Model-Based Reinforcement Learning

CS 224R

Course reminders

- Homework 3 due this Friday at 9 pm PT
- Midterm review session Monday at 4:30 pm

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

Only use offline data

Supervise policy on data actions

AWR, AWAC, IQL

Learn V for better policy w/ asymmetric loss

IQL

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

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. Data generation
      ii. Test-time planning
2. Summary & outlook

How to get a robot to do this!

Key learning goal: How to best leverage learned dynamics models

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)

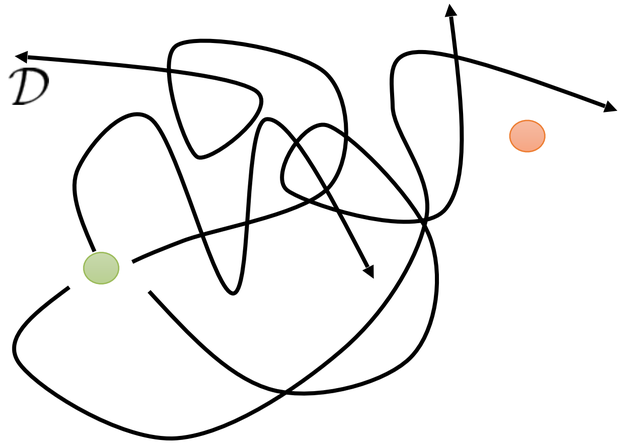
Notional “learned simulator” algorithm

1. Get some data $\mathcal{D} = \{(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})\}$ collected by some policy

2. Learn a “simulator” $p_\theta(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$

$$\min_\theta \sum_{(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) \in \mathcal{D}} -\log p_\theta(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

3. Run your favorite RL algorithm inside the “simulator” p_θ
   - 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 $\mathcal{D} = \{(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})\}$ collected by some policy

2. Learn a “simulator” $p_\theta(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$

$$\min_\theta \sum_{(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) \in \mathcal{D}} -\log p_\theta(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

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

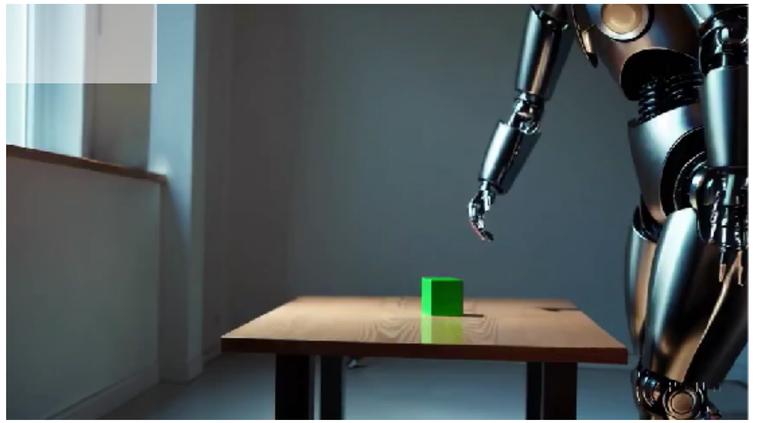
Even if we have a good simulator, this part isn’t necessarily easy!

Need to account for model inaccuracies.

$\mathcal{D}$

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.

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. Data generation
      ii. Test-time planning
2. Summary & outlook

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!

***Aside**: This kind of model called many things - dynamics model, simulator, world model, 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. Data generation
      ii. Test-time planning
2. Summary & outlook

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; add to D_{model}
4. Update policy π (and critic Q) using $D_{model} \bigcup D_{env}$

Note: - compatible with variety of model-free RL methods (step 4)

algorithm called “Dyna”

How can this approach fail?

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; add to D_{model}
4. Update policy π (and critic Q) using $D_{model} \bigcup D_{env}$

Data distribution mismatch

$$p_{\pi_\phi}(\mathbf{s}) \neq p_{\pi_{\phi'}}(\mathbf{s})$$

Going right means that we can go higher!

How to deal with this?

