Iterative LQR and Guided Policy Search

Betty Shea

UBC MLRG

29-July-2020

Outline

Introduction

MLRG theme: Optimal control

Previously

- Optimal control and dynamic systems
  - engineering perspective
  - closed-loop feedback control, stability, controllability, reachability, LQR, etc

MLRG theme: Optimal control

Previously

- Optimal control and dynamic systems
  - engineering perspective
  - closed-loop feedback control, stability, controllability, reachability, LQR, etc
- Classic control and dynamic programming
  - optimization perspective
  - HJB equation, LQG/ LQR/ Ricatti equation, Pontryagin's maximum principle, Kalman filters etc

MLRG theme: Optimal control

Previously

- Optimal control and dynamic systems
  - engineering perspective
  - closed-loop feedback control, stability, controllability, reachability, LQR, etc
- Classic control and dynamic programming
  - optimization perspective
  - HJB equation, LQG/ LQR/ Ricatti equation, Pontryagin's maximum principle, Kalman filters etc

Today: an application.

Focus of talk

A hybrid model-based and model-free algorithm for optimizing objectives in environments with unknown dynamics based on

- iterated linear quadratic Gaussian (iLQG)
- guided policy search (GPS)

Levine, S. & Abbeel, P. (2014) Learning neural network policies with guided policy search under unknown dynamics. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Examples of goals

Robot learns to walk under different conditions (video)

Heess et al. (2017) Emergence of locomotion behaviours in rich environments.

Robot learns to insert a peg into a slot (video)

Levine and Abbeel (2014)

Examples of goals

Robot learns to walk under different conditions (video)

Heess et al. (2017) Emergence of locomotion behaviours in rich environments.

Robot learns to insert a peg into a slot (video)

Levine and Abbeel (2014)

In general (part of a list from Michiel's CPSC533V lecture):

- Drive a car
- Defeat the world champion at Go
- Manage an investment portfolio
- Sequence a series of medical tests and interventions
- Control a power station or a chemical process to maximize revenue
- ...

Optimal control and reinforcement learning goals

General goal of optimal control with unknown dynamics

$$\min_{u_t} \mathbb{E}_{e_t} [l_t[x_t, u_t]] \text{ s.t. } x_{t+1} = f(x_t, u, w_t)$$

where x_0 is given, x_t is the state of the system, u_t is the action, w_t is a random disturbance or noise, and f is some function that gives the new state.

But this is very general and so we will make assumptions that builds off what we learned in the prior two talks.

Optimal control and reinforcement learning goals

General goal of optimal control with unknown dynamics

$$\min_{u_t} \mathbb{E}_{e_t} \left[l_t[x_t, u_t] \right] \text{ s.t. } x_{t+1} = f(x_t, u, w_t)$$

where x_0 is given, x_t is the state of the system, u_t is the action, w_t is a random disturbance or noise, and f is some function that gives the new state.

But this is very general and so we will make assumptions that builds off what we learned in the prior two talks.

Introductory presentation gave a list of problem class. Today's problem fits within the finite horizon-stochastic problem framework where system dynamics are unknown.

Terminology and notation

- State x_t, action u_t
- System dynamics $f(x_t, u_t)$ or $p(x_{t+1}|x_t, u_t)$
- Policy $\pi_\theta(u_t|x_t)$. A function or a distribution parameterized by θ.
- Trajectory $\tau = (x_0, u_1, x_1, u_2, \ldots, x_T)$
- Rollout τ_i. A trajectory created from running a given policy starting at some initial state x_0.
- Cost $l(x_t, u_t)$
- Cost to go $J_t = \int_{t_0}^{T} l(x_t, u_t)dt$ or $J_t = \sum_{k=t}^{T} l(x_k, u_k)$. Sum of remaining costs.

Terminology and notation

- Model-based. Dynamics and losses follow distributions known in advance.
- Model-free. Dynamics and losses not known in advance and discovered through experience.
- Regulation. Minimize error with respect to a reference goal.
- Regulator, controller or policy. Takes in a state and returns an action.

Building blocks

Linear Quadratic Regulator (LQR)

Recall from the prior two talks, LQR is a special case in the deterministic setting where there is a closed form solution. LQR has cost-to-go (for finite horizons)

$$J = x^\intercal Q_f x + \int_{t_0}^{T} (x^\intercal Q x + u^\intercal R u)\, dt \qquad \text{or} \qquad J = \frac{1}{2} x_N^\intercal Q_f x_N + \frac{1}{2} \sum_{k=0}^{N-1} x_k^\intercal Q x_k + u_k^\intercal R u_k$$

where R is symmetric positive definite, Q and Q_f are symmetric.

