Model-Based Reinforcement Learning

CS 294-112: Deep Reinforcement Learning
Sergey Levine
Class Notes

1. Homework 3 due in one week
   • Don’t put it off! It takes a while to train.

2. Project proposal due in two weeks!
Overview

1. Last lecture: choose good actions autonomously by backpropagating (or planning) through known system dynamics (e.g. known physics)

2. Today: what do we do if the dynamics are unknown?
   a. Fitting global dynamics models (“model-based RL”)
   b. Fitting local dynamics models

3. Wednesday: combining optimal control and policy search to train neural network policies with the aid of optimal control
Today’s Lecture

1. Overview of model-based RL
   • Learn only the model
   • Learn model & policy

2. What kind of models can we use?

3. Global models and local models

4. Learning with local models and trust regions
   • Goals:
     • Understand the terminology and formalism of model-based RL
     • Understand the options for models we can use in model-based RL
     • Understand practical considerations of model learning
Why learn the model?

\[
\min_{\mathbf{u}_1, \ldots, \mathbf{u}_T} \sum_{t=1}^{T} c(\mathbf{x}_t, \mathbf{u}_t) \text{ s.t. } \mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})
\]

\[
\min_{\mathbf{u}_1, \ldots, \mathbf{u}_T} c(\mathbf{x}_1, \mathbf{u}_1) + c(f(\mathbf{x}_1, \mathbf{u}_1), \mathbf{u}_2) + \cdots + c(f(f(\ldots)\ldots), \mathbf{u}_T)
\]

usual story: differentiate via backpropagation and optimize!

need \( \frac{df}{dx_t}, \frac{df}{du_t}, \frac{dc}{dx_t}, \frac{dc}{du_t} \)
Why learn the model?
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!**

- Distribution mismatch problem becomes exacerbated as we use more expressive model classes

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

1. run base policy \( \pi_0(a_t|s_t) \) (e.g., random policy) to collect \( D = \{(s,a,s')_t\} \)
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
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 $\{(x, u, x')_j\}$ 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!
That seems like a lot of work...

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. backpropagate through $f(s, a)$ to choose actions (e.g. using iLQR)

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

5. append $(s, a, s')$ to dataset $D$
Backpropagate directly into the policy?

Easy for deterministic policies, but also possible for stochastic policy (more on this later)

Model-based reinforcement learning version 2.0:

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

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 $\mathcal{D}$
Summary

• Version 0.5: collect random samples, train dynamics, plan
  • Pro: simple, no iterative procedure
  • Con: distribution mismatch problem

• Version 1.0: iteratively collect data, replan, collect data
  • Pro: simple, solves distribution mismatch
  • Con: open loop plan might perform poorly, esp. in stochastic domains

• Version 1.5: iteratively collect data using MPC (replan at each step)
  • Pro: robust to small model errors
  • Con: computationally expensive, but have a planning algorithm available

• Version 2.0: backpropagate directly into policy
  • Pro: computationally cheap at runtime
  • Con: can be numerically unstable, especially in stochastic domains (more on this later)
Case study: model-based policy search with GPs

Learning to Control a Low-Cost Manipulator using Data-Efficient Reinforcement Learning

Marc Peter Deisenroth
Dept. of Computer Science & Engineering
University of Washington
Seattle, WA, USA

Carl Edward Rasmussen
Dept. of Engineering
University of Cambridge
Cambridge, UK

Dieter Fox
Dept. of Computer Science & Engineering
University of Washington
Seattle, WA, USA
Case study: model-based policy search with GPs

Learning to Control a Low-Cost Manipulator using Data-Efficient Reinforcement Learning

Marc Peter Deisenroth
Dept. of Computer Science & Engineering
University of Washington
Seattle, WA, USA

Carl Edward Rasmussen
Dept. of Engineering
University of Cambridge
Cambridge, UK

Dieter Fox
Dept. of Computer Science & Engineering
University of Washington
Seattle, WA, USA

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

2. learn GP dynamics model $p(s'|s, a)$ to maximize $\sum_i \log p(s'_i|s_i, a_i)$

3. backpropagate through $p(s'|s, a)$ 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 $D$
Case study: model-based policy search with GPs

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

Given $p(s_t)$, use $p(s' | s, a)$ to compute $p(s_{t+1})$

If $p(s_t)$ is Gaussian, we can get a (non-Gaussian) $\overline{p}(s_{t+1})$ in closed form

Project non-Gaussian $\overline{p}(s_{t+1})$ to Gaussian $p(s_{t+1})$ using moment matching