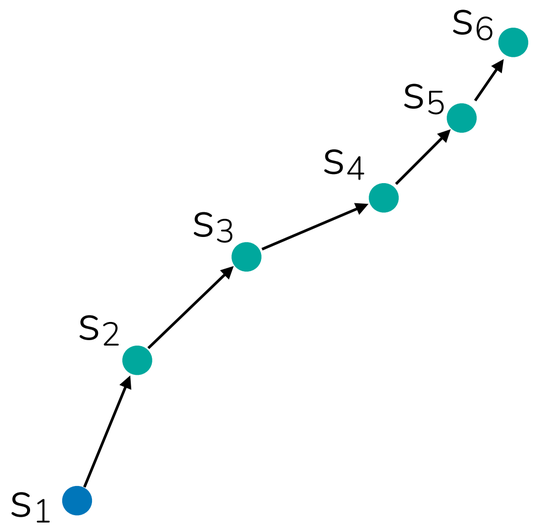
Can do same as before:

- collect data from $\pi_{\phi'}$ in subsequent iterations
- constrain policy to not change too much

But, there are other things we can do in this case!

Model-based RL: Handling distribution shift

Example real trajectory

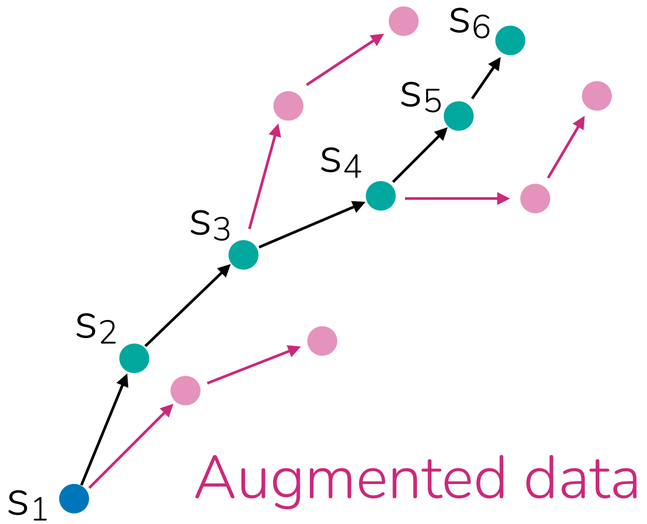


How to augment?

- 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 RL: Handling distribution shift

Example real trajectory



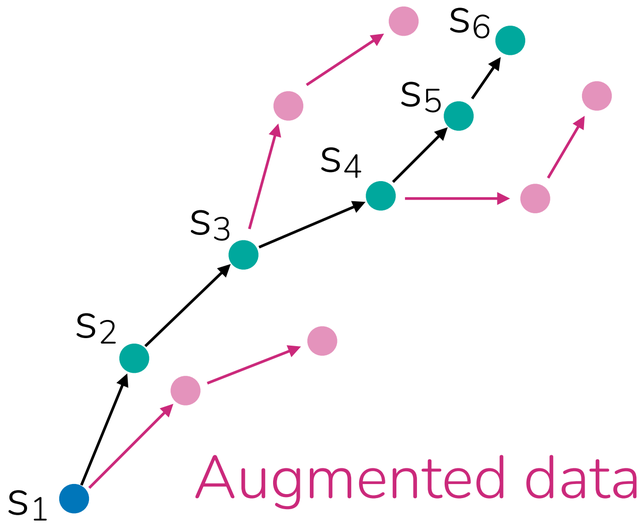
How to augment?