Linear Quadratic Regulator (LQR)

Recall from the prior two talks, LQR is a special case in the deterministic setting where there is a closed form solution. LQR has cost-to-go (for finite horizons)

$$J = x^\intercal Q_f x + \int_{t_0}^{T} (x^\intercal Q x + u^\intercal R u)\, dt \quad \text{or} \quad J = \frac{1}{2} x_N^\intercal Q_f x_N + \frac{1}{2} \sum_{k=0}^{N-1} x_k^\intercal Q x_k + u_k^\intercal R u_k$$

where R is symmetric positive definite, Q and Q_f are symmetric.

- Q and Q_f are also known as *state cost-weighting* matrices. R as *control cost-weighting* matrix.
- Within the integral/ summation, the first term is the *error cost* and the second term is the *control cost*.
- For linear dynamical systems, i.e. $x_{k+1} = f(x_k, u_k) = Ax_k + Bu_k$, the optimal solution has the form

$$u_k = -L_k x_k \text{ where } L_k = \left(R + B^\intercal V_{k+1} B\right)^{-1} B^\intercal V_{k+1} A$$

i.e. Control is linear of state and L_k is the *control gain*.

Iterated LQR

But in real life, systems are complex and nonlinear.

Idea originally from Li and Todorov (2004)

- "uses iterative linearization of the nonlinear system around a nominal trajectory"
- "computes a locally optimal feedback control law via a modified LQR technique"

Iterated LQR

But in real life, systems are complex and nonlinear.

Idea originally from Li and Todorov (2004)

- "uses iterative linearization of the nonlinear system around a nominal trajectory"
- "computes a locally optimal feedback control law via a modified LQR technique"

In other words

- assume that some localized area on a complex trajectory is linear
- optimize on this linearized local model to get the next step

Derivation of iLQR controller follows closely last week's derivation of LQR.

Iterated LQR: Derivation sketch for controller

The true dynamics are $x_{k+1} = f(x_k, u_k)$ where f is nonlinear.

1. Quadratic approximation of true cost

$$J_0 = \frac{1}{2}(x_N - x^*)^\intercal Q_f (x_N - x^*) + \frac{1}{2}\sum_{k=0}^{N-1} \left(x_k^\intercal Q x_k + u_k^\intercal R u_k\right)$$

2. Denote deviations from u_k and x_k as δu_k and δx_k. Linear approximation to deviations

$$\delta x_{k+1} = A_k \delta x_k + B_k \delta u_k$$

   where $A_k = D_x f(x_k, u_k)$, $B_k = D_u f(x_k, u_k)$ and D_x and D_u are the Jacobian of $f(\cdot)$ wrt x and u at x_k, u_k.

3. Point (2) into point (1) to get modified cost

$$\begin{aligned} J = &\frac{1}{2}(x_N + \delta x_N - x^*)^\intercal Q_f (x_N + \delta x_N - x^*) \\ &+ \frac{1}{2}\sum_{k=0}^{N-1} \left((x_k^\intercal + \delta x_k) Q (x_k + \delta x_k) + (u_k + \delta u_k)^\intercal R (u_k + \delta u_k)\right) \end{aligned}$$

Iterated LQR: Derivation sketch for controller

4. Define Hamiltonian function with Lagrangian multiplier.
5. Define boundary conditions, state equations, stationary conditions to get Hamiltonian system.
6. Assume $\delta\lambda_k = S_k \delta x_k + v_k$ for some unknown sequence S_k and v_k.
7. Solve Hamiltonian system to get

$$
\begin{aligned}
\delta u_k &= -K\delta x_k - K_v v_{k+1} - K_u u_k \\
K &= \left(B_k^\intercal S_{k+1} B_k + R\right)^{-1} B_k^\intercal S_{k+1} A_k \\
K_v &= \left(B_k^\intercal S_{k+1} B_k + R\right)^{-1} B_k^\intercal \\
K_u &= \left(B_k^\intercal S_{k+1} B_k + R\right)^{-1} R \\
S_k &= A_k^\intercal S_{k+1}(A_k - B_k K) + Q \\
v_k &= (A_k - B_k K)^\intercal v_{k+1} - K^\intercal R u_k + Q x_k
\end{aligned}
$$

8. $u_k^* = u_k + \delta u_k$ is optimal controller.

For details, refer to Li and Todorov (2004) which walks through the derivation step-by-step.

