Model-Based Reinforcement Learning

CS 285

Instructor: Sergey Levine
UC Berkeley
Today’s Lecture

1. Basics of model-based RL: learn a model, use model for control
   • Why does naïve approach not work?
   • The effect of distributional shift in model-based RL

2. Uncertainty in model-based RL

3. Model-based RL with complex observations

4. Next time: **policy learning** with model-based RL
   • Goals:
     • Understand how to build model-based RL algorithms
     • Understand the important considerations for model-based RL
     • Understand the tradeoffs between different model class choices
Why learn the model?

If we knew \( f(s_t, a_t) = s_{t+1} \), we could use the tools from last week.

(or \( p(s_{t+1}|s_t, a_t) \) in the stochastic case)

So let’s learn \( f(s_t, a_t) \) from data, and then plan through it!

model-based reinforcement learning version 0.5:

1. run base policy \( \pi_0(a_t|s_t) \) (e.g., random policy) to collect \( D = \{ (s, a, s')_i \} \)

2. learn dynamics model \( f(s, a) \) to minimize \( \sum_i \| f(s_i, a_i) - s'_i \|^2 \)

3. plan through \( f(s, a) \) to choose actions
Does it work? Yes!

- Essentially how system identification works in classical robotics
- Some care should be taken to design a good base policy
- Particularly effective if we can hand-engineer a dynamics representation using our knowledge of physics, and fit just a few parameters
Does it work? No!

1. run base policy \( \pi_0(a_t|s_t) \) (e.g., random policy) to collect \( D = \{(s_t, a, s'_t)\} \)
2. learn dynamics model \( f(s, a) \) to minimize \( \sum_i \| f(s_i, a_i) - s'_i \|^2 \)
3. plan through \( f(s, a) \) to choose actions

\[
p_{\pi_f}(s_t) \neq p_{\pi_0}(s_t)
\]

- Distribution mismatch problem becomes exacerbated as we use more expressive model classes
Can we do better?

can we make $p_{\pi_0}(s_t) = p_{\pi_f}(s_t)$?

where have we seen that before? need to collect data from $p_{\pi_f}(s_t)$

model-based reinforcement learning version 1.0:

1. run base policy $\pi_0(a_t|s_t)$ (e.g., random policy) to collect $D = \{(s, a, s')_i\}$

2. learn dynamics model $f(s, a)$ to minimize $\sum_i \| f(s_i, a_i) - s'_i \|^2$

3. plan through $f(s, a)$ to choose actions

4. execute those actions and add the resulting data $\{(s, a, s')_j\}$ to $D$
What if we make a mistake?
Can we do better?

model-based reinforcement learning version 1.5:
1. run base policy $\pi_0(a_t|s_t)$ (e.g., random policy) to collect $D = \{(s, a, s')_i\}$
2. learn dynamics model $f(s, a)$ to minimize $\sum_i \| f(s_i, a_i) - s'_i \|^2$
3. plan through $f(s, a)$ to choose actions
4. execute the first planned action, observe resulting state $s'$ (MPC)
5. append $(s, a, s')$ to dataset $D$

This will be on HW4!
How to replan?

model-based reinforcement learning version 1.5:

1. run base policy $\pi_0(a_t|s_t)$ (e.g., random policy) to collect $D = \{(s, a, s')_i\}$

2. learn dynamics model $f(s, a)$ to minimize $\sum_i \|f(s_i, a_i) - s'_i\|^2$

3. plan through $f(s, a)$ to choose actions

4. execute the first planned action, observe resulting state $s'$ (MPC)

5. append $(s, a, s')$ to dataset $D$

• The more you replan, the less perfect each individual plan needs to be

• Can use shorter horizons

• Even random sampling can often work well here!
Uncertainty in Model-Based RL
A performance gap in model-based RL

Nagabandi, Kahn, Fearing, L. ICRA 2018
Why the performance gap?

need to not overfit here...

