Model-Based Reinforcement Learning

CS 224R
Course reminders

- Project proposal due this Wednesday
  (graded fairly lightly — really for your benefit!)
- Homework 2 due next Wednesday (start early!)

Following up on high-resolution feedback:
- Optional readings posted on course website
- The most math-dense lectures are behind us.
- Unfortunately don’t have TA bandwidth to support live zoom questions
- Covering RLHF on Weds.
The plan for today

1. A brief primer on sampling-based optimization
2. Model-based reinforcement learning
   a. How to get a good dynamics model?
   b. How to use a (learned) dynamics model?
3. Case study in dexterous robotic manipulation

Key learning goals:
- model-based RL methods, and how to implement them
- the key challenges arising in model-based reinforcement learning
- tradeoffs between different model-based RL approaches
Teaser: How to get a robot to learn this?
Gradient-based vs. sampling-based optimization

Gradient-based (1st order)

- Objective: Minimize $L(\theta)$
- Strategy: Use gradient information
- Algorithm:
  1. Sample $\theta_1, \ldots, \theta_K$
  2. Rank samples according to loss $\theta_1, \ldots, \theta_K$
  3. Fit Gaussian distribution $p_i(\theta)$ to “elite” samples $\theta_1 \ldots k$
  4. Repeat

Sampling-based (0th order)

- Objective: Minimize $L(\theta)$
- Strategy: Sample and evaluate
- Algorithm:
  1. Sample $\theta_1, \ldots, \theta_K$
  2. Evaluate $L(\theta_1), \ldots, L(\theta_K)$
  3. Repeat

Cross-entropy method (CEM) (Not to be confused with the cross-entropy loss!)

1. Sample from distribution $p_i(\theta)$
2. Rank samples according to loss $\theta_1, \ldots, \theta_K$
3. Fit Gaussian distribution $p_{i+1}$ to “elite” samples $\theta_1 \ldots k$
   
   Eventually return $\theta_1$
Gradient-based vs. sampling-based optimization

**Gradient-based (1st order)**

- scalable to high dimensions
- works well *especially* in overparametrized regimes
- requires nice optimization landscape

**Sampling-based (0th order)**

- parallelizable
- requires no gradient information
- scales poorly to high dimensions
The plan for today

1. A brief primer on sampling-based optimization

2. Model-based reinforcement learning
   a. How to get a good dynamics model?
   b. How to use a (learned) dynamics model?

3. Case studies
Recap: The anatomy of a reinforcement learning algorithm

Previously: introduced model-free RL methods (policy gradient, Q-learning)

This lecture: focus on model-based RL methods
Model-based reinforcement learning

**Key idea:** It would be useful if we could approximately simulate the world! i.e. if we could predict the consequences of our actions

- Generate samples (i.e. run the policy)
- Fit a model to estimate return
- Improve the policy

**Model-based learning:**
- Estimate $p(s'|s, a)$
- Minimize $\min_{\phi} \sum_i \| f_\phi(s_i, a_i) - s_i' \|^2$
- Optimize $\pi_\theta(a|s)$
The plan for today

1. A brief primer on sampling-based optimization
2. Model-based reinforcement learning
   a. How to get a good dynamics model?
   b. How to use a (learned) dynamics model?
3. Case study in dexterous robotic manipulation
How to get a good dynamics model?

Fit a predictive model:
- input: s, a
- output: s’

Example models:
- robotics
  - video prediction model (possibly in some image representation space)
  - physics model some unknown free parameters
    (e.g. unknown coefficient of friction)
- **dialog**: large language model
- **finance**: stock market predictor
The plan for today

1. A brief primer on sampling-based optimization
2. Model-based reinforcement learning
   a. How to get a good dynamics model?
   b. How to use a (learned) dynamics model?
3. Case study
   - for planning
   - for learning a policy
Approach 1: Optimize over actions using model \[ \max_{a_{t:t+H}} \sum_t r(s_t, a_t) \]

“planning”

Approach 1a: via backpropagation
(i.e. gradient-based optimization)

Algorithm:

1. Run some policy (e.g. random policy) to collect data \( \mathcal{D} = \{(s, a, s')_i\} \)
2. Learn model \( f_\phi(s, a) \) to minimize \( \sum_i \| f_\phi(s_i, a_i) - s'_i \|^2 \)
3. Backpropagate through \( f_\phi(s, a) \) to choose actions
Approach 1: Optimize over actions using model \( \max_{a_{t:t+H}} \sum_{t} r(s_t, a_t) \) “planning”

Algorithm:

1. Run some policy (e.g. random policy) to collect data \( \mathcal{D} = \{(s, a, s')_i\} \)
2. Learn model \( f_\phi(s, a) \) to minimize \( \sum_i \|f_\phi(s_i, a_i) - s'_i\|^2 \)
3. Iteratively sample action sequences, run through model \( f_\phi(s, a) \) to choose actions
Sampling-Based Optimization

Denote $\mathbf{A} := \mathbf{a}_t, \ldots, \mathbf{a}_{t+H}$

**Version 1: guess & check**

a. Sample many $\mathbf{A}_1, \ldots, \mathbf{A}_N$ from some distribution (e.g. uniform)

b. Choose $\mathbf{A}_i$ based on $\arg \max_i \sum_{t'=t}^{t+H} r(\mathbf{s}_{t'}, \mathbf{a}_{t'})$ Can we improve this distribution?

**Version 2: cross-entropy method**

a. Sample many $\mathbf{A}_1, \ldots, \mathbf{A}_N$ from $p(\mathbf{A})$

b. Evaluate $J(\mathbf{A}_i) = \sum_{t'=t}^{t+H} r(\mathbf{s}_{t'}, \mathbf{a}_{t'})$

c. Pick the elites $\mathbf{A}_{i_1}, \ldots, \mathbf{A}_{i_M}$ with the largest $J(\mathbf{A}_i)$, where $M < N$

d. Refit $p(\mathbf{A})$ to the elites $\mathbf{A}_{i_1}, \ldots, \mathbf{A}_{i_M}$
Sampling-Based Optimization

Version 1: guess & check

Version 2: cross-entropy method

“random shooting”

Pros: + fast, if parallelized
+ simple

Cons: - doesn’t scale to high-dimensions
(Including both $H$ and $|a|$)
Approach 1: Optimize over actions using model \( \max_{a_{t:t+H}} \sum_t r(s_t, a_t) \)

![Diagram with states and actions](image)

Algorithm:

1. Run some policy (e.g. random policy) to collect data \( \mathcal{D} = \{(s, a, s')_i\} \)
2. Learn model \( f_\phi(s, a) \) to minimize \( \sum_i \|f_\phi(s_i, a_i) - s'_i\|^2 \)
3. Iteratively sample action sequences, run through model \( f_\phi(s, a) \) to choose actions
   (e.g. with random shooting or cross-entropy method)
How can this approach fail?

1. Run some policy (e.g. random policy) to collect data $\mathcal{D} = \{(s, a, s')_i\}$
2. Learn model $f_\phi(s, a)$ to minimize $\sum_i \|f_\phi(s, a_i) - s'_i\|^2$
3. Iteratively sample action sequences, run through model $f_\phi(s, a)$ to choose actions

Data distribution mismatch

$p_{\pi_0}(s) \neq p_{\pi_f}(s)$

Going right means that we can go higher!

**Thought Exercise:** How might you alleviate this issue?
Approach 1: Optimize over actions using model

\[
\max_{a_{t:t+H}} \sum_t r(s_t, a_t)
\]

Algorithm:

1. Run some policy (e.g. random policy) to collect data \( \mathcal{D} = \{(s, a, s')_i\} \)
2. Learn model \( f_\phi(s, a) \) to minimize \( \sum_i \|f_\phi(s_i, a_i) - s'_i\|^2 \)
3. Iteratively sample action sequences, run through model \( f_\phi(s, a) \) to choose actions
4. Execute planned actions, appending visiting tuples \( (s, a, s') \) to \( \mathcal{D} \)
Revisiting the cliff

1. Run some policy (e.g. random policy) to collect data $\mathcal{D} = \{(s, a, s')_i\}$
2. Learn model $f_\phi(s, a)$ to minimize $\sum_i \|f_\phi(s, a_i) - s'_i\|^2$
3. Iteratively sample action sequences, run through model $f_\phi(s, a)$ to choose actions
4. Execute planned actions, appending visiting tuples $(s, a, s')$ to $\mathcal{D}$

Going right means that we can go higher!

**Final policy:** go to the top and stop.
Can we do better?

open-loop vs. closed-loop planning
Approach 2: Plan & replan using model

*model-predictive control (MPC)*

1. run base policy $\pi_0(a_t|s_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(s, a, s')_i\}$
2. learn model $f_\phi(s, a)$ to minimize $\sum ||f_\phi(s_i, a_i) - s'_i||^2$
3. use model $f_\phi(s, a)$ to optimize action sequence
4. execute the first planned action, observe resulting state $s'$
5. append $(s, a, s')$ to dataset $\mathcal{D}$

+ replan to correct for model errors
- compute intensive
So far: Planning with learned models

1. Can *plan* $a_1$, ..., $a_H$ with *gradient-based* or *sampling-based* optimization
2. *Update the model* using data collected with planning
3. *Replan* periodically to help account for mistakes.