Iterated LQR: Optimal control

$$\delta u_k = -K\delta x_k - K_v v_{k+1} - K_u u_k$$

Main takeaway:

1. Control gain of iLQR (i.e. K) depends on the control gain in regular LQR (i.e. L_k or the Ricatti equation).
2. Unlike LQR, the optimal controller of iLQR is not linear (of state) but is composed of a term that is linear and two additional terms.

From iterated LQR to iterated LQG

- Main paper needs a stochastic environment to generate samples
- Main algorithm builds off iterated linear quadratic Gaussian (iLQG) and not iLQR.
- Recall from last week, LQG assumes linear dynamics with additive Gaussian noise, i.e. a controlled Ito diffusion

$$dx = f(x, u)dt + F(x, u)dw$$

  where $f(x, u)$ is the drift and $F(x, u)$ is the diffusion coefficient.
- Also from last week, the LQG optimal controller can be derived in closed form using the HJB equation.

To understand iLQG, you need two additional papers:
Todorov and Li (2005) and Tassa et al. (2012).

The first paper derives the iLQG optimal controller and the second paper explains how to stabilize the sequence of trajectories generated.

Iterated LQG: Optimal control and stability

Relevant points:

- iLQG uses a quadratic approximation to the cost-to-go function of LQG

Iterated LQG: Optimal control and stability

Relevant points:

- iLQG uses a quadratic approximation to the cost-to-go function of LQG
- The optimal iLQG controller is $p(x_t) = \hat{u}_t + k_t + K_t(x_t - \hat{x}_t)$ where $\hat{x}_t$ and $\hat{u}_t$ are the states and actions of the current trajectory, and terms k_t and K_t are such that the overall controller is linear

Iterated LQG: Optimal control and stability

Relevant points:

- iLQG uses a quadratic approximation to the cost-to-go function of LQG
- The optimal iLQG controller is $p(x_t) = \hat{u}_t + k_t + K_t(x_t - \hat{x}_t)$ where $\hat{x}_t$ and $\hat{u}_t$ are the states and actions of the current trajectory, and terms k_t and K_t are such that the overall controller is linear
- The fitted dynamics are only valid in a local region around the samples but the new controller generated by iLQG can be arbitrarily different from the old one.

Iterated LQG: Optimal control and stability

Relevant points:

- iLQG uses a quadratic approximation to the cost-to-go function of LQG
- The optimal iLQG controller is $p(x_t) = \hat{u}_t + k_t + K_t(x_t - \hat{x}_t)$ where $\hat{x}_t$ and $\hat{u}_t$ are the states and actions of the current trajectory, and terms k_t and K_t are such that the overall controller is linear
- The fitted dynamics are only valid in a local region around the samples but the new controller generated by iLQG can be arbitrarily different from the old one.
- Large changes in trajectories between iterations cause the algorithm to quickly fall into unstable, costly parts of the state space, preventing convergence.

Iterated LQG: Optimal control and stability

Relevant points:

- iLQG uses a quadratic approximation to the cost-to-go function of LQG
- The optimal iLQG controller is $p(x_t) = \hat{u}_t + k_t + K_t(x_t - \hat{x}_t)$ where $\hat{x}_t$ and $\hat{u}_t$ are the states and actions of the current trajectory, and terms k_t and K_t are such that the overall controller is linear
- The fitted dynamics are only valid in a local region around the samples but the new controller generated by iLQG can be arbitrarily different from the old one.
- Large changes in trajectories between iterations cause the algorithm to quickly fall into unstable, costly parts of the state space, preventing convergence.
- The original iLQG algorithm uses a line search to bound the difference between the old and new generated trajectory.

Policy Search

States and actions could continuous or high dimension and so we would like to parameterize value functions into policies. Direct policy search

- Finds parameters θ that optimizes expected reward $J(\theta)$
- Uses methods such as policy gradients, finite-differences or evolutionary methods

Policy Search

States and actions could continuous or high dimension and so we would like to parameterize value functions into policies. Direct policy search

- Finds parameters θ that optimizes expected reward $J(\theta)$
- Uses methods such as policy gradients, finite-differences or evolutionary methods

What if θ itself is high dimension? Direct policy search may

- require many samples. Can be addressed with importance sampling.
- fall into local minima

Guided Policy Search

Guided policy search (Levine and Koltun 2013)

- uses a variant of importance sampling
- uses iLQG to generate guiding samples offline. iLQG as 'differential dynamic programming (DDP)' in the paper.