...but still have high capacity over here
Why the performance gap?

model-based reinforcement learning version 1.5:

1. run base policy $\pi_0(a_t|s_t)$ (e.g., random policy) to collect $D = \{(s, a, s')_i\}$

2. learn dynamics model $f(s, a)$ to minimize $\sum_i \| f(s_i, a_i) - s'_i \|^2$

3. plan through $f(s, a)$ to choose actions

4. execute the first planned action, observe resulting state $s'$ (MPC)

5. append $(s, a, s')$ to dataset $D$

very tempting to go here...
How can uncertainty estimation help?

expected reward under high-variance prediction is very low, even though mean is the same!
Intuition behind uncertainty-aware RL

model-based reinforcement learning version 1.5:

1. run base policy $\pi_0(a_t|s_t)$ (e.g., random policy) to collect $D = \{(s, a, s')_i\}$

2. learn dynamics model $f(s, a)$ to minimize $\sum_i \|f(s_i, a_i) - s'_i\|^2$

3. plan through $f(s, a)$ to choose actions

4. execute the first planned action, observe resulting state $s'$ (MPC)

5. append $(s, a, s')$ to dataset $D$

only take actions for which we think we’ll get high reward in expectation (w.r.t. uncertain dynamics)

This avoids “exploiting” the model
The model will then adapt and get better
There are a few caveats...

Need to explore to get better

Expected value is not the same as pessimistic value

Expected value is not the same as optimistic value

...but expected value is often a good start
Uncertainty-Aware Neural Net Models
How can we have uncertainty-aware models?

Idea 1: use output entropy

Two types of uncertainty:

- \textit{aleatoric} or statistical uncertainty
- \textit{epistemic} or model uncertainty

"the model is certain about the data, but we are not certain about the model"
How can we have uncertainty-aware models?

Idea 2: estimate model uncertainty

“The model is certain about the data, but we are not certain about the model”

usually, we estimate

\[ \arg \max_\theta \log p(\theta|D) = \arg \max_\theta \log p(D|\theta) \]

can we instead estimate \( p(\theta|D) \)?

the entropy of this tells us the model uncertainty!

\[
\int p(s_{t+1}|s_t, a_t, \theta) p(\theta|D) d\theta
\]
Quick overview of Bayesian neural networks

For more, see:
Blundell et al., Weight Uncertainty in Neural Networks
Gal et al., Concrete Dropout

We’ll learn more about variational inference later!
Bootstrap ensembles

Train multiple models and see if they agree!

formally: \( p(\theta|\mathcal{D}) \approx \frac{1}{N} \sum_i \delta(\theta_i) \)

\[
\int p(s_{t+1}|s_t, a_t, \theta)p(\theta|\mathcal{D})d\theta \approx \frac{1}{N} \sum_i p(s_{t+1}|s_t, a_t, \theta_i)
\]

How to train?

Main idea: need to generate “independent” datasets to get “independent” models

\( \theta_i \) is trained on \( \mathcal{D}_i \), sampled with replacement from \( \mathcal{D} \)
Bootstrap ensembles in deep learning

This basically works

Very crude approximation, because the number of models is usually small (< 10)

Resampling with replacement is usually unnecessary, because SGD and random initialization usually makes the models sufficiently independent
Planning with Uncertainty, Examples
How to plan with uncertainty

Before: \( J(a_1, \ldots, a_H) = \sum_{t=1}^{H} r(s_t, a_t), \) where \( s_{t+1} = f(s_t, a_t) \)

Now: \( J(a_1, \ldots, a_H) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{H} r(s_{t,i}, a_t), \) where \( s_{t+1,i} = f_i(s_{t,i}, a_t) \)

In general, for candidate action sequence \( a_1, \ldots, a_H \):

Step 1: sample \( \theta \sim p(\theta|D) \)

Step 2: at each time step \( t \), sample \( s_{t+1} \sim p(s_{t+1}|s_t, a_t, \theta) \)

Step 3: calculate \( R = \sum_t r(s_t, a_t) \)

Step 4: repeat steps 1 to 3 and accumulate the average reward

Other options: moment matching, more complex posterior estimation with BNNs, etc.
Example: model-based RL with ensembles

Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models

exceeds performance of model-free after 40k steps (about 10 minutes of real time)
More recent example: PDDM

Further readings

• Deisenroth et al. PILCO: A Model-Based and Data-Efficient Approach to Policy Search.

Recent papers:

• Nagabandi et al. Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning.


• Feinberg et al. Model-Based Value Expansion for Efficient Model-Free Reinforcement Learning.

• Buckman et al. Sample-Efficient Reinforcement Learning with Stochastic Ensemble Value Expansion.
Model-Based RL with Images
What about complex observations?

