Model-Based Reinforcement Learning

CS 224R

Course reminders

- Homework 3 due next Friday, May 16
- Project milestone due Friday, May 23

High-Level Algorithm Recap

Offline

No policy collection

Online

Involves policy data collection

Offline imitation learning

Behavior cloning

Online imitation learning

DAgger

Requires expert data.
Doesn't need reward.

Offline RL

Constrain to actions in data

AWR, AWAC, IQL

Implicitly constrain via conservatism

CQL

Off-policy RL

Can reuse data from other policies

Replay buffer

DQN, SAC

Mult. grad steps

PPO, Imp. Sampling

On-policy RL

Only use data from curr. policy

REINFORCE / vanilla PG

Q-learning (critic only)

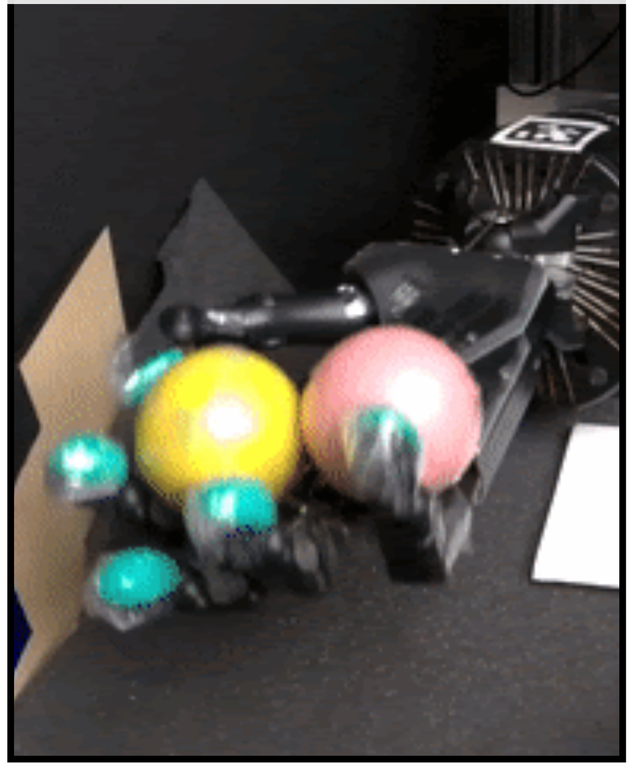
actor-critic (both)

policy gradient (actor only)

Requires more online data

The plan for today

How to get a robot to do this!



1. Model-based reinforcement learning
   a. Key idea
   b. How to learn a dynamics model? (in brief)
   c. How to use a learned dynamics model?
      i. Planning
      ii. Data generation
2. Case study in dexterous robotic manipulation

Key learning goals:

- How to learn and use dynamics models
- The key challenges and trade-offs arising in model-based RL

Can we learn a “simulator”?

i.e. predictive model of $\mathbf{s}_{t+1}$ given $\mathbf{s}_t$, $\mathbf{a}_t$

Robotics & physical systems

modeling physics of (directly observed) physical system

video prediction, conditioned on actions

Veo 2

Finance: stock market predictor

Games: rules of the game

May need to model other players —
Sometimes considered part of dynamics,
sometimes modeled separately (multi-agent RL)

Sora

May not need to learn a model! (e.g. chess)

Notional “learned simulator” algorithm

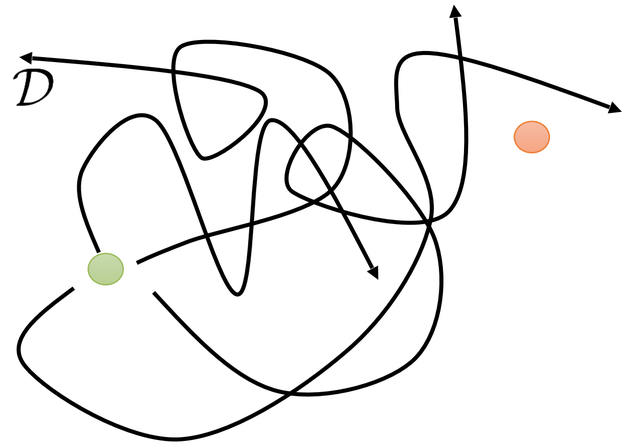
1. Get some data $\mathcal{D} = \{\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i\}$

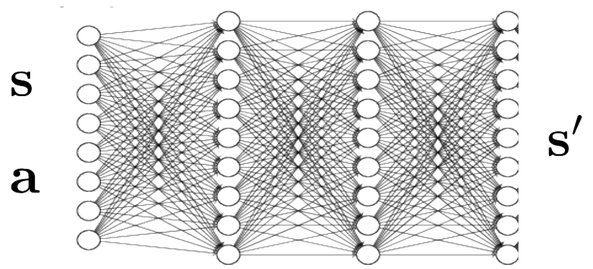
   collected using some base policy $\pi_0(\mathbf{a}|\mathbf{s})$

2. Learn “simulator” $f(\mathbf{s}, \mathbf{a}) = \mathbf{s}'$

   $$\min_f \sum_i ||f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i||^2$$

3. Run your favorite RL method inside “simulator” f
   - Using one of the RL methods we covered, or a planning method





Question: What might go wrong?