$E_{s \sim p(s)}[c(s)]$ easy if $c$ is nice and $p(s)$ Gaussian

Write $\sum_t E_{s \sim p(s_t)}[r(s_t)]$ and differentiate
Marc Peter Deisenroth, Carl Edward Rasmussen, Dieter Fox

Learning to Control a Low-Cost Manipulator using Data-efficient Reinforcement Learning
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 \((s, a)\), 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)

video prediction? more on this later in the course
Neural Network Dynamics
for Model-Based Deep Reinforcement Learning
with Model-Free Fine-Tuning

Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing, Sergey Levine
University of California, Berkeley

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 (random sampling)

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

5. append $(s, a, s')$ to dataset $D$
Model-Based Burn-In for Sample-Efficient Deep Reinforcement Learning
Break
The trouble with global models

Global model: \( f(s_t, a_t) \) represented by a big neural network

1. run base policy \( \pi_0(a_t|s_t) \) (e.g., random policy) to collect \( \mathcal{D} = \{(s, a, s')_i\} \)
2. learn dynamics model \( f(s, a) \) to minimize \( \sum \| f(s_i, a_i) - s'_i \|^2 \)
3. plan through \( f(s, a) \) to choose actions
4. execute those actions and add the resulting data \( \{(s, a, s')_j\} \) to \( \mathcal{D} \)

- 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
The trouble with global models

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

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

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

usual story: differentiate via backpropagation and optimize!

$$\frac{df}{dx_t}, \frac{df}{du_t}, \frac{dc}{dx_t}, \frac{dc}{du_t}$$
Local models


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

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

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!
What controller to execute?

Version 2.0: \( p(\mathbf{u}_t | \mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t (\mathbf{x}_t - \hat{\mathbf{x}}_t) + \mathbf{k}_t + \mathbf{\hat{u}}_t, \Sigma_t) \)

Set \( \Sigma_t = Q_{u_t,u_t}^{-1} \)

\( Q(\mathbf{x}_t, \mathbf{u}_t) \) is the cost to go: total cost we get after taking an action

\[
Q(\mathbf{x}_t, \mathbf{u}_t) = \text{const} + \frac{1}{2} \left[ \begin{array}{c} \mathbf{x}_t \\ \mathbf{u}_t \end{array} \right]^T Q_t \left[ \begin{array}{c} \mathbf{x}_t \\ \mathbf{u}_t \end{array} \right] + \left[ \begin{array}{c} \mathbf{x}_t \\ \mathbf{u}_t \end{array} \right]^T \mathbf{q}_t
\]

\( Q_{u_t,u_t} \) is big if changing \( \mathbf{u}_t \) changes the Q-value a lot!

If \( \mathbf{u}_t \) changes Q-value a lot, don’t vary \( \mathbf{u}_t \) so much

Only act randomly when it minimally affects the cost to go
What controller to execute?

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

Set \( \Sigma_t = Q_{u_t,u_t}^{-1} \)

Standard LQR solves \( \min \sum_{t=1}^{T} c(x_t, u_t) \)

Linear-Gaussian solution solves \( \min \sum_{t=1}^{T} E_{(x_t,u_t) \sim p(x_t,u_t)}[c(x_t, u_t) - \mathcal{H}(p(u_t|x_t))] \)

This is the maximum entropy solution: act as randomly as possible while minimizing cost
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\}

Version 1.0: fit \(p(x_{t+1}|x_t, u_t)\) at each time step using linear regression

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

\[ A_t \approx \frac{df}{dx_t} \quad B_t \approx \frac{df}{du_t} \]

Can we do better?

Version 2.0: fit \(p(x_{t+1}|x_t, u_t)\) using Bayesian linear regression

Use your favorite *global* model as prior (GP, deep net, GMM)
What if we go too far?
How to stay close to old controller?

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

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

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? \( D_{KL}(p(\tau)||\bar{p}(\tau)) \leq \epsilon \)
KL-divergences between trajectories

• Not just for trajectory optimization – really important for model-free policy search too! More on this in later lectures

\[ D_{KL}(p(\tau)\|\bar{p}(\tau)) = E_{p(\tau)}[\log p(\tau) - \log \bar{p}(\tau)] \]

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

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

\[ \text{dynamics & initial state are the same!} \]

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

\[ - \log p(x_1) + \sum_{t=1}^{T} - \log \bar{p}(u_t|x_t) - \log p(x_{t+1}|x_t, u_t) \]
KL-divergences between trajectories

\[ D_{KL}(p(\tau)\|\bar{p}(\tau)) = E_{p(\tau)}[\log p(\tau) - \log \bar{p}(\tau)] \]

