Model-Based Policy Learning

CS 285

Instructor: Sergey Levine
UC Berkeley
model-based reinforcement learning version 1.5:

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

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

3. plan through $f(\mathbf{s}, \mathbf{a})$ to choose actions

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

5. append $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ to dataset $\mathcal{D}$
The stochastic open-loop case

\[ p_\theta(s_1, \ldots, s_T | a_1, \ldots, a_T) = p(s_1) \prod_{t=1}^{T} p(s_{t+1} | s_t, a_t) \]

\[ a_1, \ldots, a_T = \arg \max_{a_1, \ldots, a_T} E \left[ \sum_{t} r(s_t, a_t) | a_1, \ldots, a_T \right] \]

why is this suboptimal?
The stochastic **closed-loop** case

$$p(s_1, a_1, \ldots, s_T, a_T) = p(s_1) \prod_{t=1}^{T} \pi(a_t|s_t)p(s_{t+1}|s_t, a_t)$$

$$\pi = \arg \max_{\pi} \mathbb{E}_{\tau \sim p(\tau)} \left[ \sum_{t} r(s_t, a_t) \right]$$

Form of $\pi$?

- neural net
- time-varying linear
- $K_t s_t + k_t$
Backpropagate directly into the policy?

model-based reinforcement learning version 2.0:

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. backpropagate through $f(s, a)$ into the policy to optimize $\pi_\theta(a_t|s_t)$

4. run $\pi_\theta(a_t|s_t)$, appending the visited tuples $(s, a, s')$ to $D$
What’s the problem with backprop into policy?

$\mathbf{a}_t = \pi_\theta(\mathbf{s}_t)$

$\mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t)$

$\mathbf{a}_t = \pi_\theta(\mathbf{s}_t)$

$\mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t)$

big gradients here

small gradients here
What’s the problem with backprop into policy?
What’s the problem with backprop into policy?

- Similar parameter sensitivity problems as shooting methods
  - But no longer have convenient second order LQR-like method, because policy parameters couple all the time steps, so no dynamic programming

- Similar problems to training long RNNs with BPTT
  - Vanishing and exploding gradients
  - Unlike LSTM, we can’t just “choose” a simple dynamics, dynamics are chosen by nature
What’s the solution?

• Use derivative-free ("model-free") RL algorithms, with the model used to generate synthetic samples
  • Seems weirdly backwards
  • Actually works very well
  • Essentially “model-based acceleration” for model-free RL

• Use simpler policies than neural nets
  • LQR with learned models (LQR-FLM – Fitted Local Models)
  • Train local policies to solve simple tasks
  • Combine them into global policies via supervised learning
Model-Free Learning With a Model
What’s the solution?

- Use derivative-free ("model-free") RL algorithms, with the model used to generate synthetic samples
  - Seems weirdly backwards
  - Actually works very well
  - Essentially “model-based acceleration” for model-free RL

- Use simpler policies than neural nets
  - LQR with learned models (LQR-FLM – Fitted Local Models)
  - Train local policies to solve simple tasks
  - Combine them into global policies via supervised learning
Model-free optimization with a model

Policy gradient:
\[ \nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \hat{Q}_{i,t} \]

Backprop (pathwise) gradient:
\[ \nabla_\theta J(\theta) = \sum_{t=1}^{T} \frac{d r_t}{d s_t} \prod_{t'=2}^{t} \frac{d s_{t'}}{d a_{t'-1}} \frac{d a_{t'-1}}{d s_{t'-1}} \]

- Policy gradient might be more stable (if enough samples are used) because it does not require multiplying many Jacobians

- See a recent analysis here:
  - Parmas et al. ‘18: PIPP: Flexible Model-Based Policy Search Robust to the Curse of Chaos
Model-free optimization with a model

Dyna online Q-learning algorithm that performs model-free RL with a model

1. given state $s$, pick action $a$ using exploration policy
2. observe $s'$ and $r$, to get transition $(s, a, s', r)$
3. update model $\hat{p}(s'|s, a)$ and $\hat{r}(s, a)$ using $(s, a, s')$
4. Q-update: $Q(s, a) \leftarrow Q(s, a) + \alpha E_{s', r}[r + \max_{a'} Q(s', a') - Q(s, a)]$
5. repeat $K$ times:
   6. sample $(s, a) \sim \mathcal{B}$ from buffer of past states and actions
   7. Q-update: $Q(s, a) \leftarrow Q(s, a) + \alpha E_{s', r}[r + \max_{a'} Q(s', a') - Q(s, a)]$

Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming.
General “Dyna-style” model-based RL recipe

1. collect some data, consisting of transitions \((s, a, s', r)\)
2. learn model \(\hat{p}(s'|s, a)\) (and optionally, \(\hat{r}(s, a)\))
3. repeat K times:
   4. sample \(s \sim \mathcal{B}\) from buffer
   5. choose action \(a\) (from \(\mathcal{B}\), from \(\pi\), or random)
   6. simulate \(s' \sim \hat{p}(s'|s, a)\) (and \(r = \hat{r}(s, a)\))
   7. train on \((s, a, s', r)\) with model-free RL
   8. (optional) take \(N\) more model-based steps

+ only requires short (as few as one step) rollouts from model
+ still sees diverse states
Model-Based Acceleration (MBA)  
Model-Based Value Expansion (MVE)  
Model-Based Policy Optimization (MBPO)

1. take some action $a_i$ and observe $(s_i, a_i, s'_i, r_i)$, add it to $B$  
2. sample mini-batch $\{s_j, a_j, s'_j, r_j\}$ from $B$ uniformly  
3. use $\{s_j, a_j, s'_j\}$ to update model $\hat{p}(s' | s, a)$  
4. sample $\{s_j\}$ from $B$  
5. for each $s_j$, perform model-based rollout with $a = \pi(s)$  
6. use all transitions $(s, a, s', r)$ along rollout to update $Q$-function

+ why is this a good idea?  
- why is this a bad idea?

Gu et al. Continuous deep Q-learning with model-based acceleration. ‘16  
Feinberg et al. Model-based value expansion. ’18  
Janner et al. When to trust your model: model-based policy optimization. ‘19
Local Models
What’s the solution?

• Use derivative-free (“model-free”) RL algorithms, with the model used to generate synthetic samples
  • Seems weirdly backwards
  • Actually works very well
  • Essentially “model-based acceleration” for model-free RL

• Use simpler policies than neural nets
  • LQR with learned models (LQR-FLM – Fitted Local Models)
  • Train local policies to solve simple tasks
  • Combine them into global policies via supervised learning
What’s the solution?

• Use derivative-free (“model-free”) RL algorithms, with the model used to generate synthetic samples
  • Seems weirdly backwards
  • Actually works very well
  • Essentially “model-based acceleration” for model-free RL

• Use simpler policies than neural nets
  • LQR with learned models (LQR-FLM – Fitted Local Models)
  • Train local policies to solve simple tasks
  • Combine them into global policies via supervised learning
Local models

\[
\min_{u_1, \ldots, u_T} \sum_{t=1}^{T} c(x_t, u_t) \quad \text{s.t.} \quad x_t = f(x_{t-1}, u_{t-1})
\]

\[
\min_{u_1, \ldots, u_T} c(x_1, u_1) + c(f(x_1, u_1), u_2) + \cdots + c(f(f(\cdots), \cdots), u_T)
\]

usual story: differentiate via backpropagation and optimize!

need \( \frac{df}{dx_t}, \frac{df}{du_t}, \frac{dc}{dx_t}, \frac{dc}{du_t} \)
Local models

\[
\begin{align*}
\text{need } & \frac{df}{dx_t}, \frac{df}{du_t}, \frac{dc}{dx_t}, \frac{dc}{du_t} \\
\text{idea: just fit} & \frac{df}{dx_t}, \frac{df}{du_t} \text{ around current trajectory or policy!}
\end{align*}
\]

LQR gives us a linear feedback controller

can execute in the real world!
Local models

\[ p(x_{t+1} | x_t, u_t) = \mathcal{N}(f(x_t, u_t), \Sigma) \]
\[ f(x_t, u_t) \approx A_t x_t + B_t u_t \]
\[ A_t = \frac{df}{dx_t} \quad B_t = \frac{df}{du_t} \]
What controller to execute?

iLQR produces: \( \hat{x}_t, \hat{u}_t, K_t, k_t \)

\[ u_t = K_t(x_t - \hat{x}_t) + k_t + \hat{u}_t \]

Version 0.5: \( p(u_t|x_t) = \delta(u_t = \hat{u}_t) \)

Doesn’t correct deviations or drift

Version 1.0: \( p(u_t|x_t) = \delta(u_t = K_t(x_t - \hat{x}_t) + k_t + \hat{u}_t) \)

Better, but maybe a little too good?

Version 2.0: \( p(u_t|x_t) = \mathcal{N}(K_t(x_t - \hat{x}_t) + k_t + \hat{u}_t, \Sigma_t) \)

Add noise so that all samples don’t look the same!