Notional “learned simulator” algorithm

1. Get some data $\mathcal{D} = \{\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i\}$

   collected using some base policy $\pi_0(\mathbf{a}|\mathbf{s})$

2. Learn “simulator” $f(\mathbf{s}, \mathbf{a}) = \mathbf{s}'$

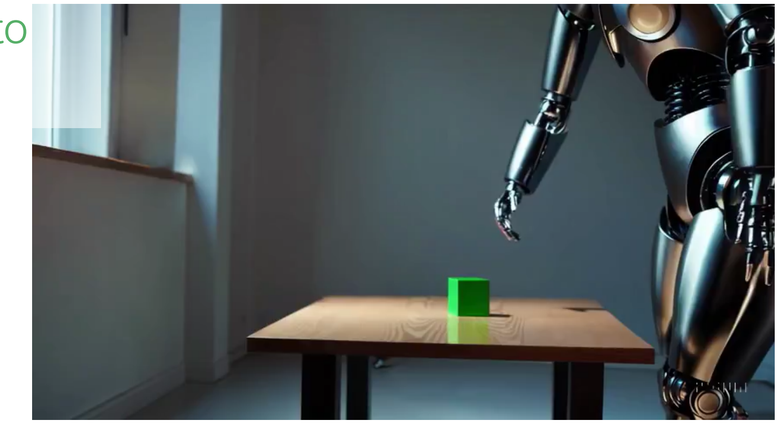
   $$\min_f \sum_i ||f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i||^2$$

3. Run your favorite RL method inside “simulator” f
   - Using one of the RL methods we covered, or a planning method

$\mathcal{D}$

Data coverage matters a lot!

Modeling is **domain dependent**. Some domains much easier to simulate than others!



A humanoid robot standing near the table with red, green and blue cubes on it performs a cube stacking task, with red in the bottom and blue on the top.

Even if we have a good simulator, this part isn’t necessarily easy!

Need to account for model inaccuracies.

The plan for today

1. Model-based reinforcement learning
   a. Key idea
   b. How to learn a dynamics model? (in brief)
   c. How to use a learned dynamics model?
      i. Planning
      ii. Data generation
2. Case study in dexterous robotic manipulation

How to learn a good dynamics model? (in brief)

- You know it already (e.g. some games)
- You approximately know most of it (e.g. certain physical models)
  - Can fit the unknown parameters using the data
- You don’t know it (almost all scenarios in practice)
  - Learn it end-to-end
  - Learn a (low-dim) state representation, then learn model over representations

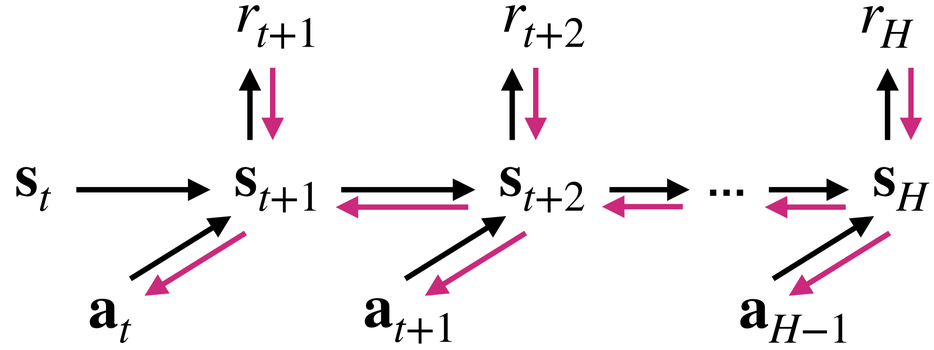
Note: Often also need to learn a reward model!

The plan for today

1. Model-based reinforcement learning
   a. Key idea
   b. How to learn a dynamics model? (in brief)
   c. How to use a learned dynamics model?
      i. Planning
      ii. Data generation
2. Case study in dexterous robotic manipulation

Approach 1: Optimize over actions using model $\max_{\mathbf{a}_{t:t+H}} \sum_t r(\mathbf{s}_t, \mathbf{a}_t)$

"planning"



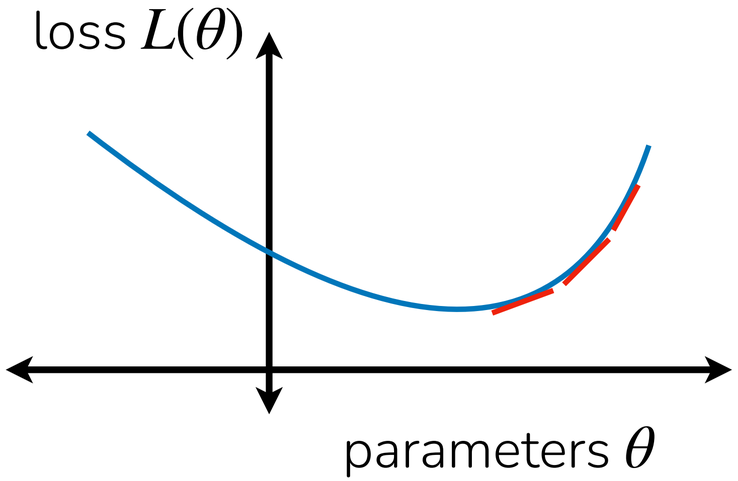
Approach 1a:
via *backpropagation*
(i.e. *gradient-based* optimization)