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

\[ - \log p(x_1) + \sum_{t=1}^{T} - \log \bar{p}(u_t|x_t) - \log p(x_{t+1}|x_t, u_t) \]

\[ D_{KL}(p(\tau)\|\bar{p}(\tau)) = E_{p(\tau)} \left[ \sum_{t=1}^{T} \log p(u_t|x_t) - \log \bar{p}(u_t|x_t) \right] \]

\[ D_{KL}(p(\tau)\|\bar{p}(\tau)) = \sum_{t=1}^{T} E_{p(x_t,u_t)} [\log p(u_t|x_t) - \log \bar{p}(u_t|x_t)] \]
KL-divergences between trajectories

\[
D_{\text{KL}}(p(\tau) \| \bar{p}(\tau)) = \sum_{t=1}^{T} E_p(x_t, u_t) \left[ \log p(u_t | x_t) - \log \bar{p}(u_t | x_t) \right]
\]

\[
D_{\text{KL}}(p(\tau) \| \bar{p}(\tau)) = \sum_{t=1}^{T} E_p(x_t, u_t) \left[ -\log \bar{p}(u_t | x_t) \right] + E_p(x_t) \left[ E_p(u_t | x_t) \left[ \log p(u_t | x_t) \right] \right]
\]

\[
D_{\text{KL}}(p(\tau) \| \bar{p}(\tau)) = \sum_{t=1}^{T} E_p(x_t, u_t) \left[ -\log \bar{p}(u_t | x_t) - \mathcal{H}(p(u_t | x_t)) \right]
\]

negative entropy
KL-divergences between trajectories

\[ D_{KL}(p(\tau)\|\bar{p}(\tau)) = \sum_{t=1}^{T} E_{p(x_t,u_t)}[-\log \bar{p}(u_t|x_t) - H(p(u_t|x_t))] \]

Reminder: Linear-Gaussian solves \( \min \sum_{t=1}^{T} E_{p(x_t,u_t)}[c(x_t,u_t) - H(p(u_t|x_t))] \)

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

If we can get \( D_{KL} \) into the cost, we can just use iLQR!

But how?

We want a constraint: \( D_{KL}(p(\tau)\|\bar{p}(\tau)) \leq \epsilon \)
Digression: dual gradient descent

\[
\min_x f(x) \text{ s.t. } C(x) = 0
\]

\[
\mathcal{L}(x, \lambda) = f(x) + \lambda C(x)
\]

\[
g(\lambda) = \inf_x \mathcal{L}(x, \lambda)
\]

\[
\lambda \leftarrow \arg \max_{\lambda} g(\lambda)
\]

how to maximize? Compute the gradient!
Digression: dual gradient descent

\[
\min_x f(x) \text{ s.t. } C(x) = 0 \quad \text{where} \quad \mathcal{L}(x, \lambda) = f(x) + \lambda C(x)
\]

\[g(\lambda) = \inf_x \mathcal{L}(x, \lambda)\]

\[g(\lambda) = \mathcal{L}(x^*(\lambda), \lambda)\]

\[
\frac{dg}{d\lambda} = \frac{d\mathcal{L}}{dx^*} \frac{dx^*}{d\lambda} + \frac{d\mathcal{L}}{d\lambda}
\]

if \(x^* = \arg \min_x \mathcal{L}(x, \lambda)\), then \(\frac{d\mathcal{L}}{dx^*} = 0\)!
Digression: dual gradient descent

\[ \min_x f(x) \text{ s.t. } C(x) = 0 \]

\[ \mathcal{L}(x, \lambda) = f(x) + \lambda C(x) \]

\[ g(\lambda) = \mathcal{L}(x^*(\lambda), \lambda) \]

\[ x^* = \arg \min_x \mathcal{L}(x, \lambda) \]

\[ \frac{dg}{d\lambda} = \frac{d\mathcal{L}}{d\lambda}(x^*, \lambda) \]

1. Find \( x^* \leftarrow \arg \min_x \mathcal{L}(x, \lambda) \)

2. Compute \( \frac{dg}{d\lambda} = \frac{d\mathcal{L}}{d\lambda}(x^*, \lambda) \)

3. \( \lambda \leftarrow \lambda + \alpha \frac{dg}{d\lambda} \)
DGD with iterative LQR