Set \( \Sigma_t = \mathbf{Q}_{u_t}^{-1} u_t \)
Local models

\[ p(x_{t+1}|x_t, u_t) = \mathcal{N}(f(x_t, u_t), \Sigma) \]

\[ f(x_t, u_t) \approx A_t x_t + B_t u_t \]

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

\[
\{ (x_t, u_t, x_{t+1}) \}_i
\]

fit \( p(x_{t+1}|x_t, u_t) \) at each time step using linear regression

\[
p(x_{t+1}|x_t, u_t) = \mathcal{N}(A_t x_t + B_t u_t + c, N_t)
\]

\[
A_t \approx \frac{df}{dx_t} \quad B_t \approx \frac{df}{du_t}
\]
What if we go too far?
How to stay close to old controller?

For details, see: “Learning Neural Network Policies with Guided Policy Search under Unknown Dynamics”

\[
p(u_t | x_t) = \mathcal{N}(K_t(x_t - \hat{x}_t) + k_t + \hat{u}_t, \Sigma_t)
\]

\[
p(\tau) = p(x_1) \prod_{t=1}^{T} p(u_t | x_t)p(x_{t+1} | x_t, u_t)
\]

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

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

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

This is easy to do if \( \bar{p}(\tau) \) also came from linear controller!
Global Policies from Local Models
What’s the solution?

• Use derivative-free ("model-free") RL algorithms, with the model used to generate synthetic samples
  • Seems weirdly backwards
  • Actually works very well
  • Essentially “model-based acceleration” for model-free RL

• Use simpler policies than neural nets
  • LQR with learned models (LQR-FLM – Fitted Local Models)
  • Train local policies to solve simple tasks
  • Combine them into global policies via supervised learning
What’s the solution?

• Use derivative-free ("model-free") RL algorithms, with the model used to generate synthetic samples
  • Seems weirdly backwards
  • Actually works very well
  • Essentially “model-based acceleration” for model-free RL

• Use simpler policies than neural nets
  • LQR with learned models (LQR-FLM – Fitted Local Models)
  • Train local policies to solve simple tasks
  • Combine them into global policies via supervised learning
Guided policy search: high-level idea
Guided policy search: algorithm sketch

1. Optimize each local policy $\pi_{LQR,i}(u_t|x_t)$ on initial state $x_{0,i}$ w.r.t. $c_{k,i}(x_t, u_t)$.
2. Use samples from step (1) to train $\pi_\theta(u_t|x_t)$ to mimic each $\pi_{LQR,i}(u_t|x_t)$.
3. Update cost function $\tilde{c}_{k+1,i}(x_t, u_t) = c(x_t, u_t) + \lambda_{k+1,i} \log \pi_\theta(u_t|x_t)$.

For details, see: "End-to-End Training of Deep Visuomotor Policies"  

Lagrange multiplier
Underlying principle: distillation

**Ensemble models:** single models are often not the most robust – instead train many models and average their predictions

  this is how most ML competitions (e.g., Kaggle) are won

  this is very expensive at test time

**Can we make a single model that is as good as an ensemble?**

**Distillation:** train on the ensemble’s predictions as “soft” targets

\[
p_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}
\]

**Intuition:** more knowledge in soft targets than hard labels!

Slide adapted from G. Hinton, see also Hinton et al. “Distilling the Knowledge in a Neural Network”
Distillation for Multi-Task Transfer

\[
\mathcal{L} = \sum_{a} \pi_{E_i}(a|s) \log \pi_{AMN}(a|s)
\]

(just supervised learning/distillation)

analogous to guided policy search, but for multi-task learning

some other details
(e.g., feature regression objective)
– see paper

Combining weak policies into a strong policy

Divide and conquer reinforcement learning algorithm sketch:

1. optimize each local policy \( \pi_{\theta_i}(a_t|s_t) \) on initial state \( s_{0,i} \) w.r.t. \( \tilde{r}_{k,i}(s_t,a_t) \)
2. use samples from step (1) to train \( \pi_{\theta}(u_t|x_t) \) to mimic each \( \pi_{\theta_i}(u_t|x_t) \)
3. update reward function \( \tilde{r}_{k+1,i}(x_t,u_t) = r(x_t,u_t) + \lambda_{k+1,i} \log \pi_{\theta}(u_t|x_t) \)

For details, see: “Divide and Conquer Reinforcement Learning”
Readings: guided policy search & distillation

• Rusu et al. Policy Distillation. 2015.