Algorithm:

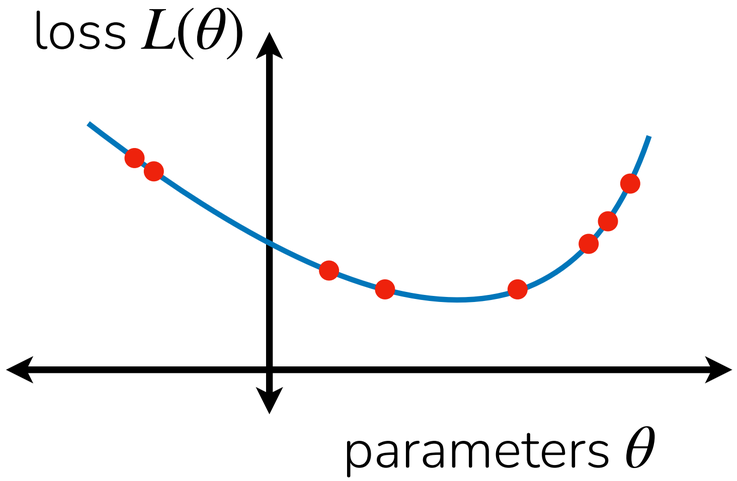
1. Run some policy (e.g. random policy) to collect data $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. Learn model $f_\phi(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. Backpropagate through $f_\phi(\mathbf{s}, \mathbf{a})$ to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

Aside: Gradient-based vs. sampling-based optimization

Gradient-based (1st order)



Sampling-based (0th order)



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

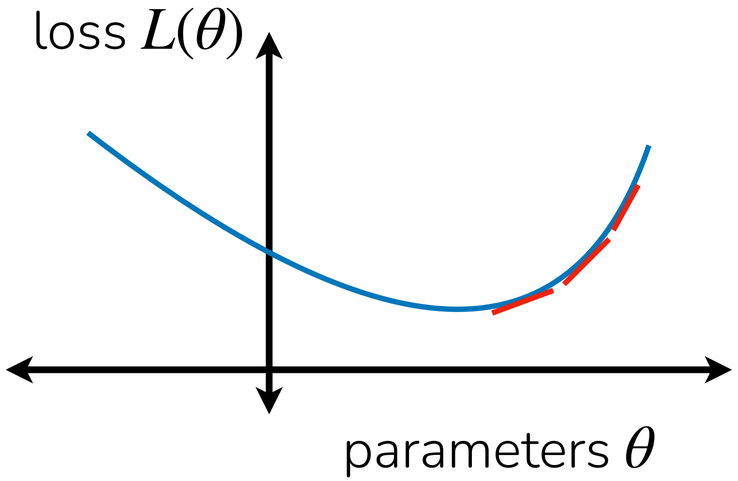
repeat

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 $\theta_1, \ldots, \theta_k$

Eventually return θ_1

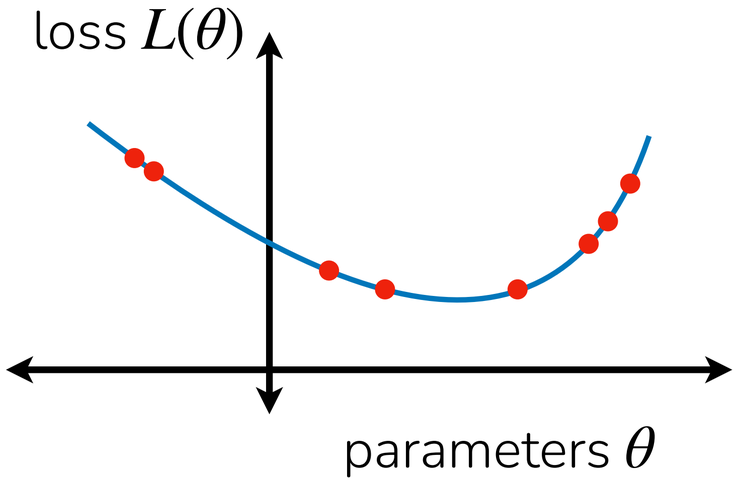
Aside: Gradient-based vs. sampling-based optimization

Gradient-based (1st order)