This is the constrained problem we want to solve:

$$\min_p \sum_{t=1}^{T} E_{p(x_t, u_t)}[c(x_t, u_t)] \text{ s.t. } D_{KL}(p(\tau)||\bar{p}(\tau)) \leq \epsilon$$

$$D_{KL}(p(\tau)||\bar{p}(\tau)) = \sum_{t=1}^{T} E_{p(x_t, u_t)}[-\log \bar{p}(u_t|x_t) - \mathcal{H}(p(u_t|x_t))]$$

$$\mathcal{L}(p, \lambda) = \sum_{t=1}^{T} E_{p(x_t, u_t)}[c(x_t, u_t) - \lambda \log \bar{p}(u_t|x_t) - \lambda \mathcal{H}(p(u_t|x_t))] - \lambda \epsilon$$
DGD with iterative LQR

\[
\min_p \sum_{t=1}^{T} E_{p(x_t, u_t)}[c(x_t, u_t)] \quad \text{s.t.} \quad D_{KL}(p(\tau) || \bar{p}(\tau)) \leq \epsilon
\]

\[
\mathcal{L}(p, \lambda) = \sum_{t=1}^{T} E_{p(x_t, u_t)}[c(x_t, u_t) - \lambda \log \bar{p}(u_t|x_t) - \lambda \mathcal{H}(p(u_t|x_t))] - \lambda \epsilon
\]

1. Find \( p^* \leftarrow \arg\min_p \mathcal{L}(p, \lambda) \)

2. Compute \( \frac{dg}{d\lambda} = \frac{d\mathcal{L}}{d\lambda}(p^*, \lambda) \)

3. \( \lambda \leftarrow \lambda + \alpha \frac{dg}{d\lambda} \)

this is the hard part, everything else is easy!
DGD with iterative LQR

1. Find $p^* \leftarrow \arg\min_p \mathcal{L}(p, \lambda)$

$$\min_p \sum_{t=1}^{T} E_{p(x_t, u_t)} [c(x_t, u_t) - \lambda \log \bar{p}(u_t | x_t) - \lambda \mathcal{H}(p(u_t | x_t))] - \lambda \epsilon$$

Reminder: Linear-Gaussian solves $\min \sum_{t=1}^{T} E_{p(x_t, u_t)} [c(x_t, u_t) - \mathcal{H}(p(u_t | x_t))]$

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

$$\min_p \sum_{t=1}^{T} E_{p(x_t, u_t)} \left[ \frac{1}{\lambda} c(x_t, u_t) - \log \bar{p}(u_t | x_t) - \mathcal{H}(p(u_t | x_t)) \right]$$

Just use LQR with cost $\tilde{c}(x_t, u_t) = \frac{1}{\lambda} c(x_t, u_t) - \log \bar{p}(u_t | x_t)$
DGD with iterative LQR

\[ \min_p \sum_{t=1}^{T} E_{p(x_t, u_t)}[c(x_t, u_t)] \quad \text{s.t.} \quad D_{KL}(p(\tau) || \bar{p}(\tau)) \leq \epsilon \]

1. Set \( \tilde{c}(x_t, u_t) = \frac{1}{\lambda} c(x_t, u_t) - \log \bar{p}(u_t|x_t) \)

2. Use LQR to find \( p^*(u_t|x_t) \) using \( \tilde{c} \)

3. \( \lambda \leftarrow \lambda + \alpha(D_{KL}(p(\tau) || \bar{p}(\tau)) - \epsilon) \)
Trust regions & trajectory distributions

• Bounding KL-divergences between two policies or controllers, whether linear-Gaussian or more complex (e.g. neural networks) is really useful

• Bounding KL-divergence between policies is equivalent to bounding KL-divergences between trajectory distributions

• We’ll use this later in the course in model-free RL too!
Case study: local models & iterative LQR

Learning Contact-Rich Manipulation Skills with Guided Policy Search

Sergey Levine, Nolan Wagener, Pieter Abbeel

run $p(u_t|x_t)$ on robot
collect $\mathcal{D} = \{r_t\}$

next iteration

fit dynamics $p(x_{t+1}|x_t, u_t)$

improve $p(u_t|x_t)$
Case study: local models & iterative LQR

linear-Gaussian controller learning curves

distance (cm)
samples

(a) (b) (c) (d) (e) (f) (g) (h) (i)
Case study: combining global and local models

One-Shot Learning of Manipulation Skills with Online Dynamics Adaptation and Neural Network Priors

Justin Fu, Sergey Levine, Pieter Abbeel
prior: $\Phi, \mu_0$

empirical estimate: $\hat{\Sigma}, \hat{\mu}$

recent experience $(x_t, u_t, x_{t+1})$

posterior: $\Sigma, \mu$