- 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 💡

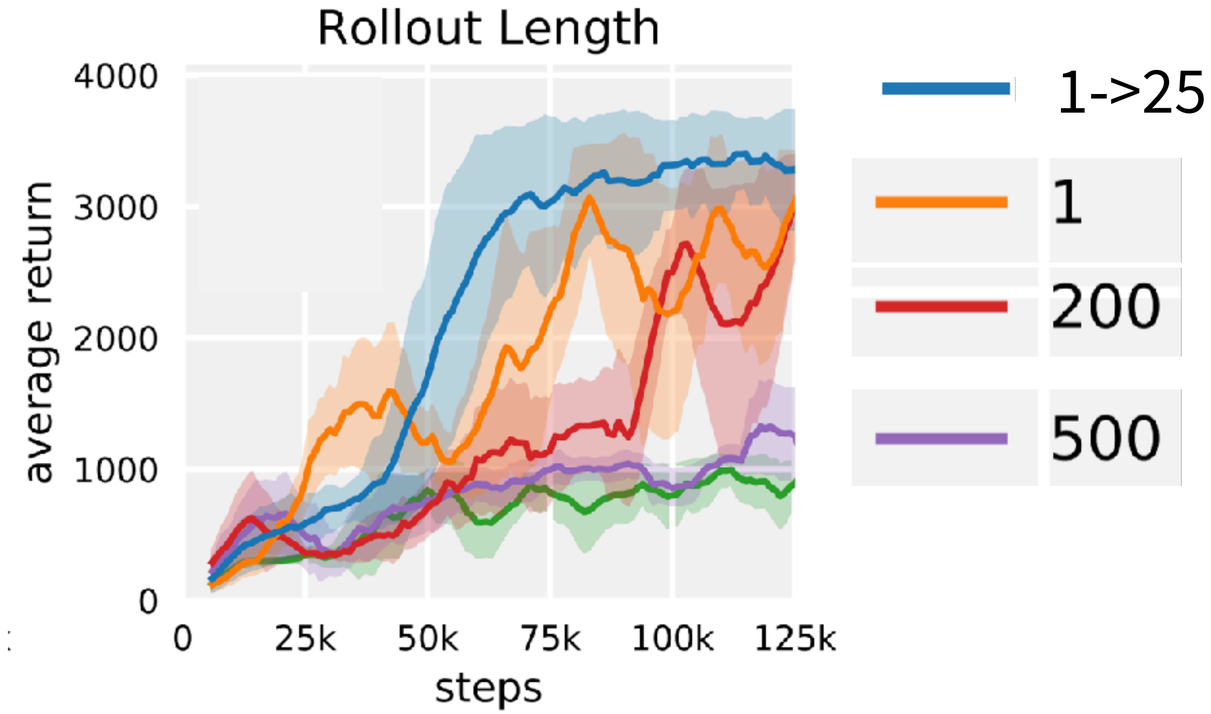
  -> don't need model to be accurate for long horizons!

Model-based RL: Handling distribution shift

Example real trajectory



How long should the partial trajectories be?



plot from: Janner et al. MBPO. '19

Model-based RL: Handling distribution shift

First idea: Use partial model roll-outs, starting from real states in replay offer

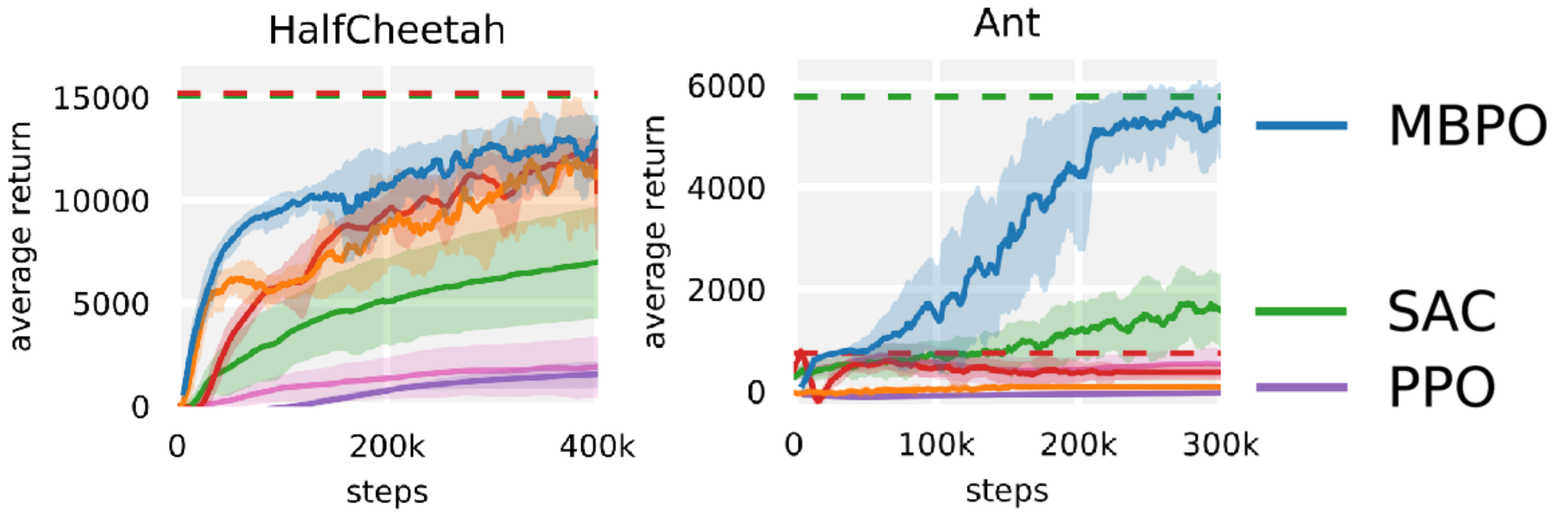
Second idea: Use an ensemble of models -> errors will average out