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

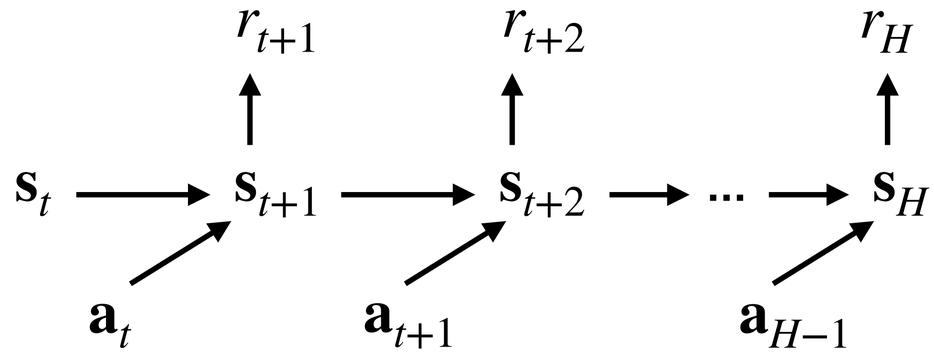
Sampling-based (0th order)



+ parallelizable
+ requires no gradient information
- scales poorly to high dimensions

Approach 1: Optimize over actions using model $\max_{\mathbf{a}_{t:t+H}} \sum_t r(\mathbf{s}_t, \mathbf{a}_t)$

"planning"



Approach 1b:
via *sampling*
(i.e. *gradient-free* optimization)

Algorithm:

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

Sampling-Based Optimization

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

Version 1: Guess & check

"random shooting"

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 $\mathbf{A}_{i_1}, \ldots, \mathbf{A}_{i_M}$

+ fast, if parallelized

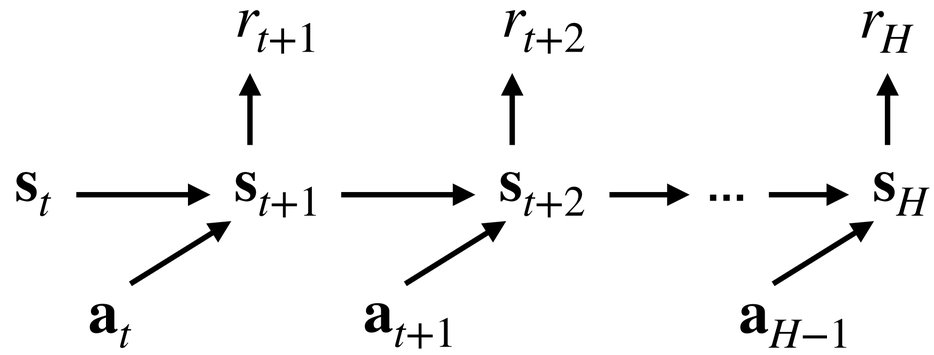
+ simple

- doesn't scale to high dimensions

(including both H and $|\mathbf{a}|$)

Approach 1: Optimize over actions using model $\max_{\mathbf{a}_{t:t+H}} \sum_t r(\mathbf{s}_t, \mathbf{a}_t)$

"planning"



Approach 1b:
via *sampling*
(i.e. *gradient-free* optimization)

Algorithm:

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

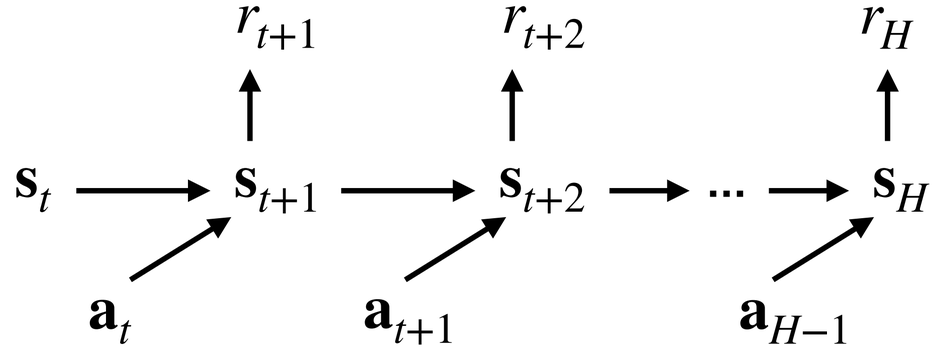
Data distribution mismatch

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

Going right means that we can go higher!

Thought Exercise: How might you alleviate this issue?

Approach 1: Optimize over actions using model $\max_{\mathbf{a}_{t:t+H}} \sum_t r(\mathbf{s}_t, \mathbf{a}_t)$

