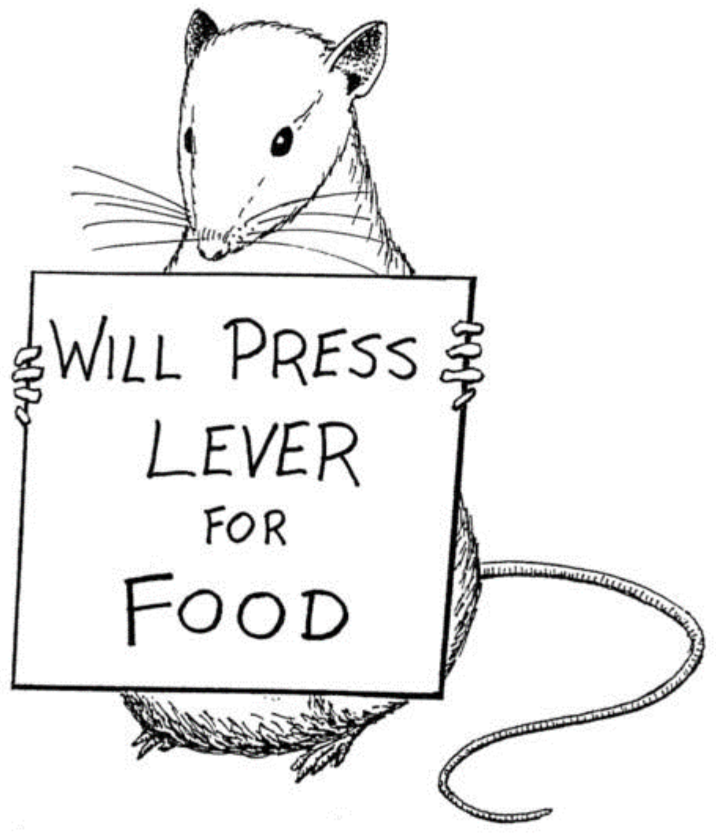
University of California, Irvine

CS 277: Control and Reinforcement Learning

Winter 2024

Lecture 10: Model-Based Methods

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



Logistics

assignments

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

Today's lecture

Model-based learning

Model-free learning with a model

Model-predictive control

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 = $O(|\mathcal{S}| \cdot |\mathcal{A}|)$ (to represent $\pi(a|s)$ or $Q(s,a)$)
  - Model-based = $\Omega(|\mathcal{S}|^2 \cdot |\mathcal{A}|)$ (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

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

  - Stochastic dynamics: minimize NLL loss

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

- 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

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

- 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 $\log \pi_\theta(a_t | s_t)$
  - Only the policy evaluation / critic is model-based

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)

Today's lecture

Model-based learning

Model-free learning with a model

Model-predictive control

How to use a learned model

- Ways to use a learned model:
  - As a fast simulator
  - As an arbitrary-reset simulator
  - As a differentiable model

Model-free RL with a model

- General scheme for using a model for model-free RL:

Algorithm Model-free RL with a model

Collect data ← interaction with environment (random policy)

Train model $\hat{p}, \hat{r}$ ← supervised learning

repeat

Sample s from the replay buffer ← seeded by initial interaction; may interact more as learner improves

Sample $(a|s) \sim \pi_\theta$

Simulate $r = \hat{r}(s, a)$ and $(s'|s, a) \sim \hat{p}$ ← use model as simulator

Perform model-free RL with (s, a, r, s')

- Benefit: get diverse off-policy s, and fresh on-policy a

Model-free RL with a model

- On-policy actions ⇒ allows n-step estimation without bias:

Algorithm Multi-step RL with a model

Collect data
Train model $\hat{p}, \hat{r}$
repeat
  Sample s from the replay buffer
  Roll out the learner's policy for n steps in the simulator
  Perform n-step model-free RL

- $\hat{r}(s_t, a_t) + \gamma \hat{r}(\hat{s}_{t+1}, a_{t+1}) + \cdots + \gamma^{n-1} \hat{r}(\hat{s}_{t+n-1}, a_{t+n-1})$ is unbiased
  - Except for model inaccuracy

Dyna

Algorithm Dyna

Collect data
Train model $\hat{p}, \hat{r}$
repeat
    Sample (s, a) from the replay buffer
    $Q(s,a) \to \hat{r}(s,a) + \gamma\, \mathbb{E}_{(s'|s,a)\sim\hat{p}}[\max_{a'} Q(s',a')]$

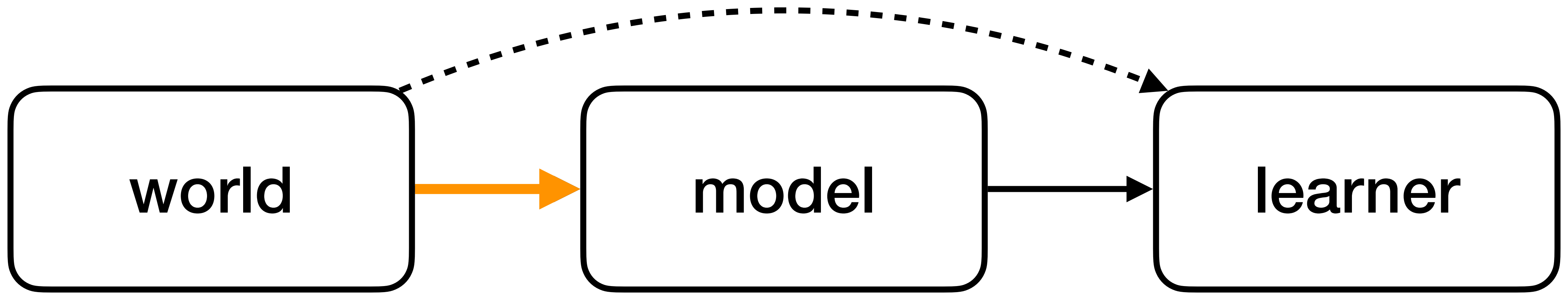
use model as simulator to estimate

- Another idea: also mix in samples generated from learner interactions
  - Benefit: keep training the model to be good for states that learner sees
  - With function approximation: feed the replay buffer and reduce covariate shift

Wait... Model-free RL... *with* a model?