Guided Policy Search: importance sampling

Uses a variant of importance sampling where the estimator has an additional regularizing term.

$$\Phi(\theta) = \sum_{t=1}^{T} \left[\frac{1}{Z_t(\theta)} \sum_{i=1}^{m} \frac{\pi_\theta(\xi_{i,1:t})}{q(\xi_{i,1:t})} r(x_t^i, u_t^i) + w_r \log Z_t(\theta) \right]$$

where $\xi_{i,1:t}$ is the *ith* truncated rollout to time t, r is the reward function, $Z_t(\theta)$ normalizes the weights, $q(x)$ is a different distribution where samples are drawn.

Guided Policy Search: importance sampling

Uses a variant of importance sampling where the estimator has an additional regularizing term.

$$\Phi(\theta) = \sum_{t=1}^{T} \left[\frac{1}{Z_t(\theta)} \sum_{i=1}^{m} \frac{\pi_\theta(\xi_{i,1:t})}{q(\xi_{i,1:t})} r(x_t^i, u_t^i) + w_r \log Z_t(\theta) \right]$$

where $\xi_{i,1:t}$ is the *ith* truncated rollout to time t, r is the reward function, $Z_t(\theta)$ normalizes the weights, $q(x)$ is a different distribution where samples are drawn.

Regularizing term acts as a soft maximum over the log weights, ensuring that at least some samples have a high probability under π_θ.

Guided Policy Search: iLQG

Recall iLQG controller is a linear controller $p(x_t) = \hat{u}_t + k_t + K_t(x_t - \hat{x}_t)$

Guided Policy Search: iLQG

Recall iLQG controller is a linear controller $p(x_t) = \hat{u}_t + k_t + K_t(x_t - \hat{x}_t)$

Under linear dynamics and quadratic rewards, the quadratic cost-to-go function in iLQG can also be minimized by a linear-Gaussian controller

$$\pi_G(u_t|x_t) = \mathcal{N}(\hat{u}_t + k_t + K_t(x_t - \hat{x}_t), Q_{u,ut}^{-1})$$

where the covariance term $Q_{u,ut}^{-1}$ is the inverse Hessian of the quadratic form of the cost-to-go with respect to action u.

Guided Policy Search: iLQG

Recall iLQG controller is a linear controller $p(x_t) = \hat{u}_t + k_t + K_t(x_t - \hat{x}_t)$

Under linear dynamics and quadratic rewards, the quadratic cost-to-go function in iLQG can also be minimized by a linear-Gaussian controller

$$\pi_G(u_t|x_t) = \mathcal{N}(\hat{u}_t + k_t + K_t(x_t - \hat{x}_t), Q_{u,ut}^{-1})$$

where the covariance term $Q_{u,ut}^{-1}$ is the inverse Hessian of the quadratic form of the cost-to-go with respect to action u.

Use this linear-Gaussian controller to generate guiding samples.

Guided Policy Search

Algorithm 1 Guided Policy Search

1: Generate DDP solutions $\pi_{\mathcal{G}_1}, \ldots, \pi_{\mathcal{G}_n}$
2: Sample $\zeta_1, \ldots, \zeta_m$ from $q(\zeta) = \frac{1}{n} \sum_i \pi_{\mathcal{G}_i}(\zeta)$
3: Initialize $\theta^\star \leftarrow \arg\max_\theta \sum_i \log \pi_{\theta^\star}(\zeta_i)$
4: Build initial sample set $\mathcal{S}$ from $\pi_{\mathcal{G}_1}, \ldots, \pi_{\mathcal{G}_n}, \pi_{\theta^\star}$
5: **for** iteration $k = 1$ to K **do**
6: Choose current sample set $\mathcal{S}_k \subset \mathcal{S}$
7: Optimize $\theta_k \leftarrow \arg\max_\theta \Phi_{\mathcal{S}_k}(\theta)$
8: Append samples from π_{θ_k} to $\mathcal{S}_k$ and $\mathcal{S}$
9: Optionally generate adaptive guiding samples
10: Estimate the values of π_{θ_k} and $\pi_{\theta^\star}$ using $\mathcal{S}_k$
11: **if** π_{θ_k} is better than $\pi_{\theta^\star}$ **then**
12: Set $\theta^\star \leftarrow \theta_k$
13: Decrease w_r
14: **else**
15: Increase w_r
16: Optionally, resample from $\pi_{\theta^\star}$
17: **end if**
18: **end for**
19: Return the best policy $\pi_{\theta^\star}$

- Line 1 uses the linear-Gaussian controller to generate guiding samples
- Line 2 generates m rollouts
- Line 4 Builds sample set from guiding samples and samples from initial policy π_{θ^*}
- Line 6 performs importance sampling
- Line 7 uses LBFGS to find optimal (for S_k) parameters θ_k which gives optimized policy π_{θ_k} for S_k
- Line 8 add samples from optimized π_{θ_k} to S_k and S
- If policy π_{θ_k} is better than policy π_{θ^*}, replace parameters θ^* with θ_k

Pause

Those are the main building blocks.