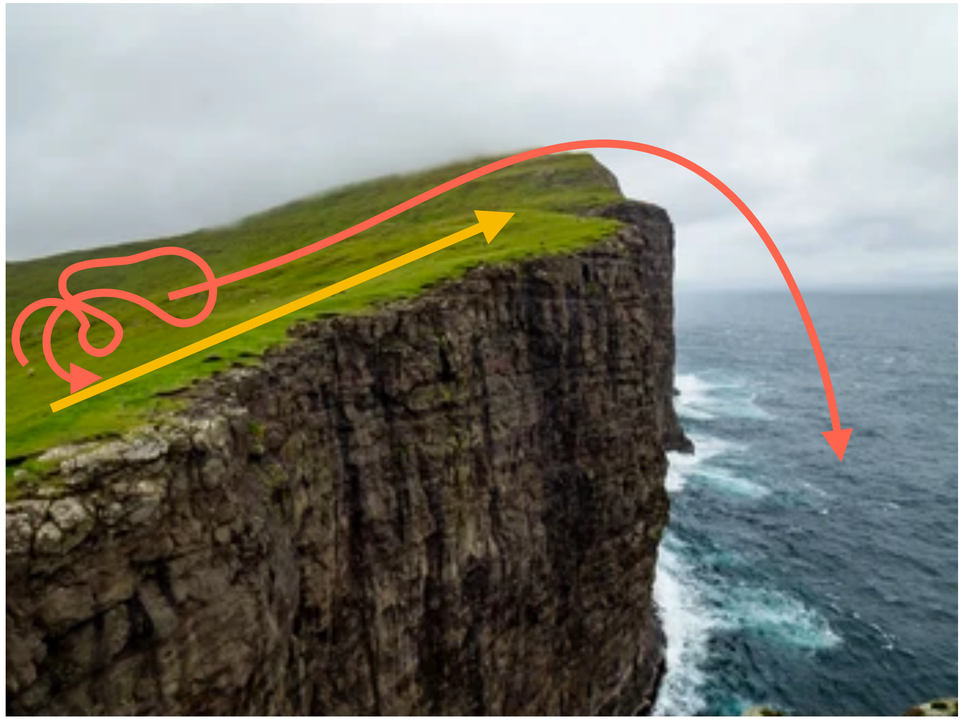


Approach 1b:
via *sampling*
(i.e. *gradient-free* optimization)

Algorithm:

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

Revisiting the cliff



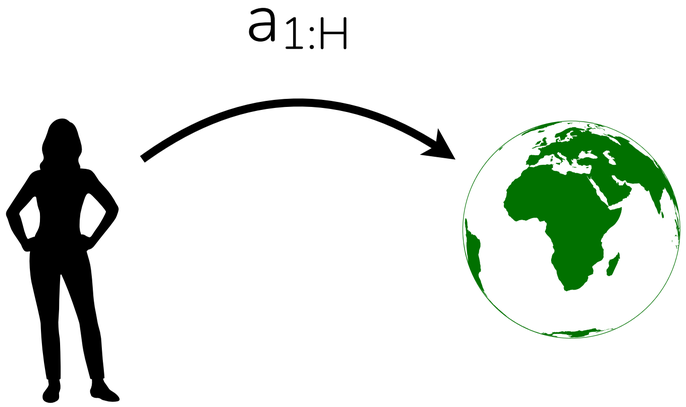
1. Run some policy (e.g. random policy) to collect data $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. Learn model $f_\phi(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}_i'\|^2$
3. Iteratively sample action sequences, run through model $f_\phi(\mathbf{s},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

Going right means that we can go higher!

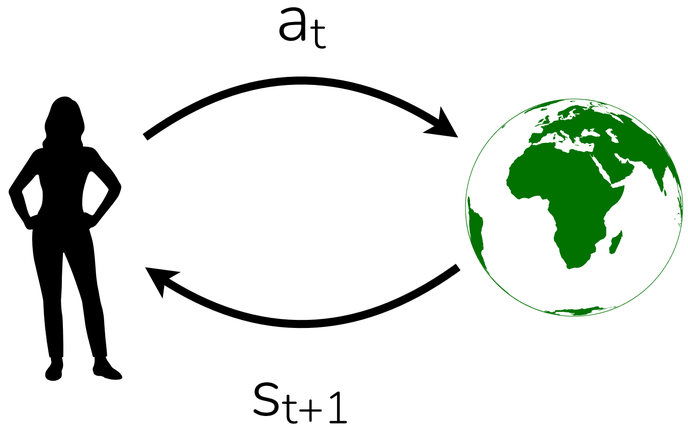
Final policy: go to the top and stop.

Can we do even better?

Open-loop



Closed-loop



When might it be bad to be open-loop?

Ideas on how to make planning closed-loop?

Approach 2: Plan & replan using model

model-predictive control (MPC)

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

+ replan to correct for model errors

- compute intensive

So far: Planning with learned models

1. Can *plan* $\mathbf{a_1}$, …, $\mathbf{a_H}$ 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?

The plan for today

1. Model-based reinforcement learning
   a. Key idea
   b. How to learn a dynamics model? (in brief)
   c. How to use a learned dynamics model?
      i. Planning
      ii. Data generation
2. Case study in dexterous robotic manipulation

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?

