

PG with a model

- Luckily, we have the Policy Gradient Theorem

$$\nabla_\theta \hat{J}_\theta = \mathbb{E}_{\xi \sim p_\theta}\left[\sum_t \gamma^t \hat{Q}_{\bar{\theta}}(s_t, a_t)\,\nabla_\theta \log \pi_\theta(a_t \mid s_t)\right]$$

- Idea: use the model as a fast simulator just to estimate $\hat{Q}_{\bar{\theta}}(s_t, a_t)$
  - E.g., by MC or TD
  - Avoids complications of gradients through the model
    - Only backprop through single-step $\log \pi_\theta(a_t \mid s_t)$

Recap

- A fast simulator is good for any RL algorithm, particularly MC
  - MCTS explores optimally in the discrete deterministic case
- An arbitrary-reset simulator has surprisingly little use
  - Notable exception: domain randomization
- An analytic model may allow direct optimization, or very fast simulation
- We can plan in a differentiable model by iterative linearization (iLQR)