Deep Model-Based Reinforcement Learning

Chelsea Finn

UC Berkeley

Google Brain

Stanford
Outline

1. Why use model-based reinforcement learning?
2. Main model-based RL approaches
3. Using local models & guided policy search
4. Handling high-dimensional observations
Outline

1. Why use model-based reinforcement learning?
2. Main model-based RL approaches
3. Using local models & guided policy search
4. Handling high-dimensional observations
Why use model-based reinforcement learning?

- a model enables you to plan
- sample efficiency
RL approaches

- gradient-free methods (e.g. NES, CMA, etc.)
- fully online methods (e.g. A3C)
- policy gradient methods (e.g. TRPO)
- replay buffer value estimation methods (Q-learning, DDPG, NAF, etc.)
- model-based deep RL (e.g. guided policy search)
- model-based “shallow” RL (e.g. PILCO)

10x gap

Wang et al. ‘17
100,000,000 steps (100,000 episodes) (~ 15 days real time)

TRPO+GAE (Schulman et al. ‘16)
10,000,000 steps (10,000 episodes) (~ 1.5 days real time)

Gu et al. ‘16
1,000,000 steps (1,000 episodes) (~ 3 hours real time)

about 20 minutes of experience on a real robot

Chebotar et al. ‘17 (note log scale)
Why use model-based reinforcement learning?

- a model enables you to plan
- sample efficiency
- transferability & generality

A model can be reused for achieving different tasks.
[more examples later]
Outline

1. Why use model-based reinforcement learning?
2. Main model-based RL approaches
3. Using local models & guided policy search
4. Handling high-dimensional observations
The Anatomy of a Reinforcement Learning Problem

compute $\hat{Q} = \sum_{t=1}^{T} \gamma^{t-t'} r_{t'}$ (MC policy gradient)
fit $Q_\phi(s, a)$ (actor-critic, Q-learning)
estimate $p(s'|s, a)$ (model-based)

$\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta)$ (policy gradient)
$\pi(s) = \arg\max Q_\phi(s, a)$ (Q-learning)
optimize $\pi_\theta(a|s)$ (model-based)
Model-Based Reinforcement Learning

1. generate samples (i.e. run the policy)
2. fit a model to estimate return
3. estimate $p(s'|s, a)$ (model-based)
4. supervised learning
   $$\min_{\phi} \sum_i \| f_{\phi}(s_i, a_i) - s'_i \|^2$$
5. improve the policy
6. optimize $\pi_\theta(a|s)$ (model-based)
Backprop through model to optimize policy

\[ \text{Algorithm v0:} \]

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

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.
   or into policy to optimize \( \pi_\theta(a_t|s_t) \).
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

Slide adapted from S. Levine
Does it work?  

No!

1. run base policy $\pi_0(a_t|s_t)$ (e.g., random policy) to collect $D = \{(s, a, s')_i\}$
2. learn model $f_\phi(s, a)$ to minimize $\sum ||f_\phi(s_i, a_i) - s'_i||^2$
3. backpropagate through $f_\phi(s, a)$ into policy to optimize $\pi_\theta(a_t|s_t)$

$\mathbb{P}_{\pi_f}(s_t) \neq \mathbb{P}_{\pi_0}(s_t)$

- **State distribution mismatch**, problem becomes exacerbated as we use more expressive model classes

Slide adapted from S. Levine
Can we do better?

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

need to collect data from $p_{\pi_f}(s_t)$

Algorithm v1:

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 $\sum_i ||f_\phi(s_i, a_i) - s'_i||^2$
3. backpropagate through $f_\phi(s, a)$ into policy to optimize $\pi_\theta(a_t|s_t)$
4. run $\pi_\theta(a_t|s_t)$, appending visited tuples $(s, a, s')$ to $\mathcal{D}$
What if we make a mistake?
Can you correct the mistake?

---

**Algorithm v2a:**

1. run base policy $\pi_0(a_t|s_t)$ (e.g., random policy) to collect $D = \{(s, a, s')_i\}$
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.
4. execute the first planned action, observe resulting state $s'$
5. append $(s, a, s')$ to dataset $D$

*model-predictive control (MPC)*
An alternative way to choose actions

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

**Can instead sample to choose actions:**

A. Sample action sequences from some distribution (e.g. uniformly at random)
B. Run actions through model to prediction future
C. Choose action leading to the best future

Nagabandi et al. ICRA ‘18
Summary so far

• Version 0: collect random samples, train dynamics, plan
  • Pro: simple, no iterative procedure
  • Con: distribution mismatch problem

• Version 1: iteratively collect data, refit model
  • Pro: simple, solves distribution mismatch
  • Con: still might make mistakes with imperfect model

• Version 2: iteratively collect data using MPC (replan at each step)
  • Pro: robust to small model errors
  • Con: computationally expensive, but have a planning algorithm available

Two ways to optimize policy w.r.t. model:
  • backprop through model into policy
  • sampling-based optimization
What kind of models can we use?