Any questions/ comments before we move on to the algorithm in the main paper?

Main algorithm

Overview of algorithm

Conceptually, there are three main components to the algorithm

1. Optimize time-varying linear-Gaussian controllers (i.e. iLQG-like controllers) and use this to generate rollouts.
2. Fit a Gaussian mixture model (GMM) to the distribution of trajectories.
3. Optimize policy parameters using GPS.

This is a hybrid approach because item 1 is model-based and item 3 is model-free. Item 2 is there to reduce sample complexity.

Algorithm

Algorithm 1 Guided policy search with unknown dynamics

1: **for** iteration $k = 1$ to K **do**
2: Generate samples $\{\tau_i^j\}$ from each linear-Gaussian controller $p_i(\tau)$ by performing rollouts
3: Fit the dynamics $p_i(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$ to the samples $\{\tau_i^j\}$
4: Minimize $\sum_{i,t} \lambda_{i,t} D_{\mathrm{KL}}(p_i(\mathbf{x}_t)\pi_\theta(\mathbf{u}_t|\mathbf{x}_t) \| p_i(\mathbf{x}_t, \mathbf{u}_t))$ with respect to θ using samples $\{\tau_i^j\}$
5: Update $p_i(\mathbf{u}_t|\mathbf{x}_t)$ using the algorithm in Section 3 and the supplementary appendix
6: Increment dual variables $\lambda_{i,t}$ by $\alpha D_{\mathrm{KL}}(p_i(\mathbf{x}_t)\pi_\theta(\mathbf{u}_t|\mathbf{x}_t) \| p_i(\mathbf{x}_t, \mathbf{u}_t))$
7: **end for**
8: **return** optimized policy parameters θ

Indices: i for controller/ rollout. j is probably k

Line 2: Generate sample trajectories

- Generate samples $\{\tau_i^j\}$ from each linear-Gaussian controller $p_i(\tau)$ by performing rollouts. In other words, each linear-Gaussian controller interacts with the live system to generate trajectories $\tau_i = \{x_{1i}, u_{1i}, \ldots, x_{Ti}, u_{Ti}\}$.

Line 2: Generate sample trajectories

- Generate samples $\{\tau_i^j\}$ from each linear-Gaussian controller $p_i(\tau)$ by performing rollouts. In other words, each linear-Gaussian controller interacts with the live system to generate trajectories $\tau_i = \{x_{1i}, u_{1i}, \ldots, x_{Ti}, u_{Ti}\}$.
- Rollouts start from different initial states or in different conditions.

Line 2: Generate sample trajectories

- Generate samples $\{\tau_i^j\}$ from each linear-Gaussian controller $p_i(\tau)$ by performing rollouts. In other words, each linear-Gaussian controller interacts with the live system to generate trajectories $\tau_i = \{x_{1i}, u_{1i}, \ldots, x_{Ti}, u_{Ti}\}$.
- Rollouts start from different initial states or in different conditions.
- Note that during training, this is the only step that interacts with the system.

Line 2: Generate sample trajectories

- Generate samples $\{\tau_i^j\}$ from each linear-Gaussian controller $p_i(\tau)$ by performing rollouts. In other words, each linear-Gaussian controller interacts with the live system to generate trajectories $\tau_i = \{x_{1i}, u_{1i}, \ldots, x_{Ti}, u_{Ti}\}$.
- Rollouts start from different initial states or in different conditions.
- Note that during training, this is the only step that interacts with the system.
- The authors argue that this is appealing from a safety point of view when the initial parameterized policy is unstable because stabilizing linear-Gaussian controllers is easier than stabilizing arbitrary policies.

Line 3: Use generated samples to model dynamics

- The linear-Gaussian controllers $p_i(\tau)$ induce approximately Gaussian distribution over trajectories. Use this to fit dynamics $p(x_{t+1}|x_t, u_t)$.
- This is where the mixture of Gaussian models prior on the distribution of trajectories is used for sample efficiency.

Line 4: Policy optimization using GPS

A supervised learning step involving a constrained version of GPS (Levine and Koltun 2014).

Enforces agreement between policy and trajectory by means of a soft KL-divergence constraint.