Model-based RL: V2

Full algorithm

1. Collect data using current policy π_ϕ, add to D_{env}
2. Update each model $p_\theta^i(s' \mid s, a)$ using minibatch B^i sampled from D_{env}
3. Collect partial rollouts using π_ϕ in model p_θ^i starting from $s \in D_{env}$; add to D_{model}
4. Update policy π (and critic Q) using $D_{model} \bigcup D_{env}$

How does this compare to PPO and SAC?



Janner et al. MBPO. '19

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. Data generation
      ii. Test-time planning
2. Summary & outlook

Another way to use models

So far at test time:

Observe state s.

Take action $a \sim \pi(\cdot \mid s)$

Observe next state s'

…

What if we are in a complicated, or new situation?

Rough sketch:

1. Consider some action candidates
2. Imagine outcome of those actions
3. Pick the action with the best outcome.

Another way to use models

Rough sketch:

1. Consider some action candidates
2. Imagine outcome of those actions
3. Pick the action with the best outcome.

<- What actions to consider? How many candidates? How long of action sequences?

<- How to measure if an outcome is good?

Planning V1

agent currently at $\mathbf{s}_t$

1. Consider some action candidates — sample $i = 1 \ldots N$ action sequences $\mathbf{a}^i_{t:t+H}$
2. Imagine outcome of those actions — generate $\hat{\mathbf{s}}^i_{t+1:t+H+1}$ by rolling actions out in p_θ
3. Pick the action with the best outcome.

execute action sequence with highest rewards

$$i^\star \leftarrow \arg\max_i \sum_{t'=t}^{t+H} r(\hat{\mathbf{s}}^i_{t'}, \mathbf{a}^i_{t'})$$

Important note: *No separate neural network* π

This process is the policy!

Planning V1

agent currently at $\mathbf{s}_t$

1. Consider some action candidates — sample $i = 1 \ldots N$ action sequences $\mathbf{a}^i_{t:t+H}$
2. Imagine outcome of those actions — generate $\hat{\mathbf{s}}^i_{t+1:t+H+1}$ by rolling actions out in p_θ
3. Pick the action with the best outcome.

sample better actions

execute action sequence with highest rewards

$$i^\star \leftarrow \arg\max_i \sum_{t'=t}^{t+H} r(\hat{\mathbf{s}}^i_{t'}, \mathbf{a}^i_{t'})$$

Note 1: Can choose to execute *only* $\mathbf{a}^{i^\star}_t$ and replan at $\mathbf{s}_{t+1}$

"receding horizon planning"

Note 2: Can iteratively optimize for better action sequences.

Planning V1

agent currently at $\mathbf{s}_t$

1. Consider some action candidates — sample $i = 1 \ldots N$ action sequences $\mathbf{a}^i_{t:t+H}$
2. Imagine outcome of those actions — generate $\hat{\mathbf{s}}^i_{t+1:t+H+1}$ by rolling actions out in p_θ
3. Pick the action with the best outcome.

execute action sequence with highest rewards

$$i^\star \leftarrow \arg\max_i \sum_{t'=t}^{t+H} r(\hat{\mathbf{s}}^i_{t'}, \mathbf{a}^i_{t'})$$

If planning horizon H is short, planning will be myopic

+ This can work well for short-horizon problems!

- For long-horizons: requires accurate long-horizon model + substantial inference compute

Planning ~~V1~~ V2 (long horizon)

agent currently at $\mathbf{s}_t$

1. Consider some action candidates — sample $i = 1 \ldots N$ action sequences $\mathbf{a}^i_{t:t+H}$
2. Imagine outcome of those actions — generate $\hat{\mathbf{s}}^i_{t+1:t+H+1}$ by rolling actions out in p_θ
3. Pick the action with the best outcome.

execute action $\mathbf{a}^{i^\star}_t$ with highest rewards

$$i^\star \leftarrow \arg\max_i \sum_{t'=t}^{t+H} r(\hat{\mathbf{s}}^i_{t'}, \mathbf{a}^i_{t'})$$