**Gaussian process**
- GP with input \((s, a)\) and output \(s'\)
- Pro: very data-efficient
- Con: not great with non-smooth dynamics
- Con: very slow when dataset is big

**neural network**
- Input is \((s, a)\) and output is \(s'\)
- Euclidean training loss corresponds to Gaussian \(p(s' | s, a)\)
- More complex losses, e.g. output parameters of Gaussian mixture
- Pro: very expressive, can use lots of data
- Con: not so great in low data regimes

**other**
- GMM over \((s, a, s')\) tuples
- Train on \((s, a, s')\), condition to get \(p(s' | s, a)\)
- For \(i^{th}\) mixture element, \(p_i(s, a)\) gives region where the mode \(p_i(s' | s, a)\) holds
- other classes: domain-specific models (e.g. physics parameters)

Slide adapted from S. Levine
Outline

1. Why use model-based reinforcement learning?
2. Main model-based RL approaches
3. Using local models & guided policy search
4. Handling high-dimensional observations
The trouble with global models

Global model: $f_\phi(s_t, a_t)$ represented by a big neural network

1. run base policy $\pi_0(a_t|s_t)$ (e.g., random policy) to collect $D = \{(s, a, s')_i\}$
2. learn model $f_\phi(s, a)$ to minimize $\sum_i ||f_\phi(s_i, a_i) - s'_i||^2$
3. backpropagate through $f_\phi(s, a)$ into policy to optimize $\pi_\theta(a_t|s_t)$
4. run $\pi_\theta(a_t|s_t)$, appending visited tuples $(s, a, s')$ to $D$

- Planner will seek out regions where the model is erroneously optimistic
- Need to find a very good model in most of the state space to converge on a good solution

Slide adapted from S. Levine
Do we need to model everything?

What if we know where our model is good and where it is bad? i.e., model uncertainty

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

Take actions that lead to high reward *in expectation*. helps avoid model exploitation  
*Caveat:* still need to explore

To get model uncertainty:
- Gaussian Processes
- Bayesian neural networks
- Bootstrap ensembles
Do we need to model everything?

In some tasks, the model is much more complex than the policy
Local models

\[
\text{backpropagate}
\]

\[
\max_\theta \sum_t r(s_t, a_t)
\]

need \( \frac{df}{ds_t}, \frac{df}{da_t}, \frac{dr}{ds_t}, \frac{dr}{da_t} \)
Local models

\[ \text{need } \left( \frac{df}{ds_t}, \frac{df}{da_t}, \frac{dr}{ds_t}, \frac{dr}{da_t} \right) \]

idea: just fit \( \frac{df}{ds_t}, \frac{df}{da_t} \), around current trajectory or policy!

\( p(a_t | s_t) \) – time-varying linear-Gaussian controller can execute on the robot!

“local policy”

Slide adapted from S. Levine
Local models

\[ p(s_{t+1} | s_t, a_t) = \mathcal{N}(f(s_t, a_t), \Sigma) \]

\[ f(s_t, a_t) \approx A_t s_t + B_t a_t \]

\[ A_t = \frac{df}{ds_t} \quad B_t = \frac{df}{da_t} \]
How to fit the dynamics?

\[
p(s_{t+1} | s_t, a_t) \]

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

Version 1.0: fit \( p(s_{t+1} | s_t, a_t) \) at each time step using linear regression

\[
p(s_{t+1} | s_t, a_t) = \mathcal{N}(A_t s_t + B_t a_t + c_t, N_t)
\]

\[A_t \approx \frac{df}{ds_t}, \quad B_t \approx \frac{df}{da_t}\]

Can we do better?

Version 2.0: fit \( p(s_{t+1} | s_t, a_t) \) using Bayesian linear regression

Use your favorite global model as prior (GP, deep net, GMM)
What if we go too far?

Slide adapted from S. Levine
How to stay close to old controller?

What if the new $p(\tau)$ is “close” to the old one $\tilde{p}(\tau)$?

If trajectory distribution is close, then dynamics will be close too!