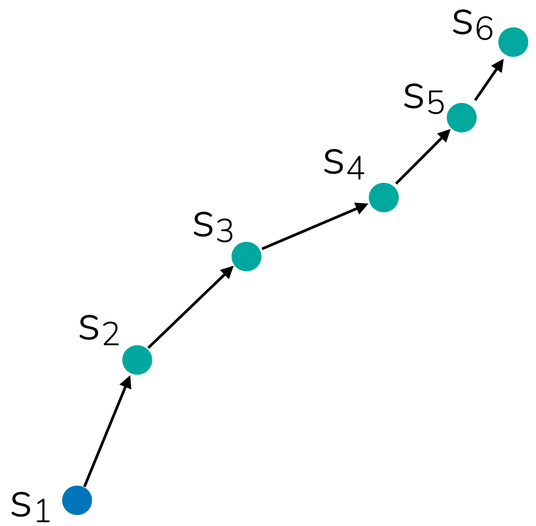
1. Plan with terminal value function
2. Augment model-free RL methods with data from model

Let's focus on #2

Model-based policy optimization

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

Example real trajectory



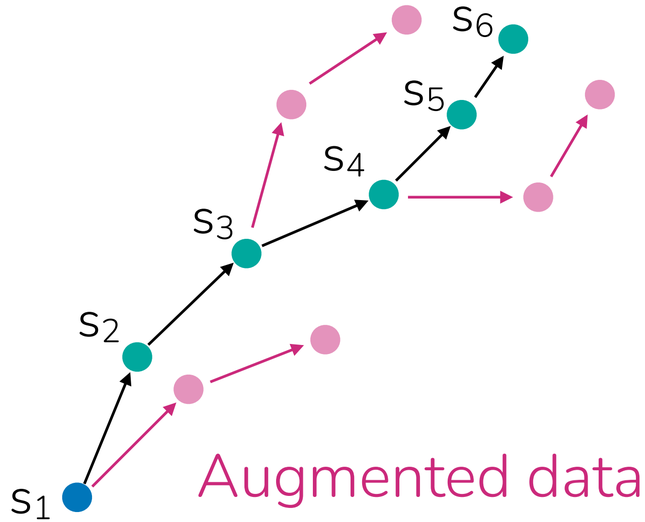
How to augment?

- generate full trajectories from initial states?
  - model may not be accurate for long horizons
- generate *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 π_ϕ, add to D_{env}
2. Update model $p_\theta(s' \mid s, a)$ using D_{env}
3. Collect synthetic roll-outs using π_ϕ in model p_θ from states in D_{env}; add to D_{model}
4. Update policy π (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?

Big upsides and big downsides

+ Immensely useful, far more data efficient if model is easy to learn
+ Model can be trained on data without reward labels (fully 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
- Another thing to train, more hyperparameters, more compute intensive

Whether to use a model depends on how hard it is to learn!

Other kinds of models

So far: modeling $p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$

Many alternatives

- inverse model $p(\mathbf{a}_t \mid \mathbf{s}_t, \mathbf{s}_{t+1})$
- multi-step inverse model $p(\mathbf{a}_t \mid \mathbf{s}_t, \mathbf{s}_{t+n})$ or $p(\mathbf{a}_{t:t+n} \mid \mathbf{s}_t, \mathbf{s}_{t+n})$
- future prediction without actions $p(\mathbf{s}_{t+1:t+n} \mid \mathbf{s}_t)$
- video interpolation $p(\mathbf{s}_{t+1:t+n} \mid \mathbf{s}_t, s_{t+n+1})$
- transition distribution $p(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$

These kinds of models can be useful too! Have various purposes.

The plan for today

1. Model-based reinforcement learning
    a. Key idea
    b. How to learn a dynamics model? (in brief)
    c. How to use a learned dynamics model?
        i. Planning
        ii. Data generation
2. **Case study in dexterous robotic manipulation**

Case study: Model-based RL for dexterous manipulation

Deep Dynamics Models for Learning Dexterous Manipulation

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

September 2019

Why this paper?

Still one of the most impressive results with five-fingered hands!

Nice comparisons on what is important for good performance

Case study: Model-based RL for dexterous manipulation

State space: hand & object positions

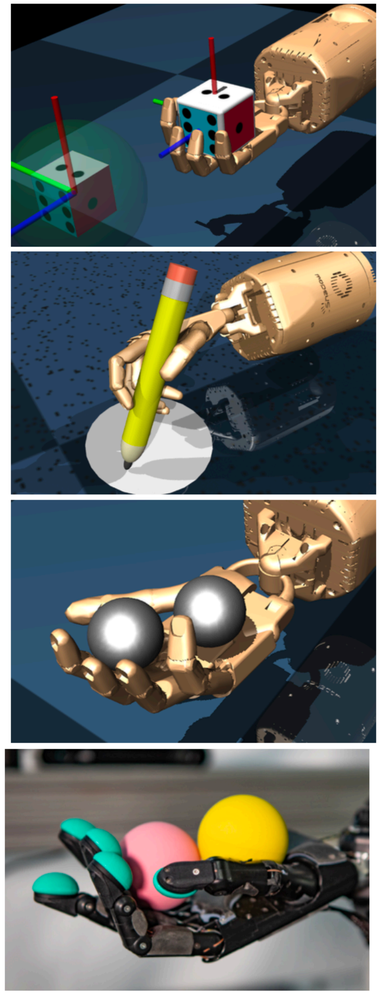
Action space: controlling 5-fingered hand (24 DoF)