+ Simple
+ Easy to plug in different goals / rewards (possibly even at test time!)

- Compute intensive at test time
- Only practical for short-horizon problems (or very shaped reward functions)

Why only short horizons?
(a) too compute expensive to make long plans
(b) model is not accurate for long horizons

Can we *train a policy* using a learned model?
Model-based policy optimization

Option 1: Distill planner’s actions into a policy
(i.e. train policy to match actions taken by planner)

+ no longer compute intensive at test time
- still limited to short-horizon problems

How might we solve longer-horizon problems using a model?

1. Plan with terminal value function
2. Augment model-free RL methods with data from model

Let’s focus on #2
Model-based policy optimization

Key idea: augment data with model-simulated roll-outs.

Example real trajectory

How to augment?
- generate full trajectories from initial states?
  - model may not be accurate for long horizons
- generate \textit{partial trajectories} from initial states?
  - may not get good coverage of later states
Model-based policy optimization

Key idea: augment data with model-simulated roll-outs.

Example real trajectory

How to augment?

- generate full trajectories from initial states?
- model may not be accurate for long horizons
- generate partial trajectories from initial states?
- may not get good coverage of later states
- generate partial trajectories from all states in the data
Model-based policy optimization

Key idea: augment data with model-simulated roll-outs.

Full algorithm

1. Collect data using current policy $\pi_\phi$, add to $D_{env}$
2. Update model $p_\theta(s'|s, a)$ using $D_{env}$
3. Collect synthetic roll-outs using $\pi_\phi$ in model $p_\theta$ from states in $D_{env}$; add to $D_{model}$
4. Update policy $\pi$ (and critic $Q$) using $D_{model}$

Notes:
- compatible with variety of model-free RL methods (step 4)
- could additionally use $D_{env}$ in policy update
When to use model-based RL?

+ Models are immensely useful if easy to learn
+ Model can be trained without reward labels (self-supervised)
+ Model is somewhat task-agnostic (can sometimes be transferred across rewards)
- Models don’t optimize for task performance
- Sometimes harder to learn than a policy

Whether to use a model depends on how hard it is to learn!
The plan for today

1. A brief primer on sampling-based optimization
2. Model-based reinforcement learning
   a. How to get a good dynamics model?
   b. How to use a (learned) dynamics model?
3. Case study in dexterous robotic manipulation
Case study: Model-based RL for dexterous manipulation

Deep Dynamics Models for Learning Dexterous Manipulation

Anusha Nagabandi, Kurt Konolgie, Sergey Levine, Vikash Kumar
Google Brain

September 2019

Still one of the most impressive results with five-fingered hands!
Case study: Model-based RL for dexterous manipulation

State space: hand & object positions

Action space: controlling 5-fingered hand (24 DoF)

Reward: track target object trajectory + penalty for dropping

Model: Ensemble of 3 neural networks, each with 2 hidden layers of size 500

Planner: modified version of CEM optimizer softer reward-weighted mean & temporal smoothing on actions

Alternate between collecting ~30 trajectories with planner & updating model.
Case study: Model-based RL for dexterous manipulation

Simulated experiments

Model-free methods:

**SAC**: actor-critic method

**NPG**: policy gradient method

Model-based methods:

**PDDM**: proposed method

**MBPO**: RL with model-generated data

**PETS**: CEM-based planner

**Nagabandi et al.**: random shooting, no ensembles

More efficient than model-free methods

More performant than other model-based methods
Case study: Model-based RL for dexterous manipulation

Simulated ablations

- Need sufficiently large model
- Need at least 3 ensemble members
- Planning horizon trade-offs
- Modified CEM is crucial
Case study: Model-based RL for dexterous manipulation

Real-world dexterous control with ShadowHand

- Efficiency is key for fragile hardware
- Learns Baoding ball rotation in ~4 hours
- Ball is reset with another robot arm
The plan for today

1. A brief primer on sampling-based optimization
2. Model-based reinforcement learning
   a. How to get a good dynamics model?
   b. How to use a (learned) dynamics model?
3. Case study in dexterous robotic manipulation

Key learning goals:
- model-based RL methods, and how to implement them
- the key challenges arising in model-based reinforcement learning
- tradeoffs between different model-based RL approaches
Course reminders

- Project proposal due this Wednesday (graded fairly lightly — really for your benefit!)
- Homework 2 due next Wednesday (start early!)

Next time: Where do rewards come from? Can we learn them?