What does “close” mean? $D_{KL}(p(\tau)\|\tilde{p}(\tau)) \leq \epsilon$

Slide adapted from S. Levine
Local Models Approach Summary

1. run base policy $\pi_0(a_t|s_t)$ to collect $D = \{(s, a, s')_i\}$
2. learn local model $f(s, a)$ to minimize $\sum_i ||f_\phi(s_i, a_i) - s'_i||^2$  
   e.g. using linear regression
3. update local policy $\pi_\theta(a_t|s_t)$ using local model $f_\phi$ with KL constraint.
4. run $\pi_\theta(a_t|s_t)$, putting visited tuples $(s, a, s')$ in $D$  
   e.g. using iterative LQR
Case study: local models & iterative LQR

Learning Contact-Rich Manipulation Skills with Guided Policy Search

Sergey Levine, Nolan Wagener, Pieter Abbeel

next iteration

run $p(a_t \mid s_t)$ on robot
collect $\mathcal{D} = \{\tau_i\}$

fit dynamics $p(s_{t+1} \mid s_t, a_t)$

improve $p(a_t \mid s_t)$

\{\tau_i\}
Case study: local models & iterative LQR
Local Models Approach Summary

1. run base policy $\pi_0(a_t|s_t)$ to collect $D = \{(s, a, s')_i\}$
2. learn local model $f(s, a)$ to minimize $\sum_i \|f_\phi(s_i, a_i) - s'_i\|^2$
   e.g. using linear regression
3. update local policy $\pi_\theta(a_t|s_t)$ using local model $f_\phi$ with KL constraint.
4. run $\pi_\theta(a_t|s_t)$, putting visited tuples $(s, a, s')$ in $D$
   e.g. using iterative LQR

end result: single local policy

Guided policy search: supervise one global policy using multiple local policies
Case study: guided policy search

Training time:

\[ s_t \rightarrow a_t \]

target pose known

Test time:

\[ o(s_t) \rightarrow a_t \]
Case study: guided policy search

Training time

- take samples for each target position
- fit local model and solve for local policy for each target position
- use supervision from local policies to train global neural network policy w/ vision
Guided Policy Search: learning

(Levine*, Finn*, et al. JMLR ’16)
Guided Policy Search: learned behaviors

(Levine*, Finn*, et al. JMLR ’16)

real time

< 300 trials = 25 min of robot time (per task)

+ efficiently learn complex vision-based skills
- requires state during training
1. Why use model-based reinforcement learning?
2. Main model-based RL approaches
3. Using local models & guided policy search
4. Handling high-dimensional observations
Only access to high-dimensional observations (i.e. images)?

Also: no reward signal with only observations
Only access to high-dimensional observations (i.e. images)?

**also:** no reward signal with only observations

**one option:** provide image of goal

**Approaches**

1. Learn model in latent space
2. Learn model of observations (e.g. video)
3. Inverse models [won’t cover]
Learning in Latent Space

**Key idea:** learn embedding \( g(o_t) \), then learn model in latent space
Learning in Latent Space

Key idea: learn embedding $s_t = g(o_t)$, then do model-based RL in latent space

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

Deep Spatial Autoencoders for Visuomotor Learning

Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, Pieter Abbeel

Fig. 1: PR2 learning to scoop a bag of rice into a bowl with a spatula (left) using a learned visual state representation (right).

NIPS 2015  ICRA 2016
Learning in Latent Space

1. run base policy \( \pi_0(a_t|o_t) \) (e.g., exploratory policy) to collect \( D = \{(o, a, o')_t\} \)
2. learn latent embedding of observation \( s_t = g(o_t) \) and dynamics model \( s' = f_\phi(s, a) \)
3. use model \( f_\phi(s, a) \) to optimize policy \( \pi_\theta(a_t|s_t) \)
4. run \( \pi_\theta(a_t|g(o_t)) \), appending visited tuples \( (o, a, o') \) to \( D \)

What is reward for optimizing policy?

**reward signal**: \( r(o, a) = r(a) + \|g(o) - g(o_{goal})\| \)

Aside: If you have reward observations (i.e. video games), can simply fit a reward model instead.
Learning in Latent Space

1. run base policy $\pi_0(a_t|s_t)$ (e.g., exploratory policy) to collect $D = \{(o, a, o')_i\}$
2. learn latent embedding of observation $s_t = g(o_t)$ and dynamics model $s' = f_\phi(s, a)$
3. use model $f_\phi(s, a)$ to optimize policy $\pi_\theta(a_t|s_t)$
4. run $\pi_\theta(a_t|g(o_t))$, appending visited tuples $(o, a, o')$ to $D$

How to optimize latent embedding $g$?