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

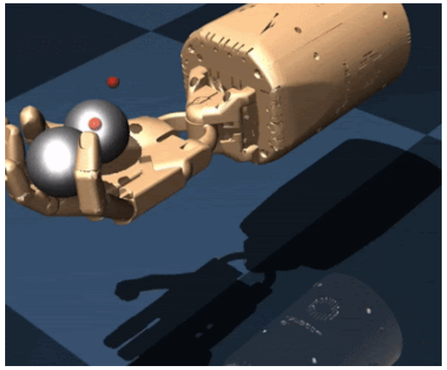
Model-based methods:

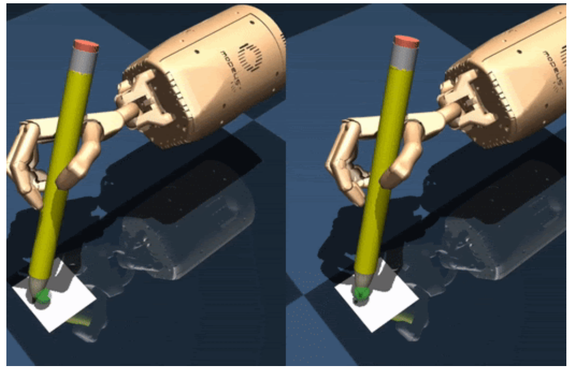
PDDM: proposed method

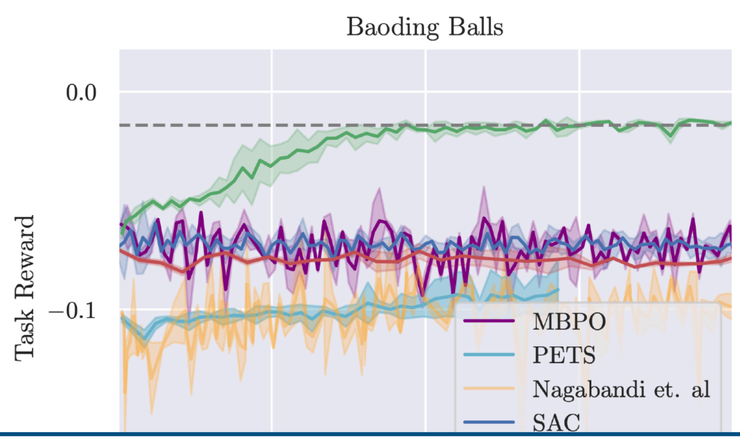
MBPO: RL with model-generated data

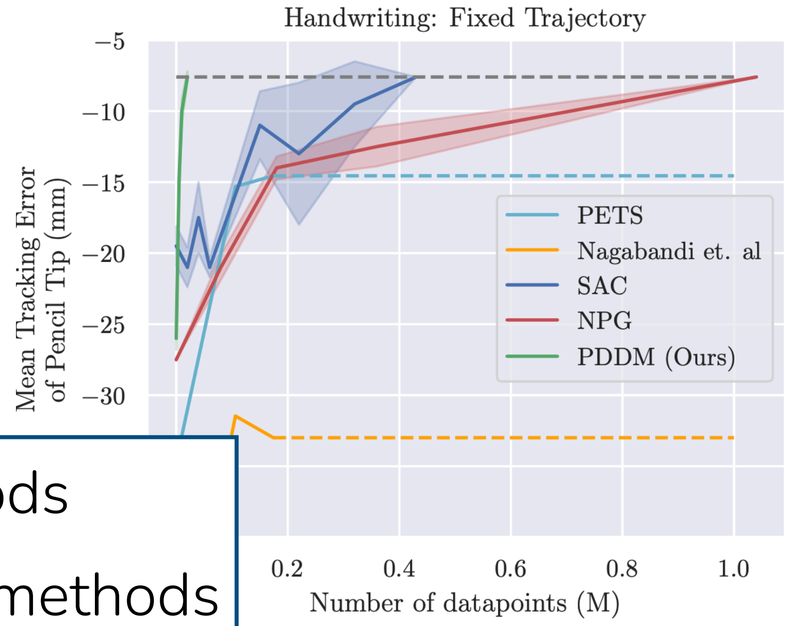
PETS: CEM-based planner

Nagabandi et al.: random shooting, no ensembles







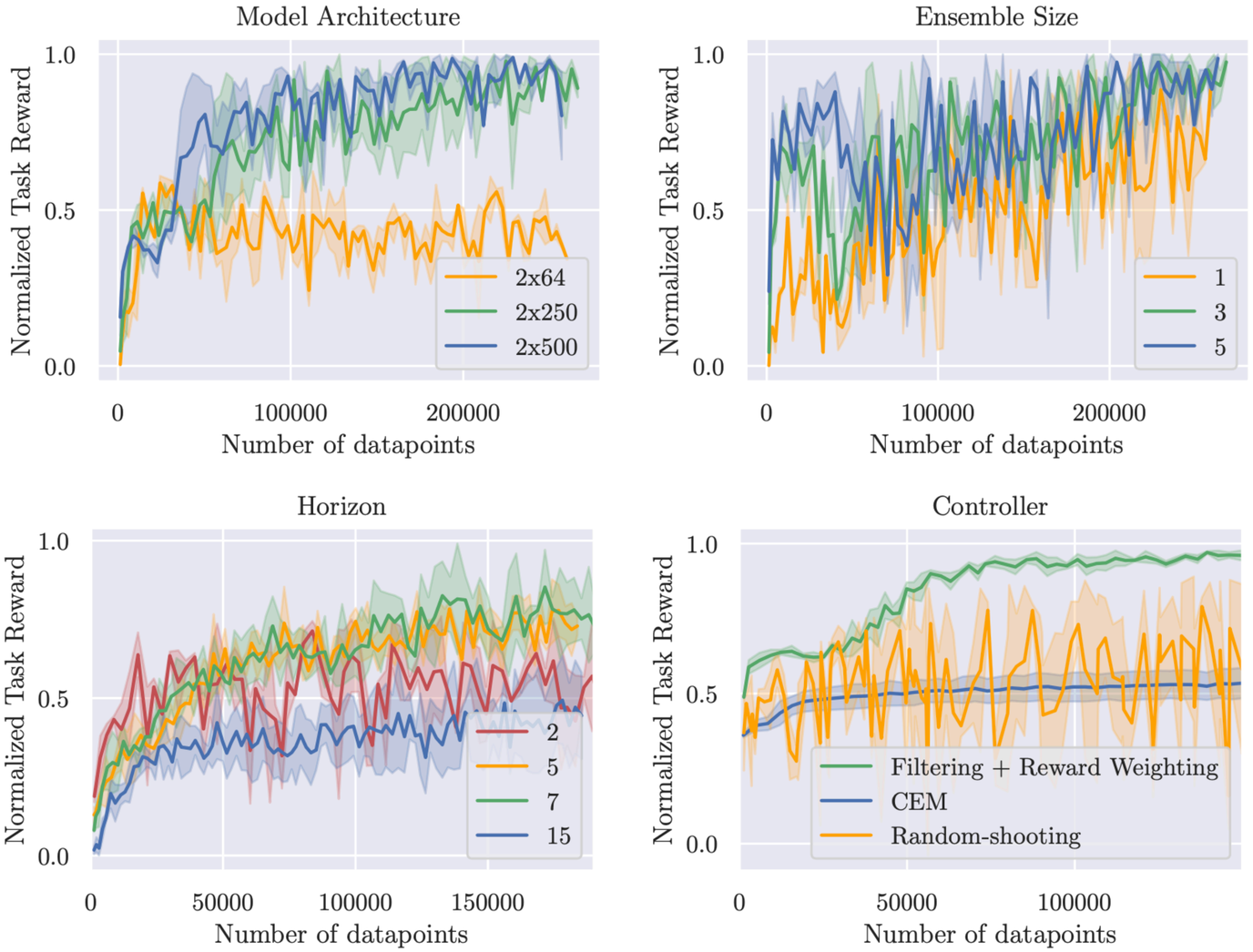


More efficient than model-free methods

More performant than other model-based methods

Case study: Model-based RL for dexterous manipulation

Simulated ablations