What is hard about this?
• High dimensionality
• Redundancy
• Partial observability

\[ f(s_t, a_t) = s_{t+1} \]

separately learn \( p(o_t|s_t) \) and \( p(s_{t+1}|s_t, a_t) \)

high-dimensional but not dynamic

low-dimensional but dynamic
State space (latent space) models

How to train?

standard (fully observed) model:
$$\max_\phi \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \log p_\phi(s_{t+1,i}|s_{t,i}, a_{t,i})$$

latent space model:
$$\max_\phi \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \mathbb{E} \left[ \log p_\phi(s_{t+1,i}|s_{t,i}, a_{t,i}) + \log p_\phi(o_{t,i}|s_{t,i}) \right]$$

expectation w.r.t. $(s_t, s_{t+1}) \sim p(s_t, s_{t+1}|o_{1:T}, a_{1:T})$

observation model
$$p(o_t|s_t)$$
dynamics model
$$p(s_{t+1}|s_t, a_t)$$
reward model
$$p(r_t|s_t, a_t)$$
Model-based RL with latent space models

\[
\max \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} E \left[ \log p_\phi(s_{t+1,i}|s_{t,i}, a_{t,i}) + \log p_\phi(o_{t,i}|s_{t,i}) \right]
\]

(expectation w.r.t. \((s_t, s_{t+1}) \sim p(s_t, s_{t+1}|o_1:T, a_1:T)\))

learn *approximate* posterior \(q_\psi(s_t|o_1:t, a_1:t)\) “encoder”

many other choices for approximate posterior:

- \(q_\psi(s_t, s_{t+1}|o_1:T, a_1:T)\) full smoothing posterior
  - + most accurate
  - - most complicated
- \(q_\psi(s_t|o_{t})\) single-step encoder
  - + simplest
  - - least accurate

we’ll talk about this one for now

**We will discuss variational inference in more detail next week!**
Model-based RL with latent space models

$$\max_{\phi} \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} E \left[ \log p_\phi(s_{t+1,i}|s_{t,i}, a_{t,i}) + \log p_\phi(o_{t,i}|s_{t,i}) \right]$$

(expectation w.r.t. $s_t \sim q_\psi(s_t|o_t), s_{t+1} \sim q_\psi(s_{t+1}|o_{t+1})$)

$q_\psi(s_t|o_t)$

simple special case: $q(s_t|o_t)$ is deterministic

stochastic case requires variational inference (next week)

$q_\psi(s_t|o_t) = \delta(s_t = g_\psi(o_t)) \Rightarrow s_t = g_\psi(o_t)$  deterministic encoder

$$\max_{\phi,\psi} \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \log p_\phi(g_\psi(o_{t+1,i})|g_\psi(o_{t,i}), a_{t,i}) + \log p_\phi(o_{t,i}|g_\psi(o_{t,i}))$$

Everything is differentiable, can train with backprop
Model-based RL with latent space models

Many practical methods use a stochastic encoder to model uncertainty
Model-based RL with latent space models

model-based reinforcement learning with latent state:

1. run base policy $\pi_0(a_t|o_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(o, a, o')_t\}$

2. learn $p_\phi(s_{t+1}|s_t, a_t)$, $p_\phi(r_t|s_t)$, $p(o_t|s_t)$, $g_\psi(o_t)$

3. plan through the model to choose actions

4. execute the first planned action, observe resulting $o'$ (MPC)

5. append $(o, a, o')$ to dataset $\mathcal{D}$
Embed to Control: A Locally Linear Latent Dynamics Model for Control from Raw Images

Manuel Watter*  Jost Tobias Springenberg*  Joschka Boedecker
University of Freiburg, Germany
{watterm, springj, jboedeck}@cs.uni-freiburg.de

Martin Riedmiller
Google DeepMind
London, UK
riedmiller@google.com
Swing-up with the E2C algorithm
SOLAR: Deep Structured Latent Representations for Model-Based Reinforcement Learning
Learn directly in observation space

**Key idea:** learn embedding $g(o_t) = s_t$

directly learn $p(o_{t+1} | o_t, a_t)$


Use predictions to complete tasks

Designated Pixel ⭐
Goal Pixel ⬤
Task execution