Planning ~~V1~~ V2 (long horizon)

agent currently at $\mathbf{s}_t$

1. Consider some action candidates — sample $i = 1 \ldots N$ action sequences $\mathbf{a}^i_{t:t+H}$
2. Imagine outcome of those actions — generate $\hat{\mathbf{s}}^i_{t+1:t+H+1}$ by rolling actions out in p_θ
3. Pick the action with the best outcome.

execute action $\mathbf{a}_t^{i^\star}$ with highest rewards

$$i^\star \leftarrow \arg\max_i \sum_{t'=t}^{t+H} r(\hat{\mathbf{s}}_{t'}^{\,i}, \mathbf{a}^i_{t'}) + \hat{V}(\hat{\mathbf{s}}_{t+H+1})$$

- Additional complexity of learning a value function

\+ Only need to learn V, not Q

\+ Applicable to long-horizon problems!

Full RL Algorithm with Planning

1. Collect data using current planner & model, add to D_{env}
2. Update model $p_\theta(s' \mid s, a)$ using D_{env}
3. Optionally, update $\hat{V}_\psi$ with Monte Carlo or TD learning using D_{env}
4. Optionally, update a policy using D_{env}, if planner uses it to sample actions

Planner at $\mathbf{s}_t$

sample $i = 1 \ldots N$ action sequences $\mathbf{a}^i_{t:t+H}$

generate $\hat{\mathbf{s}}^i_{t+1:t+H+1}$ by rolling actions out in p_θ

execute action $\mathbf{a}^{i^\star}_t$ with highest rewards

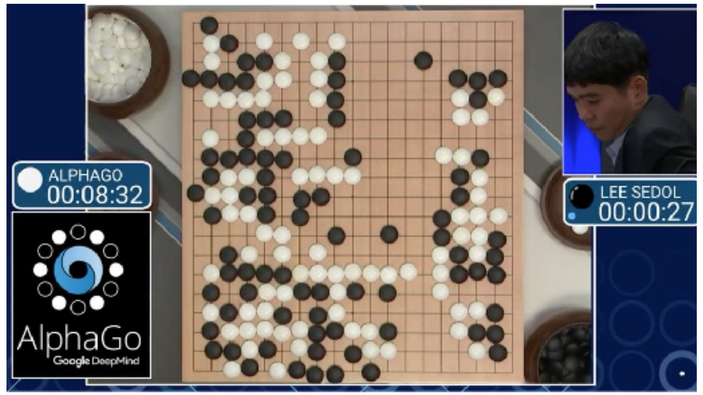
$$i^\star \leftarrow \arg\max_i \sum_{t'=t}^{t+H} r(\hat{\mathbf{s}}^i_{t'}, \mathbf{a}^i_{t'}) + \hat{V}(\hat{\mathbf{s}}_{t+H+1})$$

RL with Planning Examples

Nagabandi et al. 2019

- **Model**: Learned dynamics model: ensemble of three neural nets
- **Planning:**
  - Short-horizon with shaped reward (no value function)
  - Iterative sampling-based optimization (“cross-entropy method”)
- **Algorithm**: Alternate between with planner & updating model

RL with Planning Examples



AlphaGo

+ many other games!!

- **Model**: Known rules of the game + other agent's move
  - Play against itself or a pool of past versions of itself
- **Planning:**
  - horizon of 1-40 moves + terminal value function
  - policy network narrows down action space
  - heuristics to balance exploration & exploitation
- **Algorithm**:
  - Value function trained with Monte Carlo regression
  - Policy trained to match actions selected by planner
  - Trained for 3 weeks (AlphaGo), 40 days (AlphaGo Zero)

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. Data generation
      ii. Test-time planning
2. **Summary & outlook**

Summary of Model-Based RL

Key idea: learn a simulator of the dynamics $p_\theta(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$

Any method that learns this is a "model-based RL" algorithm

Use it to simulate additional data

Simulate data starting from all states seen in the data

Use it to do planning (i.e. lookahead)

Use with value functions for long-horizon planning

Mitigating model errors

- Use short synthetic roll-outs
- Ensembles of models can help average out errors

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

Only use offline data

Supervise policy on data actions

AWR, AWAC, IQL

Learn V for better policy w/ asymmetric loss

IQL

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

When to use model-based RL?

Big upsides and big downsides

+ Models are 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.

Course reminders

- Homework 3 due this Friday at 9 pm PT
- Midterm review session Monday at 4:30 pm

Next time

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