We use the more standard expected cost objective, resulting in the following optimization objective

$$\min_{\theta, p(\tau)} E_{p(\tau)}[l(\tau)] \text{ s.t. } D_{KL}(p(x_t)\pi_\theta(u_t|x_t)\|p(x_t, u_t)) = 0 \quad \forall t$$

Line 4: Policy optimization using GPS

Turn constrained GPS to unconstrained with Lagrangian

$$\mathcal{L}_{GPS}(\theta, p, \lambda) = E_{p(\tau)}[l(\tau)] + \sum_{t=1}^{T} \lambda_t D_{KL}(p(x_t)\pi_\theta(u_t|x_t) \| p(x_t, u_t))$$

where

$$p(\tau) = \underset{p(\tau)\in\mathcal{N}(\tau)}{\arg\min} \; E_p[l(\tau)] - \mathcal{H}(p(\tau)) \text{ s.t. } p(x_{t+1}|x_t, u_t) = \mathcal{N}(x_{t+1} : \nabla f_{x,t} \cdot x_t + \nabla f_{u,t} \cdot u_t, F_t)$$

where $\mathcal{H}$ is the differential entropy, which is an attempt to extend the idea of Shannon entropy to continuous probability distributions. Guided policy search

uses alternating optimization:

1. Optimizing $\mathcal{L}_{GPS}$ wrt to $p(\tau)$ corresponds to trajectory optimization, which in our case involves dual gradient descent on $\mathcal{L}_{traj}$ in Section 3.1,
2. and optimizing with respect to θ corresponds to supervised policy optimization to minimize the weighted sum of KL-divergence.

Line 5: Optimize linear Gaussian controllers

- This step performs trajectory optimization with unknown dynamics step and is the part of the paper that is novel.
- Now that the dynamics have been updated, we need to update the linear-Gaussian controllers $p_i(u_t|x_t) = \mathcal{N}(K_t x_t + k_t, C_t)$ that will generate new trajectories at the start of the next iteration of the loop.
- We also want to exploit the linear-Gaussian structure and use dynamic programming. That would make this step more efficient than general policy optimization.
- But we cannot use iLQG directly. iLQG uses a line search to ensure stability, i.e. new trajectory generated is not too different from old. But a line search 'is impractical when the rollouts must be stochastically sampled from the real system'.

Line 5: Optimize linear Gaussian controllers

Instead, authors impose constraint on the change in the KL divergence of controller between iterations

$$\min_{p(\tau)\in\tau} E_p[l(\tau)] \text{ s.t. } D_{KL}(p(\tau)\|\hat{p}(\tau)) \leq \epsilon$$

Again, use Lagrangian to go from constrained to unconstrained optimization

$$\mathcal{L}_{traj}(p(\tau),\eta) = E_p[l(\tau)] + \eta\,[D_{KL}(p(\tau)l\hat{p}(\tau)) - \epsilon]$$

Authors show that $\mathcal{L}_{traj}$ can still use DP to optimize $p(\tau)$.

So this step involves

- Run gradient descent to optimize η
- Run DP to optimize $p(\tau)$.

Line 6: Proxy for dual gradient descent (practical detail)

In practice, the algorithm does not use dual gradient descent to update the dual variable $\lambda_{i,t}$ in line 4.

It takes too long ('unnecessary and costly') to optimize $\mathcal{L}_{GPS}(\theta, p, \lambda)$ wrt $p(\tau)$ and to θ to convergence before each update of λ.

Instead, they just increment every λ_i after each iteration with a multiple α of the constraint of the original (constrained) objective. This approach amounts to using a penalty method instead.

Use a Gaussian mixture model prior (practical detail)

- Sample efficiency generally depends on system dimension. But could use priors to reduce sample complexity.
- Recall that the distribution of sampled trajectories is roughly Gaussian. This suggests a mixture of Gaussian model (GMM).
- If we use the GMM to obtain a prior for linear regression, it is easy to determine the correct linear model from the covariance of (x_{ti}, u_{ti}) with x_{t+1i} in the current samples at time step t.
- Construct GMM prior using experience tuples $\{x_{t+1}, x_t, u_t\}$ from samples.
- GMM is used to produce a normal-inverse-Wishart prior for the mean and covariance of this Gaussian at each time step.
- Authors claimed that GMM prior reduces the number of samples needed at each iteration by a factor of 4 to 8.

Experimental results

Comparison methods

- iLQG with a known model. Baseline.
- REPS(100s) and REPS(20+500). Model-free method that enforces KL-divergence constraint between new and old policy. The 500 is the number of pseudo-samples.
- Reward-weighted regression. (RWR). An EM algorithm that fits the policy to previous samples weighted by the exponential of their reward.
- Cross-entropy method. (CEM) Fits policy to the best samples in each batch.
- PILCO. Model-based method that uses a Gaussian process to learn a global dynamics model that is used to optimize the policy.