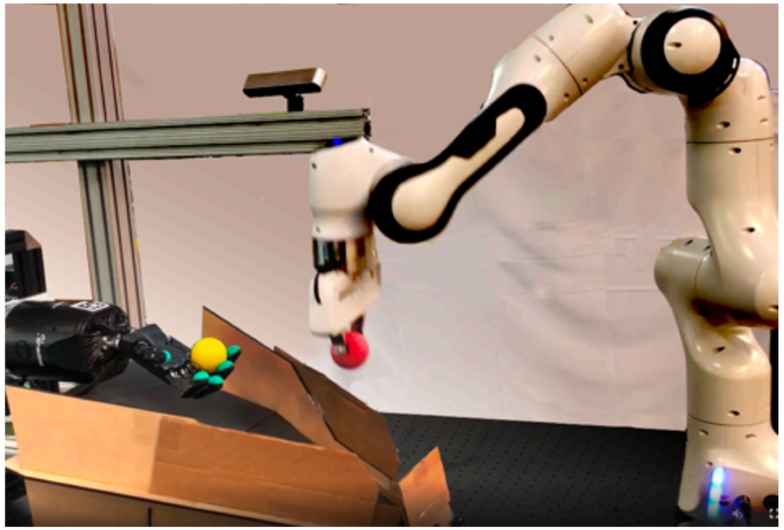


- Need sufficiently large model
- Need at least 3 ensemble members
- Planning horizon trade-offs
- Modified CEM is crucial

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. Model-based reinforcement learning
   a. Key idea
   b. How to learn a dynamics model? (in brief)
   c. How to use a learned dynamics model?
      i. Planning
      ii. Data generation
2. Case study in dexterous robotic manipulation

Course reminders

- Homework 3 due next Friday, May 16
- Project milestone due Friday, May 23

Next time

RL for multiple tasks and goals (e.g. learning generalist policies)