- Why be model-free if we have the model?
- Learning to control is inherently model-free
  - Policy gradient is 0 for the $\log p(s' | s, a)$ terms of $\log p_\theta(\xi)$
  - Same in Imitation Learning: optimize NLL $\mathscr{L}_\theta(s_t, a_t) = -\log \pi_\theta(a_t | s_t)$
  - As opposed to planning, which requires averaging over futures
- The model still gives benefits
  - It can diversify the experience data, like a replay buffer but more so
  - Indirect benefits: generalization, transfer

Optimal exploration for model learning



- How to explore optimally for learning the model?
- Explicit Explore or Exploit (E[3]):
  - Maintain set S of sufficiently explored states
  - The model $\hat{M}$ has the empirical transitions and rewards on S
  - Other states collapsed to absorbing state with reward 0 (in $\hat{M}$) or $r_{\max}$ (in $\hat{M}'$)
- Principle of optimism under uncertainty

Explicit Explore or Exploit (E³)

Algorithm E³

$S \leftarrow \emptyset$
repeat
  $\pi \leftarrow$ optimal plan in $\hat{M}$ pessimistic model
  if Pr(π reaches absorbing state) $< \epsilon$ **then**
    Terminate
  else
    Execute optimal plan in $\hat{M}'$ optimistic model
    if $s \notin S$ reached **then**
      Take least tried action
      if each action tried K times **then**
        Empirically estimate $\hat{p}(\cdot|s,\cdot)$, $\hat{r}(s,\cdot)$
        Add s to S

- When probability to explore is low, optimal policy in $\hat{M}$ is truly near-optimal
- For provable guarantees, ϵ and K can be determined from real number of states
  - Or updated every time the number of visited states is doubled

R-max

- E[3] takes different actions when it explores or exploits
  - $\Rightarrow$ needs to know which at start of episode, many steps ahead
- Instead, plan only in optimistic $\hat{M}'$
  - Implicit explore or exploit: either

Algorithm R-MAX

mark all states *unknown*
repeat
  Execute $\pi \leftarrow$ optimal plan in $\hat{M}'$
  Record (s, a, r, s') in *unknown* states
  if $n(s) = K$ **then**
    Empirically estimate $\hat{p}(\cdot|s, \cdot)$, $\hat{r}(s, \cdot)$
    Mark s *known*

Today's lecture

Model-based learning

Model-free learning with a model

Model-predictive control

Issues with approximate models (1)

- In large state / action spaces, we can only approximate the dynamics
- No guarantees outside of training distribution
  - We can't be too far off-policy
- Solution: keep interacting using learner policy and updating the model

Issues with approximate models (2)

- Model inaccuracy accumulates
  - If $|p_\phi(s'|s,a) - p(s'|s,a)|_1 \leq \epsilon$ then $|p_\phi(s_t) - p(s_t)|_1 \leq \epsilon t$
  - We have to plan far enough ahead to realize the consequences of actions
  - But we don't have to execute those plans far ahead!

Algorithm Model-Predictive Control (MPC)

$\mathcal{D} \leftarrow$ collect data
repeat
  $\hat{M} \leftarrow$ train model $\hat{p}, \hat{r}$ from $\mathcal{D}$
  repeat
    $\pi \leftarrow$ plan in $\hat{M}$ from current state s to horizon H
    Take *one action* a according to π
    Add empirical (s, a, r, s') to $\mathcal{D}$

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

Local models

- Can we use a learned model for iLQR?
  - Idea 1: learn global model, linearize locally ⇒ wasteful
  - Idea 2: directly learn local linearizations:

Algorithm Local Models

Initialize a policy $\pi(u_t|x_t)$
repeat
    Roll out π to horizon T for N trajectories
    Fit $p(x_{t+1}|x_t, u_t)$
    Plan new policy π

How to fit local dynamics

- Idea 1: linear regression
  - Find $(A_t, B_t)_{t=0}^{T-1}$ such that $x_{t+1} \approx A_t x_t + B_t u_t$
  - Do we care about the process noise ω_t?
    - If we assume it's Gaussian, doesn't affect policy; but could help evaluate the method
- Idea 2: Bayesian linear regression
  - Learn global model, use it as prior for local model
  - More data efficient across time steps and across iterations

How to plan with local models

- Idea 1: as in iLQR, find optimal control sequence $\hat{u}$ and its trajectory $\hat{x}$
  - Problem: model errors will cause actual trajectory to diverge from $\hat{x}$
- Idea 2: find $\hat{x}$ by executing the optimal policy directly in the environment
  - Problem: need spread for linear regression, dynamics may be too deterministic
- Idea 3: make control stochastic by injecting Gaussian noise
  - E.g., have $\epsilon_t \sim \mathcal{N}(0, R^{-1})$, shaped by the control cost
    - Optimal for the incurred costs, not for the spread needed for regression

Recap

- Model-based RL schemes:
  - Plan in a learned model
  - Improve model-free RL using a learned model
- Good theory for how to explore optimally for learning a model