Finn et al.’16

Watter et al.’15

learn embedding & model jointly

embedding is smooth and structured
Learning in Latent Space

~300 trials = ~25 min of robot time (per task)  

Watter et al. NIPS ‘15
Learning in Latent Space

125 trials = 11 min of robot time (per task)

Finn et al. ICRA’16
Learning in Latent Space

Pros:
+ Learn complex visual skills very efficiently
+ Structured representation enables effective learning

Cons:
- Reconstruction objectives might not recover the right representation
Aside: Low-dimensional embedding can also be useful for model-free approaches

**model-free RL in latent space**
- **FQI in latent space**
  - Lange et al. ‘12
- **TRPO in latent space**
  - Ghadirzadeh et al. ‘17

**use embedding for reward function**
- **Sermanet et al. RSS ’17**
  - video demonstration
  - learned policy
  - acquire reward using ImageNet features
  - + model-free RL

If you have a reward, you can predict it to form better latent space
- Jaderberg et al. ’17, Shelhamer et al. ’17
Modeling directly in observation space

Recall MPC

1. run base policy $\pi_0(a_t|o_t)$ (e.g., random policy) to collect $D = \{(o, a, o')_i\}$
2. learn model $f_\phi(o, a)$ to minimize $\sum_i \|f_\phi(o_i, a_i) - o'_i\|^2$
3. backpropagate through $f_\phi(o, a)$ to choose actions.
4. execute the first planned action, observe resulting state $o'$
5. append $(o, a, o')$ to dataset $D$

action-conditioned video prediction

Finn et al. NIPS '16, Finn & Levine ICRA '17, Ebert et al. '18
Models capture **general purpose** knowledge about the world.

Use **all** of the available supervision signal.

Also: No assumptions about task **representations**.
1. Consider potential action sequences
2. Predict the future for each action sequence
3. Pick best future & execute corresponding action
4. Repeat 1-3 to replan in real time

visual “model-predictive control” (MPC)

**Overall System:** Collect data, Train predictive model, Plan to achieve goals
Which future is the best one?

Human specifies a goal by:

- Selecting where pixels should move.
- Providing an image of the goal.
- Providing a few examples of success.

Finn & Levine ICRA ’17
Ebert, Lee, Levine, Finn CoRL ’18
Xie, Singh, Levine, Finn CoRL ’18
Modeling directly in Observation Space

Specify goal

Visual MPC execution

Visual MPC w.r.t. goal

\~2 weeks of *unsupervised* robot time

*Only human involvement*: programming initial motions and providing objects to play with.

Ebert*, Finn*, Dasari, Xie, Lee, Levine. ‘18
Planning with a **single model** for many tasks

Ebert*, Finn*, Dasari, Xie, Lee, Levine. ‘18
Modeling directly in observation space

**Pros:**
+ Entirely self-supervised
+ Learn for a variety of tasks
+ More efficient than single-task model-free learning

**Cons:**
- Can’t [yet] handle as complex skills as model-free methods
Predict alternative quantities

If I take a set of actions:

Pinto et al. ’16

Will I successfully grasp?

Kahn et al. ’17

Will I collide?

Dosovitskiy & Koltun ’17

What will health/damage/etc. be?

Pros:
+ Only predict task-relevant quantities!

Cons:
- Need to manually pick quantities, must be able to directly observe them
Outline

1. Why use model-based reinforcement learning?
2. Main model-based RL approaches
3. Using local models & guided policy search
4. Handling high-dimensional observations
Model-based RL Review

Correcting for model errors:
refit model with new data, replan with MPC, use local models or uncertainty

Model-based RL from raw observations:
learn latent space, typically with unsupervised learning, or model & plan directly in observational space
Model-Based vs. Model-Free Algorithms

Models:
+ Easy to collect data in a scalable way (self-supervised)
+ Possibility to transfer across tasks
+ Typically require a smaller quantity of supervised data
  - Models don’t optimize for task performance
  - Sometimes harder to learn than a policy
  - Often need assumptions to learn complex skills (continuity, resets)

Model-Free:
+ Makes little assumptions beyond a reward function
+ Effective for learning complex policies
  - Require a lot of experience (slower)
  - Not transferable across tasks

Ultimately we will want both!
Challenges & Frontiers

Long-horizon prediction & planning
- Structured latent representations
need:
- Uncertainty
- Compositionality

Internal reward representations

Combining elements of model-based & model-free
- use roll-outs from model as experience:
  Sutton ‘90, Gu et al. ICML ’16, Kurutach et al. ICLR ’18
- model-free policy with planning capabilities:
  Tamar et al. NIPS ’16, Pascanu et al. ‘17
- model-based look-ahead:
  Guo et al. NIPS ’14, Silver et al. Nature ’16, Buckman et al. NIPS ’18

Exploration (models can help!)
Stadie et al. arXiv ’15, Oh et al. NIPS ’16, Burda et al. ’18
Questions?