Experimental results

Comparison tasks

- Insertion tasks. Test method's ability to handle discontinuities.
- Octopus arm control. Test method's ability to handle high dimensionalities.
- Swimming and walking tasks. Challenge comes from underactuation.

Experimental results

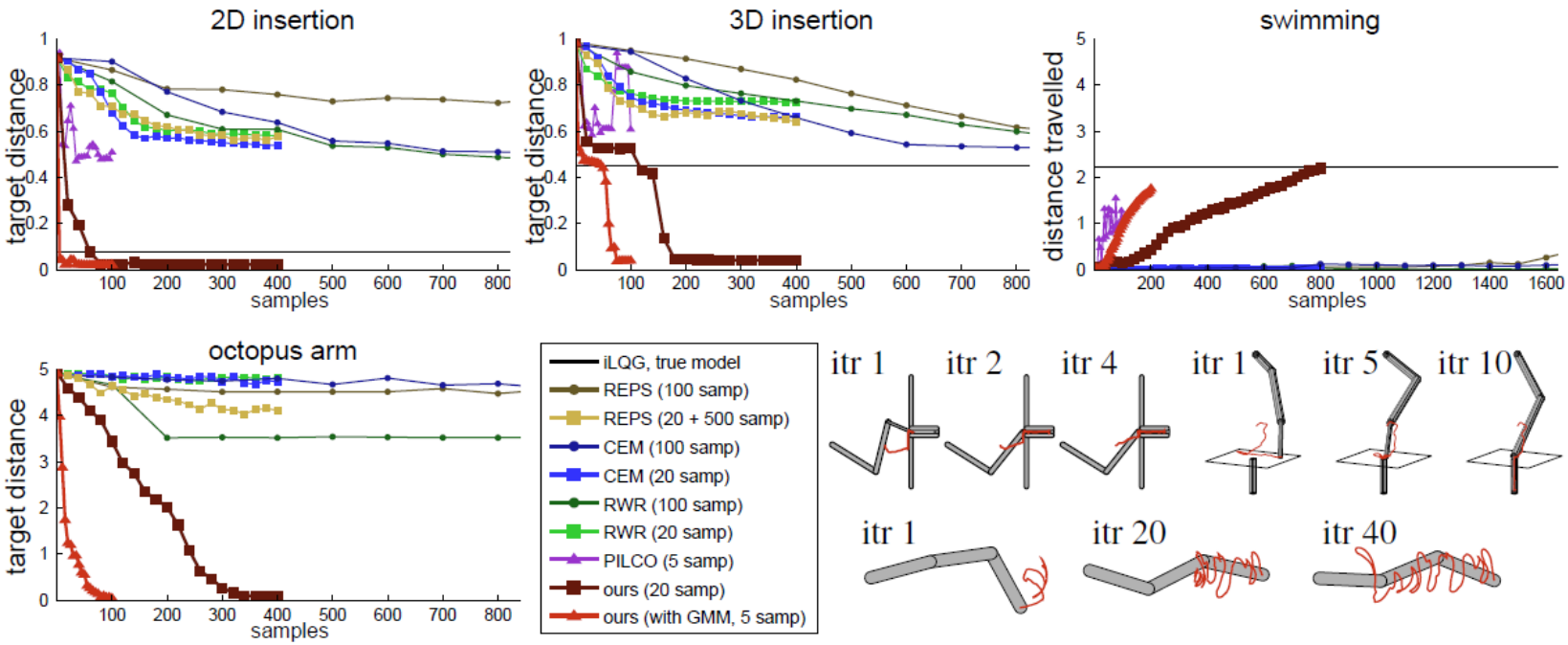

Figure 1: Results for learning linear-Gaussian controllers for 2D and 3D insertion, octopus arm, and swimming. Our approach uses fewer samples and finds better solutions than prior methods, and the GMM further reduces the required sample count. Images in the lower-right show the last time step for each system at several iterations of our method, with red lines indicating end effector trajectories.

Experimental results

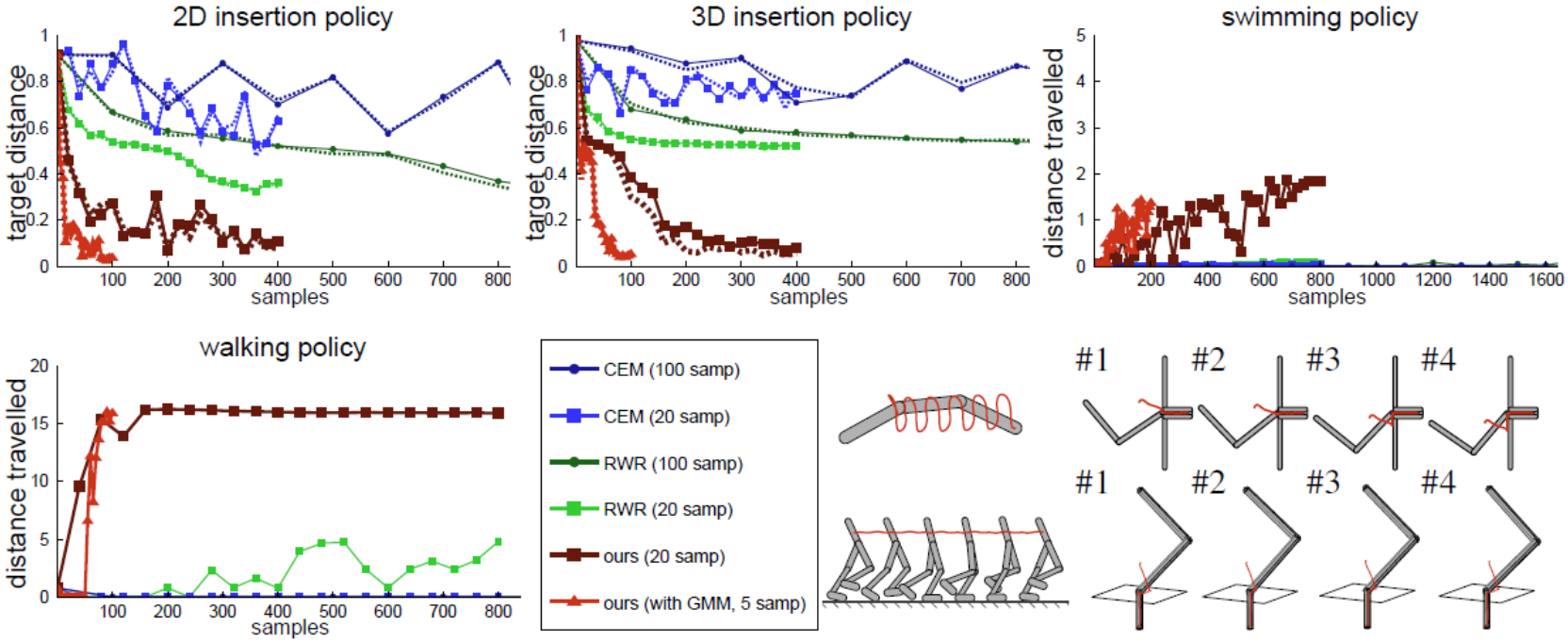

Figure 2: Comparison on neural network policies. For insertion, the policy was trained to search for an unknown slot position on four slot positions (shown above), and generalization to new positions is graphed with dashed lines. Note how the end effector (in red) follows the surface to find the slot, and how the swimming gait is smoother due to the stationary policy (also see supplementary video).

Questions/ discussion

Questions/ discussion

Thank you.

References

- Heess, N., Sriram, S., Lemmon, J., Merel, J., Wayne, G., Tassa, Y., Erez, T., Wang, Z., Eslami, A., Riedmiller, M., & Silver, D. (2017) Emergence of locomotion behaviours in rich environments. In *arXiv:1707.02286*.
- Levine, S. (2018) CS 285 lectures. `http://rail.eecs.berkeley.edu/deeprlcourse`
- Levine, S. & Abbeel, P. (2014) Learning neural network policies with guided policy search under unknown dynamics. In *Advances in Neural Information Processing Systems (NeurIPS)*.
- Levine, S. & Koltun, V. (2013) Guided policy search. In *International Conference on Machine Learning (ICML)*.
- Levine, S. & Koltun, V. (2014) Learning complex neural network policies with trajectory optimization. In *International Conference on Machine Learning (ICML)*.
- Li, W. & Todorov, E. (2004) Iterative Linear Quadratic Regulator Design for Nonlinear Biological Movement Systems. In *ICINCO (1)*, pages 222-229.
- Tassa, Y., Erez, T. & Todorov, E. (2012) Synthesis and stabilization of complex behaviors through online trajectory optimization. In *IEEE/ RSJ International Conference on Intelligent Robots and Systems*.
- Todorov, E. & Li, W. (2005) A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *Proceedings of the 2005 American Control Conference*.
- Van de Panne, M. (2020) CPSC 533-V lectures. `https://www.cs.ubc.ca/~van/cpsc533/index.html`